Intel® oneAPI Math Kernel Library (oneMKL) - Data Parallel C++ Developer Reference

December 1, 2022

Release 2023.0

Contents

The Intel® oneAPI Math Kernel Library (oneMKL) improves performance with math routines for software applications that solve large computational problems. oneMKL provides BLAS and LAPACK linear algebra routines, fast Fourier transforms, vectorized math functions, random number generation functions, and other functionality.

What's New

This publication describes the Data Parallel C++ (DPC++) interface.

DPC++ is ISO C++ plus Khronos SYCL with Intel extensions. For more information, see Intel® oneAPI DPC++ Compiler.

Basic Linear Algebra Subprograms (BLAS) The BLAS routines provide vector, matrix-vector, and matrix-matrix operations.

Sparse BLAS The Sparse BLAS routines provide basic operations on sparse vectors and matrices.

LAPACK The LAPACK routines solve systems of linear equations, least square problems, eigenvalue and singular value problems, and Sylvester's equations.

Random Number Generators The Random Number Generators provides a set of routines implementing commonly used pseudorandom and quasi-random generators with continuous and discrete distributions.

Summary Statistics Summary Statistics provides routines that compute basic statistical estimates for single and double precision multi-dimensional datasets.

Vector Mathematics Functions The Vector Mathematical Functions compute core mathematical functions on vector arguments.

Fourier Transform Functions The Fourier Transform Functions offer several options for computing Fast Fourier Transforms (FFTs).

Data Fitting The Data Fitting provides spline-based interpolation capabilities that can be used for spline construction (Linear, Cubic, Quadratic etc.), to perform cell-search operations, and to approximate functions, function derivatives, or integrals.

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

1.0 What's New

This developer reference documents the Intel® oneAPI Math Kernel Library (oneMKL) release for the Data Parallel C++ (DPC++) interface. The manual has been updated to reflect enhancements to the product as well as improvements and error corrections.

Product and Performance Information
Performance varies by use, configuration, and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

2.0 Introduction to the Intel® oneAPI Math Kernel Library (oneMKL) BLAS and LAPACK with DPC++

This guide provides an overview of the Intel® oneAPI Math Kernel Library (oneMKL) BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra Package) application programming interfaces for the Data Parallel C++ (DPC++) implementation of SYCL. It is aimed at users who have had some prior experience with the standard BLAS and LAPACK APIs.

In general, the DPC++ APIs for BLAS and LAPACK are similar to the standard BLAS and LAPACK APIs, sharing the same routine names and argument orders. Unlike standard routines, however, DPC++ routines are designed to run asynchronously on a compute device (CPU or GPU) and typically use device memory for inputs and outputs. To support this functionality, the data types of many arguments have changed and each routine takes an additional argument (a DPC++ queue), which specifies where the routine should be executed. There are several smaller API changes that are described below.

In oneMKL, all DPC++ routines and associated data types belong to the `oneapi::mkl` namespace. CPU-based oneMKL routines are still available via the C interface (which uses the global namespace). Additionally, each BLAS-specific routine is in the `oneapi::mkl::blas`, `oneapi::mkl::blas::column_major`, and `oneapi::mkl::blas::row_major` namespaces.

By default, column major layout is assumed for all BLAS functions in the `oneapi::mkl::blas` namespace. BLAS functions in the `oneapi::mkl::blas::column_major` namespace can also be used when matrices are stored using column major layout. To use row major layout to store matrices, BLAS functions in the `oneapi::mkl::blas::row_major` namespace must be used. For example, `oneapi::mkl::blas::gemm` is the DPC++ routine for matrix multiplication using column major layout for storing matrices, while `::{cblas_}{s, d, c, z}gemm` is the traditional CPU-based version. Currently, LAPACK DPC++ APIs do not support matrices stored using row major layout.

2.1 Differences between Standard BLAS/LAPACK and DPC++ oneMKL APIs

2.1.1 Naming—BLAS Only

DPC++ BLAS APIs are templated on precision. For example, unlike standard BLAS API having four different routines for GEMM computation with names based on precision (`sgemm`, `dgemm`, `cgemm` and `zgemm`), the DPC++ BLAS has only one entry point for GEMM computation named `gemm` accepting `float`, `double`, `half`, `bfloat16`, `std::complex<float>`, and `std::complex<double>` data types.

2.1.2 References

All DPC++ objects (buffers and queues) are passed by reference, rather than by pointer. Other parameters are typically passed by value.

2.1.3 Queues

Every DPC++ BLAS and LAPACK routine has an extra parameter at the beginning: A DPC++ queue (type `queue&`), where computational tasks are submitted. A queue can be associated with the host device, a CPU device, or a GPU device. In the pre-alpha release of DPC++, the CPU and GPU devices are supported for all BLAS functions. Refer to the **Overview of Intel® oneAPI Math Kernel Library (oneMKL) LAPACK for DPC++** documentation for a complete list of supported LAPACK functions on CPU and GPU devices.

2.1.4 Vector and Matrix Types

DPC++ has two APIs for storing data on a device and sharing data between devices and the host: the buffer API and the unified shared memory (USM) API. DPC++ BLAS and LAPACK routines support both APIs.

With the buffer API, vector and matrix inputs to DPC++ BLAS and LAPACK routines are DPC++ buffer types. Currently, all buffers must be one-dimensional, but you can use DPC++'s `buffer::reinterpret()` member function to convert a higher-dimensional buffer to a one-dimensional one.

For the USM API, vector and matrix inputs to DPC++ BLAS and LAPACK routines are pointers of the appropriate type, but the pointers must point to memory allocated by one of the DPC++ USM allocation routines (eg `malloc_host`, `malloc_shared`, or `malloc_device`). Memory that is allocated with the usual `malloc` or `new` routines cannot be used in the Intel® oneAPI Math Kernel Library (oneMKL) DPC++ interfaces.

For example, the `gemv` routine takes a matrix `A` and vectors `x`, `y`. For the real double precision case, each of these parameters has types:

- `double*` in standard BLAS;
- `buffer<double,1>&` in DPC++ BLAS with the buffer API;
- `double*` in DPC++ BLAS with the USM API, with the restriction that the memory the pointer refers to must be allocated in a device-accessible way using a DPC++ USM allocation routine.

2.1.5 Scalars

Scalar inputs are passed by value for all BLAS functions.

2.1.6 Complex Numbers

In DPC++, complex numbers are represented with C++ `std::complex` types. For instance, `MKL_Complex8` can be replaced by `std::complex<float>`.

This is true for scalar, vector, and matrix arguments. For instance, a double-precision complex vector would have type `buffer<std::complex<double>,1>`.

2.1.7 Return Values

Some BLAS and LAPACK routines (`dot`, `nrm2`, `lapy2`, `asum`, `iamax`) return a scalar result as their return value. In DPC++, to support asynchronous computation, these routines take an additional buffer argument that occurs at the end of the argument list. The result value is stored in this buffer when the computation completes. These routines, like the other DPC++ routines, have a return type of `void`.

2.1.8 Computation Options (Character Parameters)

Standard BLAS and LAPACK use special alphabetic characters to control operations: transposition of matrices, storage of symmetric and triangular matrices, etc. In DPC++, these special characters are replaced by scoped enum types for extra type safety.

For example, the BLAS matrix-vector multiplication `dgemv` takes a character argument `trans`, which can be one of `N` or `T`, specifying whether the input matrix `A` should be transposed before multiplication.

In DPC++, `trans` is a member of the scoped `enum` type `oneapi::mkl::transpose`. You can use the traditional character-based names `oneapi::mkl::transpose::N` and `oneapi::mkl::transpose::T`, or the equivalent, more descriptive names `oneapi::mkl::transpose::nontrans` and `oneapi::mkl::transpose::trans`.

See the Data Types for more information on the new types.

2.2 Matrix Layout (Row Major and Column Major)

The standard BLAS and LAPACK APIs require a Fortran layout for matrices (column major), where matrices are stored column-by-column in memory and the entries in each column are stored in consecutive memory locations. oneMKL for DPC++ likewise assumes this matrix layout. Row major layout is not supported directly, but you can still use DPC++ BLAS by treating row major matrices as transposed column major matrices.

2.2.1 Example for BLAS

Below is a short excerpt of a program calling standard BLAS `dgemm`:

```
double *A = …, *B = …, *C = …;
double alpha = 2.0, beta = 3.0;


int m = 16, n = 20, k = 24;
int lda = m, ldb = n, ldc = m;


dgemm(“N”, “T”, &m, &n, &k, &alpha, A, &lda, B, &ldb, &beta, C, &ldc);
```

The DPC++ equivalent of this excerpt would be as follows:

```
using namespace cl::sycl;
using namespace mkl;


queue Q(…);
buffer A = …, B = …, C = …;
int m = 16, n = 20, k = 24;
int lda = m, ldb = n, ldc = m;
blas::gemm(Q, transpose::N, transpose::T, m, n, k, 2.0, A, lda, B, ldb, 3.0, C, ldc);
```

2.2.2 Example for LAPACK

Below is a short excerpt of a program calling a standard LAPACK `dgetrf` (LU factorization):

```
double *A = …;


MKL_INT m = 16, n = 20;
MKL_INT lda = m;
MKL_INT ipiv[m];


dgetrf(&m, &n, A, &lda, ipiv, &info);
```

The DPC++ equivalent of this excerpt would be as follows:

```
using namespace cl::sycl;
using namespace mkl;


queue Q(…);
buffer<double,1> A = …;
int64_t m = 16, n = 20;
int64_t lda = m;
buffer<int64_t,1> ipiv(range<1>(m));


dgetrf(Q, m, n, A, lda, ipiv, info);
```

3.0 Overview of Intel® oneMKL BLAS Routines for Data Parallel C++

The following pages describe the oneMKL BLAS routines for Data Parallel C++ (DPC++), all of which are declared in the header file `oneapi/mkl/blas.hpp`.

Several conventions are used throughout this document:

- All oneMKL for DPC++ data types and non domain specific functions are inside the `oneapi::mkl::` namespace.
- All oneMKL BLAS functions for DPC++ are inside the `oneapi::mkl::blas` namespace.
- The routines are templated on precision. Each routine has a table detailing the supported precisions.

3.1 Device Support

DPC++ supports the following types of devices:

- CPU device: Performs computations on a CPU using OpenCL™.
- GPU device: Performs computations on a GPU.

In the current release of oneMKL BLAS for DPC++, all standard Level1, Level2, and Level3 BLAS routines and the BLAS extensions support CPU and GPU devices.

4.0 Overview of Intel® oneAPI Math Kernel Library (oneMKL) Sparse BLAS for DPC++

The following pages describe the oneMKL Sparse BLAS computational routines for DPC++ in detail. These routines, along with other helper routines (see **Sparse BLAS Routines** for the full list), are declared in the header file `oneapi/mkl/spblas.hpp`.

Several conventions are used throughout this document:

- All oneMKL DPC++ functions and data types are inside the `mkl` namespace.
- For brevity, the `sycl` namespace is omitted from DPC++ object types such as buffers and queues. For example, a single-precision, 1D buffer `A` would be written `buffer<float,1> &A` instead of `sycl::buffer<float,1> &A`.
- The routines are overloaded on precision. Each routine has a table detailing the supported precisions.

4.1 Device Support

DPC++ supports several types of devices:

- CPU device: Performs computations on a CPU using OpenCL™.
- GPU device: Performs computations on a GPU using OpenCL™ or Level Zero.

Each routine details the device types that are currently supported.

In the current release of oneMKL DPC++ Sparse BLAS, all listed routines support use on CPU and GPU devices (unless otherwise noted) with the Compressed Sparse Row (CSR) matrix format.

Routine	Data Types	Description
Level 2:		
sparse::gemv	float, double	General sparse matrix-dense vector product
sparse::gemvdot	float, double	General sparse matrix-dense vector product with fused dot product
sparse::symv	float, double	Symmetric sparse matrix-dense vector product
sparse::trmv	float, double	Triangular sparse matrix-dense vector product
sparse::trsv	float, double	Triangular solve of sparse matrix against a dense vector.
Level 3:		
sparse::gemm	float, double	General sparse matrix-dense matrix product with dense matrix output
sparse::matmat	float, double	General sparse matrix-sparse matrix product with sparse matrix output. Supports CPU and GPU devices.

5.0 Overview of Intel® oneAPI Math Kernel Library (oneMKL) LAPACK for DPC++

The following pages describe the oneMKL LAPACK routines for DPC++ in detail, all of which are declared in the header file `oneapi/mkl/lapack.hpp`.

Several conventions are used throughout this document:

- All oneMKL for DPC++ functions and data types are inside the `mkl` namespace.
- For brevity, the `cl::sycl` namespace is omitted from DPC++ object types, such as buffers and queues. For example a single-precision, 1D buffer `A` would be written `buffer<float,1> &A` instead of `sycl::buffer<float,1> &A`.
- A question mark `(?)` in a routine name stands for one or more characters (typically one of s, d, c, z) specifying the precision of the operation. Each routine has a table detailing the supported precisions and associated routine names.

5.1 Device Support

DPC++ supports several types of devices:

- CPU device: Performs computations on a CPU using OpenCL™.
- GPU device: Performs computations on a GPU.

Each routine details the device types are currently supported.

In the current release of oneMKL LAPACK for DPC++, all routines support at least CPU devices. GPU devices are supported in the current release only for the following routines:

Routines
`getrf` (all precisions)
`getrs` (all precisions)
`trtrs` (all precisions)
`potrf` (real precisions)
`potrs` (real precisions)
`getri` (real precisions)

6.0 Data Types

6.1 BLAS and LAPACK Data Types

Intel® oneAPI Math Kernel Library (oneMKL) BLAS and LAPACK for Data Parallel C++ (DPC++) introduces several new enumeration data types, which are type-safe versions of the traditional Fortran characters in BLAS and LAPACK. They are declared in `oneapi/mkl/types.hpp`, which is included automatically when you include `oneapi/mkl/blas.hpp` or `oneapi/mkl/lapack.hpp`. Like all oneMKL DPC++ functionality, they belong to the namespace `oneapi::mkl::`.

Each enumeration value comes with two names: A single-character name (the traditional BLAS/LAPACK character) and a longer, descriptive name. The two names are exactly equivalent and may be used interchangeably.

6.1.1 Transpose

The `transpose` type specifies whether an input matrix should be transposed and/or conjugated. It can take the following values:

Short Name	Long Name	Description
`transpose::N`	`transpose::nontrans`	Do not transpose or conjugate the matrix.
`transpose::T`	`transpose::trans`	Transpose the matrix.
`transpose::C`	`transpose::conjtrans`	Perform Hermitian transpose (transpose and conjugate). Only applicable to complex matrices.

6.1.2 Uplo

The `uplo` type specifies whether the lower or upper triangle of a triangular, symmetric, or Hermitian matrix should be accessed. It can take the following values:

Short Name	Long Name	Description
`uplo::U`	`uplo::upper`	Access the upper triangle of the matrix.
`uplo::L`	`uplo::lower`	Access the lower triangle of the matrix.

In both cases, elements that are not in the selected triangle are not accessed or updated.

6.1.3 Diag

The `diag` type specifies the values on the diagonal of a triangular matrix. It can take the following values:

Short Name	Long Name	Description
`diag::N`	`diag::nonunit`	The matrix is not unit triangular. The diagonal entries are stored with the matrix data.
`diag::U`	`diag::unit`	The matrix is unit triangular (the diagonal entries are all 1s). The diagonal entries in the matrix data are not accessed.

6.1.4 Side

The `side` type specifies the order of matrix multiplication when one matrix has a special form (triangular, symmetric, or Hermitian):

Short Name	Long Name	Description
`side::L`	`side::left`	The special form matrix is on the left in the multiplication.
`side::R`	`side::right`	The special form matrix is on the right in the multiplication.

6.1.5 Offset

The `offset` type specifies whether the offset to apply to an output matrix is a fix offset, column offset or row offset. It can take the following values

Short Name	Long Name	Description
`offset::F`	`offset::fix`	The offset to apply to the output matrix is fix, all the inputs in the `C_offset` matrix has the same value given by the first element in the `co` array.
`offset::C`	`offset::column`	The offset to apply to the output matrix is a column offset, that is to say all the columns in the `C_offset` matrix are the same and given by the elements in the `co` array.
`offset::R`	`offset::row`	The offset to apply to the output matrix is a row offset, that is to say all the rows in the `C_offset` matrix are the same and given by the elements in the `co` array.

6.2 Vector Math Data Types

oneMKL VM for Data Parallel C++ (DPC++) introduces a slice type, available in the `oneapi::mkl` namespace. Slices are used in the DPC++ VM Strided APIs. oneMKL slices accept positive, zero, and negative strides, for forward, static, and backward traversals of a vector, respectively.

Constructors	Description
slice()	Default constructor equivalent to slice(0, 0, 0).
slice(`std::size_t start, std::size_t size`, `std::int64_t stride`)	Slice defining the start index, the number of values to select, and the stride between two elements.
slice(`const slice & other`)	Copy constructor.

For example:

- slice(1, 5, 2) defines a selector of elements at indices 1, 3, 5, 7, 9 in a buffer or array;
- slice(0, 5, 0) defines a selector of a single element at index 0, repeated five times;
- slice(9, 4, -3) defines a selector of elements at indices 9, 6, 3, 0.

A slice is considered invalid if it produces negative indices. A slice of size 0 selects no element. If a slice may cause out-of-bounds memory accesses, the behavior is undefined.

The behavior of VM Strided APIs used with invalid or non-equal slices is configurable with several oneMKL VM mode values. See the **set_mode** function for possible values and their descriptions.

In the `slice_cyclic` mode, zero-sized slices are invalid.

7.0 Matrix Storage

The oneMKL BLAS and LAPACK routines for DPC++ use several matrix and vector storage formats. These are the same formats used in traditional Fortran BLAS/LAPACK.

7.1 General Matrix

A general matrix `A` of `m` rows and `n` columns with leading dimension `lda` is represented as a one dimensional array a of size of at least `lda` * `n` if column major layout is used and at least `lda`*`m` if row major layout is used. Before entry in any BLAS function using a general matrix, the leading `m` by `n` part of the array a must contain the matrix `A`. For column (respectively, row) major layout, the elements of each column (respectively, row) are contiguous in memory while the elements of each row (respectively, column) are at distance `lda` from the same elements in the previous row (respectively, column).

Visually, the matrix

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2n} \\ A_{31} & A_{32} & A_{33} & \dots & A_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & A_{m3} & \dots & A_{mn} \end{bmatrix}$$

is stored in memory as an array

For column major layout

$$a=[\underbrace{A_{11}, A_{21}, A_{31}, ..., A_{m1}, *, ..., *}_{\text{lda}}, \underbrace{A_{12}, A_{22}, A_{32}, ..., A_{m2}, *, ..., *}_{\text{lda}}, ..., \underbrace{A_{1n}, A_{2n}, A_{3n}, ..., A_{mn}, *, ..., *}_{\text{lda}}]$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{lda x n}}$$

For row major layout

$$a=[\underbrace{A_{11}, A_{12}, A_{13}, ..., A_{1n}, *, ..., *}_{\text{lda}}, \underbrace{A_{21}, A_{22}, A_{23}, ..., A_{2n}, *, ..., *}_{\text{lda}}, ..., \underbrace{A_{m1}, A_{m2}, A_{m3}, ..., A_{mn}, *, ..., *}_{\text{lda}}]$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{m x lda}}$$

7.2 Triangular Matrix

A triangular matrix `A` of `n` rows and `n` columns with leading dimension `lda` is represented as a one dimensional array a, of a size of at least `lda` * `n`. When column (respectively, row) major layout is used, the elements of each column (respectively, row) are contiguous in memory while the elements of each row (respectively, column) are at distance `lda` from the same elements in the previous row (respectively, column).

Before entry in any BLAS function using a triangular matrix,

- If `upper_lower = uplo::upper`, the leading `n` by `n` upper triangular part of the array `a` must contain the upper triangular part of the matrix `A`. The strictly lower triangular part of the array `a` is not referenced. In other words, the matrix

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1n} \\ * & A_{22} & A_{23} & \dots & A_{2n} \\ * & * & A_{33} & \dots & A_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \dots & A_{nn} \end{bmatrix}$$

  is stored in memory as the array

  - For column major layout

$$a=[\underbrace{\underbrace{A_{11}, *, ..., *}_{\text{lda}}, \underbrace{A_{12}, A_{22}, *, ..., *}_{\text{lda}}, ..., \underbrace{A_{1n}, A_{2n}, A_{3n}, ..., A_{nn}, *, ..., *}_{\text{lda}}}_{\text{lda x n}}]$$

  - For row major layout

$$a=[\underbrace{\underbrace{A_{11}, A_{12}, A_{13}, ..., A_{1n}, *, ..., *}_{\text{lda}}, \underbrace{*, A_{22}, A_{23}, ..., A_{2n}, *, ..., *}_{\text{lda}}, ..., \underbrace{*, ..., *, A_{nn}, *, ..., *}_{\text{lda}}}_{\text{lda x n}}]$$

- If `upper_lower = uplo::lower`, the leading `n` by `n` lower triangular part of the array `a` must contain the lower triangular part of the matrix `A`. The strictly upper triangular part of the array `a` is not referenced. That is, the matrix

$$A = \begin{bmatrix} A_{11} & * & * & \dots & * \\ A_{21} & A_{22} & * & \dots & * \\ A_{31} & A_{32} & A_{33} & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \dots & A_{nn} \end{bmatrix}$$

  is stored in memory as the array

  - For column major layout

$$a=[\underbrace{\underbrace{A_{11}, A_{21}, A_{31}, .., A_{n1}, *, ..., *}_{\text{lda}}, \underbrace{*, A_{22}, A_{32}, ..., A_{n2}, *, ..., *}_{\text{lda}}, ..., \underbrace{*, ..., *, A_{nn}, *, ..., *}_{\text{lda}}}_{\text{lda x n}}]$$

  - For row major layout

$$a=[\underbrace{\underbrace{A_{11}, *, ..., *}_{\text{lda}}, \underbrace{A_{21}, A_{22}, *, ..., *}_{\text{lda}}, ..., \underbrace{A_{n1}, A_{n2}, A_{n3}, ..., A_{nn}, *, ..., *}_{\text{lda}}}_{\text{lda x n}}]$$

7.3 Band Matrix

A general band matrix `A` of `m` rows and `n` columns with `kl` sub-diagonals, `ku` super-diagonals and leading dimension `lda` is represented as a one dimensional array a of size at least `lda` * `n` (respectively, `lda` * `m`) if column (respectively, row) major layout is used.

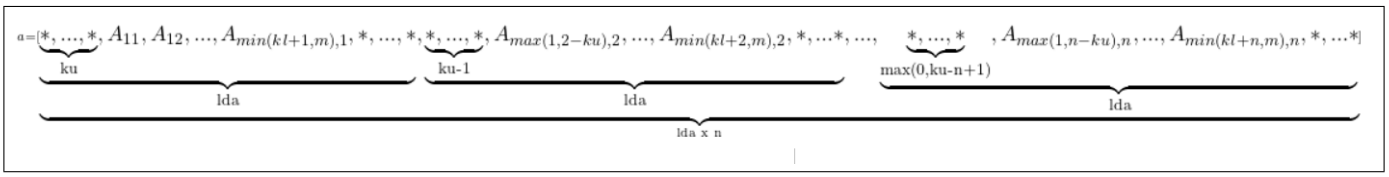
Before entry in any BLAS function using a general band matrix, the leading (`kl` + `ku` + 1) by `n` (respectively, `m`) part of the array a must contain the matrix `A`. This matrix must be supplied column-by-column (respectively, row-by-row), with the main diagonal of the matrix in row `ku` (respectively, column `kl`) of the array (0-based indexing), the first super-diagonal starting at position 1 (respectively, 0) in row (`ku` – 1) (respectively, column (`kl` + 1)), the first sub-diagonal starting at position 0 (respectively, 1) in row (`ku` + 1) (respectively, column (`kl` – 1)), and so on. Elements in the array a that do not correspond to elements in the band matrix (such as the top left `ku`-by-`ku` triangle) are not referenced.

Visually, the matrix `A`

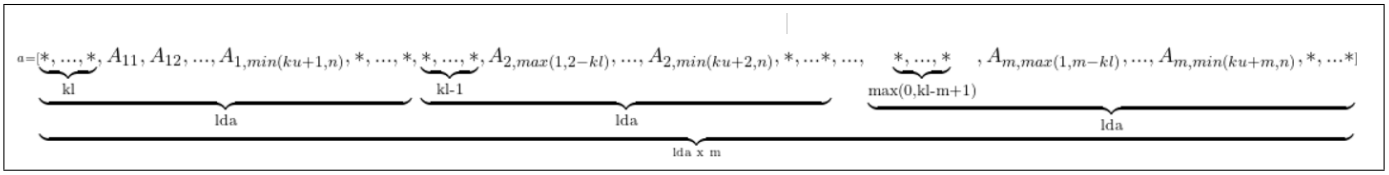
$$A = \begin{bmatrix}
A_{11} & A_{12} & A_{13} & \dots & A_{1,ku+1} & * & \dots & \dots & \dots & \dots & \dots & * \\
A_{21} & A_{22} & A_{23} & A_{24} & \dots & A_{2,ku+2} & * & \dots & \dots & \dots & \dots & * \\
A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & \dots & A_{3,ku+3} & * & \dots & \dots & \dots & * \\
\vdots & A_{42} & A_{43} & \ddots & \ddots & \ddots & \ddots & \ddots & * & \dots & \dots & \vdots \\
A_{kl+1,1} & \vdots & A_{53} & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & * & \dots & \vdots \\
* & A_{kl+2,2} & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & * & A_{kl+3,3} & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & * \\
\vdots & \vdots & * & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & A_{n-ku,n} \\
\vdots & \vdots & \vdots & * & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \vdots & \vdots & \vdots & * & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & A_{m-2,n} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & A_{m-1,n} \\
* & * & * & \dots & \dots & \dots & * & A_{m,m-kl} & \dots & A_{m,n-2} & A_{m,n-1} & A_{m,n}
\end{bmatrix}$$

is stored in memory as an array

For column major layout



For row major layout



The following program segment transfers a band matrix from conventional full matrix storage (variable `matrix`, with leading dimension `ldm`) to band storage (variable a, with leading dimension `lda`):

- Using column major layout

```
for (j = 0; j < n; j++) {
    k = ku – j;
    for (i = max(0, j – ku); i < min(m, j + kl + 1); i++) {
        a[(k + i) + j * lda] = matrix[i + j * ldm];
    }
}
```

- Using row major layout

```
for (i = 0; i < m; i++) {
    k = kl – i;
    for (j = max(0, i – kl); j < min(n, i + ku + 1); j++) {
        a[(k + j) + i * lda] = matrix[j + i * ldm];
    }
}
```

7.4 Triangular Band Matrix

A triangular band matrix `A` of `n` rows and `n` columns with `k` sub/super-diagonals and leading dimension `lda` is represented as a one dimensional array a of size at least `lda` * `n`.

Before entry in any BLAS function using a triangular band matrix,

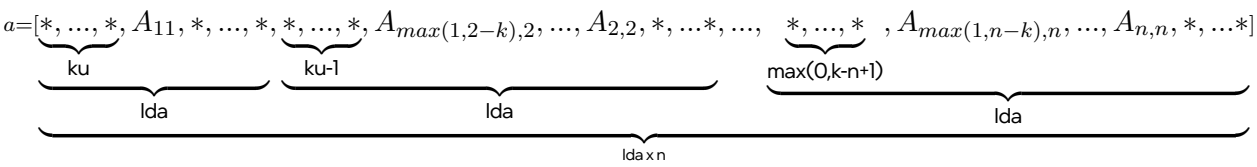
- If `upper_lower = uplo::upper`, the leading (`k` + 1) by `n` part of the array a must contain the upper triangular band part of the matrix `A`. When using column major layout, this matrix must be supplied column-by-column (respectively, row-by-row) with the main diagonal of the matrix in row (`k`) (respectively, column 0) of the array, the first super-diagonal starting at position 1 (respectively, 0) in row (`k` - 1) (respectively, column 1), and so on. Elements in the array a that do not correspond to elements in the triangular band matrix (such as the bottom left `k` by `k` triangle) are not referenced.

Visually, the matrix

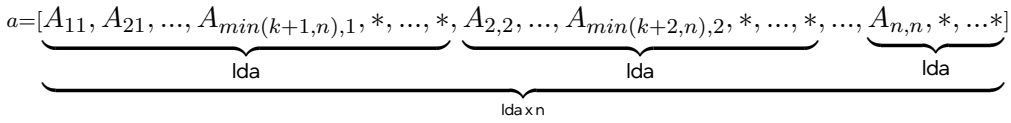
$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1,k+1} & * & \dots & \dots & \dots & \dots & \dots & * \\ * & A_{22} & A_{23} & A_{24} & \dots & A_{2,k+2} & * & \dots & \dots & \dots & \dots & * \\ \vdots & * & A_{33} & A_{34} & A_{35} & \dots & A_{3,k+3} & * & \dots & \dots & \dots & * \\ \vdots & \vdots & * & \ddots & \ddots & \ddots & \ddots & \ddots & * & \dots & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & * & \dots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & A_{n-k,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & A_{n-2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & A_{n-1,n} \\ * & * & * & \dots & \dots & \dots & \dots & \dots & \dots & \dots & * & A_{n,n} \end{bmatrix}$$

is stored as an array

- For column major layout



- For row major layout



The following program segment transfers a band matrix from conventional full matrix storage (variable `matrix`, with leading dimension `ldm`) to band storage (variable a, with leading dimension `lda`):

- Using column major layout

```
for (j = 0; j < n; j++) {
    m = k – j;
    for (i = max(0, j – k); i <= j; i++) {
        a[(m + i) + j * lda] = matrix[i + j * ldm];
    }
}
```

- Using row major layout

```
for (i = 0; i < n; i++) {
    m = –i;
    for (j = i; j < min(n, i + k + 1); j++) {
        a[(m + j) + i * lda] = matrix[j + i * ldm];
    }
}
```

- If `upper_lower = uplo::lower`, the leading (k + 1) by n part of the array a must contain the upper triangular band part of the matrix A. This matrix must be supplied column-by-column with the main diagonal of the matrix in row 0 of the array, the first sub-diagonal starting at position 0 in row 1, and so on. Elements in the array a that do not correspond to elements in the triangular band matrix (such as the bottom right k by k triangle) are not referenced.

That is, the matrix

$$A = \begin{bmatrix} A_{11} & * & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & * \\ A_{21} & A_{22} & * & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & * \\ A_{31} & A_{32} & A_{33} & * & \dots & \dots & \dots & \dots & \dots & \dots & \dots & * \\ \vdots & A_{42} & A_{43} & \ddots & \ddots & \dots & \dots & \dots & \dots & \dots & \dots & \vdots \\ A_{k+1,1} & \vdots & A_{53} & \ddots & \ddots & \ddots & \dots & \dots & \dots & \dots & \dots & \vdots \\ * & A_{k+2,2} & \vdots & \ddots & \ddots & \ddots & \ddots & \dots & \dots & \dots & \dots & \vdots \\ \vdots & * & A_{k+3,3} & \ddots & \ddots & \ddots & \ddots & \ddots & \dots & \dots & \dots & \vdots \\ \vdots & \vdots & * & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \dots & \dots & \vdots \\ \vdots & \vdots & \vdots & * & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \dots & \vdots \\ \vdots & \vdots & \vdots & \vdots & * & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & * \\ * & * & * & \dots & \dots & \dots & * & A_{n,n-k} & \dots & A_{n,n-2} & A_{n,n-1} & A_{n,n} \end{bmatrix}$$

is stored as the array

- For column major layout

$$a=[\underbrace{A_{11}, A_{21}, ..., A_{min(k+1,n),1}, *, ..., *}_{\text{lda}}, \underbrace{A_{2,2}, ..., A_{min(k+2,n),2}, *, ..., *}_{\text{lda}}, ..., \underbrace{A_{n,n}, *, ...*}_{\text{lda}}]$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{lda x n}}$$

- For row major layout

$$a=[\underbrace{*, ..., *}_{\text{k}}, A_{11}, *, ..., *, \underbrace{*, ..., *}_{\text{k-1}}, A_{max(1,2-k),2}, ..., A_{2,2}, *, ...*, ..., \underbrace{*, ..., *}_{\text{max(0,k-n+1)}}, A_{max(1,n-k),n}, ..., A_{n,n}, *, ...*]$$
$$\underbrace{\qquad\qquad}_{\text{lda}} \quad \underbrace{\qquad\qquad\qquad\qquad}_{\text{lda}} \quad \underbrace{\qquad\qquad\qquad\qquad}_{\text{lda}}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{lda x n}}$$

The following program segment transfers a band matrix from conventional full matrix storage (variable `matrix`, with leading dimension `ldm`) to band storage (variable a, with leading dimension `lda`):

- Using column major layout

```
for (j = 0; j < n; j++) {
    m = –j;
    for (i = j; i < min(n, j + k + 1); i++) {
        a[(m + i) + j * lda] = matrix[i + j * ldm];
    }
}
```

- Using row major layout

```
for (i = 0; i < n; i++) {
   m = k – i;
   for (j = max(0, i – k); j <= i; j++) {
       a[(m + j) + i * lda] = matrix[j + i * ldm];
   }
}
```

7.5 Packed Triangular Matrix

A triangular matrix A of n rows and n columns is represented in packed format as a one dimensional array a of size at least (n*(n + 1))/2. All elements in the upper or lower part of the matrix A are stored contiguously in the array a.

Before entry in any BLAS function using a triangular packed matrix,

- If `upper_lower = uplo::upper`, the first (n*(n + 1))/2 elements in the array a must contain the upper triangular part of the matrix A packed sequentially, column by column so that a[0] contains A_{11}, a[1] and a[2] contain A_{12} and A_{22} respectively, and so on. Hence, the matrix

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1n} \\ * & A_{22} & A_{23} & \dots & A_{2n} \\ * & * & A_{33} & \dots & A_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \dots & A_{nn} \end{bmatrix}$$

  is stored as the array

- For column major layout

$$a=[A_{11},A_{12},A_{22},A_{13},A_{23},A_{33},...,A_{(n-1),n},A_{nn}]$$

- For row major layout

$$a=[A_{11},A_{12},A_{13},...,A_{1n},A_{22},A_{23},...,A_{2n},...,A_{(n-1),(n-1)},A_{(n-1),n},A_{nn}]$$

- If `upper_lower = uplo::lower`, if column (respectively, row) major layout is used, the first (n*(n + 1))/2 elements in the array a must contain the lower triangular part of the matrix A packed sequentially, column by column (respectively, row by row) so that a[0] contains A_{11}, a[1] and a[2] contain A_{21} and A_{31} respectively, and so on. The matrix

$$A = \begin{bmatrix} A_{11} & * & * & \dots & * \\ A_{21} & A_{22} & * & \dots & * \\ A_{31} & A_{32} & A_{33} & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \dots & A_{nn} \end{bmatrix}$$

  is stored as the array

- For column major layout

$$a=[A_{11},A_{21},A_{31},...,A_{n1},A_{22},A_{32},...,A_{n2},...,A_{(n-1),(n-1)},A_{n,(n-1)},A_{nn}]$$

- For row major layout

$$a=[A_{11},A_{21},A_{22},A_{31},A_{32},A_{33},...,A_{n,(n-1)},A_{nn}]$$

7.6 Vector

A vector X of n elements with increment `incx` is represented as a one dimensional array x of size at least (1 + (n - 1) * abs(`incx`)).

Visually, the vector

$$X = (X_1, X_2, X_3, ..., X_n)$$

is stored in memory as an array

$$x=[\underbrace{\underbrace{X_1, *, ..., *}_{\text{incx}}, \underbrace{X_2, *, ..., *}_{\text{incx}}, ..., \underbrace{X_{n-1}, *, ..., *}_{\text{incx}}, X_n}_{1+(n-1)\,x\,\text{incx}}] \quad if\ incx > 0$$

$$x=[\underbrace{\underbrace{X_n, *, ..., *}_{|\text{incx}|}, \underbrace{X_{n-1}, *, ..., *}_{|\text{incx}|}, ..., \underbrace{X_2, *, ..., *}_{|\text{incx}|}, X_1}_{1+(1-n)\,x\,\text{incx}}] \quad if\ incx < 0$$

8.0 Error Handling

oneMKL error handling relies on the mechanism of C++ exceptions. Should errors occur, they are propagated at the point of a function call where they are caught using standard C++ error handling mechanisms.

8.1 Exception Classification

Exception classification in oneMKL is aligned with C++ Standard Library classification. oneMKL introduces a class that defines the base class in the hierarchy of oneMKL exception classes. All oneMKL routines throw exceptions inherited from this base class.

In the hierarchy of oneMKL exceptions, `oneapi::mkl::exception` is the base class inherited from the `std::exception` class. All other oneMKL exception classes are derived from this base class.

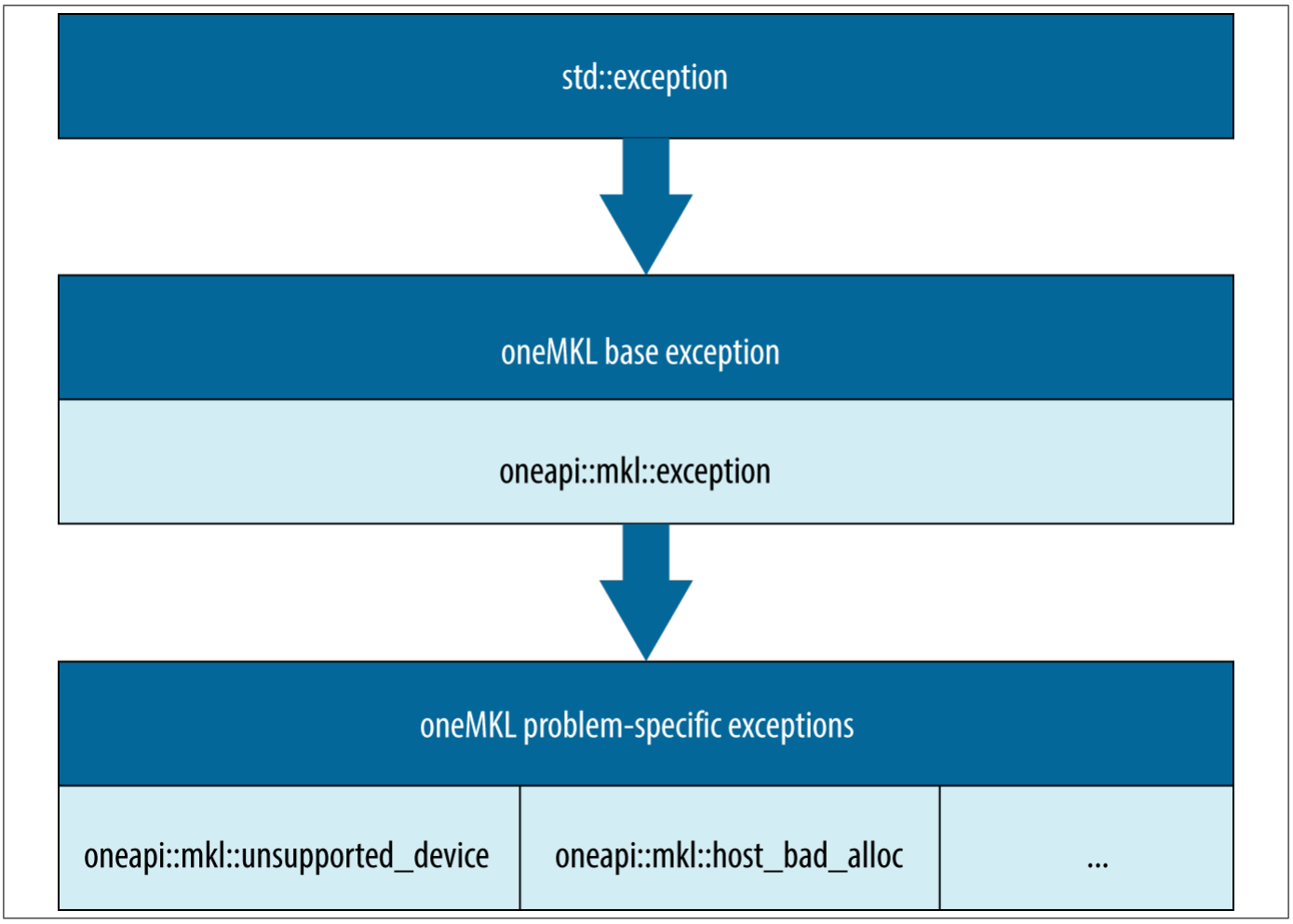


Fig. 1: Hierarchy of oneMKL exceptions

All oneMKL problem-specific exceptions are listed in the following table.

Exception Class	Description
`oneapi::mkl::unsupported_device`	Reports a problem when the routine is not supported on a specific device
`oneapi::mkl::host_bad_alloc`	Reports a problem that occurred during memory allocation on the host
`oneapi::mkl::device_bad_alloc`	Reports a problem that occurred during memory allocation on a specific device
`oneapi::mkl::unimplemented`	Reports a problem when a specific routine has not been implemented for the specified parameters
`oneapi::mkl::invalid_argument`	Reports problem when arguments to the routine were rejected
`oneapi::mkl::uninitialized`	Reports problem when a handle (descriptor) has not been initialized
`oneapi::mkl::computation_error`	Reports any computation error that occurred inside the oneMKL routine
`oneapi::mkl::batch_error`	Reports errors that occurred inside batch oneMKL routines

9.0 BLAS Routines

The Intel® oneAPI Math Kernel Library provides a Data Parallel C++ interface to some of the BLAS routines. The routine descriptions are arranged in several sections:

- BLAS Level 1 Routines (vector-vector operations)
- BLAS Level 2 Routines (matrix-vector operations)
- BLAS Level 3 Routines (matrix-matrix operations)
- BLAS-like Extensions

9.1 BLAS Level 1 Routines

This section describes BLAS Level 1 routines, which perform vector-vector operations. The following table lists the BLAS Level 1 routine groups and the data types associated with them.

Routine Group	Data Types	Description
asum	float, double, mixed float and std::complex<float>, mixed double and std::complex<double>	Sum of vector magnitudes
axpy	sycl::half, oneapi::mkl::bfloat16, float, double, std::complex<float>, std::complex<double>	Scalar-vector product
copy	float, double, std::complex<float>, std::complex<double>	Copy vector
dot	sycl::half, oneapi::mkl::bfloat16, float, double, mixed float and double	Dot product
sdsdot	mixed float and double	Dot product with double precision
dotc	std::complex<float>, std::complex<double>	Dot product conjugated
dotu	std::complex<float>, std::complex<double>	Dot product unconjugated
nrm2	sycl::half, oneapi::mkl::bfloat16, float, double, mixed float and std::complex<float>, mixed double and std::complex<double>	Vector 2-norm (Euclidean norm)
rot	sycl::half, oneapi::mkl::bfloat16, float, double, mixed float and std::complex<float>, mixed double and std::complex<double>	Plane rotation of points
rotg	float, double, std::complex<float>, std::complex<double>	Generate Givens rotation of points
rotm	float, double	Modified Givens plane rotation of points
rotmg	float, double	Generate modified Givens plane rotation of points
scal	sycl::half, oneapi::mkl::bfloat16, float, double, std::complex<float>, std::complex<double>, mixed float and std::complex<float>, mixed double and std::complex<double>	Vector-scalar product
swap	float, double, std::complex<float>, std::complex<double>	Vector-vector swap
iamax	float, double, std::complex<float>, std::complex<double>	Index of the maximum absolute value element of a vector
iamin	float, double, std::complex<float>, std::complex<double>	Index of the minimum absolute value element of a vector

9.1.1 asum

Computes the sum of magnitudes of the vector elements.

Description

The asum routine computes the sum of the magnitudes of elements of a real vector, or the sum of magnitudes of the real and imaginary parts of elements of a complex vector. The operation is defined as:

$$\text{result} \leftarrow \sum_{i=1}^{n} (|\text{Re}(X_i)| + |\text{Im}(X_i)|)$$

where:

- x is a vector with n elements

asum supports the following precisions:

T	Tres
float	float
double	double
std::complex<float>	float
std::complex<double>	double

asum (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void asum(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<Tres,1> &result)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void asum(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<Tres,1> &result)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

Output Parameters

result Buffer where the scalar result is stored.

asum (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event asum(sycl::queue &queue,
                     std::int64_t n,
                     const T *x,
                     std::int64_t incx,
                     Tres *result,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event asum(sycl::queue &queue,
                     std::int64_t n,
                     const T *x,
                     std::int64_t incx,
                     Tres *result,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Pointer to input vector x. Size of the array holding vector x must be least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

result Pointer to where the scalar result is stored.

Return Values

Output event to wait on to ensure computation is complete.

9.1.2 axpy

Computes a vector-scalar product and adds the result to a vector.

Description

The axpy routines compute a scalar-vector product and add the result to a vector. The operation is defined as:

$$y \leftarrow alpha * x + y$$

where:

- x and y are vectors of n elements
- alpha is a scalar

axpy supports the following precisions:

T
sycl::half
oneapi::mkl::bfloat16
float
double
std::complex<float>
std::complex<double>

axpy (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void axpy(sycl::queue &queue,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void axpy(sycl::queue &queue,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

alpha Specifies scalar `alpha`.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n – 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

y Buffer holding input vector y. Size of the buffer must be at least (1 + (n – 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

Output Parameters

y Buffer holding updated vector y.

Examples

An example of how to use `axpy` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/blas/source/axpy.cpp
```

axpy (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event axpy(sycl::queue &queue,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event axpy(sycl::queue &queue,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

alpha Specifies scalar `alpha`.

x Pointer to input vector x. Size of the array holding vector x must be at least (1 + (n – 1)*abs(`incx`)). See **Matrix Storage** for more details.

incx Stride of vector x.

y Pointer to input vector y. Size of the array holding vector y must be at least (1 + (n – 1)*abs(`incy`)). See **Matrix Storage** for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.1.3 copy

Copies a vector to another vector.

Description

The `copy` routines copy one vector to another. The operation is defined as:

$$y \leftarrow x$$

where:

- x and y are vectors of n elements

`copy` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

copy (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void copy(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void copy(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n – 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

incy Stride of vector y.

Output Parameters

y Buffer holding updated vector y.

copy (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event copy(sycl::queue &queue,
                     std::int64_t n,
                     const T *x,
                     std::int64_t incx,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event copy(sycl::queue &queue,
                     std::int64_t n,
                     const T *x,
                     std::int64_t incx,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Pointer to the input vector x. Size of the array holding vector x must be at least (1 + (n – 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.1.4 dot

Computes the `dot` product of two real vectors.

Description

The `dot` routines perform a dot product between two vectors. The operation is defined as:

$$\text{result} = \sum_{i=1}^{n} X_i Y_i$$

`dot` supports the following precisions:

T	Tres
`sycl::half`	`sycl::half`
`oneapi::mkl::bfloat16`	`oneapi::mkl::bfloat16`
`float`	`float`
`double`	`double`
`float`	`double`

Note: For mixed precision version (inputs are float while result is double), `dot` product is computed with double precision.

dot (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void dot(sycl::queue &queue,
             std::int64_t n,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &y,
             std::int64_t incy,
             sycl::buffer<Tres,1> &result)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void dot(sycl::queue &queue,
             std::int64_t n,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &y,
             std::int64_t incy,
             sycl::buffer<Tres,1> &result)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors x and y.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n – 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

y Buffer holding input vector y. Size of the buffer must be at least (1 + (n – 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

Output Parameters

result Buffer where the result (a scalar) will be stored.

dot (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event dot(sycl::queue &queue,
                    std::int64_t n,
                    const T *x,
                    std::int64_t incx,
                    const T *y,
                    std::int64_t incy,
                    Tres *result,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event dot(sycl::queue &queue,
                    std::int64_t n,
                    const T *x,
                    std::int64_t incx,
                    const T *y,
                    std::int64_t incy,
                    Tres *result,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors x and y.

x Pointer to input vector x. Size of the array holding vector x must be least (1 + (n – 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

y Pointer to input vector y. Size of the array holding vector y must be at least (1 + (n – 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

result Pointer to where the result (a scalar) will be stored.

Return Values

Output event to wait on to ensure computation is complete.

9.1.5 dotc

Computes the dot product of two complex vectors, conjugating the first vector.

Description

The `dotc` routines perform a dot product between two complex vectors, conjugating the first one. The operation is defined as:

$$\text{result} = \sum_{i=1}^{n} \overline{X_i} Y_i$$

`dotc` supports the following precisions:

T
std::complex<float>
std::complex<double>

dotc (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void dotc(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &result)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void dotc(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &result)
}
```

Input Parameters

queue The queue where the routine should be executed.

n The number of elements in vectors x and y.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx The stride of vector x.

y Buffer holding input vector y. Size of the buffer must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy The stride of vector y.

Output Parameters

result The buffer where the result (a scalar) is stored.

dotc (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event dotc(sycl::queue &queue,
                     std::int64_t n,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *result,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event dotc(sycl::queue &queue,
                     std::int64_t n,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *result,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n The number of elements in vectors x and y.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx The stride of vector x.

y Pointer to input vector y. Size of the array holding input vector y must be at least (1 + (n - 1)*abs(`incy`)). See Matrix Storage for more details.

incy The stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

result The pointer to where the result (a scalar) is stored.

Return Values

Output event to wait on to ensure computation is complete.

9.1.6 dotu

Computes the dot product of two complex vectors.

Description

The `dotu` routines perform a dot product between two complex vectors. The operation is defined as:

$$\text{result} = \sum_{i=1}^{n} X_i Y_i$$

`dotu` supports the following precisions:

T
`std::complex<float>`
`std::complex<double>`

dotu (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void dotu(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &result)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void dotu(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &result)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors x and y.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

y Buffer holding input vector y. The buffer must have size at least (1 + (n - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

Output Parameters

result Buffer where the result (a scalar) is stored.

dotu (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event dotu(sycl::queue &queue,
                     std::int64_t n,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *result,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event dotu(sycl::queue &queue,
                     std::int64_t n,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *result,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors x and y.

x Pointer to the input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

y Pointer to input vector y. Size of the array holding input vector y must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

result Pointer to where the result (a scalar) is stored.

Return Values

Output event to wait on to ensure computation is complete.

9.1.7 iamax

Finds the index of the element with the largest absolute value in a vector.

Description

The `iamax` routines return an index `i` such that `x[i]` has the maximum absolute value of all elements in vector `x` (real variants), or such that (|Re(`x[i]`)| + |Im(`x[i]`)|) is maximal (complex variants).

`iamax` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std:complex<double>`

Note: The index is zero-based.

If either `n` or `incx` are not positive, the routine returns `0`.

If more than one vector element is found with the same largest absolute value, the index of the first one encountered is returned.

If the vector contains `NaN` values, then the routine returns the index of the first `NaN`.

---

iamax (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void iamax(sycl::queue &queue,
               std::int64_t n,
               sycl::buffer<T, 1> &x,
               std::int64_t incx,
               sycl::buffer<std::int64_t, 1> &result)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void iamax(sycl::queue &queue,
               std::int64_t n,
               sycl::buffer<T, 1> &x,
               std::int64_t incx,
               sycl::buffer<std::int64_t, 1> &result)
}
```

Input Parameters

exec_queue The queue where the routine should be executed.

n The number of elements in vector x.

x The buffer that holds the input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx The stride of vector x.

Output Parameters

result The buffer where the zero-based index `i` of the maximal element is stored.

Examples

An example of how to use `iamax` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/blas/source/iamax.cpp
```

iamax (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event iamax(sycl::queue &queue,
                      std::int64_t n,
                      const T *x,
                      std::int64_t incx,
                      std::int64_t *result,
                      const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event iamax(sycl::queue &queue,
                      std::int64_t n,
                      const T *x,
                      std::int64_t incx,
                      std::int64_t *result,
                      const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n The number of elements in vector x.

x The pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx The stride of vector x.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

result The pointer to where the zero-based index i of the maximal element is stored.

Return Values

Output event to wait on to ensure computation is complete.

9.1.8 iamin

Finds the index of the element with the smallest absolute value in a vector

Description

The iamin routines return an index i such that x[i] has the minimum absolute value of all elements in vector x (real variants), or such that (|Re(x[i])| + |Im(x[i])|) is minimal (complex variants).

iamin supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

Note: The index is zero-based.

If either `n` or `incx` are not positive, the routine returns `0`.

If more than one vector element is found with the same smallest absolute value, the index of the first one encountered is returned.

If the vector contains `NaN` values, then the routine returns the index of the first `NaN`.

iamin (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void iamin(sycl::queue &queue,
               std::int64_t n,
               sycl::buffer<T,1> &x,
               std::int64_t incx,
               sycl::buffer<std::int64_t,1> &result)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void iamin(sycl::queue &queue,
               std::int64_t n,
               sycl::buffer<T,1> &x,
               std::int64_t incx,
               sycl::buffer<std::int64_t,1> &result)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

Output Parameters

result Buffer where the zero-based index `i` of the minimum element is stored.

iamin (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event iamin(sycl::queue &queue,
                      std::int64_t n,
                      const T *x,
                      std::int64_t incx,
                      std::int64_t *result,
                      const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event iamin(sycl::queue &queue,
                      std::int64_t n,
                      const T *x,
                      std::int64_t incx,
                      std::int64_t *result,
                      const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x The pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

Output Parameters

result Pointer to where the zero-based index i of the minimum element is stored.

Return Values

Output event to wait on to ensure computation is complete.

9.1.9 nrm2

Computes the Euclidean norm of a vector.

Description

The nrm2 routines compute Euclidean norm of a vector. The operation is defined as:

$$result = ||x||$$

where:

- x is a vector of n elements

nrm2 supports the following precisions:

T	Tres
sycl::half	sycl::half
oneapi::mkl::bfloat16	oneapi::mkl::bfloat16
float	float
double	double
std::complex<float>	float
std::complex<double>	double

nrm2 (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void nrm2(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<Tres,1> &result)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void nrm2(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<Tres,1> &result)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

Output Parameters

result Buffer where the Euclidean norm of the vector x is stored.

nrm2 (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event nrm2(sycl::queue &queue,
                     std::int64_t n,
                     const T *x,
                     std::int64_t incx,
                     Tres *result,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event nrm2(sycl::queue &queue,
                     std::int64_t n,
                     const T *x,
                     std::int64_t incx,
                     Tres *result,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

result Pointer to where the Euclidean norm of the vector x is stored.

Return Values

Output event to wait on to ensure computation is complete.

9.1.10 rot

Performs rotation of points in the plane.

Description

Given two vectors x and y of n elements, the `rot` routines compute four scalar-vector products and update the input vectors with the sum of two of these scalar-vector products. The operation is defined as:

$$\begin{bmatrix} x \\ y \end{bmatrix} \leftarrow \begin{bmatrix} c * x + s * y \\ -s * x + c * y \end{bmatrix}$$

If s is a complex type, the operation is defined as:

$$\begin{bmatrix} x \\ y \end{bmatrix} \leftarrow \begin{bmatrix} c * x + s * y \\ -conj(s) * x + c * y \end{bmatrix}$$

`rot` supports the following precisions:

T	Tc	Ts
`sycl::half`	`sycl::half`	`sycl::half`
`oneapi::mkl::bfloat16`	`oneapi::mkl::bfloat16`	`oneapi::mkl::bfloat16`
`float`	`float`	`float`
`double`	`double`	`double`
`std::complex<float>`	`float`	`std::complex<float>`
`std::complex<double>`	`double`	`std::complex<double>`
`std::complex<float>`	`float`	`float`
`std::complex<double>`	`double`	`double`

rot (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void rot(sycl::queue &queue,
             std::int64_t n,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &y,
             std::int64_t incy,
             Tc c,
             Ts s)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void rot(sycl::queue &queue,
             std::int64_t n,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &y,
             std::int64_t incy,
             Tc c,
             Ts s)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

y Buffer holding input vector y. Size of the buffer must be at least (1 + (n - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

c Scaling factor.

s Scaling factor.

Output Parameters

x Buffer holding updated buffer x.

y Buffer holding updated buffer y.

rot (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event rot(sycl::queue &queue,
                    std::int64_t n,
                    T *x,
                    std::int64_t incx,
                    T *y,
                    std::int64_t incy,
                    Tc c,
                    Ts s,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event rot(sycl::queue &queue,
                    std::int64_t n,
                    T *x,
                    std::int64_t incx,
                    T *y,
                    std::int64_t incy,
                    Tc c,
                    Ts s,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

y Pointer to input vector y. Size of the array holding input vector y must be at least (1 + (n - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

c Scaling factor.

s Scaling factor.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

x Pointer to updated vector x.

y Pointer to updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.1.11 rotg

Computes the parameters for a Givens rotation.

Description

Given the Cartesian coordinates `(a, b)` of a point, the `rotg` routines return the parameters c, s, r, and z associated with the Givens rotation. The parameters c and s define a unitary matrix such that:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

The parameter z is defined such that if |a| > |b|, z is s; otherwise if c is not 0 z is 1/c; otherwise z is 1.

`rotg` supports the following precisions:

T	Tc
`float`	`float`
`double`	`double`
`std::complex<float>`	`float`
`std::complex<double>`	`double`

rotg (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void rotg(sycl::queue &queue,
              sycl::buffer<T,1> &a,
              sycl::buffer<T,1> &b,
              sycl::buffer<Tc,1> &c,
              sycl::buffer<T,1> &s)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void rotg(sycl::queue &queue,
              sycl::buffer<T,1> &a,
              sycl::buffer<T,1> &b,
              sycl::buffer<Tc,1> &c,
              sycl::buffer<T,1> &s)
}
```

Input Parameters

queue The queue where the routine should be executed

a Buffer holding x-coordinate of the point.

b Buffer holding y-coordinate of the point.

Output Parameters

a Buffer holding parameter r associated with the Givens rotation.

b Buffer holding parameter z associated with the Givens rotation.

c Buffer holding parameter c associated with the Givens rotation.

s Buffer holding parameter s associated with the Givens rotation.

rotg (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event rotg(sycl::queue &queue,
                     T *a,
                     T *b,
                     Tc *c,
                     T *s,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event rotg(sycl::queue &queue,
                     T *a,
                     T *b,
                     Tc *c,
                     T *s,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed

a Pointer to x-coordinate of the point.

b Pointer to y-coordinate of the point.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to parameter `r` associated with the Givens rotation.

b Pointer to parameter `z` associated with the Givens rotation.

c Pointer to parameter `c` associated with the Givens rotation.

s Pointer to parameter `s` associated with the Givens rotation.

Return Values

Output event to wait on to ensure computation is complete.

9.1.12 rotm

Performs modified Givens rotation of points in the plane.

Description

Given two vectors `x` and `y`, each vector element of these vectors is replaced as follows:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = H \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

for `i` from `1` to `n`, where `H` is a modified Givens transformation matrix.

`rotm` supports the following precisions:

T
float
double

rotm (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void rotm(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &param)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void rotm(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &param)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

y Buffer holding input vector y. Size of the buffer must be at least (1 + (n - 1)*abs(`incy`)). See **:ref:'matrix-storage** for more details.

incy Stride of vector y.

param Buffer holding an array of size 5. The elements of the `param` array are:

`param[0]` contains a switch, `flag`. The other array elements `param[1-4]` contain the components of the modified Givens transformation matrix `H`: h_{11}, h_{21}, h_{12}, and h_{22}, respectively.

Depending on the values of `flag`, the components of `H` are set as follows:

`flag = -1.0`:

$$H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$$

`flag = 0.0`:

$$H = \begin{bmatrix} 1.0 & h_{12} \\ h_{21} & 1.0 \end{bmatrix}$$

`flag = 1.0`:

$$H = \begin{bmatrix} h_{11} & 1.0 \\ -1.0 & h_{22} \end{bmatrix}$$

`flag = -2.0`:

$$H = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$$

In the last three cases, the matrix entries of 1.0, -1.0, 0.0 are assumed based on the value of `flag` and are not required to be set in the `param` vector.

Output Parameters

x Buffer holding updated buffer x.

y Buffer holding updated buffer y.

rotm (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event rotm(sycl::queue &queue,
                     std::int64_t n,
                     T *x,
                     std::int64_t incx,
                     T *y,
                     std::int64_t incy,
                     const T *param,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event rotm(sycl::queue &queue,
                     std::int64_t n,
                     T *x,
                     std::int64_t incx,
                     T *y,
                     std::int64_t incy,
                     const T *param,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

x Pointer to the input vector x. Size of the array holding vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

yparam Pointer to the input vector y. Size of the array holding vector y must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

param Pointer to an array of size 5. The elements of the `param` array are:

`param[0]` contains a switch, `flag`. The other array elements `param[1-4]` contain the components of the modified Givens transformation matrix H: h_{11}, h_{21}, h_{12}, and h_{22}, respectively.

Depending on the values of `flag`, the components of H are set as follows:

`flag = -1.0`:

$$H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$$

`flag = 0.0`:

$$H = \begin{bmatrix} 1.0 & h_{12} \\ h_{21} & 1.0 \end{bmatrix}$$

`flag = 1.0`:

$$H = \begin{bmatrix} h_{11} & 1.0 \\ -1.0 & h_{22} \end{bmatrix}$$

`flag = -2.0`:

$$H = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$$

In the last three cases, the matrix entries of 1.0, -1.0, 0.0 are assumed based on the value of `flag` and are not required to be set in the `param` vector.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

x Pointer to updated array x.

y Pointer to updated array y.

Return Values

Output event to wait on to ensure computation is complete.

9.1.13 rotmg

Computes the parameters for a modified Givens rotation.

Description

Given Cartesian coordinates (`x1`, `y1`) of an input vector, the `rotmg` routines compute the components of a modified Givens transformation matrix `H` that zeros the y-component of the resulting vector:

$$\begin{bmatrix} x1 \\ 0 \end{bmatrix} = H \begin{bmatrix} x1 & \sqrt{d1} \\ y1 & \sqrt{d2} \end{bmatrix}$$

`rotmg` supports the following precisions:

T
float
double

rotmg (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void rotmg(sycl::queue &queue,
               sycl::buffer<T,1> &d1,
               sycl::buffer<T,1> &d2,
               sycl::buffer<T,1> &x1,
               sycl::buffer<T,1>  y1,
               sycl::buffer<T,1> &param)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void rotmg(sycl::queue &queue,
               sycl::buffer<T,1> &d1,
               sycl::buffer<T,1> &d2,
               sycl::buffer<T,1> &x1,
               sycl::buffer<T,1>  y1,
               sycl::buffer<T,1> &param)
}
```

Input Parameters

queue The queue where the routine should be executed.

d1 Buffer holding the scaling factor for x-coordinate of the input vector.

d2 Buffer holding the scaling factor for y-coordinate of the input vector.

x1 Buffer holding x-coordinate of the input vector.

y1 Scalar specifying y-coordinate of the input vector.

Output Parameters

d1 Buffer holding the first diagonal element of the updated matrix.

d2 Buffer holding the second diagonal element of the updated matrix.

x1 Buffer holding x-coordinate of the rotated vector before scaling

param Buffer holding an array of size 5.

The elements of the `param` array are:

`param[0]` contains a switch, `flag`. The other array elements `param[1-4]` contain the components of the array `H`: h_{11}, h_{21}, h_{12}, and h_{22}, respectively.

Depending on the values of `flag`, the components of `H` are set as follows:

`flag = -1.0`:

$$H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$$

`flag = 0.0`:

$$H = \begin{bmatrix} 1.0 & h_{12} \\ h_{21} & 1.0 \end{bmatrix}$$

`flag = 1.0`:

$$H = \begin{bmatrix} h_{11} & 1.0 \\ -1.9 & h_{22} \end{bmatrix}$$

`flag = -2.0`:

$$H = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$$

In the last three cases, the matrix entries of 1.0, -1.0, and 0.0 are assumed based on the value of `flag` and are not required to be set in the `param` vector.

rotmg (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event rotmg(sycl::queue &queue,
                      T *d1,
                      T *d2,
                      T *x1,
                      T  y1,
                      T *param,
                      const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event rotmg(sycl::queue &queue,
                      T *d1,
                      T *d2,
                      T *x1,
                      T  y1,
                      T *param,
                      const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

d1 Pointer to the scaling factor for x-coordinate of the input vector.

d2 Pointer to the scaling factor for y-coordinate of the input vector.

x1 Pointer to x-coordinate of the input vector.

y1 Scalar specifying y-coordinate of the input vector.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

d1 Pointer to the first diagonal element of the updated matrix.

d2 Pointer to the second diagonal element of the updated matrix.

x1 Pointer to x-coordinate of the rotated vector before scaling

param Pointer to an array of size 5.

The elements of the `param` array are:

`param[0]` contains a switch, `flag`. The other array elements `param[1-4]` contain the components of the array `H`: h_{11}, h_{21}, h_{12}, and h_{22}, respectively.

Depending on the values of `flag`, the components of `H` are set as follows:

`flag = -1.0`:

$$H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$$

`flag = 0.0`:

$$H = \begin{bmatrix} 1.0 & h_{12} \\ h_{21} & 1.0 \end{bmatrix}$$

`flag = 1.0`:

$$H = \begin{bmatrix} h_{11} & 1.0 \\ -1.0 & h_{22} \end{bmatrix}$$

`flag = -2.0`:

$$H = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$$

In the last three cases, the matrix entries of 1.0, -1.0, and 0.0 are assumed based on the value of `flag` and are not required to be set in the `param` vector.

Return Values

Output event to wait on to ensure computation is complete.

9.1.14 scal

Computes the product of a vector by a scalar.

Description

The `scal` routines computes a scalar-vector product. The operation is defined as:

$$x \leftarrow alpha * x$$

where:

- x is a vector of n elements
- `alpha` is a scalar

`scal` supports the following precisions:

T	Ts
`sycl::half`	`sycl::half`
`oneapi::mkl::bfloat16`	`oneapi::mkl::bfloat16`
`float`	`float`
`double`	`double`
`std::complex<float>`	`std::complex<float>`
`std::complex<double>`	`std::complex<double>`
`std::complex<float>`	`float`
`std::complex<double>`	`double`

scal (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void scal(sycl::queue &queue,
              std::int64_t n,
              Ts alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void scal(sycl::queue &queue,
              std::int64_t n,
              Ts alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

alpha Specifies the scalar `alpha`.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See ref:**matrix-storage** for more details.

incx Stride of vector x.

Output Parameters

x Buffer holding updated buffer x.

scal (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event scal(sycl::queue &queue,
                     std::int64_t n,
                     Ts alpha,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event scal(sycl::queue &queue,
                     std::int64_t n,
                     Ts alpha,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vector x.

alpha Specifies the scalar `alpha`.

x Pointer to the input vector x. Size of the array must be at least (1 + (n - 1)*abs(`incx`)). See ref:**matrix-storage** for more details.

incx Stride of vector x.

Output Parameters

x Pointer to updated array x.

Return Values

Output event to wait on to ensure computation is complete.

9.1.15 sdsdot

Computes a vector-vector dot product with double precision.

Description

The `sdsdot` routines perform a dot product between two vectors with double precision. The operation is defined as:

$$\text{result} = sb + \sum_{i=1}^{n} X_i Y_i$$

sdsdot (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void sdsdot(sycl::queue &queue,
                std::int64_t n,
                float sb,
                sycl::buffer<float,1> &x,
                std::int64_t incx,
                sycl::buffer<float,1> &y,
                std::int64_t incy,
                sycl::buffer<float,1> &result)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void sdsdot(sycl::queue &queue,
                std::int64_t n,
                float sb,
                sycl::buffer<float,1> &x,
                std::int64_t incx,
                sycl::buffer<float,1> &y,
                std::int64_t incy,
                sycl::buffer<float,1> &result)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors x and y.

sb Single precision scalar to be added to the dot product.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See ref:**matrix-storage** for more details.

incx Stride of vector x.

y Buffer holding input vector y. Size of the buffer must be at least (1 + (n - 1)*abs(`incxy`)). See ref:**matrix-storage** for more details.

incy Stride of vector y.

Output Parameters

result Buffer where the result (a scalar) will be stored. If n < 0 the result is `sb`.

sdsdot (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event sdsdot(sycl::queue &queue,
                       std::int64_t n,
                       float sb,
                       const float *x,
                       std::int64_t incx,
                       const float *y,
                       std::int64_t incy,
                       float *result,
                       const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event sdsdot(sycl::queue &queue,
                       std::int64_t n,
                       float sb,
                       const float *x,
                       std::int64_t incx,
                       const float *y,
                       std::int64_t incy,
                       float *result,
                       const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors x and y.

sb Single precision scalar to be added to the dot product.

x Pointer to input vector x. Size of the array must be at least (1 + (n - 1)*abs(`incx`)). See ref:**matrix-storage** for more details.

incx Stride of vector x.

y Pointer to the input vector y. Size of the array must be at least (1 + (n - 1)*abs(`incy`)). See ref:**matrix-storage** for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

result Pointer to where the result (a scalar) will be stored. If n < 0 the result is `sb`.

Return Values

Output event to wait on to ensure computation is complete.

9.1.16 swap

Swaps a vector with another vector.

Description

Given two vectors of n elements, x and y, the `swap` routines return vectors y and x swapped, each replacing the other. The operation is defined as:

$$\begin{bmatrix} y \\ x \end{bmatrix} \leftarrow \begin{bmatrix} x \\ y \end{bmatrix}$$

`swap` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std::complex<double>`

swap (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void swap(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void swap(sycl::queue &queue,
              std::int64_t n,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors x and y.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See ref:**matrix-storage** for more details.

incx Stride of vector x.

y Buffer holding input vector y. Size of the buffer must be at least (1 + (n - 1)*abs(`incy`)). See ref:**matrix-storage** for more details.

incy Stride of vector y.

Output Parameters

x Buffer holding updated buffer x, that is, the input vector y.

y Buffer holding updated buffer y, that is, the input vector x.

swap (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event swap(sycl::queue &queue,
                     std::int64_t n,
                     T *x,
                     std::int64_t incx,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event swap(sycl::queue &queue,
                     std::int64_t n,
                     T *x,
                     std::int64_t incx,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors x and y.

x Pointer to input vector x. Size of the array must be at least (1 + (n - 1)*abs(`incx`)). See ref:**matrix-storage** for more details.

incx Stride of vector x.

y Pointer to input vector y. Size of the array must be at least (1 + (n - 1)*abs(`incy`)). See ref:**matrix-storage** for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

x Pointer to updated array x, that is, the input vector y.

y Pointer to updated array y, that is, the input vector x.

Return Values

Output event to wait on to ensure computation is complete.

9.2 BLAS Level 2 Routines

This section describes BLAS Level 2 routines, which perform matrix-vector operations. The following table lists the BLAS Level 2 routine groups and the data types associated with them.

Routine Groups	Data Types	Description
gbmv	`float, double, std::complex<float>, std::complex<double>`	Matrix-vector product using a general band matrix
gemv	`float, double, std::complex<float>, std::complex<double>`	Matrix-vector product using a general matrix
ger	`float, double`	Rank-1 update of a general matrix
gerc	`std::complex<float>, std::complex<double>`	Rank-1 update of a conjugated general matrix
geru	`std::complex<float>, std::complex<double>`	Rank-1 update of a general matrix, unconjugated
hbmv	`std::complex<float>, std::complex<double>`	Matrix-vector product using a Hermitian band matrix
hemv	`std::complex<float>, std::complex<double>`	Matrix-vector product using a Hermitian matrix
her	`std::complex<float>, std::complex<double>`	Rank-1 update of a Hermitian matrix
her2	`std::complex<float>, std::complex<double>`	Rank-2 update of a Hermitian matrix
hpmv	`std::complex<float>, std::complex<double>`	Matrix-vector product using a Hermitian packed matrix

continues on next page

Table 2 – continued from previous page

Routine Groups	Data Types	Description
hpr	`std::complex<float>`, `std::complex<double>`	Rank-1 update of a Hermitian packed matrix
hpr2	`std::complex<float>`, `std::complex<double>`	Rank-2 update of a Hermitian packed matrix
sbmv	`float`, `double`	Matrix-vector product using symmetric band matrix
spmv	`float`, `double`	Matrix-vector product using a symmetric packed matrix
spr	`float`, `double`	Rank-1 update of a symmetric packed matrix
spr2	`float`, `double`	Rank-2 update of a symmetric packed matrix
symv	`float`, `double`	Matrix-vector product using a symmetric matrix
syr	`float`, `double`	Rank-1 update of a symmetric matrix
syr2	`float`, `double`	Rank-2 update of a symmetric matrix
tbmv	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Matrix-vector product using a triangular band matrix
tbsv	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Solution of a linear system of equations with a triangular band matrix
tpmv	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Matrix-vector product using a triangular packed matrix
tpsv	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Solution of a linear system of equations with a triangular packed matrix
trmv	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Matrix-vector product using a triangular matrix
trsv	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Solution of a linear system of equations with a triangular matrix

9.2.1 gbmv

Computes a matrix-vector product with a general band matrix.

Description

The gbmv routines compute a scalar-matrix-vector product and add the result to a scalar-vector product, with a general band matrix. The operation is defined as:

$$y \leftarrow alpha * op(A) * x + beta * y$$

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- alpha and beta are scalars
- A is m x n matrix with kl sub-diagonals and ku super-diagonals
- x and y are vectors

gbmv supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

gbmv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void gbmv(queue &queue,
              oneapi::mkl::transpose trans,
              std::int64_t m,
              std::int64_t n,
              std::int64_t kl,
              std::int64_t ku,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void gbmv(queue &queue,
              oneapi::mkl::transpose trans,
              std::int64_t m,
```

(continues on next page)

(continued from previous page)

```
                std::int64_t n,
                std::int64_t kl,
                std::int64_t ku,
                T alpha,
                sycl::buffer<T,1> &a,
                std::int64_t lda,
                sycl::buffer<T,1> &x,
                std::int64_t incx,
                T beta,
                sycl::buffer<T,1> &y,
                std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

m Number of rows of matrix A. Must be at least zero.

n Number of columns of matrix A. Must be at least zero.

kl Number of sub-diagonals of matrix A. Must be at least zero.

ku Number of super-diagonals of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Buffer holding input matrix A. Size of the buffer must be at least `lda` * `n` if column major layout is used, or at least `lda` * `m` if row major layout is used. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least (`kl` + `ku` + 1) and positive.

x Buffer holding input vector x. The length(`len`) of vector x is `n` if A is not transposed, and `m` if A is transposed. Size of the buffer must be at least (1 + (`len` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Buffer holding input/output vector y. The length(`len`) of vector y is `m`, if A is not transposed, and `n` if A is transposed. Size of the buffer must be at least (1 + (`len` - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

Output Parameters

y Buffer holding updated vector y.

gbmv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gbmv(queue &queue,
                     oneapi::mkl::transpose trans,
                     std::int64_t m,
                     std::int64_t n,
                     std::int64_t kl,
                     std::int64_t ku,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event gbmv(queue &queue,
                     oneapi::mkl::transpose trans,
                     std::int64_t m,
                     std::int64_t n,
                     std::int64_t kl,
                     std::int64_t ku,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

m Number of rows of matrix A. Must be at least zero.

n Number of columns of matrix A. Must be at least zero.

kl Number of sub-diagonals of matrix A. Must be at least zero.

ku Number of super-diagonals of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Pointer to input matrix A. Size of the array must be at least `lda` * `n` if column major layout is used, or at least `lda` * `m` if row major layout is used. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least (`kl` + `ku` + 1) and positive.

x Pointer to input vector x. The length `len` of vector x is `n` if A is not transposed, and `m` if A is transposed. Size of the array holding input vector x must be at least (1 + (`len` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Pointer to input/output vector y. The length `len` of vector y is `m`, if A is not transposed, and `n` if A is transposed. Size of the array holding input/output vector y must be at least (1 + (`len` - 1)*abs(`incy`)) where `len` is this length. See Matrix Storage for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.2.2 gemv

Computes a matrix-vector product using a general matrix.

Description

The gemv routines compute a scalar-matrix-vector product and add the result to a scalar-vector product, with a general matrix. The operation is defined as:

$$y \leftarrow alpha * op(A) * x + beta * y$$

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- alpha and beta are scalars
- A is m x n matrix
- x and y are vectors

gemv supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

gemv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void gemv(sycl::queue &queue,
              oneapi::mkl::transpose trans,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void gemv(sycl::queue &queue,
              oneapi::mkl::transpose trans,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

m Number of rows of matrix A. Must be at least zero.

n Number of columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Buffer holding input matrix A. Size of the buffer must be at least `lda` * `n` if column major layout is used, or at least `lda` * `m` if row major layout is used. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be positive and at least `m` if column major layout is used or at least `n` if row major layout is used.

x Buffer holding input vector x. The length `len` of vector x is `n` if A is not transposed, and `m` if A is transposed. Size of the buffer must be at least (1 + (`len` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

beta The scaling factor for vector y.

y Buffer holding input/output vector y. The length `len` of vector y is `m`, if A is not transposed, and `n` if A is transposed. Size of the buffer must be at least (1 + (`len` - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

Output Parameters

y Buffer holding updated vector y.

Examples

An example of how to use gemv can be found in the oneMKL installation directory, under:

```
examples/dpcpp/blas/source/gemv.cpp
```

gemv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemv(sycl::queue &queue,
                     oneapi::mkl::transpose trans,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event gemv(sycl::queue &queue,
                     oneapi::mkl::transpose trans,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

m Number of rows of matrix A. Must be at least zero.

n Number of columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Pointer to input matrix A. Size of the array must be at least `lda` * `n` if column major layout is used, or at least `lda` * `m` if row major layout is used.

lda Leading dimension of matrix A. Must be positive and at least `m` if column major layout is used or at least `n` if row major layout is used.

x Pointer to the input vector x. The length `len` of vector x is `n` if A is not transposed, and `m` if A is transposed. Size of the array holding vector x must be at least (1 + (`len` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Pointer to input/output vector y. The length `len` of vector y is `m`, if A is not transposed, and `n` if A is transposed. Size of the array holding input/output vector y must be at least (1 + (`len` - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.2.3 ger

Computes a rank-1 update of a general matrix.

Description

The `ger` routines compute a scalar-vector-vector product and add the result to a general matrix. The operation is defined as:

$$A \leftarrow alpha * x * y^T + A$$

where:

- `alpha` is scalar
- A is `m` x `n` matrix
- x is a vector length `m`
- y is a vector length `n`

`ger` supports the following precisions:

T
float
double

ger (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void ger(sycl::queue &queue,
             std::int64_t m,
             std::int64_t n,
             T alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &y,
             std::int64_t incy,
             sycl::buffer<T,1> &a,
             std::int64_t lda)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void ger(sycl::queue &queue,
             std::int64_t m,
             std::int64_t n,
             T alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &y,
             std::int64_t incy,
             sycl::buffer<T,1> &a,
             std::int64_t lda)
}
```

Input Parameters

queue The queue where the routine should be executed.

m Number of rows of matrix `A`. Must be at least zero.

n Number of columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector `x`. Size of the buffer must be at least `(1 + (m - 1)*abs(incx))`. See Matrix Storage for more details.

incx Stride of vector `x`.

y Buffer holding input/output vector `y`. Size of the buffer must be at least `(1 + (n - 1)*abs(incy))`. See Matrix Storage for more details.

incy Stride of vector `y`.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda * n` if column major layout is used, or at least `lda * m` if row major layout is used.

lda Leading dimension of matrix `A`. Must be positive and at least `m` if column major layout is used or at least `n` if row major layout is used.

Output Parameters

a Buffer holding updated matrix `A`.

ger (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event ger(sycl::queue &queue,
                    std::int64_t m,
                    std::int64_t n,
                    T alpha,
                    const T *x,
                    std::int64_t incx,
                    const T *y,
                    std::int64_t incy,
                    T *a,
                    std::int64_t lda,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event ger(sycl::queue &queue,
                    std::int64_t m,
                    std::int64_t n,
                    T alpha,
                    const T *x,
                    std::int64_t incx,
                    const T *y,
                    std::int64_t incy,
                    T *a,
                    std::int64_t lda,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

m Number of rows of matrix `A`. Must be at least zero.

n Number of columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector `x`. Size of the array holding input vector `x` must be at least (1 + (`m` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector `x`.

y Pointer to input/output vector `y`. Size of the array holding input/output vector `y` must be at least (1 + (`n` - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector `y`.

a Pointer to input matrix `A`. Size of the array must be at least `lda` * `n` if column major layout is used, or at least `lda` * `m` if row major layout is used.

lda Leading dimension of matrix `A`. Must be positive and at least `m` if column major layout is used or at least `n` if row major layout is used.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated matrix `A`.

Return Values

Output event to wait on to ensure computation is complete.

9.2.4 gerc

Computes a rank-1 update (conjugated) of a general complex matrix.

Description

The `gerc` routines compute a scalar-vector-vector product and add the result to a general matrix. The operation is defined as:

$$A \leftarrow alpha * x * y^H + A$$

where:

- `alpha` is a scalar
- `A` is `m` x `n` matrix
- `x` is a vector of length `m`
- `y` is vector of length `n`

`gerc` supports the following precisions:

T
std::complex<float>
std::complex<double>

gerc (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void gerc(sycl::queue &queue,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a,
              std::int64_t lda)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void gerc(sycl::queue &queue,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a,
              std::int64_t lda)
}
```

Input Parameters

queue The queue where the routine should be executed.

m Number of rows of matrix `A`. Must be at least zero.

n Number of columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (`m` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

y Buffer holding input/output vector y. Size of the buffer must be at least (1 + (`n` - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda` * `n` if column major layout is used, or at least `lda` * `m` if row major layout is used. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be positive and at least `m` if column major layout is used or at least `n` if row major layout is used.

Output Parameters

a Buffer holding updated matrix `A`.

gerc (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gerc(sycl::queue &queue,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     std::int64_t lda,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event gerc(sycl::queue &queue,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     std::int64_t lda,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

m Number of rows of matrix `A`. Must be at least zero.

n Number of columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector `x`. Size of the array holding input vector `x` must be at least `(1 + (m - 1)*abs(incx))`. See Matrix Storage for more details.

incx Stride of vector `x`.

y Pointer to input/output vector `y`. Size of the array holding input/output vector `y` must be at least `(1 + (n - 1)*abs(incy))`. See Matrix Storage for more details.

incy Stride of vector `y`.

a Pointer to input matrix `A`. Size of the array must be at least `lda * n` if column major layout is used, or at least `lda * m` if row major layout is used.

lda Leading dimension of matrix `A`. Must be positive and at least `m` if column major layout is used or at least `n` if row major layout is used.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated matrix `A`.

Return Values

Output event to wait on to ensure computation is complete.

9.2.5 geru

Computes a rank-1 update (unconjugated) of a general complex matrix.

Description

The `geru` routines routines compute a scalar-vector-vector product and add the result to a general matrix. The operation is defined as:

$$A \leftarrow alpha * x * y^T + A$$

where:

- `alpha` is a scalar
- `A` is `m` x `n` matrix
- `x` is a vector of length `m`
- `y` is a vector of length `n`

`geru` supports the following precisions:

T
`std::complex<float>`
`std::complex<double>`

geru (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void geru(sycl::queue &queue,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a,
              std::int64_t lda)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void geru(sycl::queue &queue,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a,
              std::int64_t lda)
}
```

Input Parameters

queue The queue where the routine should be executed.

m Number of rows of matrix `A`. Must be at least zero.

n Number of columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector `x`. Size of the buffer must be at least `(1 + (m - 1)*abs(incx))`. See **Matrix Storage** for more details.

incx Stride of vector `x`.

y Buffer holding input/output vector `y`. Size of the buffer must be at least `(1 + (n - 1)*abs(incy))`. See **Matrix Storage** for more details.

incy Stride of vector `y`.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda * n` if column major layout is used, or at least `lda * m` if row major layout is used.

lda Leading dimension of matrix `A`. Must be positive and at least `m` if column major layout is used or at least `n` if row major layout is used.

Output Parameters

a Buffer holding updated matrix A.

geru (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event geru(sycl::queue &queue,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     std::int64_t lda,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event geru(sycl::queue &queue,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     std::int64_t lda,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

m Number of rows of matrix A. Must be at least zero.

n Number of columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (m - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

y Pointer to input/output vector y. Size of the array holding input/output vector y must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

a Pointer to input matrix A. Size of the array must be at least lda * n if column major layout is used, or at least lda * m if row major layout is used.

lda Leading dimension of matrix A. Must be positive and at least m if column major layout is used or at least n if row major layout is used.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated matrix A.

Return Values

Output event to wait on to ensure computation is complete.

9.2.6 hbmv

Computes a matrix-vector product using a hermitian band matrix.

Description

The hbmv routines compute a scalar-matrix-vector product and add the result to a scalar-vector product, with a hermitian band matrix. The operation is defined as:

$$y \leftarrow alpha * A * x + beta * y$$

where:

- alpha and beta are scalars
- A is n x n hermitian band matrix, with k super-diagonals
- x and y are vectors of length n

hbmv supports the following precisions:

T
std::complex<float>
std::complex<double>

hbmv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void hbmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              std::int64_t k,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void hbmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              std::int64_t k,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix `A`. Must be at least zero.

k Number of super-diagonals of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda` * `n`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least (`k` + 1) and positive.

x Buffer holding input vector `x`. Size of the buffer must be at least (1 + (`n` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Buffer holding input/output vector y. Size of the buffer must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

Output Parameters

y Buffer holding updated vector y.

hbmv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event hbmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     std::int64_t k,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event hbmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     std::int64_t k,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix `A`. Must be at least zero.

k Number of super-diagonals of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Pointer to input matrix `A`. Size of the array holding input matrix `A` must be at least `lda` * `n`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least (`k` + 1) and positive.

x Pointer to input vector `x`. Size of the array holding input vector `x` must be at least (1 + (`n` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector `x`.

beta Scaling factor for vector `y`.

y Pointer to input/output vector `y`. Size of the array holding input/output vector `y` must be at least (1 + (`n` - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector `y`.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to updated vector `y`.

Return Values

Output event to wait on to ensure computation is complete.

9.2.7 hemv

Computes a matrix-vector product using a hermitian matrix.

Description

The hemv routines compute a scalar-matrix-vector product and add the result to a scalar-vector product, with a hermitian matrix. The operation is defined as:

$$y \leftarrow alpha * A * x + beta * y$$

where:

- alpha and beta are scalars
- A is n x n hermitian matrix
- x and y are vectors of length n

hemv supports the following precisions:

T
std::complex<float>
std::complex<double>

hemv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void hemv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void hemv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Buffer holding input matrix A. Size of the buffer must be at least lda * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Buffer holding input/output vector y. Size of the buffer must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

Output Parameters

y Buffer holding updated vector y.

hemv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event hemv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event hemv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least `lda` * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Pointer to input/output vector y. Size of the array holding input/output vector y must be at least (1 + (n - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.2.8 her

Computes a rank-1 update of a hermitian matrix.

Description

The her routines compute a scalar-vector-vector product and add the result to a hermitian matrix. The operation is defined as:

$$A \leftarrow alpha * x * x^H + A$$

where:

- alpha is scalar
- A is n x n hermitian matrix
- x is a vector of length n

her supports the following precisions:

T	Treal
std::complex<float>	float
std::complex<double>	double

her (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void her(sycl::queue &queue,
             oneapi::mkl::uplo upper_lower,
             std::int64_t n,
             Treal alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &a,
             std::int64_t lda)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void her(sycl::queue &queue,
             oneapi::mkl::uplo upper_lower,
             std::int64_t n,
             Treal alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &a,
             std::int64_t lda)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda` * n. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least n and positive.

Output Parameters

a Buffer holding updated upper triangular part of the hermitian matrix `A` if `upper_lower=upper` or updated lower triangular part of the hermitian matrix `A` if `upper_lower=lower`.

The imaginary parts of the diagonal elements are set to zero.

her (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event her(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    std::int64_t n,
                    Treal alpha,
                    const T *x,
                    std::int64_t incx,
```

(continues on next page)

(continued from previous page)

```
                    T *a,
                    std::int64_t lda,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event her(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    std::int64_t n,
                    Treal alpha,
                    const T *x,
                    std::int64_t incx,
                    T *a,
                    std::int64_t lda,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least lda * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated upper triangular part of the hermitian matrix A if upper_lower=upper or updated lower triangular part of the hermitian matrix A if upper_lower=lower.

The imaginary parts of the diagonal elements are set to zero.

Return Values

Output event to wait on to ensure computation is complete.

9.2.9 her2

Computes a rank-2 update of a hermitian matrix.

Description

The `her2` routines compute two scalar-vector-vector products and add them to a hermitian matrix. The operation is defined as:

$$A \leftarrow alpha * x * y^H + conjg(alpha) * y * x^H + A$$

where:

- `alpha` is a scalar
- `A` is n x n hermitian matrix
- x and y are vectors or length n

`her2` supports the following precisions:

T
std::complex<float>
std::complex<double>

her2 (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void her2(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a,
              std::int64_t lda)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void her2(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a,
              std::int64_t lda)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

y Buffer holding input/output vector y. Size of the buffer must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

a Buffer holding input matrix A. Size of the buffer must be at least lda * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

Output Parameters

a Buffer holding updated upper triangular part of the hermitian matrix A if upper_lower=upper, or updated lower triangular part of the hermitian matrix A if upper_lower=lower.

The imaginary parts of the diagonal elements are set to zero.

her2 (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event her2(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     std::int64_t lda,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event her2(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     std::int64_t lda,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

y Pointer to input/output vector y. Size of the array holding input/output vector y must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

a Pointer to input matrix `A`. Size of the array holding input matrix `A` must be at least `lda * n`. See **Matrix Storage** for more details.

lda Leading dimension of matrix `A`. Must be at least `n` and positive.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated upper triangular part of the hermitian matrix `A` if `upper_lower=upper`, or updated lower triangular part of the hermitian matrix `A` if `upper_lower=lower`.

The imaginary parts of the diagonal elements are set to zero.

Return Values

Output event to wait on to ensure computation is complete.

9.2.10 hpmv

Computes a matrix-vector product using a hermitian packed matrix.

Description

The `hpmv` routines compute a scalar-matrix-vector product and add the result to a scalar-vector product, with a hermitian packed matrix. The operation is defined as:

$$y \leftarrow alpha * A * x + beta * y$$

where:

- `alpha` and `beta` are scalars
- `A` is `n` x `n` hermitian matrix supplied in packed form
- `x` and `y` are vectors of length `n`

`hpmv` supports the following precisions:

T
`std::complex<float>`
`std::complex<double>`

hpmv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void hpmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void hpmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Buffer holding input matrix A. Size of the buffer must be at least (n*(n+1))/2. See Matrix Storage for more details.

The imaginary parts of the diagonal elements need not be set and are assumed to be zero.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Buffer holding input/output vector y. Size of the buffer must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

Output Parameters

y Buffer holding updated vector y.

hpmv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event hpmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event hpmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least (n*(n+1))/2. See Matrix Storage for more details.

The imaginary parts of the diagonal elements need not be set and are assumed to be zero.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Pointer to input/output vector y. Size of the array holding input/output vector y must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.2.11 hpr

Computes a rank-1 update of a hermitian packed matrix.

Description

The `hpr` routines compute a scalar-vector-vector product and add the result to a hermitian packed matrix. The operation is defined as:

$$A \leftarrow alpha * x * x^H + A$$

where:

- `alpha` is scalar
- A is n x n hermitian matrix, supplied in packed form
- x is a vector of length n

`hpr` supports the following precisions:

T	Treal
`std::complex<float>`	`float`
`std::complex<double>`	`double`

hpr (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void hpr(sycl::queue &queue,
             oneapi::mkl::uplo upper_lower,
             std::int64_t n,
             Treal alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &a)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void hpr(sycl::queue &queue,
             oneapi::mkl::uplo upper_lower,
             std::int64_t n,
             Treal alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &a)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

a Buffer holding input matrix A. Size of the buffer must be at least (n*(n-1))/2. See Matrix Storage for more details.

The imaginary parts of the diagonal elements do not need to be set and are assumed to be zero.

Output Parameters

a Buffer holding updated upper triangular part of the hermitian matrix A if `upper_lower=upper`, or updated lower triangular part of the hermitian matrix A if `upper_lower=lower`.

The imaginary parts of the diagonal elements are set to zero.

hpr (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event hpr(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    std::int64_t n,
                    Treal alpha,
                    const T *x,
                    std::int64_t incx,
                    T *a,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event hpr(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    std::int64_t n,
                    Treal alpha,
                    const T *x,
                    std::int64_t incx,
                    T *a,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least (n*(n-1))/2. See Matrix Storage for more details.

The imaginary parts of the diagonal elements do not need to be set and are assumed to be zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated upper triangular part of the hermitian matrix A if `upper_lower=upper`, or updated lower triangular part of the hermitian matrix A if `upper_lower=lower`.

The imaginary parts of the diagonal elements are set to zero.

Return Values

Output event to wait on to ensure computation is complete.

9.2.12 hpr2

Performs a rank-2 update of a hermitian packed matrix.

Description

The hpr2 routines compute two scalar-vector-vector products and add them to a hermitian packed matrix. The operation is defined as:

$$A \leftarrow alpha * x * y^H + conjg(alpha) * y * x^H + A$$

where:

- alpha is a scalar
- A is n x n hermitian matrix, supplied in packed form

- x and y are vectors of length n

hpr2 supports the following precisions:

T
std::complex<float>
std::complex<double>

hpr2 (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void hpr2(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void hpr2(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

y Buffer holding input/output vector y. Size of the buffer must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

a Buffer holding input matrix A. Size of the buffer must be at least (n*(n-1))/2. See Matrix Storage for more details.

The imaginary parts of the diagonal elements need not be set and are assumed to be zero.

Output Parameters

a Buffer holding updated upper triangular part of the hermitian matrix A if upper_lower=upper, or updated lower triangular part of the hermitian matrix A if upper_lower=lower.

The imaginary parts of the diagonal elements are set to zero.

hpr2 (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event hpr2(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event hpr2(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector `x`. Size of the array holding input vector `x` must be at least `(1 + (n - 1)*abs(incx))`. See Matrix Storage for more details.

incx Stride of vector `x`.

y Pointer to input/output vector `y`. Size of the array holding input/output vector `y` must be at least `(1 + (n - 1)*abs(incy))`. See Matrix Storage for more details.

incy Stride of vector `y`.

a Pointer to input matrix `A`. Size of the array holding input matrix `A` must be at least `(n*(n-1))/2`. See Matrix Storage for more details.

The imaginary parts of the diagonal elements need not be set and are assumed to be zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated upper triangular part of the hermitian matrix `A` if `upper_lower=upper`, or updated lower triangular part of the hermitian matrix `A` if `upper_lower=lower`.

The imaginary parts of the diagonal elements are set to zero.

Return Values

Output event to wait on to ensure computation is complete.

9.2.13 sbmv

Computes a matrix-vector product with a symmetric band matrix.

Description

The sbmv routines compute a scalar-matrix-vector product and add the result to a scalar-vector product, with a symmetric band matrix. The operation is defined as:

$$y \leftarrow alpha * A * x + beta * y$$

where:

- alpha and beta are scalars
- A is n x n symmetric matrix with k super-diagonals
- x and y are vectors of length n

sbmv supports the following precisions:

T
float
double

sbmv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void sbmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              std::int64_t k,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void sbmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              std::int64_t k,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
```

(continues on next page)

(continued from previous page)

```
                T beta,
                sycl::buffer<T,1> &y,
                std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

k Number of super-diagonals of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Buffer holding input matrix A. Size of the buffer must be at least `lda` * n. See ref:**matrix-storage** for more details.

lda Leading dimension of matrix A. Must be at least (k + 1) and positive.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See ref:**matrix-storage** for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Buffer holding input/output vector y. Size of the buffer must be at least (1 + (n - 1)*abs(`incy`)). See ref:**matrix-storage** for more details.

incy Stride of vector y.

Output Parameters

y Buffer holding updated vector y.

sbmv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event sbmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     std::int64_t k,
                     T alpha,
                     const T *a,
```

(continues on next page)

(continued from previous page)

```
                    std::int64_t lda,
                    const T *x,
                    std::int64_t incx,
                    T beta,
                    T *y,
                    std::int64_t incy,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event sbmv(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    std::int64_t n,
                    std::int64_t k,
                    T alpha,
                    const T *a,
                    std::int64_t lda,
                    const T *x,
                    std::int64_t incx,
                    T beta,
                    T *y,
                    std::int64_t incy,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

k Number of super-diagonals of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least `lda` * `n`. See ref:**matrix-storage** for more details.

lda Leading dimension of matrix A. Must be at least (k + 1) and positive.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(`incx`)). See ref:**matrix-storage** for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Pointer to input/output vector y. Size of the array holding input/output vector y must be at least (1 + (n - 1)*abs(`incy`)). See ref:**matrix-storage** for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.2.14 spmv

Computes a matrix-vector product with a symmetric packed matrix.

Description

The `spmv` routines compute a scalar-matrix-vector product and add the result to a scalar-vector product, with a symmetric packed matrix. The operation is defined as:

$$y \leftarrow alpha * A * x + beta * y$$

where:

- `alpha` and `beta` are scalars
- `A` is n x n symmetric matrix, supplied in packed form
- x and y are vectors of length n

`spmv` supports the following precisions:

T
float
double

spmv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void spmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void spmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              T beta,
              sycl::buffer<T,1> &y,
              std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Buffer holding input matrix A. Size of the buffer must be at least (n*(n+1))/2. See Matrix Storage for more details.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Buffer holding input/output vector y. Size of the buffer must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

Output Parameters

y Buffer holding updated vector y.

spmv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event spmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event spmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least (n*(n+1))/2. See Matrix Storage for more details.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

beta Scaling factor for vector y.

y Pointer to input/output vector y. Size of the array holding input/output vector y must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.2.15 spr

Performs a rank-1 update of a symmetric packed matrix.

Description

The spr routines compute a scalar-vector-vector product and add the result to a symmetric packed matrix. The operation is defined as;

$$A \leftarrow alpha * x * x^T + A$$

where:

- alpha is scalar
- A is n x n symmetric matrix, supplied in packed form
- x is a vector of length n

spr supports the following precisions:

T
float
double

spr (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void spr(sycl::queue &queue,
             oneapi::mkl::uplo upper_lower,
             std::std::int64_t n,
             T alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &a)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void spr(sycl::queue &queue,
             oneapi::mkl::uplo upper_lower,
             std::std::int64_t n,
             T alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &a)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

a Buffer holding input matrix A. Size of the buffer must be at least (n*(n-n))/2. See Matrix Storage for more details.

Output Parameters

a Buffer holding updated upper triangular part of the symmetric matrix A if `upper_lower=upper`, or updated lower triangular part of the symmetric matrix A if `upper_lower=lower`.

spr (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event spr(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    std::int64_t n,
                    T alpha,
                    const T *x,
                    std::int64_t incx,
                    T *a,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event spr(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    std::int64_t n,
                    T alpha,
                    const T *x,
                    std::int64_t incx,
                    T *a,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least (n*(n-n))/2. See Matrix Storage for more details.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated upper triangular part of the symmetric matrix `A` if `upper_lower=upper`, or updated lower triangular part of the symmetric matrix `A` if `upper_lower=lower`.

Return Values

Output event to wait on to ensure computation is complete.

9.2.16 spr2

Computes a rank-2 update of a symmetric packed matrix.

Description

The `spr2` routines compute two scalar-vector-vector products and add them to a symmetric packed matrix. The operation is defined as:

$$A \leftarrow alpha * x * y^T + alpha * y * x^T + A$$

where:

- `alpha` is scalar
- `A` is `n` x `n` symmetric matrix, supplied in packed form
- `x` and `y` are vectors of length `n`

`spr2` supports the following precisions:

T
float
double

spr2 (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void spr2(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void spr2(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

y Buffer holding input/output vector y. Size of the buffer must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

a Buffer holding input matrix A. Size of the buffer must be at least (n*(n-1))/2. See Matrix Storage for more details.

Output Parameters

a Buffer holding updated upper triangular part of the symmetric matrix A if upper_lower=upper or updated lower triangular part of the symmetric matrix A if upper_lower=lower.

spr2 (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event spr2(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event spr2(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

y Pointer to input/output vector y. Size of the array holding input/output vector y must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least (n*(n-1))/2. See Matrix Storage for more details.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated upper triangular part of the symmetric matrix `A` if `upper_lower=upper` or updated lower triangular part of the symmetric matrix `A` if `upper_lower=lower`.

Return Values

Output event to wait on to ensure computation is complete.

9.2.17 symv

Computes a matrix-vector product for a symmetric matrix.

Description

The `symv` routines compute a scalar-matrix-vector product and add the result to a scalar-vector product, with a symmetric matrix. The operation is defined as:

$$y \leftarrow alpha * A * x + beta * y$$

where:

- `alpha` and `beta` are scalars
- `A` is `n` x `n` symmetric matrix
- `x` and `y` are vectors of length `n`

`symv` supports the following precisions:

T
float
double

symv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void symv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
```

(continues on next page)

(continued from previous page)

```
            T beta,
            sycl::buffer<T,1> &y,
            std::int64_t incy)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void symv(sycl::queue &queue,
            oneapi::mkl::uplo upper_lower,
            std::int64_t n,
            T alpha,
            sycl::buffer<T,1> &a,
            std::int64_t lda,
            sycl::buffer<T,1> &x,
            std::int64_t incx,
            T beta,
            sycl::buffer<T,1> &y,
            std::int64_t incy)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda` * `n`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least `n` and positive.

x Buffer holding input vector `x`. Size of the buffer must be at least (1 + (`n` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector `x`.

y Buffer holding input/output vector `y`. Size of the buffer must be at least (1 + (`n` - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride of vector `y`.

Output Parameters

y Buffer holding updated vector y.

symv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event symv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event symv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *x,
                     std::int64_t incx,
                     T beta,
                     T *y,
                     std::int64_t incy,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

n Number of rows and columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

a Pointer to input matrix `A`. Size of the array holding input matrix `A` must be at least `lda` * `n`. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

y Pointer to input/output vector y. Size of the array holding input/output vector y must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.2.18 syr

Computes a rank-1 update of a symmetric matrix.

Description

The syr routines compute a scalar-vector-vector product and add the result to a matrix, with a symmetric matrix. The operation is defined as:

$$A \leftarrow alpha * x * x^T + A$$

where:

- alpha is scalar
- A is n x n symmetric matrix
- x is a vector of length n

syr supports the following precisions:

T
float
double

syr (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void syr(sycl::queue &queue,
             oneapi::mkl::uplo upper_lower,
             std::int64_t n,
             T alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &a,
             std::int64_t lda)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void syr(sycl::queue &queue,
             oneapi::mkl::uplo upper_lower,
             std::int64_t n,
             T alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             sycl::buffer<T,1> &a,
             std::int64_t lda)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

a Buffer holding input matrix A. Size of the buffer must be at least lda * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

Output Parameters

a Buffer holding updated upper triangular part of the symmetric matrix `A` if `upper_lower=upper` or updated lower triangular part of the symmetric matrix `A` if `upper_lower=lower`.

syr (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event syr(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    std::int64_t n,
                    T alpha,
                    const T *x,
                    std::int64_t incx,
                    T *a,
                    std::int64_t lda,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event syr(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    std::int64_t n,
                    T alpha,
                    const T *x,
                    std::int64_t incx,
                    T *a,
                    std::int64_t lda,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

n Number of columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector `x`. Size of the array holding input vector `x` must be at least (1 + (`n` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector `x`.

a Pointer to input matrix `A`. Size of the array holding input matrix `A` must be at least `lda` * `n`. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated upper triangular part of the symmetric matrix A if `upper_lower=upper` or updated lower triangular part of the symmetric matrix A if `upper_lower=lower`.

Return Values

Output event to wait on to ensure computation is complete.

9.2.19 syr2

Computes a rank-2 update of a symmetric matrix.

Description

The `syr2` routines compute two scalar-vector-vector products, add them and add the result to a matrix, with a symmetric matrix. The operation is defined as:

$$A \leftarrow alpha * x * y^T + alpha * y * x^T + A$$

where:

- `alpha` is a scalar
- A is n x n symmetric matrix
- x and y are vectors of length n

`syr2` supports the following precisions:

T
`float`
`double`

syr2 (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void syr2(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a,
              std::int64_t lda)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void syr2(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx,
              sycl::buffer<T,1> &y,
              std::int64_t incy,
              sycl::buffer<T,1> &a,
              std::int64_t lda)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

n Number of columns of matrix A. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

y Buffer holding input/output vector y. Size of the buffer must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

a Buffer holding input matrix A. Size of the buffer must be at least lda * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

Output Parameters

a Buffer holding updated upper triangular part of the symmetric matrix `A` if `upper_lower=upper`, or updated lower triangular part of the symmetric matrix `A` if `upper_lower=lower`.

syr2 (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event syr2(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     std::int64_t lda,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event syr2(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t n,
                     T alpha,
                     const T *x,
                     std::int64_t incx,
                     const T *y,
                     std::int64_t incy,
                     T *a,
                     std::int64_t lda,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

n Number of columns of matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix-vector product.

x Pointer to input vector `x`. Size of the array holding input vector `x` must be at least `(1 + (n - 1)*abs(incx))`. See Matrix Storage for more details.

incx Stride of vector x.

y Pointer to input/output vector y. Size of the array holding input/output vector y must be at least (1 + (n - 1)*abs(incy)). See Matrix Storage for more details.

incy Stride of vector y.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least lda * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

a Pointer to updated upper triangular part of the symmetric matrix A if upper_lower=upper, or updated lower triangular part of the symmetric matrix A if upper_lower=lower.

Return Values

Output event to wait on to ensure computation is complete.

9.2.20 tbmv

Computes a matrix-vector product using a triangular band matrix.

Description

The tbmv routines compute a matrix-vector product with a triangular band matrix. The operation is defined as:

$$x \leftarrow op(A) * x$$

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- A is n x n unit or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals
- x is a vector of length n

tbmv supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

tbmv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void tbmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              std::int64_t k,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void tbmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              std::int64_t k,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details..

trans Specifies op(`A`), the transposition operation applied to matrix `A`. See Data Types for more details.

unit_diag Specifies whether matrix `A` is unit triangular or not. See Data Types for more details.

n Numbers of rows and columns of matrix `A`. Must be at least zero.

k Number of sub/super-diagonals of matrix `A`. Must be at least zero.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda` * `n`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least (`k` + 1) and positive.

x Buffer holding input vector `x`. Size of the buffer must be at least (1 + (`n` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector `x`.

Output Parameters

x Buffer holding updated vector x.

tbmv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event tbmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t n,
                     std::int64_t k,
                     const T *a,
                     std::int64_t lda,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event tbmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t n,
                     std::int64_t k,
                     const T *a,
                     std::int64_t lda,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

unit_diag Specifies whether matrix A is unit triangular or not. See Data Types for more details.

n Numbers of rows and columns of matrix A. Must be at least zero.

k Number of sub/super-diagonals of matrix A. Must be at least zero.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least `lda` * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least (k + 1) and positive.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

x Pointer to updated vector x.

Return Values

Output event to wait on to ensure computation is complete.

9.2.21 tbsv

Solves a system of linear equations whose coefficients are in a triangular band matrix.

Description

The `tbsv` routines solve a system of linear equations whose coefficients are in a triangular band matrix. The operation is defined as:

$$op(A) * x = b$$

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- A is n x n unit or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals
- b and x are vectors of length n

`tbsv` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std::complex<double>`

tbsv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void tbsv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              std::int64_t k,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void tbsv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              std::int64_t k,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

unit_diag Specifies whether matrix A is unit triangular or not. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

k Number of sub/super-diagonals of matrix A. Must be at least zero.

a Buffer holding input matrix A. Size of the buffer must be at least `lda` * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least (k + 1) and positive.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

Output Parameters

x Buffer holding solution vector x.

tbsv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event tbsv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t n,
                     std::int64_t k,
                     const T *a,
                     std::int64_t lda,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event tbsv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t n,
                     std::int64_t k,
                     const T *a,
                     std::int64_t lda,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

unit_diag Specifies whether matrix A is unit triangular or not. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

k Number of sub/super-diagonals of matrix A. Must be at least zero.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least lda * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least (k + 1) and positive.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

x Pointer to solution vector x.

Return Values

Output event to wait on to ensure computation is complete.

9.2.22 tpmv

Computes a matrix-vector product using a triangular packed matrix.

Description

The tpmv routines compute a matrix-vector product with a triangular packed matrix. The operation is defined as:

$$x \leftarrow op(A) * x$$

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- A is n x n unit or non-unit, upper or lower triangular band matrix, supplied in packed form
- x is a vector of length n

tpmv supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

tpmv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void tpmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              sycl::buffer<T,1> &a,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void tpmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              sycl::buffer<T,1> &a,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

unit_diag Specifies whether matrix A is unit triangular or not. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

a Buffer holding input matrix A. Size of the buffer must be at least (n*(n+1))/2. See Matrix Storage for more details.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

Output Parameters

x Buffer holding updated vector x.

tpmv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event tpmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t n,
                     const T *a,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event tpmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t n,
                     const T *a,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

unit_diag Specifies whether matrix A is unit triangular or not. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least (n*(n+1))/2. See Matrix Storage for more details.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

x Pointer to updated vector x.

Return Values

Output event to wait on to ensure computation is complete.

9.2.23 tpsv

Solves a system of linear equations whose coefficients are in a triangular packed matrix.

Description

The `tpsv` routines solve a system of linear equations whose coefficients are in a triangular packed matrix. The operation is defined as:

$$op(A) * x = b$$

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- A is n x n unit or non-unit, upper or lower triangular band matrix, supplied in packed form
- b and x are vectors of length n

`tpsv` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

tpsv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void tpsv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              std::int64_t k,
              sycl::buffer<T,1> &a,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void tpsv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              std::int64_t k,
              sycl::buffer<T,1> &a,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

unit_diag Specifies whether matrix A is unit triangular or not. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

a Buffer holding input matrix A. Size of the buffer must be at least (n*(n+1))/2. See Matrix Storage for more details.

x Buffer holding the n-element right-hand side vector b. Size of the buffer must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

Output Parameters

x Buffer holding solution vector x.

tpsv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event tpsv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t n,
                     std::int64_t k,
                     const T *a,
```

(continues on next page)

(continued from previous page)

```
                    T *x,
                    std::int64_t incx,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event tpsv(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    oneapi::mkl::transpose trans,
                    oneapi::mkl::diag unit_diag,
                    std::int64_t n,
                    std::int64_t k,
                    const T *a,
                    T *x,
                    std::int64_t incx,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

unit_diag Specifies whether matrix A is unit triangular or not. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least (n*(n+1))/2. See Matrix Storage for more details.

x Pointer to the n-element right-hand side vector b. Size of the array holding the n-element right-hand side vector b must be at least (1 + (n - 1)*abs(incx)). See Matrix Storage for more details.

incx Stride of vector x.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

x Pointer to solution vector x.

Return Values

Output event to wait on to ensure computation is complete.

9.2.24 trmv

Computes a matrix-vector product using a triangular matrix.

Description

The `trmv` routines compute a matrix-vector product with a triangular matrix. The operation is defined as:

$$x \leftarrow op(A) * x$$

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- A is n x n unit or non-unit, upper or lower triangular band matrix
- x is a vector of length n

`trmv` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

trmv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void trmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void trmv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

unit_diag Specifies whether matrix A is unit triangular or not. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

a Buffer holding input matrix A. Size of the buffer must be at least `lda` * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

x Buffer holding input vector x. Size of the buffer must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

Output Parameters

x Buffer holding updated vector x.

trmv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event trmv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t n,
                     const T *a,
```

(continues on next page)

(continued from previous page)

```
                    std::int64_t lda,
                    T *x,
                    std::int64_t incx,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event trmv(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    oneapi::mkl::transpose trans,
                    oneapi::mkl::diag unit_diag,
                    std::int64_t n,
                    const T *a,
                    std::int64_t lda,
                    T *x,
                    std::int64_t incx,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

unit_diag Specifies whether matrix A is unit triangular or not. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least `lda` * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

x Pointer to input vector x. Size of the array holding input vector x must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

x Pointer to updated vector x.

Return Values

Output event to wait on to ensure computation is complete.

9.2.25 trsv

Solves a system of linear equations whose coefficients are in a triangular matrix.

Description

The `trsv` routines solve a system of linear equations whose coefficients are in a triangular matrix. The operation is defined as:

$$op(A) * x = b$$

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- A is n x n unit or non-unit, upper or lower triangular matrix
- b and x are vectors of length n

`trsv` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

trsv (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void trsv(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t n,
              std::int64_t k,
```

(continues on next page)

(continued from previous page)

```
            sycl::buffer<T,1> &a,
            std::int64_t lda,
            sycl::buffer<T,1> &x,
            std::int64_t incx)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void trsv(sycl::queue &queue,
            oneapi::mkl::uplo upper_lower,
            oneapi::mkl::transpose trans,
            oneapi::mkl::diag unit_diag,
            std::int64_t n,
            std::int64_t k,
            sycl::buffer<T,1> &a,
            std::int64_t lda,
            sycl::buffer<T,1> &x,
            std::int64_t incx)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), the transposition operation applied to matrix `A`. See Data Types for more details.

unit_diag Specifies whether matrix `A` is unit triangular or not. See Data Types for more details.

n Number of rows and columns of matrix `A`. Must be at least zero.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda` * `n`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least `n` and positive.

x Buffer holding the `n`-element right-hand side vector `b`. Size of the buffer must be at least (1 + (`n` - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector `x`.

Output Parameters

x Buffer holding solution vector `x`.

trsv (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event trsv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t n,
                     std::int64_t k,
                     const T *a,
                     std::int64_t lda,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event trsv(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t n,
                     std::int64_t k,
                     const T *a,
                     std::int64_t lda,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix A is upper or lower triangular. See Data Types for more details.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

unit_diag Specifies whether matrix A is unit triangular or not. See Data Types for more details.

n Number of rows and columns of matrix A. Must be at least zero.

a Pointer to input matrix A. Size of the array holding input matrix A must be at least `lda` * n. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least n and positive.

x Pointer to the n-element right-hand side vector b. Size of the array holding the n-element right-hand side vector b must be at least (1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride of vector x.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

x Pointer to solution vector x.

Return Values

Output event to wait on to ensure computation is complete.

9.3 BLAS Level 3 Routines

BLAS Level 3 routines perform matrix-matrix operations. The following table lists the BLAS Level 3 routine groups and the data types associated with them.

Routine Group	Data Types	Description
gemm	`std::int8_t`, `oneapi::mkl::bfloat16`, `sycl::half`, `float`, `double`, `std::complex<float>`, `std::complex<double>`, mixed	Computes a matrix-matrix product with general matrices.
hemm	`std::complex<float>`, `std::complex<double>`	Computes a matrix-matrix product where one input matrix is Hermitian and one is general.
herk	`std::complex<float>`, `std::complex<double>`	Performs a Hermitian rank-k update.
her2k	`std::complex<float>`, `std::complex<double>`	Performs a Hermitian rank-2k update.
symm	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Computes a matrix-matrix product where one input matrix is symmetric and one matrix is general.
syrk	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Performs a symmetric rank-k update.
syr2k	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Performs a symmetric rank-2k update.
trmm	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Computes a matrix-matrix product where one input matrix is triangular and one input matrix is general.
trsm	`float`, `double`, `std::complex<float>`, `std::complex<double>`	Solves a triangular matrix equation (forward or backward solve).

The BLAS functions are blocked where possible to restructure the code in a way that increases the localization of data reference, enhances cache memory use, and reduces the dependency on the memory bus.

The code is distributed across the processors to maximize parallelism.

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

9.3.1 gemm

Computes a matrix-matrix product with general matrices.

Description

The `gemm` routines compute a scalar-matrix-matrix product and add the result to a scalar-matrix product, with general matrices. The operation is defined as:

$$C \leftarrow alpha * op(A) * op(B) + beta * C$$

where:

- op(`X`) is one of op(`X`) = `X`, or op(`X`) = X^T, or op(`X`) = X^H
- `alpha` and `beta` are scalars
- `A`, `B` and `C` are matrices
- op(`A`) is `m` x `k` matrix
- op(`B`) is `k` x `n` matrix
- `C` is `m` x `n` matrix

`gemm` supports the following precisions:

Ta (A matrix)	Tb (B matrix)	Tc (C matrix)	Ts (alpha/beta)
`sycl::half`	`sycl::half`	`sycl::half`	`sycl::half`
`sycl::half`	`sycl::half`	`float`	`float`
`oneapi::mkl::bfloat16`	`oneapi::mkl::bfloat16`	`oneapi::mkl::bfloat16`	`float`
`oneapi::mkl::bfloat16`	`oneapi::mkl::bfloat16`	`float`	`float`
`std::int8_t`	`std::int8_t`	`std::int32_t`	`float`
`std::int8_t`	`std::int8_t`	`float`	`float`
`float`	`float`	`float`	`float`
`double`	`double`	`double`	`double`
`std::complex<float>`	`std::complex<float>`	`std::complex<float>`	`std::complex<float>`
`std::complex<double>`	`std::complex<double>`	`std::complex<double>`	`std::complex<double>`

gemm (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void gemm(sycl::queue &queue,
              oneapi::mkl::transpose transa,
              oneapi::mkl::transpose transb,
              std::int64_t m,
              std::int64_t n,
              std::int64_t k,
              Ts alpha,
              sycl::buffer<Ta,1> &a,
              std::int64_t lda,
              sycl::buffer<Tb,1> &b,
              std::int64_t ldb,
              Ts beta,
              sycl::buffer<Tc,1> &c,
              std::int64_t ldc,
              compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void gemm(sycl::queue &queue,
              oneapi::mkl::transpose transa,
              oneapi::mkl::transpose transb,
              std::int64_t m,
              std::int64_t n,
              std::int64_t k,
              Ts alpha,
              sycl::buffer<Ta,1> &a,
              std::int64_t lda,
              sycl::buffer<Tb,1> &b,
              std::int64_t ldb,
              Ts beta,
              sycl::buffer<Tc,1> &c,
              std::int64_t ldc,
              compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

transa Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

transb Specifies op(B), the transposition operation applied to matrix B. See Data Types for more details.

m Number of rows of matrix op(A) and matrix C. Must be at least zero.

n Number of columns of matrix op(B) and matrix C. Must be at least zero.

k Number of columns of matrix op(A) and rows of matrix op(B). Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Buffer holding input matrix A. See **Matrix Storage** for more details.

	transa = transpose::nontrans	transa = transpose::trans **or** trans = transpose::conjtrans
Column major	A is m x k matrix. Size of array a must be at least lda * k	A is k x m matrix. Size of array a must be at least lda * m
Row major	A is m x k matrix. Size of array a must be at least lda * m	A is k x m matrix. Size of array a must be at least lda * k

lda Leading dimension of matrix A. Must be positive.

	transa = transpose::nontrans	transa = transpose::trans **or** trans = transpose::conjtrans
Column major	Must be at least m	Must be at least k
Row major	Must be at least k	Must be at least m

b Buffer holding input matrix B. See **Matrix Storage** for more details.

	transb = transpose::nontrans	transb = transpose::trans **or** trans = transpose::conjtrans
Column major	B is k x n matrix. Size of array b must be at least ldb * n	B is n x k matrix. Size of array b must be at least ldb * k
Row major	B is k x n matrix. Size of array b must be at least ldb * k	B is n x k matrix. Size of array b must be at least ldb * n

ldb Leading dimension of matrix B. Must be positive.

	transb = transpose::nontrans	transb = transpose::trans **or** trans = transpose::conjtrans
Column major	Must be at least k	Must be at least n
Row major	Must be at least n	Must be at least k

beta Scaling factor for matrix C.

c Buffer holding input/output matrix C. See **Matrix Storage** for more details.

Column major	C is m x n matrix. Size of array c must be at least ldc * n
Row major	C is m x n matrix. Size of array c must be at least ldc * m

ldc Leading dimension of matrix `C`. Must be positive.

Column major	Must be at least `m`
Row major	Must be at least `n`

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

Output Parameters

c Output buffer overwritten by `alpha` * op(`A`)*op(`B`) + `beta` * `C`.

Note: If `beta` = 0, matrix `C` does not need to be initialized before calling `gemm`.

Examples

An example of how to use buffer version of `gemm` can be found in oneMKL installation directory, under:

```
examples/dpcpp/blas/source/gemm.cpp
```

gemm (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemm(sycl::queue &queue,
                     oneapi::mkl::transpose transa,
                     oneapi::mkl::transpose transb,
                     std::int64_t m,
                     std::int64_t n,
                     std::int64_t k,
                     Ts alpha,
                     const Ta *a,
                     std::int64_t lda,
                     const Tb *b,
                     std::int64_t ldb,
                     Ts beta,
                     Tc *c,
                     std::int64_t ldc,
                     compute_mode mode = compute_mode::unset,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event gemm(sycl::queue &queue,
                     oneapi::mkl::transpose transa,
                     oneapi::mkl::transpose transb,
                     std::int64_t m,
                     std::int64_t n,
                     std::int64_t k,
                     Ts alpha,
                     const Ta *a,
                     std::int64_t lda,
                     const Tb *b,
                     std::int64_t ldb,
                     Ts beta,
                     Tc *c,
                     std::int64_t ldc,
                     compute_mode mode = compute_mode::unset,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

transa Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

transb Specifies op(B), the transposition operation applied to matrix B. See Data Types for more details.

m Number of rows of matrix op(A) and matrix C. Must be at least zero.

n Number of columns of matrix op(B) and matrix C. Must be at least zero.

k Number of columns of matrix op(A) and rows of matrix op(B). Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Pointer to input matrix A. See Matrix Storage for more details.

	A not transposed	A transposed
Column major	A is m x k matrix. Size of array a must be at least lda * k	A is k x m matrix. Size of array a must be at least lda * m
Row major	A is m x k matrix. Size of array a must be at least lda * m	A is k x m matrix. Size of array a must be at least lda * k

lda Leading dimension of matrix A. Must be positive.

	A not transposed	A transposed
Column major	Must be at least m	Must be at least k
Row major	Must be at least k	Must be at least m

b Pointer to input matrix B. See **Matrix Storage** for more details.

	B not transposed	B transposed
Column major	B is k x n matrix. Size of array b must be at least ldb * n	B is n x k matrix. Size of array b must be at least ldb * k
Row major	B is k x n matrix. Size of array b must be at least ldb * k	B is n x k matrix. Size of array b must be at least ldb * n

ldb Leading dimension of matrix B. Must be positive.

	B not transposed	B transposed
Column major	Must be at least k	Must be at least n
Row major	Must be at least n	Must be at least k

beta Scaling factor for matrix C.

c Pointer to input/output matrix C. See **Matrix Storage** for more details.

Column major	C is m x n matrix. Size of array c must be at least ldc * n
Row major	C is m x n matrix. Size of array c must be at least ldc * m

ldc Leading dimension of matrix C. Must be positive.

Column major	Must be at least m
Row major	Must be at least n

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

mode and dependencies may be omitted independently; it is not necessary to specify mode in order to provide dependencies.

Output Parameters

c Pointer to output matrix overwritten by alpha * op(A)*op(B) + beta * C.

Note: If beta = 0, matrix C does not need to be initialized before calling gemm.

Return Values

Output event to wait on to ensure computation is complete.

Examples

An example of how to use USM version of `gemm` can be found in oneMKL installation directory, under:

```
examples/dpcpp/blas/source/gemm_usm.cpp
```

9.3.2 hemm

Computes a matrix-matrix product where one input matrix is hermitian and one is general.

Description

The `hemm` routines compute a scalar-matrix-matrix product and add the result to a scalar-matrix product, where one of the matrices in the multiplication is hermitian. The argument `left_right` determines if the hermitian matrix, `A`, is on the left of the multiplication (`left_right` = `side::left`) or on the right (`left_right` = `side::right`). The operation is defined as:

If (`left_right` = `side::left`),

$$C \leftarrow alpha * A * B + beta * C$$

If (`left_right` = `side::right`),

$$C \leftarrow alpha * B * A + beta * C$$

where:

- `alpha` and `beta` are scalars
- `A` is either `m` x `m` or `n` x `n` hermitian matrix
- `B` and `C` are `m` x `n` matrices

`hemm` supports the following precisions:

T
std::complex<float>
std::complex<double>

hemm (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void hemm(sycl::queue &queue,
              oneapi::mkl::side left_right,
              oneapi::mkl::uplo upper_lower,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &b,
              std::int64_t ldb,
              T beta,
              sycl::buffer<T,1> &c,
              std::int64_t ldc,
              compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void hemm(sycl::queue &queue,
              oneapi::mkl::side left_right,
              oneapi::mkl::uplo upper_lower,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &b,
              std::int64_t ldb,
              T beta,
              sycl::buffer<T,1> &c,
              std::int64_t ldc,
              compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies whether matrix `A` is on the left side or right side of the multiplication. See Data Types for more details.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

m Number of rows of matrix `B` and matrix `C`. Must be at least zero.

n Number of columns of matrix `B` and matrix `C`. Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda * m` if `left_right = side::left` or `lda * n` if `left_right = side::right`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least `m` if `left_right` = `side::left` or at least `n` if `left_right` = `side::right`. Must be positive.

b Buffer holding input matrix `B`. Size of the buffer must be at least `ldb` * `n` if column major layout or at least `ldb` * `m` if row major layout is used. See **Matrix Storage** for more details.

ldb Leading dimension of matrix `B`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

beta Scaling factor for matrix `C`.

c Buffer holding input/output matrix `C`. Size of the buffer must be at least `ldc` * `n` if column major layout or at least `ldc` * `m` if row major layout is used. See **Matrix Storage** for more details.

ldc Leading dimension of matrix `C`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

Output Parameters

c Output buffer overwritten by `alpha` * `A` * `B` + `beta` * `C` if `left_right` = `side::left` or `alpha` * `B` * `A` + `beta` * `C` if `left_right` = `side::right`.

Note: If `beta` = 0, matrix `C` does not need to be initialized before calling `hemm`.

---

hemm (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event hemm(sycl::queue &queue,
                     oneapi::mkl::side left_right,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *b,
                     std::int64_t ldb,
                     T beta,
                     T *c,
                     std::int64_t ldc,
                     compute_mode mode = compute_mode::unset,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event hemm(sycl::queue &queue,
                     oneapi::mkl::side left_right,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *b,
                     std::int64_t ldb,
                     T beta,
                     T *c,
                     std::int64_t ldc,
                     compute_mode mode = compute_mode::unset,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies whether matrix `A` is on the left side or right side of the multiplication. See Data Types for more details.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

m Number of rows of matrix `B` and matrix `C`. Must be at least zero.

n Number of columns of matrix `B` and matrix `C`. Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Pointer to input matrix `A`. Size of the array must be at least `lda * m` if `left_right = side::left` or `lda * n` if `left_right = side::right`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least `m` if `left_right = side::left` or at least `n` if `left_right = side::right`. Must be positive.

b Pointer to input matrix `B`. Size of the array must be at least `ldb * n` if column major layout or at least `ldb * m` if row major layout is used. See Matrix Storage for more details.

ldb Leading dimension of matrix `B`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

beta Scaling factor for matrix `C`.

c Pointer to input/output matrix `C`. Size of the array must be at least `ldc * n` if column major layout or at least `ldc * m` if row major layout is used. See Matrix Storage for more details.

ldc Leading dimension of matrix `C`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Pointer to output matrix overwritten by `alpha` * `A` * `B` + `beta` * `C` if `left_right` = `side::left` or `alpha` * `B` * `A` + `beta` * `C` if `left_right` = `side::right`.

Note: If `beta` = 0, matrix `C` does not need to be initialized before calling `hemm`.

Return Values

Output event to wait on to ensure computation is complete.

9.3.3 her2k

Performs a hermitian rank-2k update.

Description

The `her2k` routines perform a rank-2k update of an `n` x `n` hermitian matrix `C` by general matrices `A` and `B`. The operation is defined as:

If `trans` = `transpose::nontrans`,

$$C \leftarrow alpha * A * B^H + conjg(alpha)B * A^H + beta * C$$

where:

- `A` is `n` x `k` and `B` is `k` x `n`.

If `trans` = `transpose::conjtrans`,

$$C \leftarrow alpha * A^H * B + conjg(alpha) * B^H * A + beta * C$$

where:

- `A` is `k` x `n` and `B` is `n` x `k`.

In both cases:

- `alpha` is a complex scalar and `beta` is a real scalar

- C is hermitian matrix, A and B are general matrices
- The inner dimension of both matrix multiplications is k

her2k supports the following precisions:

T	Treal
std::complex<float>	float
std::complex<double>	double

her2k (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void her2k(sycl::queue &queue,
               oneapi::mkl::uplo upper_lower,
               oneapi::mkl::transpose trans,
               std::int64_t n,
               std::int64_t k,
               T alpha,
               sycl::buffer<T,1> &a,
               std::int64_t lda,
               sycl::buffer<T,1> &b,
               std::int64_t ldb,
               Treal beta,
               sycl::buffer<T,1> &c,
               std::int64_t ldc,
               compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void her2k(sycl::queue &queue,
               oneapi::mkl::uplo upper_lower,
               oneapi::mkl::transpose trans,
               std::int64_t n,
               std::int64_t k,
               T alpha,
               sycl::buffer<T,1> &a,
               std::int64_t lda,
               sycl::buffer<T,1> &b,
               std::int64_t ldb,
               Treal beta,
               sycl::buffer<T,1> &c,
               std::int64_t ldc,
               compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `C` is upper or lower triangular. See Data Types for more details.

trans Specifies the transposition operation applied as described above. Supported operations are `transpose::nontrans` and `transpose::conjtrans`.

n Number of rows and columns of matrix `C`. Must be at least zero.

k Inner dimension of matrix multiplications. Must be at least zero.

alpha Complex scaling factor for the rank-2k update.

a Buffer holding input matrix `A`. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::conjtrans`
Column major	`A` is n x k matrix. Size of array a must be at least `lda * k`	`A` is k x n matrix. Size of array a must be at least `lda * n`
Row major	`A` is n x k matrix. Size of array a must be at least `lda * n`	`A` is k x n matrix. Size of array a must be at least `lda * k`

lda Leading dimension of matrix `A`. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::conjtrans`
Column major	Must be at least n	Must be at least k
Row major	Must be at least k	Must be at least n

beta Real scaling factor for matrix `C`.

b Buffer holding input matrix `B`. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::conjtrans`
Column major	`B` is k x n matrix. Size of array b must be at least `ldb * n`	`B` is n x k matrix. Size of array b must be at least `ldb * k`
Row major	`B` is k x n matrix. Size of array b must be at least `ldb * k`	`B` is n x k matrix. Size of array b must be at least `ldb * n`

ldb Leading dimension of matrix `B`. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::conjtrans`
Column major	Must be at least k	Must be at least n
Row major	Must be at least n	Must be at least k

c Buffer holding input/output matrix `C`. Size of the buffer must be at least `ldc` * `n`. See **Matrix Storage** for more details.

ldc Leading dimension of matrix `C`. Must be positive and at least `n`.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

Output Parameters

c Output buffer overwritten by updated `C` matrix.

her2k (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event her2k(sycl::queue &queue,
                      oneapi::mkl::uplo upper_lower,
                      oneapi::mkl::transpose trans,
                      std::int64_t n,
                      std::int64_t k,
                      T alpha,
                      const T *a,
                      std::int64_t lda,
                      const T *b,
                      std::int64_t ldb,
                      Treal beta,
                      T *c,
                      std::int64_t ldc,
                      compute_mode mode = compute_mode::unset,
                      const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event her2k(sycl::queue &queue,
                      oneapi::mkl::uplo upper_lower,
                      oneapi::mkl::transpose trans,
                      std::int64_t n,
                      std::int64_t k,
                      T alpha,
                      const T *a,
                      std::int64_t lda,
                      const T *b,
                      std::int64_t ldb,
                      Treal beta,
                      T *c,
                      std::int64_t ldc,
                      compute_mode mode = compute_mode::unset,
                      const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `C` is upper or lower triangular. See Data Types for more details.

trans Specifies the transposition operation applied as described above. Supported operations are `transpose::nontrans` and `transpose::conjtrans`.

n Number of rows and columns of matrix `C`. Must be at least zero.

k Inner dimension of matrix multiplications. Must be at least zero.

alpha Complex scaling factor for the rank-2k update.

a Pointer to input matrix `A`. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::conjtrans`
Column major	`A` is n x k matrix. Size of array a must be at least `lda` * k	`A` is k x n matrix. Size of array a must be at least `lda` * n
Row major	`A` is n x k matrix. Size of array a must be at least `lda` * n	`A` is k x n matrix. Size of array a must be at least `lda` * k

lda Leading dimension of matrix `A`. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::conjtrans`
Column major	Must be at least n	Must be at least k
Row major	Must be at least k	Must be at least n

beta Real scaling factor for matrix `C`.

b Pointer to input matrix `B`. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::conjtrans`
Column major	`B` is k x n matrix. Size of array b must be at least `ldb` * n	`B` is n x k matrix. Size of array b must be at least `ldb` * k
Row major	`B` is k x n matrix. Size of array b must be at least `ldb` * k	`B` is n x k matrix. Size of array b must be at least `ldb` * n

ldb Leading dimension of matrix `B`. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::conjtrans`
Column major	Must be at least k	Must be at least n
Row major	Must be at least n	Must be at least k

c Pointer to input/output matrix `C`. Size of the array must be at least `ldc * n`. See **Matrix Storage** for more details.

ldc Leading dimension of matrix `C`. Must be positive and at least `n`.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Pointer to output matrix overwritten by updated `C` matrix.

Return Values

Output event to wait on to ensure computation is complete.

9.3.4 herk

Performs a hermitian rank-k update.

Description

The `herk` routines compute a rank-k update of a hermitian matrix `C` by a general matrix `A`. The operation is defined as:

$$C \leftarrow alpha * op(A) * op(A)^H + beta * C$$

where:

- op(X) is one of op(X) = X or op(X) = X^H
- `alpha` and `beta` are real scalars
- `C` is n x n hermitian matrix
- op(`A`) is n x k general matrix

`herk` supports the following precisions:

T	Treal
`std::complex<float>`	`float`
`std::complex<double>`	`double`

herk (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void herk(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              std::int64_t n,
              std::int64_t k,
              Treal alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              Treal beta,
              sycl::buffer<T,1> &c,
              std::int64_t ldc,
              compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void herk(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              std::int64_t n,
              std::int64_t k,
              Treal alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              Treal beta,
              sycl::buffer<T,1> &c,
              std::int64_t ldc,
              compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `C` is upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), the transposition operation applied to matrix `A`. Supported operations are `transpose::nontrans` and `transpose::conjtrans`. See Data Types for more details.

n Number of rows and columns of matrix `C`. Must be at least zero.

k Number of columns of matrix op(`A`). Must be at least zero.

alpha Complex scaling factor for the rank-k update.

a Buffer holding input matrix A. See Matrix Storage for more details.

	trans = transpose::nontrans	trans = transpose::conjtrans
Column major	A is n x k matrix. Size of array a must be at least lda * k	A is k x n matrix. Size of array a must be at least lda * n
Row major	A is n x k matrix. Size of array a must be at least lda * n	A is k x n matrix. Size of array a must be at least lda * k

lda Leading dimension of matrix A. Must be positive.

	trans = transpose::nontrans	trans = transpose::conjtrans
Column major	Must be at least n	Must be at least k
Row major	Must be at least k	Must be at least n

beta Real scaling factor for matrix C.

c Buffer holding input/output matrix C. Size of the buffer must be at least ldc * n. See Matrix Storage for more details.

ldc Leading dimension of matrix C. Must be positive and at least n.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

Output Parameters

c Output buffer overwritten by alpha * op(A) * op(A)^H + beta * C. The imaginary parts of the diagonal elements are set to zero.

herk (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event herk(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     std::int64_t n,
                     std::int64_t k,
                     Treal alpha,
                     const T *a,
                     std::int64_t lda,
                     Treal beta,
                     T *c,
                     std::int64_t ldc,
                     compute_mode mode = compute_mode::unset,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event herk(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     std::int64_t n,
                     std::int64_t k,
                     Treal alpha,
                     const T *a,
                     std::int64_t lda,
                     Treal beta,
                     T *c,
                     std::int64_t ldc,
                     compute_mode mode = compute_mode::unset,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `C` is upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), the transposition operation applied to matrix `A`. Supported operations are `transpose::nontrans` and `transpose::conjtrans`. See Data Types for more details.

n Number of rows and columns of matrix `C`. Must be at least zero.

k Number of columns of matrix op(`A`). Must be at least zero.

alpha Complex scaling factor for the rank-k update.

a Pointer to input matrix `A`. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::conjtrans`
Column major	`A` is `n` x `k` matrix. Size of array `a` must be at least `lda * k`	`A` is `k` x `n` matrix. Size of array `a` must be at least `lda * n`
Row major	`A` is `n` x `k` matrix. Size of array `a` must be at least `lda * n`	`A` is `k` x `n` matrix. Size of array `a` must be at least `lda * k`

lda Leading dimension of matrix `A`. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::conjtrans`
Column major	Must be at least `n`	Must be at least `k`
Row major	Must be at least `k`	Must be at least `n`

beta Real scaling factor for matrix `C`.

c Pointer to input/output matrix `C`. Size of the array must be at least `ldc * n`. See Matrix Storage for more details.

ldc Leading dimension of matrix `C`. Must be positive and at least `n`.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Pointer to output matrix overwritten by `alpha` * op(`A`) * op(`A`)^H + `beta` * `C`. The imaginary parts of the diagonal elements are set to zero.

Return Values

Output event to wait on to ensure computation is complete.

9.3.5 symm

Computes a matrix-matrix product where one input matrix is symmetric and one matrix is general.

Description

The `symm` routines compute a scalar-matrix-matrix product and add the result to a scalar-matrix product, where one of the matrices in the multiplication is symmetric. The argument `left_right` determines if the symmetric matrix `A` is on the left of the multiplication (`left_right` = `side::left`) or on the right (`left_right` = `side::right`). The operation is defined as:

If (`left_right` = `side::left`),

$$C \leftarrow alpha * A * B + beta * C$$

If (`left_right` = `side::right`),

$$C \leftarrow alpha * B * A + beta * C$$

where:

- `alpha` and `beta` are scalars
- `A` is either `m` x `m` or `n` x `n` symmetric matrix
- `B` and `C` are `m` x `n` matrices

`symm` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

symm (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void symm(sycl::queue &queue,
              oneapi::mkl::side left_right,
              oneapi::mkl::uplo upper_lower,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &b,
              std::int64_t ldb,
              T beta,
              sycl::buffer<T,1> &c,
              std::int64_t ldc,
              compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void symm(sycl::queue &queue,
              oneapi::mkl::side left_right,
              oneapi::mkl::uplo upper_lower,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &b,
              std::int64_t ldb,
              T beta,
              sycl::buffer<T,1> &c,
              std::int64_t ldc,
              compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies whether matrix `A` is on the left side or right side of the multiplication. See Data Types for more details.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

m Number of rows of matrix `B` and matrix `C`. Must be at least zero.

n Number of columns of matrix `B` and matrix `C`. Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda * m` if `left_right = side::left` or `lda * n` if `left_right = side::right`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least `m` if `left_right = side::left` or at least `n` if `left_right = side::right`. Must be positive.

b Buffer holding input matrix `B`. Size of the buffer must be at least `ldb * n` if column major layout or at least `ldb * m` if row major layout is used. See Matrix Storage for more details.

ldb Leading dimension of matrix `B`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

beta Scaling factor for matrix `C`.

c Buffer holding input/output matrix `C`. Size of the buffer must be at least `ldc * n` if column major layout or at least `ldc * m` if row major layout is used. See Matrix Storage for more details.

ldc Leading dimension of matrix `C`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

Output Parameters

c Output buffer overwritten by `alpha * A * B + beta * C` if `left_right = side::left` or `alpha * B * A + beta * C` if `left_right = side::right`.

> **Note:** If `beta = 0`, matrix `C` does not need to be initialized before calling `symm`.

symm (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event symm(sycl::queue &queue,
                     oneapi::mkl::side left_right,
                     oneapi::mkl::uplo upper_lower,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     const T *b,
                     std::int64_t ldb,
                     T beta,
                     T *c,
                     std::int64_t ldc,
```

(continues on next page)

(continued from previous page)

```
                    compute_mode mode = compute_mode::unset,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event symm(sycl::queue &queue,
                    oneapi::mkl::side left_right,
                    oneapi::mkl::uplo upper_lower,
                    std::int64_t m,
                    std::int64_t n,
                    T alpha,
                    const T *a,
                    std::int64_t lda,
                    const T *b,
                    std::int64_t ldb,
                    T beta,
                    T *c,
                    std::int64_t ldc,
                    compute_mode mode = compute_mode::unset,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies whether matrix `A` is on the left side or right side of the multiplication. See Data Types for more details.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

m Number of rows of matrix `B` and matrix `C`. Must be at least zero.

n Number of columns of matrix `B` and matrix `C`. Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Pointer to input matrix `A`. Size of the array must be at least `lda * m` if `left_right = side::left` or `lda * n` if `left_right = side::right`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least `m` if `left_right = side::left` or at least `n` if `left_right = side::right`. Must be positive.

b Pointer to input matrix `B`. Size of the array must be at least `ldb * n` if column major layout or at least `ldb * m` if row major layout is used. See Matrix Storage for more details.

ldb Leading dimension of matrix `B`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

beta Scaling factor for matrix `C`.

c Pointer to input/output matrix `C`. Size of the array must be at least `ldc` * `n` if column major layout or at least `ldc` * `m` if row major layout is used. See Matrix Storage for more details.

ldc Leading dimension of matrix `C`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Pointer to output matrix overwritten by `alpha` * `A` * `B` + `beta` * `C` if `left_right` = `side::left` or `alpha` * `B` * `A` + `beta` * `C` if `left_right` = `side::right`.

Note: If `beta` = 0, matrix `C` does not need to be initialized before calling `symm`.

Return Values

Output event to wait on to ensure computation is complete.

9.3.6 syr2k

Performs a symmetric rank-2k update.

Description

The `syr2k` routines perform a rank-2k update of an `n` x `n` symmetric matrix `C` by general matrices `A` and `B`. The operation is defined as:

If `trans` = `transpose::nontrans`,

$$C \leftarrow alpha * (A * B^T + B * A^T) + beta * C$$

where `A` and `B` are `n` x `k` matrices.

If `trans` = `transpose::trans`,

$$C \leftarrow alpha * (A^T * B + B^T * A) + beta * C$$

where `A` and `B` are `k` x `n` matrices.

In both cases:

- `alpha` and `beta` are scalars
- `C` is symmetric matrix, `A` and `B` are general matrices
- The inner dimension of both matrix multiplications is `k`

`syr2k` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

syr2k (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void syr2k(sycl::queue &queue,
               oneapi::mkl::uplo upper_lower,
               oneapi::mkl::transpose trans,
               std::int64_t n,
               std::int64_t k,
               T alpha,
               sycl::buffer<T,1> &a,
               std::int64_t lda,
               sycl::buffer<T,1> &b,
               std::int64_t ldb,
               T beta,
               sycl::buffer<T,1> &c,
               std::int64_t ldc,
               compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void syr2k(sycl::queue &queue,
               oneapi::mkl::uplo upper_lower,
               oneapi::mkl::transpose trans,
               std::int64_t n,
               std::int64_t k,
               T alpha,
               sycl::buffer<T,1> &a,
               std::int64_t lda,
               sycl::buffer<T,1> &b,
               std::int64_t ldb,
               T beta,
               sycl::buffer<T,1> &c,
               std::int64_t ldc,
               compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `C` is upper or lower triangular. See Data Types for more details.

trans Specifies the transposition operation applied as described above. Conjugation is never performed even if `trans` = `transpose::conjtrans`.

n Number of rows and columns of matrix `C`. Must be at least zero.

k Inner dimension of matrix multiplications. Must be at least zero.

alpha Complex scaling factor for the rank-2k update.

a Buffer holding input matrix `A`. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	`A` is `n` x `k` matrix. Size of array `a` must be at least `lda` * `k`	`A` is `k` x `n` matrix. Size of array `a` must be at least `lda` * `n`
Row major	`A` is `n` x `k` matrix. Size of array `a` must be at least `lda` * `n`	`A` is `k` x `n` matrix. Size of array `a` must be at least `lda` * `k`

lda Leading dimension of matrix `A`. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `n`	Must be at least `k`
Row major	Must be at least `k`	Must be at least `n`

b Buffer holding input matrix `B`. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	`B` is `n` x `k` matrix. Size of array `b` must be at least `ldb` * `k`	`B` is `k` x `n` matrix. Size of array `b` must be at least `ldb` * `n`
Row major	`B` is `n` x `k` matrix. Size of array `b` must be at least `ldb` * `n`	`B` is `k` x `n` matrix. Size of array `b` must be at least `ldb` * `k`

ldb Leading dimension of matrix `B`. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `n`	Must be at least `k`
Row major	Must be at least `k`	Must be at least `n`

c Buffer holding input/output matrix `C`. Size of the buffer must be at least `ldc` * `n`. See **Matrix Storage** for more details.

ldc Leading dimension of matrix `C`. Must be positive and at least `n`.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

Output Parameters

c Output buffer overwritten by updated `C` matrix.

syr2k (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event syr2k(sycl::queue &queue,
                      oneapi::mkl::uplo upper_lower,
                      oneapi::mkl::transpose trans,
                      std::int64_t n,
                      std::int64_t k,
                      T alpha,
                      const T *a,
                      std::int64_t lda,
                      const T *b,
                      std::int64_t ldb,
                      T beta,
                      T *c,
                      std::int64_t ldc,
                      compute_mode mode = compute_mode::unset,
                      const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event syr2k(sycl::queue &queue,
                      oneapi::mkl::uplo upper_lower,
                      oneapi::mkl::transpose trans,
                      std::int64_t n,
                      std::int64_t k,
                      T alpha,
                      const T *a,
                      std::int64_t lda,
                      const T *b,
                      std::int64_t ldb,
                      T beta,
                      T *c,
                      std::int64_t ldc,
                      compute_mode mode = compute_mode::unset,
                      const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix C is upper or lower triangular. See Data Types for more details.

trans Specifies the transposition operation applied as described above. Conjugation is never performed even if `trans` = `transpose::conjtrans`.

n Number of rows and columns of matrix C. Must be at least zero.

k Inner dimension of matrix multiplications. Must be at least zero.

alpha Complex scaling factor for the rank-2k update.

a Pointer to input matrix A. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	A is n x k matrix. Size of array a must be at least `lda` * k	A is k x n matrix. Size of array a must be at least `lda` * n
Row major	A is n x k matrix. Size of array a must be at least `lda` * n	A is k x n matrix. Size of array a must be at least `lda` * k

lda Leading dimension of matrix A. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least n	Must be at least k
Row major	Must be at least k	Must be at least n

b Pointer to input matrix B. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	B is n x k matrix. Size of array b must be at least `ldb` * k	B is k x n matrix. Size of array b must be at least `ldb` * n
Row major	B is n x k matrix. Size of array b must be at least `ldb` * n	B is k x n matrix. Size of array b must be at least `ldb` * k

ldb Leading dimension of matrix B. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least n	Must be at least k
Row major	Must be at least k	Must be at least n

c Pointer to input/output matrix `C`. Size of the array must be at least `ldc * n`. See **Matrix Storage** for more details.

ldc Leading dimension of matrix `C`. Must be positive and at least `n`.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Pointer to output matrix overwritten by updated `C` matrix.

Return Values

Output event to wait on to ensure computation is complete.

9.3.7 syrk

Performs a symmetric rank-k update.

Description

The `syrk` routines perform a rank-k update of a symmetric matrix `C` by a general matrix `A`. The operation is defined as:

$$C \leftarrow alpha * op(A) * op(A)^T + beta * C$$

where:

- op(X) is one of op(X) = X or op(X) = X^T
- `alpha` and `beta` are scalars
- `C` is n x n symmetric matrix,
- op(`A`) is n x k general matrix

`syrk` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std::complex<double>`

syrk (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void syrk(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              std::int64_t n,
              std::int64_t k,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              T beta,
              sycl::buffer<T,1> &c,
              std::int64_t ldc,
              compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void syrk(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              std::int64_t n,
              std::int64_t k,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              T beta,
              sycl::buffer<T,1> &c,
              std::int64_t ldc,
              compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `C` is upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), the transposition operation applied to matrix `A`. Conjugation is never performed even if `trans` = `transpose::conjtrans`. See Data Types for more details.

n Number of rows and columns of matrix `C`. Must be at least zero.

k Number of columns of matrix op(`A`). Must be at least zero.

alpha Scaling factor for the rank-k update.

a Buffer holding input matrix A. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	A is n x k matrix. Size of array a must be at least `lda` * k	A is k x n matrix. Size of array a must be at least `lda` * n
Row major	A is n x k matrix. Size of array a must be at least `lda` * n	A is k x n matrix. Size of array a must be at least `lda` * k

lda Leading dimension of matrix A. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least n	Must be at least k
Row major	Must be at least k	Must be at least n

beta Scaling factor for matrix C.

c Buffer holding input/output matrix C. Size of the buffer must be at least `ldc` * n. See Matrix Storage for more details.

ldc Leading dimension of matrix C. Must be positive and at least n.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

Output Parameters

c Output buffer overwritten by `alpha` * op(A) * op(A)^T + `beta` * C.

syrk (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event syrk(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     std::int64_t n,
                     std::int64_t k,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     T beta,
                     T *c,
```

(continues on next page)

(continued from previous page)

```
                    std::int64_t ldc,
                    compute_mode mode = compute_mode::unset,
                    const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event syrk(sycl::queue &queue,
                    oneapi::mkl::uplo upper_lower,
                    oneapi::mkl::transpose trans,
                    std::int64_t n,
                    std::int64_t k,
                    T alpha,
                    const T *a,
                    std::int64_t lda,
                    T beta,
                    T *c,
                    std::int64_t ldc,
                    compute_mode mode = compute_mode::unset,
                    const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `C` is upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), the transposition operation applied to matrix `A`. Conjugation is never performed even if `trans` = `transpose::conjtrans`. See Data Types for more details.

n Number of rows and columns of matrix `C`. Must be at least zero.

k Number of columns of matrix op(`A`). Must be at least zero.

alpha Scaling factor for the rank-k update.

a Pointer to input matrix `A`. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	`A` is n x k matrix. Size of array a must be at least `lda` * k	`A` is k x n matrix. Size of array a must be at least `lda` * n
Row major	`A` is n x k matrix. Size of array a must be at least `lda` * n	`A` is k x n matrix. Size of array a must be at least `lda` * k

lda Leading dimension of matrix `A`. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `n`	Must be at least `k`
Row major	Must be at least `k`	Must be at least `n`

beta Scaling factor for matrix `C`.

c Pointer to input/output matrix `C`. Size of the array must be at least `ldc * n`. See **Matrix Storage** for more details.

ldc Leading dimension of matrix `C`. Must be positive and at least `n`.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Pointer to output matrix, overwritten by `alpha` * op(`A`) * op(`A`)^T + `beta` * `C`.

Return Values

Output event to wait on to ensure computation is complete.

9.3.8 trmm

Computes a matrix-matrix product where one input matrix is triangular and other matrix is general.

Description

The `trmm` routines compute a scalar-matrix-matrix product where one of the matrices in the multiplication is triangular. The argument `left_right` determines if the triangular matrix, `A`, is on the left of the multiplication (`left_right` = `side::left`) or on the right (`left_right` = `side::right`). The operation is defined as:

If (`left_right` = `side::left`),

$$B \leftarrow alpha * op(A) * B$$

If (`left_right` = `side::right`),

$$B \leftarrow alpha * B * op(A)$$

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- alpha is a scalar
- A is either m x m or n x n triangular matrix
- B is m x n general matrix

trmm supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

trmm (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void trmm(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &b,
              std::int64_t ldb,
              compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void trmm(sycl::queue &queue,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &b,
              std::int64_t ldb,
              compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies whether matrix `A` is on the left side or right side of the multiplication. See Data Types for more details.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), the transposition operation applied to matrix `A`. See Data Types for more details.

unit_diag Specifies whether matrix `A` is unit triangular or not. See Data Types for more details.

m Number of rows of matrix `B`. Must be at least zero.

n Number of columns of matrix `B`. Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda * m` if `left_right = side::left` or `lda * n` if `left_right = side::right`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least `m` if `left_right = side::left` or at least `n` if `left_right = side::right`. Must be positive.

b Buffer holding input matrix `B`. Size of the buffer must be at least `ldb * n` if column major layout or at least `ldb * m` if row major layout is used. See Matrix Storage for more details.

ldb Leading dimension of matrix `B`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

Output Parameters

b Output buffer overwritten by `alpha * op(A) * B` if `left_right = side::left` or `alpha * B * op(A)` if `left_right = side::right`.

Note: If `alpha = 0`, matrix `B` is set to zero, and `A` and `B` do not need to be initialized at entry.

---

trmm (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event trmm(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     T *b,
                     std::int64_t ldb,
                     compute_mode mode = compute_mode::unset,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event trmm(sycl::queue &queue,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     T *b,
                     std::int64_t ldb,
                     compute_mode mode = compute_mode::unset,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies whether matrix `A` is on the left side or right side of the multiplication. See Data Types for more details.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), the transposition operation applied to matrix `A`. See Data Types for more details.

unit_diag Specifies whether matrix `A` is unit triangular or not. See Data Types for more details.

m Number of rows of matrix `B`. Must be at least zero.

n Number of columns of matrix `B`. Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Pointer to input matrix `A`. Size of the array must be at least `lda * m` if `left_right = side::left` or `lda * n` if `left_right = side::right`. See Matrix Storage for more details.

lda Leading dimension of matrix A. Must be at least m if left_right = side::left or at least n if left_right = side::right. Must be positive.

b Pointer to input matrix B. Size of the array must be at least ldb * n if column major layout or at least ldb * m if row major layout is used. See Matrix Storage for more details.

ldb Leading dimension of matrix B. Must be at least m if column major layout or at least n if row major layout is used. Must be positive.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

mode and dependencies may be omitted independently; it is not necessary to specify mode in order to provide dependencies.

Output Parameters

b Pointer to output matrix overwritten by alpha * op(A) * B if left_right = side::left or alpha * B * op(A) if left_right = side::right.

Note: If alpha = 0, matrix B is set to zero, and A and B do not need to be initialized at entry.

Return Values

Output event to wait on to ensure computation is complete.

9.3.9 trsm

Solves a triangular matrix equation (forward or backward solve).

Description

The trsm routines solve triangular matrix equations where one of the matrices in the multiplication is triangular. The argument left_right determines if the triangular matrix, A, is on the left of the multiplication (left_right = side::left) or on the right (left_right = side::right). The operation is defined as:

If (left_right = side::left),

$$op(A) * X = alpha * B$$

If (left_right = side::right),

$$X * op(A) = alpha * B$$

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- alpha is a scalar
- A is either m x m or n x n triangular matrix
- B and X are m x n general matrices

On return, matrix B is overwritten by solution matrix X.

trsm supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

trsm (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void trsm(sycl::queue &queue,
              oneapi::mkl::side left_right,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose transa,
              oneapi::mkl::diag unit_diag,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &b,
              std::int64_t ldb,
              compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void trsm(sycl::queue &queue,
              oneapi::mkl::side left_right,
              oneapi::mkl::uplo upper_lower,
              oneapi::mkl::transpose trans,
              oneapi::mkl::diag unit_diag,
              std::int64_t m,
              std::int64_t n,
              T alpha,
              sycl::buffer<T,1> &a,
              std::int64_t lda,
              sycl::buffer<T,1> &b,
```

(continues on next page)

(continued from previous page)

```
                std::int64_t ldb,
                compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies whether matrix `A` is on the left side or right side of the multiplication. See Data Types for more details.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), the transposition operation applied to matrix `A`. See Data Types for more details.

unit_diag Specifies whether matrix `A` is unit triangular or not. See Data Types for more details.

m Number of rows of matrix `B`. Must be at least zero.

n Number of columns of matrix `B`. Must be at least zero.

alpha Scaling factor for the solution.

a Buffer holding input matrix `A`. Size of the buffer must be at least `lda` * `m` if `left_right` = `side::left` or `lda` * `n` if `left_right` = `side::right`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least `m` if `left_right` = `side::left` or at least `n` if `left_right` = `side::right`. Must be positive.

b Buffer holding input matrix `B`. Size of the buffer must be at least `ldb` * `n` if column major layout or at least `ldb` * `m` if row major layout is used. See Matrix Storage for more details.

ldb Leading dimension of matrix `B`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

Output Parameters

b Output buffer overwritten by solution matrix `X`.

Note: If `alpha` = 0, matrix `B` is set to zero, and `A` and `B` do not need to be initialized before calling `trsm`.

---

trsm (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event trsm(sycl::queue &queue,
                     oneapi::mkl::side left_right,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     T *b,
                     std::int64_t ldb,
                     compute_mode mode = compute_mode::unset,
                     const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event trsm(sycl::queue &queue,
                     oneapi::mkl::side left_right,
                     oneapi::mkl::uplo upper_lower,
                     oneapi::mkl::transpose trans,
                     oneapi::mkl::diag unit_diag,
                     std::int64_t m,
                     std::int64_t n,
                     T alpha,
                     const T *a,
                     std::int64_t lda,
                     T *b,
                     std::int64_t ldb,
                     compute_mode mode = compute_mode::unset,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies whether matrix `A` is on the left side or right side of the multiplication. See Data Types for more details.

upper_lower Specifies whether matrix `A` is upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), the transposition operation applied to matrix `A`. See Data Types for more details.

unit_diag Specifies whether matrix `A` is unit triangular or not. See Data Types for more details.

m Number of rows of matrix `B`. Must be at least zero.

n Number of columns of matrix `B`. Must be at least zero.

alpha Scaling factor for the solution.

a Pointer to input matrix `A`. Size of the array must be at least `lda * m` if `left_right = side::left` or `lda * n` if `left_right = side::right`. See Matrix Storage for more details.

lda Leading dimension of matrix `A`. Must be at least `m` if `left_right = side::left` or at least `n` if `left_right = side::right`. Must be positive.

b Pointer to input matrix `B`. Size of the array must be at least `ldb * n` if column major layout or at least `ldb * m` if row major layout is used. See Matrix Storage for more details.

ldb Leading dimension of matrix `B`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

b Pointer to output matrix overwritten by solution matrix `X`.

Note: If `alpha = 0`, matrix `B` is set to zero, and `A` and `B` do not need to be initialized before calling `trsm`.

Return Values

Output event to wait on to ensure computation is complete.

9.4 BLAS-like Extensions

Intel® oneAPI Math Kernel Library (oneMKL) DPC++ provides additional routines to extend the functionality of the BLAS routines. These include routines to compute many independent vector-vector, vector-matrix, matrix-matrix operations.

The following table lists these routines.

Routine	Data Types	Description
axpby	float, double, std::complex<float>, std::complex<double>	Computes a vector-scalar product added to a scaled-vector.
axpy_batch	float, double, std::complex<float>, std::complex<double>	Computes groups of vector-scalar product added to a vector.
copy_batch	float, double, std::complex<float>, std::complex<double>	Computes groups of vector copies.
dgmm_batch	float, double, std::complex<float>, std::complex<double>	Computes a group of diagonal matrix-matrix product.
gemm_batch	std::int8_t, oneapi::mkl::bfloat16, sycl::half, float, double, std::complex<float>, std::complex<double>, mixed	Computes groups of matrix-matrix product with general matrices.
gemm_bias	mixed std::int8_t, std::uint8_t, and std::int32_t	Computes a matrix-matrix product with general matrices and mixed precision.
gemmt	float, double, std::complex<float>, std::complex<double>	Computes a matrix-matrix product with general matrices, but updates only the upper/lower triangular part of the output matrix.
gemv_batch	float, double, std::complex<float>, std::complex<double>	Computes a group of matrix-vector product using general matrices.
syrk_batch	float, double, std::complex<float>, std::complex<double>	Computes rank-k updates on a group of symmetric matrices by a group of general matrices.
trsm_batch	float, double, std::complex<float>, std::complex<double>	Solves a triangular matrix equation for a group of matrices.
omatcopy	float, double, std::complex<float>, std::complex<double>	Computes an out-of-place matrix copy or transposition.
imatcopy	float, double, std::complex<float>, std::complex<double>	Computes an in-place matrix copy or transposition.
omatadd	float, double, std::complex<float>, std::complex<double>	Computes a sum of two general matrices, with optional transposes.
omatcopy_batch	float, double, std::complex<float>, std::complex<double>	Computes groups of out-of-place matrix copies or transpositions.
imatcopy_batch	float, double, std::complex<float>, std::complex<double>	Computes groups of in-place matrix copies or transpositions.
omatadd_batch	float, double, std::complex<float>, std::complex<double>	Computes groups of matrix additions.

9.4.1 axpby

Computes a vector-scalar product added to a scaled-vector.

Description

The axpby routines compute two scalar-vector products and add them:

$$y \leftarrow beta * y + alpha * x$$

where:

- x and y are vectors of n elements
- `alpha` and `beta` are scalars

axpby supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

axpby (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
  void axpby(sycl::queue &queue,
             std::int64_t n,
             T alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             T beta,
             sycl::buffer<T,1> &y,
             std::int64_t incy);
  }
```

```
namespace oneapi::mkl::blas::row_major {
  void axpby(sycl::queue &queue,
             std::int64_t n,
             T alpha,
             sycl::buffer<T,1> &x,
             std::int64_t incx,
             T beta,
             sycl::buffer<T,1> &y,
             std::int64_t incy);
  }
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors x and y.

alpha Specifies the scalar `alpha`.

x Buffer holding input vector x. Size of the buffer must be at least 1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride between two consecutive elements of vector x.

beta Specifies the scalar `beta`.

y Buffer holding input vector y. Size of the buffer must be at least 1 + (n - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride between two consecutive elements of vector y.

Output Parameters

y Buffer holding updated vector y.

axpby (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
  sycl::event axpby(sycl::queue &queue,
                    std::int64_t n,
                    T alpha,
                    const T *x,
                    std::int64_t incx,
                    T beta,
                    T *y,
                    std::int64_t incy,
                    const std::vector<event> &dependencies = {});
  }
```

```
namespace oneapi::mkl::blas::row_major {
  sycl::event axpby(sycl::queue &queue,
                    std::int64_t n,
                    T alpha,
                    const T *x,
                    std::int64_t incx,
                    T beta,
                    T *y,
                    std::int64_t incy,
                    const std::vector<event> &dependencies = {});
  }
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors x and y.

alpha Specifies the scalar `alpha`.

x Pointer to the input vector x. Size of the array must be at least 1 + (n - 1)*abs(`incx`)). See Matrix Storage for more details.

incx Stride between two consecutive elements of vector x.

beta Specifies the scalar `beta`.

y Pointer to the input vector y. Size of the array must be at least 1 + (n - 1)*abs(`incy`)). See Matrix Storage for more details.

incy Stride between two consecutive elements of vector y.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Array holding updated vector y.

Return Values

Output event to wait on to ensure computation is complete.

9.4.2 axpy_batch

Computes a group of `axpy` operations.

Description

The `axpy_batch` routines are batched versions of axpy, performing multiple `axpy` operations in a single call. Each `axpy` operation adds a scalar-vector product to a vector.

`axpy_batch` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std::complex<double>`

axpy_batch (Buffer Version)

Buffer version of `axpy_batch` supports only strided API.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    X and Y are vectors at offset i * stridex and i * stridey in x and y
    Y = alpha * X + Y
end for
```

where:

- `alpha` is scalar
- X and Y are vectors

For strided API, all vectors X and Y have same parameters (size, increments) and are stored at constant stride given by `stridex` and `stridey` from each other. The x and y arrays contain all the input vectors. Total number of vectors in x and y are given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void axpy_batch(sycl::queue &queue,
                    std::int64_t n,
                    T alpha,
                    sycl::buffer<T, 1> &x,
                    std::int64_t incx,
                    std::int64_t stridex,
                    sycl::buffer<T, 1> &y,
                    std::int64_t incy,
                    std::int64_t stridey,
                    std::int64_t batch_size)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void axpy_batch(sycl::queue &queue,
                    std::int64_t n,
                    T alpha,
                    sycl::buffer<T, 1> &x,
                    std::int64_t incx,
                    std::int64_t stridex,
                    sycl::buffer<T, 1> &y,
                    std::int64_t incy,
                    std::int64_t stridey,
                    std::int64_t batch_size)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors X and Y.

alpha Specifies the scalar `alpha`.

x Buffer holding input vectors X. Size of the buffer must be at least `batch_size` * `stridex`.

incx Stride between two consecutive elements of X vectors.

stridex Stride between two consecutive X vectors. Must be at least (1 + (n-1)*abs(`incx`)). See Matrix Storage for more details.

y Buffer holding input/output vectors Y. Size of the buffer must be at least `batch_size` * `stridey`.

incy Stride between two consecutive elements of Y vectors.

stridey Stride between two consecutive Y vectors. Must be at least (1 + (n-1)*abs(`incy`)). See Matrix Storage for more details.

batch_size Number of `axpy` computations to perform. Must be at least zero.

Output Parameters

y Output buffer overwritten by `batch_size` `axpy` operations of the form `alpha` * X + Y.

axpy_batch (USM Version)

USM version of `axpy_batch` supports group API and strided API.

Group API

Group API operation is defined as:

```
idx = 0
for i = 0 … group_count – 1
     for j = 0 … group_size – 1
         X and Y are vectors at x[idx] and y[idx]
         Y = alpha[i] * X + Y
         idx = idx + 1
     end for
end for
```

where:

- `alpha` is scalar
- X and Y are vectors

For group API, each group contains vectors with the same parameters (size and increment). The x and y arrays contain the pointers for all the input vectors. Total number of vectors in x and y are given by:

$$total\_batch\_count = \sum_{i=0}^{group\_count-1} group\_size[i]$$

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event axpy_batch(sycl::queue &queue,
                           const std::int64_t *n,
                           const T *alpha,
                           const T **x,
                           const std::int64_t *incx,
                           T **y,
                           const std::int64_t *incy,
                           std::int64_t group_count,
                           const std::int64_t *group_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event axpy_batch(sycl::queue &queue,
                           const std::int64_t *n,
                           const T *alpha,
                           const T **x,
                           const std::int64_t *incx,
                           T **y,
                           const std::int64_t *incy,
                           std::int64_t group_count,
                           const std::int64_t *group_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Array of `group_count` integers. `n[i]` specifies number of elements in vectors X and Y for every vector in group `i`.

alpha Array of `group_count` scalar elements. `alpha[i]` specifies scaling factor for vector X in group `i`.

x Array of pointers to input vectors X with size `total_batch_count`. Size of the array allocated for the X vector of the group `i` must be at least (1 + (`n[i]` – 1)*abs(`incx[i]`)). See **Matrix Storage** for more details.

incx Array of `group_count` integers. `incx[i]` specifies stride of vector X in group `i`.

y Array of pointers to input/output vectors Y with size `total_batch_count`. Size of the array allocated for the Y vector of the group `i` must be at least (1 + (`n[i]` – 1)*abs(`incy[i]`)). See **Matrix Storage** for more details.

incy Array of `group_count` integers. `incy[i]` specifies the stride of vector Y in group `i`.

group_count Number of groups. Must be at least zero.

group_size Array of `group_count` integers. `group_size[i]` specifies the number of axpy operations in group `i`. Each element in `group_size` must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Array of pointers holding Y vectors, overwritten by `total_batch_count` axpy operations of the form `alpha` * X + Y.

Return Values

Output event to wait on to ensure computation is complete.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    X and Y are vectors at offset i * stridex and i * stridey in x and y
    Y = alpha * X + Y
end for
```

where:

- `alpha` is scalar
- X and Y are vectors

For strided API, all vectors X and Y have same parameters (size, increments) and are stored at constant stride given by `stridex` and `stridey` from each other. The x and y arrays contain all the input vectors. Total number of vectors in x and y are given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event axpy_batch(sycl::queue &queue,
                           std::int64_t n,
                           T alpha,
                           const T *x,
                           std::int64_t incx,
                           std::int64_t stridex,
                           T *y,
                           std::int64_t incy,
                           std::int64_t stridey,
                           std::int64_t batch_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event axpy_batch(sycl::queue &queue,
                           std::int64_t n,
                           T alpha,
                           const T *x,
                           std::int64_t incx,
                           std::int64_t stridex,
                           T *y,
                           std::int64_t incy,
                           std::int64_t stridey,
                           std::int64_t batch_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors X and Y.

alpha Specifies the scalar `alpha`.

x Pointer to input vectors X. Size of the array must be at least `batch_size` * `stridex`.

incx Stride between two consecutive elements of X vectors.

stridex Stride between two consecutive X vectors. Must be at least (1 + (n-1)*abs(`incx`)). See Matrix Storage for more details.

y Pointer to input/output vectors Y. Size of the array must be at least `batch_size` * `stridey`.

incy Stride between two consecutive elements of Y vectors.

stridey Stride between two consecutive Y vectors. Must be at least (1 + (n-1)*abs(`incy`)). See Matrix Storage for more details.

batch_size Number of `axpy` computations to perform. Must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to output vectors Y overwritten by `batch_size` `axpy` operations of the form `alpha` * X + Y.

Return Values

Output event to wait on to ensure computation is complete.

9.4.3 copy_batch

Computes a group of `copy` operations.

Description

The `copy_batch` routines are batched versions of **copy**, performing multiple `copy` operations in a single call. Each `copy` operation copies one vector to another.

`copy_batch` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

copy_batch (Buffer Version)

Buffer version of `copy_batch` supports only strided API.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    X and Y are vectors at offset i * stridex and i * stridey in x and y
    Y = X
end for
```

where:

- X and Y are vectors

For strided API, all vectors x (y) have same parameters (size, increments) and are stored at constant `stridex` (`stridey`) from each other. The x and y arrays contain all the input vectors. Total number of vectors in x and y are given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void copy_batch(sycl::queue &queue,
                    std::int64_t n,
                    sycl::buffer<T, 1> &x,
                    std::int64_t incx,
                    std::int64_t stridex,
                    sycl::buffer<T, 1> &y,
                    std::int64_t incy,
                    std::int64_t stridey,
                    std::int64_t batch_size)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void copy_batch(sycl::queue &queue,
                    std::int64_t n,
                    sycl::buffer<T, 1> &x,
                    std::int64_t incx,
                    std::int64_t stridex,
                    sycl::buffer<T, 1> &y,
                    std::int64_t incy,
                    std::int64_t stridey,
                    std::int64_t batch_size)
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors X and Y.

x Buffer holding input vectors X. Size of the buffer must be at least `batch_size` * `stridex`.

incx Stride between two consecutive elements of X vectors.

stridex Stride between two consecutive X vectors. Must be at least (1 + (n-1)*abs(`incx`)). See Matrix Storage for more details.

y Buffer holding input/output vectors Y. Size of the buffer must be at least `batch_size` * `stridey`.

incy Stride between two consecutive elements of Y vectors.

stridey Stride between two consecutive Y vectors. Must be at least (1 + (n-1)*abs(`incy`)). See Matrix Storage for more details.

batch_size Number of `copy` computations to perform. Must be at least zero.

Output Parameters

y Output buffer overwritten by `batch_size` `copy` operations.

copy_batch (USM Version)

USM version of `copy_batch` supports group API and strided API.

Group API

Group API operation is defined as:

```
idx = 0
for i = 0 … group_count – 1
     for j = 0 … group_size – 1
         X and Y are vectors at x[idx] and y[idx]
         Y = X
         idx = idx + 1
     end for
end for
```

where:

- X and Y are vectors

For group API, each group contains vectors with the same parameters (size and increment). The x and y arrays contain the pointers for all the input vectors. Total number of vectors in x and y are given by:

$$total\_batch\_count = \sum_{i=0}^{group\_count-1} group\_size[i]$$

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event copy_batch(sycl::queue &queue,
                           const std::int64_t *n,
                           const T **x,
                           const std::int64_t *incx,
                           T **y,
                           const std::int64_t *incy,
                           std::int64_t group_count,
                           const std::int64_t *group_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event copy_batch(sycl::queue &queue,
                           const std::int64_t *n,
                           const T **x,
                           const std::int64_t *incx,
                           T **y,
                           const std::int64_t *incy,
                           std::int64_t group_count,
                           const std::int64_t *group_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Array of `group_count` integers. `n[i]` specifies number of elements in vectors X and Y for every vector in group `i`.

x Array of pointers to input vectors X with size `total_batch_count`. Size of the array allocated for the X vector of the group `i` must be at least (1 + (`n[i]` – 1)*abs(`incx[i]`)). See Matrix Storage for more details.

incx Array of `group_count` integers. `incx[i]` specifies stride of vector X in group `i`.

y Array of pointers to input/output vectors Y with size `total_batch_count`. Size of the array allocated for the Y vector of the group `i` must be at least (1 + (`n[i]` – 1)*abs(`incy[i]`)). See Matrix Storage for more details.

incy Array of `group_count` integers. `incy[i]` specifies the stride of vector Y in group `i`.

group_count Number of groups. Must be at least zero.

group_size Array of `group_count` integers. `group_size[i]` specifies the number of `copy` operations in group `i`. Each element in `group_size` must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Array of pointers holding Y vectors, overwritten by `total_batch_count` `copy` operations.

Return Values

Output event to wait on to ensure computation is complete.

Examples

An example of how to use USM version of `copy_batch` can be found in oneMKL installation directory, under:

```
examples/dpcpp/blas/source/copy_batch_usm.cpp
```

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    X and Y are vectors at offset i * stridex and i * stridey in x and y
    Y = X
end for
```

where:

- X and Y are vectors

For strided API, all vectors x (y) have same parameters (size, increments) and are stored at constant `stridex` (`stridey`) from each other. The x and y arrays contain all the input vectors. Total number of vectors in x and y are given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event copy_batch(sycl::queue &queue,
                           std::int64_t n,
                           const T *x,
                           std::int64_t incx,
                           std::int64_t stridex,
                           T *y,
                           std::int64_t incy,
                           std::int64_t stridey,
                           std::int64_t batch_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event copy_batch(sycl::queue &queue,
                           std::int64_t n,
                           const T *x,
                           std::int64_t incx,
                           std::int64_t stridex,
                           T *y,
                           std::int64_t incy,
                           std::int64_t stridey,
                           std::int64_t batch_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

n Number of elements in vectors X and Y.

x Pointer to input vectors X. Size of the array must be at least `batch_size` * `stridex`.

incx Stride between two consecutive elements of X vectors.

stridex Stride between two consecutive X vectors. Must be at least (1 + (n-1)*abs(`incx`)). See Matrix Storage for more details.

y Pointer to input/output vectors Y. Size of the array must be at least `batch_size` * `stridey`.

incy Stride between two consecutive elements of Y vectors.

stridey Stride between two consecutive Y vectors. Must be at least (1 + (n-1)*abs(`incy`)). See Matrix Storage for more details.

batch_size Number of `copy` computations to perform. Must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Output vectors overwritten by `batch_size` `copy` operations.

Return Values

Output event to wait on to ensure computation is complete.

9.4.4 dgmm_batch

Computes a group of (diagonal matrix-matrix product (`dgmm`) operations.

Description

The `dgmm_batch` routines perform multiple diagonal matrix-matrix product (`dgmm`) operations in a single call. The diagonal matrices are stored as dense vectors and the operations are performed with groups of matrices and vectors.

`dgmm_batch` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std::complex<double>`

dgmm_batch (Buffer Version)

Buffer version of `dgmm_batch` supports only strided API.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    A and C are matrices at offset i * stridea in a, i * stridec in c.
    X is a vector at offset i * stridex in x
    if (left_right == side::left)
        C = diag(X) * A
    else
        C = A * diag(X)
end for
```

where:

- `A` is a matrix
- `X` is a diagonal matrix stored as a vector

For strided API, all matrices `A` and `C` and vector `X` have the same parameters (size, increments) and are stored at a constant stride given by `stridea`, `stridec` and `stridex` from each other.

The a and x buffers contain all the input matrices. Total number of matrices in a and x are given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void dgmm_batch(sycl::queue &queue,
                    oneapi::mkl::side left_right,
                    std::inte64_t m,
                    std::int64_t n,
                    sycl::buffer<T,1> &a,
                    std::int64_t lda,
                    std::int64_t stridea,
                    sycl::buffer<T,1> &x,
                    std::int64_t incx,
                    std::int64_t stridex,
                    sycl::buffer<T,1> &c,
                    std::int64_t ldc,
                    std::int64_t stridec,
                    std::int64_t batch_size);
}
```

```
namespace oneapi::mkl::blas::row_major {
    void dgmm_batch(sycl::queue &queue,
                    oneapi::mkl::side left_right,
                    std::inte64_t m,
                    std::int64_t n,
                    sycl::buffer<T,1> &a,
                    std::int64_t lda,
                    std::int64_t stridea,
                    sycl::buffer<T,1> &x,
                    std::int64_t incx,
                    std::int64_t stridex,
                    sycl::buffer<T,1> &c,
                    std::int64_t ldc,
                    std::int64_t stridec,
                    std::int64_t batch_size);
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies the position of the diagonal matrix in the product. See Data Types for more details.

m Number of rows of matrix `A` and matrix `C`. Must be at least zero.

n Number of columns of matrix `A` and matrix `C`. Must be at least zero.

a Buffer holding input matrices `A`. Size of the buffer must be at least `lda` * k + `stridea` * (`batch_size` - 1) where k is `n` if column major layout or `m` if row major layout is used.

lda Leading dimension of matrices `A`. Must be at least `m` if column major layout or `n` if row major layout is used. Must be positive.

stridea Stride between two consecutive `A` matrices. Must be at least zero. See Matrix Storage for more details.

x Buffer holding input matrices `X`. Size of the buffer must be at least (1 + (`len` - 1)*abs(`incx`)) + `stridex` * (`batch_size` - 1) where `len` is `n` if the diagonal matrix is on the right of the product or `m` otherwise.

incx Stride between two consecutive elements of the `X` vectors.

stridex Stride between two consecutive `X` vectors. Must be at least zero. See Matrix Storage for more details.

c Buffer holding input/output matrices `C`. Size of the buffer must be at least `batch_size` * `stridec`.

ldc Leading dimension of matrices `C`. Must be at least `m` if column major layout or `n` if row major layout is used. Must be positive.

stridec Stride between two consecutive `C` matrices. Must be at least `ldc` * `n` if column major layout or `ldc` * `m` if row major layout is used. See Matrix Storage for more details.

batch_size Number of `dgmm` computations to perform. Must be at least zero.

Output Parameters

c Buffer holding output matrices `C` overwritten by `batch_size` `dgmm` operations.

dgmm_batch (USM Version)

USM version of `dgmm_batch` supports group API and strided API.

Group API

Group API operation is defined as:

```
idx = 0
for i = 0 … group_count – 1
     for j = 0 … group_size – 1
         A and C are matrices at a[idx] and c[idx]
         X is a vector at x[idx]
         if (left_right[idx] == side::left)
             C = diag(X) * A
         else
             C = A * diag(X)
         idx = idx + 1
     end for
end for
```

where:

- `A` is a matrix
- `X` is a diagonal matrix stored as a vector

For group API, each group contain matrices and vectors with the same parameters (size, increment). The a and x arrays contain the pointers for all the input matrices. Total number of matrices in a and x are given by:

$$total\_batch\_count = \sum_{i=0}^{group\_count-1} group\_size[i]$$

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event dgmm_batch(sycl::queue &queue,
                           const oneapi::mkl::side *left_right,
                           const std::int64_t *m,
                           const std::int64_t *n,
                           const T **a,
                           const std::int64_t *lda,
                           const T **x,
```

(continues on next page)

(continued from previous page)

```
                            const std::int64_t *incx,
                            T **c,
                            const std::int64_t *ldc,
                            std::int64_t group_count,
                            const std::int64_t *group_size,
                            const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event dgmm_batch(sycl::queue &queue,
                            const oneapi::mkl::side *left_right,
                            const std::int64_t *m,
                            const std::int64_t *n,
                            const T **a,
                            const std::int64_t *lda,
                            const T **x,
                            const std::int64_t *incx,
                            T **c,
                            const std::int64_t *ldc,
                            std::int64_t group_count,
                            const std::int64_t *group_size,
                            const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Array of `group_count` parameters. `left_right[i]` specifies the position of the diagonal matrix in group `i`. See Data Types for more details.

m Array of `group_count` integers. `m[i]` specifies number of rows of `A` for every matrix in group `i`. All entries must be at least zero.

n Array of `group_count` integers. `n[i]` specifies number of columns of `A` for every matrix in group `i`. All entries must be at least zero.

a Array of pointers to input matrices `A` with size `total_batch_count`. Size of the array must be at least `lda[i]` * `n[i]` if column major layout or at least `lda[i]` * `m[i]` if row major layout is used. See Matrix Storage for more details.

lda Array of `group_count` integers. `lda[i]` specifies the leading dimension of `A` for every matrix in group `i`. All entries must be positive and at least `m[i]` if column major layout or at least `n[i]` if row major layout is used.

x Array of pointers to input vectors `X` with size `total_batch_count`. Size of the array must be at least (1+`len[i]` – 1)*abs(`incx[i]`)) where `len[i]` is `n[i]` if diagonal matrix is on the right of the product or `m[i]` otherwise. See Matrix Storage for more details.

incx Array of `group_count` integers. `incx[i]` specifies the stride of `X` for every vector in group `i`. All entries must be positive.

c Array of pointers to input/output matrices `C` with size `total_batch_count`. Size of the array must be least `ldc[i]` * `n[i]` if column major layout or at least `ldc[i]` * `m[i]` if row major layout is used. See **Matrix Storage** for more details.

ldc Array of `group_count` integers. `ldc[i]` specifies the leading dimension of `C` for every matrix in group `i`. All entries must be positive and at least `m[i]` if column major layout or at least `n[i]` if row major layout is used.

group_count Specifies number of groups. Must be at least zero.

group_size Array of `group_count` integers. `group_size[i]` specifies the number of diagonal matrix-matrix product operations in group `i`. All entries must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

c Array of pointers to output matrices `C` overwritten by `total_batch_count` `dgmm` operations.

Return Values

Output event to wait on to ensure computation is complete.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    A and C are matrices at offset i * stridea in a, i * stridec in c.
    X is a vector at offset i * stridex in x
    if (left_right == side::left)
        C = diag(X) * A
    else
        C = A * diag(X)
end for
```

where:

- `A` is a matrix
- `X` is a diagonal matrix stored as a vector

For strided API, all matrices `A` and `C` and vector `X` have the same parameters (size, increments) and are stored at a constant stride given by `stridea`, `stridec` and `stridex` from each other.

The a and x buffers contain all the input matrices. Total number of matrices in a and x are given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event dgmm_batch(sycl::queue &queue,
                           oneapi::mkl::side left_right,
                           std::inte64_t m,
                           std::int64_t n,
                           const T *a,
                           std::int64_t lda,
                           std::int64_t stridea,
                           const T *x,
                           std::int64_t incx,
                           std::int64_t stridex,
                           T *c,
                           std::int64_t ldc,
                           std::int64_t stridec,
                           std::int64_t batch_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event dgmm_batch(sycl::queue &queue,
                           oneapi::mkl::side left_right,
                           std::inte64_t m,
                           std::int64_t n,
                           const T *a,
                           std::int64_t lda,
                           std::int64_t stridea,
                           const T *x,
                           std::int64_t incx,
                           std::int64_t stridex,
                           T *c,
                           std::int64_t ldc,
                           std::int64_t stridec,
                           std::int64_t batch_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies the position of the diagonal matrix in the product. See Data Types for more details.

m Number of rows of matrix `A` and matrix `C`. Must be at least zero.

n Number of columns of matrix `A` and matrix `C`. Must be at least zero.

a Pointer to input matrices `A`. Size of the array must be at least `lda` * k + `stridea` * (`batch_size` - 1) where k is `n` if column major layout or `m` if row major layout is used.

lda Leading dimension of matrices A. Must be at least m if column major layout or n if row major layout is used. Must be positive.

stridea Stride between two consecutive A matrices. Must be at least zero. See Matrix Storage for more details.

x Pointer to input matrices X. Size of the array must be at least (1 + (len - 1)*abs(incx)) + stridex * (batch_size - 1) where len is n if the diagonal matrix is on the right of the product or m otherwise.

incx Stride between two consecutive elements of the X vectors.

stridex Stride between two consecutive X vectors. Must be at least zero. See Matrix Storage for more details.

c Pointer to input/output matrices C. Size of the array must be at least batch_size * stridec.

ldc Leading dimension of matrices C. Must be at least m if column major layout or n if row major layout is used. Must be positive.

stridec Stride between two consecutive C matrices. Must be at least ldc * n if column major layout or ldc * m if row major layout is used. See Matrix Storage for more details.

batch_size Number of dgmm computations to perform. Must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

c Pointer to output matrices C overwritten by batch_size dgmm operations.

Return Values

Output event to wait on to ensure computation is complete.

9.4.5 gemm_batch

Computes groups of matrix-matrix product with general matrices.

Description

The gemm_batch routines are batched versions of gemm, performing multiple gemm operations in a single call. Each gemm operation performs a matrix-matrix product with general matrices.

gemm_batch supports the following precisions:

Ta (A matrices)	Tb (B matrices)	Tc (C matrices)	Ts (alpha/beta)
sycl::half	sycl::half	sycl::half	sycl::half
sycl::half	sycl::half	float	float
oneapi::mkl::bfloat16	oneapi::mkl::bfloat16	oneapi::mkl::bfloat16	float
oneapi::mkl::bfloat16	oneapi::mkl::bfloat16	float	float
std::int8_t	std::int8_t	std::int32_t	float
std::int8_t	std::int8_t	float	float
float	float	float	float
double	double	double	double
std::complex<float>	std::complex<float>	std::complex<float>	std::complex<float>
std::complex<double>	std::complex<double>	std::complex<double>	std::complex<double>

gemm_batch (Buffer Version)

Buffer version of gemm_batch supports only strided API.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    A, B and C are matrices at offset i * stridea, i * strideb, i * stridec in a, b and c.
    C = alpha * op(A) * op(B) + beta * C
end for
```

where:

- op(X) is one of op(X) = X, or op(X) = X^T, or op(X) = X^H
- alpha and beta are scalars
- A, B, and C are matrices
- op(A) is m x k, op(B) is k x n, and C is m x n

For strided API, a, b and c buffers contains all the input matrices. The stride between matrices is given by the stride parameters. Total number of matrices in a, b and c buffers is given by batch_size parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void gemm_batch(sycl::queue &queue,
                    oneapi::mkl::transpose transa,
                    oneapi::mkl::transpose transb,
                    std::int64_t m,
                    std::int64_t n,
                    std::int64_t k,
                    Ts alpha,
                    sycl::buffer<Ta,1> &a,
                    std::int64_t lda,
                    std::int64_t stridea,
                    sycl::buffer<Tb,1> &b,
                    std::int64_t ldb,
                    std::int64_t strideb,
                    Ts beta,
                    sycl::buffer<Tc,1> &c,
                    std::int64_t ldc,
                    std::int64_t stridec,
                    std::int64_t batch_size,
                    compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void gemm_batch(sycl::queue &queue,
                    oneapi::mkl::transpose transa,
                    oneapi::mkl::transpose transb,
                    std::int64_t m,
                    std::int64_t n,
                    std::int64_t k,
                    Ts alpha,
                    sycl::buffer<Ta,1> &a,
                    std::int64_t lda,
                    std::int64_t stridea,
                    sycl::buffer<Tb,1> &b,
                    std::int64_t ldb,
                    std::int64_t strideb,
                    Ts beta,
                    sycl::buffer<Tc,1> &c,
                    std::int64_t ldc,
                    std::int64_t stridec,
                    std::int64_t batch_size,
                    compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

transa Specifies op(A), transposition operation applied to matrices A. See Data Types for more details.

transb Specifies op(B), transposition operation applied to matrices B. See Data Types for more details.

m Number of rows of matrices op(A) and matrices C. Must be at least zero.

n Number of columns of matrices op(B) and matrices C. Must be at least zero.

k Number of columns of matrices op(A) and rows of matrices op(B). Must be at least zero.

alpha Scaling factor for matrix-matrix products.

a Buffer holding input matrices A. Size of the buffer must be at least `stridea` * `batch_size`.

lda Leading dimension of matrices A. Must be positive.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `k`
Row major	Must be at least `k`	Must be at least `m`

stridea Stride between two consecutive A matrices.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `lda` * `k`	Must be at least `lda` * `m`
Row major	Must be at least `lda` * `m`	Must be at least `lda` * `k`

b Buffer holding input matrices B. Size of the buffer must be at least `strideb` * `batch_size`.

ldb Leading dimension of matrices B. Must be positive.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `k`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `k`

strideb Stride between two consecutive B matrices.

	transa = transpose::nontrans	transa = transpose::trans **or** trans = transpose::conjtrans
Column major	Must be at least ldb * n	Must be at least ldb * k
Row major	Must be at least ldb * k	Must be at least ldb * n

beta Scaling factor for matrices C.

c Buffer holding input/output matrices C. Size of the buffer must be at least stridec * batch_size.

ldc Leading dimension of matrices C. Must be positive.

Column major	Must be at least m
Row major	Must be at least n

stridec Stride between two consecutive C matrices.

Column major	Must be at least ldc * n
Row major	Must be at least ldc * m

batch_size Specifies the number of matrix multiply operations to perform. Must be at least zero.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

Output Parameters

c Output buffer overwritten by batch_size gemm operations of the form alpha * op(A) * op(B) + beta * C.

Note: If beta = 0, matrices C do not need to be initialized before calling gemm_batch.

---

gemm_batch (USM Version)

USM version of gemm_batch supports group API and strided API.

Group API

Group API operation is defined as:

```
idx = 0
for i = 0 … group_count – 1
    for j = 0 … group_size – 1
        A, B, and C are matrices in a[idx], b[idx] and c[idx]
        C = alpha[i] * op(A) * op(B) + beta[i] * C
        idx = idx + 1
    end for
end for
```

where:

- op(X) is one of op(X) = X, or op(X) = X^T, or op(X) = X^H
- `alpha` and `beta` are scalars
- A, B, and C are matrices
- op(A) is `m` x k, op(B) is k x `n`, and C is `m` x `n`

For group API, a, b and c arrays contain the pointers for all the input matrices. The total number of matrices in a, b and c are given by:

$$total\_batch\_count = \sum_{i=0}^{group\_count-1} group\_size[i]$$

Group API supports both pointer and span inputs.

The advantage of using span instead of pointer is that the sizes of the array can vary and the size of the span can be queried at runtime. For each `gemm` parameter, except the output matrices, span can be of size 1, the number of groups or the total batch size. For output matrices, to ensure all computation are independent, size of the span must be the total batch size.

Depending on the size of spans, each parameter for the `gemm` computation is used as follows:

span size = 1	Parameter is reused for all `gemm` computations
span size = `group_count`	Parameter is reused for all `gemm` within a group. Each group will have a different value for this parameter. This is same as `gemm_batch` group API with pointers.
span size = `total_batch_count`	Each `gemm` computation use a different value for this parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemm_batch(sycl::queue &queue,
                           const oneapi::mkl::transpose *transa,
                           const oneapi::mkl::transpose *transb,
                           const std::int64_t *m,
                           const std::int64_t *n,
```

(continues on next page)

(continued from previous page)

```
                            const std::int64_t *k,
                            const Ts *alpha,
                            const Ta **a,
                            const std::int64_t *lda,
                            const Tb **b,
                            const std::int64_t *ldb,
                            const Ts *beta,
                            Tc **c,
                            const std::int64_t *ldc,
                            std::int64_t group_count,
                            const std::int64_t *group_size,
                            compute_mode mode = compute_mode::unset,
                            const std::vector<sycl::event> &dependencies = {})

    sycl::event gemm_batch(sycl::queue &queue,
                            const sycl::span<oneapi::mkl::transpose> &transa,
                            const sycl::span<oneapi::mkl::transpose> &transb,
                            const sycl::span<std::int64_t> &m,
                            const sycl::span<std::int64_t> &n,
                            const sycl::span<std::int64_t> &k,
                            const sycl::span<Ts> &alpha,
                            const sycl::span<const Ta*> &a,
                            const sycl::span<std::int64_t> &lda,
                            const sycl::span<const Tb*> &b,
                            const sycl::span<std::int64_t> &ldb,
                            const sycl::span<Ts> &beta,
                            sycl::span<Tc*> &c,
                            const sycl::span<std::int64_t> &ldc,
                            size_t group_count,
                            const sycl::span<size_t> &group_sizes,
                            compute_mode mode = compute_mode::unset,
                            const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event gemm_batch(sycl::queue &queue,
                            const oneapi::mkl::transpose *transa,
                            const oneapi::mkl::transpose *transb,
                            const std::int64_t *m,
                            const std::int64_t *n,
                            const std::int64_t *k,
                            const Ts *alpha,
                            const Ta **a,
                            const std::int64_t *lda,
                            const Tb **b,
                            const std::int64_t *ldb,
                            const Ts *beta,
                            Tc **c,
                            const std::int64_t *ldc,
                            std::int64_t group_count,
                            const std::int64_t *group_size,
```

(continues on next page)

(continued from previous page)

```
                          compute_mode mode = compute_mode::unset,
                          const std::vector<sycl::event> &dependencies = {})

    sycl::event gemm_batch(sycl::queue &queue,
                          const sycl::span<oneapi::mkl::transpose> &transa,
                          const sycl::span<oneapi::mkl::transpose> &transb,
                          const sycl::span<std::int64_t> &m,
                          const sycl::span<std::int64_t> &n,
                          const sycl::span<std::int64_t> &k,
                          const sycl::span<Ts> &alpha,
                          const sycl::span<const Ta*> &a,
                          const sycl::span<std::int64_t> &lda,
                          const sycl::span<const Tb*> &b,
                          const sycl::span<std::int64_t> &ldb,
                          const sycl::span<Ts> &beta,
                          sycl::span<Tc*> &c,
                          const sycl::span<std::int64_t> &ldc,
                          size_t group_count,
                          const sycl::span<size_t> &group_sizes,
                          compute_mode mode = compute_mode::unset,
                          const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

transa Array or span of `group_count` `oneapi::mkl::transpose` values. `transa[i]` specifies op(`A`), the transposition operation applied to matrices `A` in group `i`. See Data Types for more details.

transb Array or span of `group_count` `oneapi::mkl::transpose` values. `transb[i]` specifies op(`B`), the transposition operation applied to matrices `B` in group `i`. See Data Types for more details.

m Array or span of `group_count` integers. `m[i]` specifies number of rows of matrices op(`A`) and matrices `C` in group `i`. All entries must be at least zero.

n Array or span of `group_count` integers. `n[i]` specifies number of columns of matrices op(`B`) and matrices `C` in group `i`. All entries must be at least zero.

k Array or span of `group_count` integers. `k[i]` specifies number of columns of matrices op(`A`) and rows of matrices op(`B`) in group `i`. All entries must be at least zero.

alpha Array or span of `group_count` scalar elements. `alpha[i]` specifies scaling factor for matrix-matrix products in group `i`.

a Array of `total_batch_count` pointers for input matrices `A` or span of `total_batch_count` input matrices `A`. See Matrix Storage for more details.

	transa = transpose::nontrans	transa = transpose::trans or trans = transpose::conjtrans
Column major	Size of array A[i] must be at least lda[i]*k[i]	Size of array A[i] must be at least lda[i]*m[i]
Row major	Size of array A[i] must be at least lda[i]*m[i]	Size of array A[i] must be at least lda[i]*k[i]

lda Array or span of group_count integers. lda[i] specifies leading dimension of matrices A in group i. Must be positive.

	transa = transpose::nontrans	transa = transpose::trans or trans = transpose::conjtrans
Column major	Must be at least m[i]	Must be at least k[i]
Row major	Must be at least k[i]	Must be at least m[i]

b Array of total_batch_count pointers for input matrices B or span of total_batch_count input matrices B. See Matrix Storage for more details.

	transa = transpose::nontrans	transa = transpose::trans or trans = transpose::conjtrans
Column major	Size of array B[i] must be at least ldb[i]*n[i]	Size of array B[i] must be at least ldb[i]*k[i]
Row major	Size of array B[i] must be at least ldb[i]*k[i]	Size of array B[i] must be at least ldb[i]*n[i]

ldb Array or span of group_count integers. ldb[i] specifies leading dimension of matrices B in group i. Must be positive.

	transa = transpose::nontrans	transa = transpose::trans or trans = transpose::conjtrans
Column major	Must be at least k[i]	Must be at least n[i]
Row major	Must be at least n[i]	Must be at least k[i]

beta Array or span of group_count scalar elements. beta[i] specifies scaling factor for matrices C in group i.

c Array of total_batch_count pointers for input/output matrices C or span of total_batch_count input/output matrices C. See Matrix Storage for more details.

Column major	Size of array C[i] must be at least ldc[i]*n[i]
Row major	Size of array C[i] must be at least ldc[i]*m[i]

ldc Array or span of `group_count` integers. `ldc[i]` specifies leading dimension of matrices `C` in group `i`. Must be positive.

Column major	Must be at least `m[i]`
Row major	Must be at least `n[i]`

group_count Number of groups. Must be at least zero.

group_size Array or span of `group_count` integers. `group_size[i]` specifies the number of `gemm` operations in group `i`. Each element in `group_size` must be at least zero.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Array of pointers to output matrices `C` overwritten by `total_batch_count` `gemm` operations of the form `alpha` * op(`A`) * op(`B`) + `beta` * `C`.

Note: If `beta` = 0, matrices `C` do not need to be initialized before calling `gemm_batch`.

Return Values

Output event to wait on to ensure computation is complete.

Examples

An example of how to use USM version of `gemm_batch` can be found in oneMKL installation directory, under:

```
examples/dpcpp/blas/source/gemm_batch.cpp
```

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    A, B and C are matrices at offset i * stridea, i * strideb, i * stridec in a, b and c.
    C = alpha * op(A) * op(B) + beta * C
end for
```

where:

- op(X) is one of op(X) = X, or op(X) = X^T, or op(X) = X^H
- `alpha` and `beta` are scalars
- A, B, and C are matrices
- op(A) is `m` x k, op(B) is k x `n`, and C is `m` x `n`

For strided API, a, b and c arrays contains all the input matrices. The stride between matrices is either given by the stride parameters. Total number of matrices in a, b and c arrays is given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemm_batch(sycl::queue &queue,
                           oneapi::mkl::transpose transa,
                           oneapi::mkl::transpose transb,
                           std::int64_t m,
                           std::int64_t n,
                           std::int64_t k,
                           Ts alpha,
                           const Ta *a,
                           std::int64_t lda,
                           std::int64_t stridea,
                           const Tb *b,
                           std::int64_t ldb,
                           std::int64_t strideb,
                           Ts beta,
                           Tc *c,
                           std::int64_t ldc,
                           std::int64_t stridec,
                           std::int64_t batch_size,
                           compute_mode mode = compute_mode::unset,
                           const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event gemm_batch(sycl::queue &queue,
                           oneapi::mkl::transpose transa,
                           oneapi::mkl::transpose transb,
                           std::int64_t m,
                           std::int64_t n,
                           std::int64_t k,
                           Ts alpha,
                           const Ta *a,
                           std::int64_t lda,
                           std::int64_t stridea,
                           const Tb *b,
                           std::int64_t ldb,
```

(continues on next page)

(continued from previous page)

```
                          std::int64_t strideb,
                          Ts beta,
                          Tc *c,
                          std::int64_t ldc,
                          std::int64_t stridec,
                          std::int64_t batch_size,
                          compute_mode mode = compute_mode::unset,
                          const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

transa Specifies op(A), transposition operation applied to matrices A. See Data Types for more details.

transb Specifies op(B), transposition operation applied to matrices B. See Data Types for more details.

m Number of rows of matrices op(A) and matrices C. Must be at least zero.

n Number of columns of matrices op(B) and matrices C. Must be at least zero.

k Number of columns of matrices op(A) and rows of matrices op(B). Must be at least zero.

alpha Scaling factor for matrix-matrix products.

a Pointer to input matrices A. Size of the array must be at least `stridea * batch_size`.

lda Leading dimension of matrices A. Must be positive.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `k`
Row major	Must be at least `k`	Must be at least `m`

stridea Stride between two consecutive A matrices.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `lda * k`	Must be at least `lda * m`
Row major	Must be at least `lda * m`	Must be at least `lda * k`

b Pointer to input matrices B. Size of the array must be at least `strideb * batch_size`.

ldb Leading dimension of matrices B. Must be positive.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least k	Must be at least n
Row major	Must be at least n	Must be at least k

strideb Stride between two consecutive B matrices.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `ldb` * n	Must be at least `ldb` * k
Row major	Must be at least `ldb` * k	Must be at least `ldb` * n

beta Scaling factor for matrices C.

c Pointer to input/output matrices C. Size of the array must be at least `stridec` * `batch_size`.

ldc Leading dimension of matrices C. Must be positive.

Column major	Must be at least m
Row major	Must be at least n

stridec Stride between two consecutive C matrices.

Column major	Must be at least `ldc` * n
Row major	Must be at least `ldc` * m

batch_size Specifies the number of matrix multiply operations to perform. Must be at least zero.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Pointer to output matrices `C` overwritten by `batch_size` `gemm` operations of the form `alpha` * op(`A`) * op(`B`) + `beta` * `C`.

Note: If `beta` = 0, matrices `C` do not need to be initialized before calling `gemm_batch`.

Return Values

Output event to wait on to ensure computation is complete.

9.4.6 gemm_bias

Computes a matrix-matrix product using general integer matrices with bias.

Description

The `gemm_bias` routines compute a scalar-matrix-matrix product and add the result to a scalar-matrix product, using general integer matrices with biases/offsets. The operation is defined as:

$$C \leftarrow alpha * (op(A) - A\_offset) * (op(B) - B\_offset) + beta * C + C\_offset$$

where:

- op(`X`) is one of op(`X`) = `X`, or op(`X`) = X^T, or op(`X`) = X^H
- `alpha` and `beta` are scalars
- `A_offset` is `m` x `k` matrix with every element equal to the value `ao`
- `B_offset` is `k` x `n` matrix with every element equal to the value `bo`
- `C_offset` is `m` x `n` matrix defined by the `co` buffer. See Data Types for more details.
- `A`, `B`, and `C` are matrices
- op(`A`) is `m` x `k`, op(`B`) is `k` x `n`, and `C` is `m` x `n`

`gemm_bias` supports the following precisions:

Ta	Tb
`std::uint8_t`	`std::uint8_t`
`std::int8_t`	`std::uint8_t`
`std::uint8_t`	`std::int8_t`
`std::int8_t`	`std::int8_t`

gemm_bias (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void gemm_bias(sycl::queue &queue,
                   oneapi::mkl::transpose transa,
                   oneapi::mkl::transpose transb,
                   oneapi::mkl::offset offsetc,
                   std::int64_t m,
                   std::int64_t n,
                   std::int64_t k,
                   float alpha,
                   sycl::buffer<Ta,1> &a,
                   std::int64_t lda,
                   Ta ao,
                   sycl::buffer<Tb,1> &b,
                   std::int64_t ldb,
                   Tb bo,
                   float beta,
                   sycl::buffer<std::int32_t,1> &c,
                   std::int64_t ldc,
                   sycl::buffer<std::int32_t,1> &co,
                   compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void gemm_bias(sycl::queue &queue,
                   oneapi::mkl::transpose transa,
                   oneapi::mkl::transpose transb,
                   oneapi::mkl::offset offsetc,
                   std::int64_t m,
                   std::int64_t n,
                   std::int64_t k,
                   float alpha,
                   sycl::buffer<Ta,1> &a,
                   std::int64_t lda,
                   Ta ao,
                   sycl::buffer<Tb,1> &b,
                   std::int64_t ldb,
                   Tb bo,
                   float beta,
                   sycl::buffer<std::int32_t,1> &c,
                   std::int64_t ldc,
                   sycl::buffer<std::int32_t,1> &co,
                   compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

transa Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

transb Specifies op(B), the transposition operation applied to matrix B. See Data Types for more details.

offsetc Specifies the form of C_offset used in the matrix multiplication. See Data Types for more details.

m Number of rows of matrix op(A) and matrix C. Must be at least zero.

n Number of columns of matrix op(B) and matrix C. Must be at least zero.

k Number of columns of matrix op(A) and rows of matrix op(B). Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Buffer holding input matrix A. See Matrix Storage for more details.

	transa = transpose::nontrans	transa = transpose::trans **or** trans = transpose::conjtrans
Column major	A is m x k matrix. Size of array a must be at least lda * k	A is k x m matrix. Size of array a must be at least lda * m
Row major	A is m x k matrix. Size of array a must be at least lda * m	A is k x m matrix. Size of array a must be at least lda * k

lda Leading dimension of matrix A. Must be positive.

	transa = transpose::nontrans	transa = transpose::trans **or** trans = transpose::conjtrans
Column major	Must be at least m	Must be at least k
Row major	Must be at least k	Must be at least m

ao Specifies the scalar offset value for matrix A.

b Buffer holding input matrix B. See Matrix Storage for more details.

	transb = transpose::nontrans	transb = transpose::trans **or** trans = transpose::conjtrans
Column major	B is k x n matrix. Size of array b must be at least ldb * n	B is n x k matrix. Size of array b must be at least ldb * k
Row major	B is k x n matrix. Size of array b must be at least ldb * k	B is n x k matrix. Size of array b must be at least ldb * n

ldb Leading dimension of matrix B. Must be positive.

	transb = transpose::nontrans	transb = transpose::trans **or** trans = transpose::conjtrans
Column major	Must be at least k	Must be at least n
Row major	Must be at least n	Must be at least k

bo Specifies the scalar offset value for matrix B.

beta Scaling factor for matrix C.

c Buffer holding input/output matrix C. See **Matrix Storage** for more details.

Column major	C is m x n matrix. Size of array c must be at least ldc * n
Row major	C is m x n matrix. Size of array c must be at least ldc * m

ldc Leading dimension of matrix C. Must be positive.

Column major	Must be at least m
Row major	Must be at least n

co Buffer holding the offset values for matrix C.

If offset_type = offset::fix, size of co array must be at least 1.

If offset_type = offset::col, size of co array must be at least max(1,m).

If offset_type = offset::row, size of co array must be at least max(1,n).

See **Data Types** for more details.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

Output Parameters

c Output buffer overwritten by alpha * (op(A) - A_offset) * (op(B) - B_offset) + beta * C + C_offset.

Note: If beta = 0, matrix C does not need to be initialized before calling gemm_bias.

---

gemm_bias (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemm_bias(sycl::queue &queue,
                          oneapi::mkl::transpose transa,
                          oneapi::mkl::transpose transb,
                          oneapi::mkl::offset offsetc,
                          std::int64_t m,
                          std::int64_t n,
                          std::int64_t k,
                          float alpha,
                          const Ta *a,
                          std::int64_t lda,
                          Ta ao,
                          const Tb *b,
                          std::int64_t ldb,
                          Tb bo,
                          float beta,
                          std::int32_t *c,
                          std::int64_t ldc,
                          const std::int32_t *co,
                          compute_mode mode = compute_mode::unset,
                          const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event gemm_bias(sycl::queue &queue,
                          oneapi::mkl::transpose transa,
                          oneapi::mkl::transpose transb,
                          oneapi::mkl::offset offsetc,
                          std::int64_t m,
                          std::int64_t n,
                          std::int64_t k,
                          float alpha,
                          const Ta *a,
                          std::int64_t lda,
                          Ta ao,
                          const Tb *b,
                          std::int64_t ldb,
                          Tb bo,
                          float beta,
                          std::int32_t *c,
                          std::int64_t ldc,
                          const std::int32_t *co,
                          compute_mode mode = compute_mode::unset,
                          const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

transa Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

transb Specifies op(B), the transposition operation applied to matrix B. See Data Types for more details.

offsetc Specifies the form of C_offset used in the matrix multiplication. See Data Types for more details.

m Number of rows of matrix op(A) and matrix C. Must be at least zero.

n Number of columns of matrix op(B) and matrix C. Must be at least zero.

k Number of columns of matrix op(A) and rows of matrix op(B). Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Pointer to input matrix A. See Matrix Storage for more details.

	transa = transpose::nontrans	transa = transpose::trans **or** trans = transpose::conjtrans
Column major	A is m x k matrix. Size of array a must be at least lda * k	A is k x m matrix. Size of array a must be at least lda * m
Row major	A is m x k matrix. Size of array a must be at least lda * m	A is k x m matrix. Size of array a must be at least lda * k

lda Leading dimension of matrix A. Must be positive.

	transa = transpose::nontrans	transa = transpose::trans **or** trans = transpose::conjtrans
Column major	Must be at least m	Must be at least k
Row major	Must be at least k	Must be at least m

ao Specifies the scalar offset value for matrix A.

b Pointer to input matrix B. See Matrix Storage for more details.

	transb = transpose::nontrans	transb = transpose::trans **or** trans = transpose::conjtrans
Column major	B is k x n matrix. Size of array b must be at least ldb * n	B is n x k matrix. Size of array b must be at least ldb * k
Row major	B is k x n matrix. Size of array b must be at least ldb * k	B is n x k matrix. Size of array b must be at least ldb * n

ldb Leading dimension of matrix `B`. Must be positive.

	`transb = transpose::nontrans`	`transb = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `k`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `k`

bo Specifies the scalar offset value for matrix `B`.

beta Scaling factor for matrix `C`.

c Pointer to input/output matrix `C`. See **Matrix Storage** for more details.

Column major	`C` is `m` x `n` matrix. Size of array `c` must be at least `ldc * n`
Row major	`C` is `m` x `n` matrix. Size of array `c` must be at least `ldc * m`

ldc Leading dimension of matrix `C`. Must be positive.

Column major	Must be at least `m`
Row major	Must be at least `n`

co Pointer to array holding offset values for matrix `C`.

If `offset_type = offset::fix`, size of `co` array must be at least 1.

If `offset_type = offset::col`, size of `co` array must be at least `max(1,m)`.

If `offset_type = offset::row`, size of `co` array must be at least `max(1,n)`.

See **Data Types** for more details.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Pointer to output matrix `C` overwritten by `alpha` * (op(`A`) - `A_offset`) * (op(`B`) - `B_offset`) + `beta` * `C` + `C_offset`.

Note: If `beta` = 0, matrix `C` does not need to be initialized before calling `gemm_bias`.

Return Values

Output event to wait on to ensure computation is complete.

9.4.7 gemmt

Computes a matrix-matrix product with general matrices, but updates only the upper or lower triangular part of the result matrix.

Description

The `gemmt` routines compute a scalar-matrix-matrix product and add the result to the upper or lower part of a scalar-matrix product, with general matrices. The operation is defined as:

$$C \leftarrow alpha * op(A) * op(B) + beta * C$$

where:

- op(X) is one of op(X) = X, or op(X) = X^T, or op(X) = X^H
- `alpha` and `beta` are scalars
- A, B, and C are matrices
- op(A) is n x k, op(B) is k x n, and C is n x n

`gemmt` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

gemmt (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void gemmt(sycl::queue &queue,
               oneapi::mkl::uplo upper_lower,
               oneapi::mkl::transpose transa,
               oneapi::mkl::transpose transb,
               std::int64_t n,
               std::int64_t k,
               T alpha,
               sycl::buffer<T,1> &a,
               std::int64_t lda,
```

(continues on next page)

(continued from previous page)

```
                sycl::buffer<T,1> &b,
                std::int64_t ldb,
                T beta,
                sycl::buffer<T,1> &c,
                std::int64_t ldc,
                compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void gemmt(sycl::queue &queue,
                oneapi::mkl::uplo upper_lower,
                oneapi::mkl::transpose transa,
                oneapi::mkl::transpose transb,
                std::int64_t n,
                std::int64_t k,
                T alpha,
                sycl::buffer<T,1> &a,
                std::int64_t lda,
                sycl::buffer<T,1> &b,
                std::int64_t ldb,
                T beta,
                sycl::buffer<T,1> &c,
                std::int64_t ldc,
                compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix `C` is upper or lower triangular. See Data Types for more details.

transa Specifies op(`A`), the transposition operation applied to matrix `A`. See Data Types for more details.

transb Specifies op(`B`), the transposition operation applied to matrix `B`. See Data Types for more details.

n Number of rows of matrix op(`A`) and matrix `C`. Must be at least zero.

k Number of columns of matrix op(`A`) and rows of matrix op(`B`). Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Buffer holding input matrix `A`. See Matrix Storage for more details.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	`A` is n x k matrix. Size of array a must be at least `lda` * k	`A` is k x n matrix. Size of array a must be at least `lda` * n
Row major	`A` is n x k matrix. Size of array a must be at least `lda` * n	`A` is k x n matrix. Size of array a must be at least `lda` * k

lda Leading dimension of matrix A. Must be positive.

	transa = transpose::nontrans	transa = transpose::trans **or** trans = transpose::conjtrans
Column major	Must be at least n	Must be at least k
Row major	Must be at least k	Must be at least n

b Buffer holding input matrix B. See **Matrix Storage** for more details.

	transb = transpose::nontrans	transb = transpose::trans **or** trans = transpose::conjtrans
Column major	B is k x n matrix. Size of array b must be at least ldb * n	B is n x k matrix. Size of array b must be at least ldb * k
Row major	B is k x n matrix. Size of array b must be at least ldb * k	B is n x k matrix. Size of array b must be at least ldb * n

ldb Leading dimension of matrix B. Must be positive.

	transb = transpose::nontrans	transb = transpose::trans **or** trans = transpose::conjtrans
Column major	Must be at least k	Must be at least n
Row major	Must be at least n	Must be at least k

beta Scaling factor for matrix C.

c Buffer holding input/output matrix C. See **Matrix Storage** for more details.

Column major	C is m x n matrix. Size of array c must be at least ldc * n
Row major	C is m x n matrix. Size of array c must be at least ldc * m

ldc Leading dimension of matrix C. Must be positive.

Column major	Must be at least m
Row major	Must be at least n

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

Output Parameters

c Output buffer overwritten by upper or lower triangular part of `alpha` * op(`A`)*op(`B`) + `beta` * `C`.

Note: If `beta` = 0, matrix `C` does not need to be initialized before calling `gemmt`.

---

gemmt (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemmt(sycl::queue &queue,
                      oneapi::mkl::uplo upper_lower,
                      oneapi::mkl::transpose transa,
                      oneapi::mkl::transpose transb,
                      std::int64_t n,
                      std::int64_t k,
                      T alpha,
                      const T* a,
                      std::int64_t lda,
                      const T* b,
                      std::int64_t ldb,
                      T beta,
                      T* c,
                      std::int64_t ldc,
                      compute_mode mode = compute_mode::unset,
                      const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event gemmt(sycl::queue &queue,
                      oneapi::mkl::uplo upper_lower,
                      oneapi::mkl::transpose transa,
                      oneapi::mkl::transpose transb,
                      std::int64_t n,
                      std::int64_t k,
                      T alpha,
                      const T* a,
                      std::int64_t lda,
                      const T* b,
                      std::int64_t ldb,
                      T beta,
                      T* c,
                      std::int64_t ldc,
                      compute_mode mode = compute_mode::unset,
                      const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrix C is upper or lower triangular. See Data Types for more details.

transa Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

transb Specifies op(B), the transposition operation applied to matrix B. See Data Types for more details.

n Number of rows of matrix op(A) and matrix C. Must be at least zero.

k Number of columns of matrix op(A) and rows of matrix op(B). Must be at least zero.

alpha Scaling factor for matrix-matrix product.

a Pointer to input matrix A. See Matrix Storage for more details.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	A is n x k matrix. Size of array a must be at least `lda` * k	A is k x n matrix. Size of array a must be at least `lda` * n
Row major	A is n x k matrix. Size of array a must be at least `lda` * n	A is k x n matrix. Size of array a must be at least `lda` * k

lda Leading dimension of matrix A. Must be positive.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least n	Must be at least k
Row major	Must be at least k	Must be at least n

b Pointer to input matrix B. See Matrix Storage for more details.

	`transb = transpose::nontrans`	`transb = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	B is k x n matrix. Size of array b must be at least `ldb` * n	B is n x k matrix. Size of array b must be at least `ldb` * k
Row major	B is k x n matrix. Size of array b must be at least `ldb` * k	B is n x k matrix. Size of array b must be at least `ldb` * n

ldb Leading dimension of matrix B. Must be positive.

	`transb = transpose::nontrans`	`transb = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `k`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `k`

beta Scaling factor for matrix `C`.

c Pointer to input/output matrix `C`. See **Matrix Storage** for more details.

Column major	`C` is `m` x `n` matrix. Size of array `c` must be at least `ldc * n`
Row major	`C` is `m` x `n` matrix. Size of array `c` must be at least `ldc * m`

ldc Leading dimension of matrix `C`. Must be positive.

Column major	Must be at least `m`
Row major	Must be at least `n`

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Pointer to output matrix `C` overwritten by upper or lower triangular part of `alpha` * op(`A`)*op(`B`) + `beta` * `C`.

Note: If `beta` = 0, matrix `C` does not need to be initialized before calling `gemmt`.

Return Values

Output event to wait on to ensure computation is complete.

9.4.8 gemv_batch

Computes a group of `gemv` operations.

Description

The `gemv_batch` routines are batched versions of **gemv**, performing multiple `gemv` operations in a single call. Each `gemv` operations perform a scalar-matrix-vector product and add the result to a scalar-vector product.

`gemv_batch` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std::complex<double>`

gemv_batch (Buffer Version)

Buffer version of `gemv_batch` supports only strided API.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    A is a matrix at offset i * stridea in a.
    X and Y are vctors at offset i * stridex, i * stridey, in x and y.
    Y = alpha * op(A) * X + beta * Y
end for
```

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- `alpha` and `beta` are scalars
- A is matrix and X and Y are vectors

For strided API, x and y buffers contain all the input vectors. The stride between vectors is either given by the stride parameters. Total number of vectors in x and y buffers is given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void gemv_batch(sycl::queue &queue,
                    oneapi::mkl::transpose trans,
                    std::int64_t m,
                    std::int64_t n,
```

(continues on next page)

(continued from previous page)

```
                    T alpha,
                    sycl::buffer<T,1> &a,
                    std::int64_t lda,
                    std::int64_t stridea,
                    sycl::buffer<T,1> &x,
                    std::int64_t incx,
                    std::int64_t stridex,
                    T beta,
                    sycl::buffer<T,1> &y,
                    std::int64_t incy,
                    std::int64_t stridey,
                    std::int64_t batch_size)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void gemv_batch(sycl::queue &queue,
                    oneapi::mkl::transpose trans,
                    std::int64_t m,
                    std::int64_t n,
                    T alpha,
                    sycl::buffer<T,1> &a,
                    std::int64_t lda,
                    std::int64_t stridea,
                    sycl::buffer<T,1> &x,
                    std::int64_t incx,
                    std::int64_t stridex,
                    T beta,
                    sycl::buffer<T,1> &y,
                    std::int64_t incy,
                    std::int64_t stridey,
                    std::int64_t batch_size)
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Specifies op(`A`), the transposition operation applied to matrices `A`. See Data Types for more details.

m Number of rows of matrices op(`A`). Must be at least zero.

n Number of columns of matrices op(`A`). Must be at least zero.

alpha Scaling factor for matrix-vector product.

a The buffer holding input matrices `A`. Size of the buffer must be at least `stridea` * `batch_size`.

lda Leading dimension of matrices `A`. Must be positive and at least `m` if column major layout or at least `n` if row major layout is used.

stridea Stride between two consecutive `A` matrices.

x Buffer holding input vectors X. Size of the buffer must be at least `stridex` * `batch_size`.

incx Stride between two consecutive elements of X vectors.

stridex Stride between two consecutive X vectors. Must be at least zero.

beta Scaling factor for vectors Y.

y Buffer holding input/output vectors Y. Size of the buffer must be at least `stridey` * `batch_size`.

incy Stride between two consecutive elements of Y vectors.

stridey Stride between two consecutive Y vectors. Must be at least zero.

batch_size Number of `gemv` computations to perform. Must be at least zero.

Output Parameters

y Output buffer overwritten by `batch_size` `gemv` operations of the form `alpha` * op(A) * X + `beta` * Y.

gemv_batch (USM Version)

USM version of `gemv_batch` supports group API strided API.

Group API

Group API operation is defined as:

```
idx = 0
for i = 0 … group_count – 1
    for j = 0 … group_size – 1
        A is an m x n matrix in a[idx]
        X and Y are vectors in x[idx] and y[idx]
        Y = alpha[i] * op(A) * X + beta[i] * Y
        idx = idx + 1
    end for
end for
```

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- `alpha` and `beta` are scalars
- A is matrix and X and Y are vectors

For group API, x and y arrays contain the pointers for all the input vectors. a array contains the pointers to all input matrices. The total number of vectors in x and y and matrices in a is given by:

$$total\_batch\_count = \sum_{i=0}^{group\_count-1} group\_size[i]$$

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemv_batch(sycl::queue &queue,
                           const oneapi::mkl::transpose *trans,
                           const std::int64_t *m,
                           const std::int64_t *n,
                           const T *alpha,
                           const T **a,
                           const std::int64_t *lda,
                           const T **x,
                           const std::int64_t *incx,
                           const T *beta,
                           T **y,
                           const std::int64_t *incy,
                           std::int64_t group_count,
                           const std::int64_t *group_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event gemv_batch(sycl::queue &queue,
                           const oneapi::mkl::transpose *trans,
                           const std::int64_t *m,
                           const std::int64_t *n,
                           const T *alpha,
                           const T **a,
                           const std::int64_t *lda,
                           const T **x,
                           const std::int64_t *incx,
                           const T *beta,
                           T **y,
                           const std::int64_t *incy,
                           std::int64_t group_count,
                           const std::int64_t *group_size,
                           const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Array of `group_count` `oneapi::mkl::transpose` values. `transa[i]` specifies op(`A`), the transposition operation applied to matrices `A` in group `i`. See Data Types for more details.

m Array of `group_count` integers. `m[i]` specifies number of rows of matrices op(`A`) in group `i`. All entries must be at least zero.

n Array of `group_count` integers. `n[i]` specifies number of columns of matrices op(`A`) in group `i`. All entries must be at least zero.

alpha Array of `group_count` scalar elements. `alpha[i]` specifies scaling factor for matrix-vector products in group `i`.

a Array of `total_batch_count` pointers for input matrices `A`. See Matrix Storage for more details.

lda Array of `group_count` integers. `lda[i]` specifies leading dimension of matrices `A` in group `i`. Must be positive and at least `m[i]` if column major layout or at least `n[i]` if row major layout is used.

x Array of `total_batch_count` pointers for input vectors `X`. See Matrix Storage for more details.

incx Array of `group_count` integers. `incx[i]` specifies stride of vectors `X` in group `i`.

beta Array of `group_count` scalar elements. `beta[i]` specifies scaling factor for vectors `Y` in group `i`.

y Array of `total_batch_count` pointers for input/output vectors `Y`. See Matrix Storage for more details.

incy Array of `group_count` integers. `incy[i]` specifies stride of vectors `Y` in group `i`.

group_count Number of groups. Must be at least zero.

group_size Array of `group_count` integers. `group_size[i]` specifies the number of `gemv` operations in group `i`. Each element in `group_size` must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Array of pointers to output vectors `Y` overwritten by `total_batch_count` `gemv` operations of the form `alpha * op(A) * X + beta * Y`.

Return Values

Output event to wait on to ensure computation is complete.

Examples

An example of how to use USM version of `gemv_batch` can be found in oneMKL installation directory, under:

```
examples/dpcpp/blas/source/gemv_batch_usm.cpp
```

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    A is a matrix at offset i * stridea in a.
    X and Y are vctors at offset i * stridex, i * stridey, in x and y.
    Y = alpha * op(A) * X + beta * Y
end for
```

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- alpha and beta are scalars
- A is matrix and X and Y are vectors

For strided API, x and y arrays contain all the input vectors. The stride between vectors is given by the stride parameters. Total number of vectors in x and y arrays is given by batch_size parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemv_batch(sycl::queue &queue,
                           oneapi::mkl::transpose trans,
                           std::int64_t m,
                           std::int64_t n,
                           T alpha,
                           const T *a,
                           std::int64_t lda,
                           std::int64_t stridea,
                           const T *x,
                           std::int64_t incx,
                           std::int64_t stridex,
                           T beta,
                           T *y,
                           std::int64_t incy,
                           std::int64_t stridey,
                           std::int64_t batch_size)
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event gemv_batch(sycl::queue &queue,
                           oneapi::mkl::transpose trans,
                           std::int64_t m,
                           std::int64_t n,
                           T alpha,
                           const T *a,
                           std::int64_t lda,
                           std::int64_t stridea,
                           const T *x,
                           std::int64_t incx,
                           std::int64_t stridex,
                           T beta,
                           T *y,
                           std::int64_t incy,
                           std::int64_t stridey,
                           std::int64_t batch_size)
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Specifies op(A), the transposition operation applied to matrices A. See Data Types for more details.

m Number of rows of matrices op(A). Must be at least zero.

n Number of columns of matrices op(A). Must be at least zero.

alpha Scaling factor for matrix-vector product.

a Pointer to input matrices A. Size of the array must be at least `stridea * batch_size`.

lda Leading dimension of matrices A. Must be positive and at least `m` if column major layout or at least `n` if row major layout is used.

stridea Stride between two consecutive A matrices.

x Pointer to input vectors X. Size of the array must be at least `stridex * batch_size`.

incx Stride between two consecutive elements of X vectors.

stridex Stride between two consecutive X vectors. Must be at least zero.

beta Scaling factor for vectors Y.

y Pointer to input/output vectors Y. Size of the array must be at least `stridey * batch_size`.

incy Stride between two consecutive elements of Y vectors.

stridey Stride between two consecutive Y vectors. Must be at least zero.

batch_size Number of `gemv` computations to perform. Must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

y Pointer to output vectors Y overwritten by `batch_size gemv` operations of the form `alpha * op(A) * X + beta * Y`.

Return Values

Output event to wait on to ensure computation is complete.

9.4.9 syrk_batch

Computes a group of `syrk` operations.

Description

The `syrk_batch` routines are batched versions of **syrk**, performing multiple `syrk` operations in a single call. Each `syrk` operation performs a rank-k update with general matrices.

`syrk_batch` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std::complex<double>`

syrk_batch (Buffer Version)

Buffer version of `syrk_batch` supports only strided API.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    A and C are matrices at offset i * stridea and i * stridec in a and c.
    C = alpha * op(A) * op(A)^T + beta * C
end for
```

where:

- op(X) is one of op(X) = X, or op(X) = X^T, or op(X) = X^H
- `alpha` and `beta` are scalars
- `A` is general matrix and `C` is symmetric matrix
- op(`A`) is n x k and `C` is n x n

For strided API, a and c buffers contain all the input matrices. The stride between matrices is given by the stride parameters. Total number of matrices in a and c buffers is given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
   void syrk_batch(sycl::queue &queue,
                   oneapi::mkl::uplo upper_lower,
                   oneapi::mkl::transpose trans,
                   std::int64_t n,
                   std::int64_t k,
                   T alpha,
                   sycl::buffer<T,1> &a,
                   std::int64_t lda,
                   std::int64_t stridea,
                   T beta,
                   sycl::buffer<T,1> &c,
                   std::int64_t ldc,
                   std::int64_t stridec,
                   std::int64_t batch_size,
                   compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
   void syrk_batch(sycl::queue &queue,
                   oneapi::mkl::uplo upper_lower,
                   oneapi::mkl::transpose trans,
                   std::int64_t n,
                   std::int64_t k,
                   T alpha,
                   sycl::buffer<T,1> &a,
                   std::int64_t lda,
                   std::int64_t stridea,
                   T beta,
                   sycl::buffer<T,1> &c,
                   std::int64_t ldc,
                   std::int64_t stridec,
                   std::int64_t batch_size,
                   compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrices `C` are upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), transposition operation applied to matrices `A`. Conjugation is never performed even if `trans` = `transpose::conjtrans`. See Data Types for more details.

n Number of rows and columns of matrices `C`. Must be at least zero.

k Number of columns of matrices op(`A`). Must be at least zero.

alpha Scaling factor for rank-k update.

a Buffer holding input matrices `A`. Size of the buffer must be at least `stridea` * `batch_size`.

lda Leading dimension of matrices A. Must be positive.

	transa = transpose::nontrans	transa = transpose::trans **or** trans = transpose::conjtrans
Column major	Must be at least n	Must be at least k
Row major	Must be at least k	Must be at least n

stridea Stride between two consecutive A matrices.

	transa = transpose::nontrans	transa = transpose::trans **or** trans = transpose::conjtrans
Column major	Must be at least lda * k	Must be at least lda * n
Row major	Must be at least lda * n	Must be at least lda * k

beta Scaling factor for matrices C.

c Buffer holding input/output matrices C. Size of the buffer must be at least stridec * batch_size.

ldc Leading dimension of matrices C. Must be positive and at least n.

stridec Stride between two consecutive C matrices. Must be least ldc * n.

batch_size Specifies the number of matrix multiply operations to perform.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

Output Parameters

c Output buffer overwritten by batch_size syrk operations of the form alpha * op(A) * op(A)^T + beta * C.

syrk_batch (USM Version)

USM version of syrk_batch supports group API and strided API.

Group API

Group API operation is defined as:

```
idx = 0
for i = 0 … group_count – 1
    for j = 0 … group_size – 1
        A, and C are matrices in a[idx] and c[idx]
        C = alpha[i] * op(A) * op(A)^T + beta[i] * C
        idx := idx + 1
    end for
end for
```

where:

- op(X) is one of op(X) = X, or op(X) = X^T, or op(X) = X^H
- `alpha` and `beta` are scalars
- A is general matrix and C is symmetric matrix
- op(A) is n x k and C is n x n

For group API, a and c arrays contain the pointers for all the input matrices. The total number of matrices in a and c are given by:

$$total\_batch\_count = \sum_{i=0}^{group\_count-1} group\_size[i]$$

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event syrk_batch(sycl::queue &queue,
                           const oneapi::mkl::uplo *upper_lower,
                           const oneapi::mkl::transpose *trans,
                           const std::int64_t *n,
                           const std::int64_t *k,
                           const T *alpha,
                           const T **a,
                           const std::int64_t *lda,
                           const T *beta,
                           T **c,
                           const std::int64_t *ldc,
                           std::int64_t group_count,
                           const std::int64_t *group_size,
                           compute_mode mode = compute_mode::unset,
                           const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event syrk_batch(sycl::queue &queue,
                           const oneapi::mkl::uplo *upper_lower,
                           const oneapi::mkl::transpose *trans,
                           const std::int64_t *n,
```

(continues on next page)

(continued from previous page)

```
                            const std::int64_t *k,
                            const T *alpha,
                            const T **a,
                            const std::int64_t *lda,
                            const T *beta,
                            T **c,
                            const std::int64_t *ldc,
                            std::int64_t group_count,
                            const std::int64_t *group_size,
                            compute_mode mode = compute_mode::unset,
                            const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Array of `group_count` `oneapi::mkl::uplo` values. `upper_lower[i]` specifies whether matrices `C` are upper or lower triangular in group `i`. See Data Types for more details.

trans Array of `group_count` `oneapi::mkl::transpose` values. `trans[i]` specifies op(`A`), transposition operation applied to matrices `A` in group `i`. See Data Types for more details.

n Array of `group_count` integers. `n[i]` specifies number of rows and columns of matrices `C` in group `i`. All entries must be at least zero.

k Array of `group_count` integers. `k[i]` specifies number of columns of matrices op(`A`) in group `i`. All entries must be at least zero.

alpha Array of `group_count` scalar elements. `alpha[i]` specifies scaling factor for every rank-k update in group `i`.

a Array of `total_batch_count` pointers for input matrices `A`. See Matrix Storage for more details.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Size of array `A[i]` must be at least `lda[i]*k[i]`	Size of array `A[i]` must be at least `lda[i]*n[i]`
Row major	Size of array `A[i]` must be at least `lda[i]*n[i]`	Size of array `A[i]` must be at least `lda[i]*k[i]`

lda Array of `group_count` integers. `lda[i]` specifies leading dimension of matrices `A` in group `i`. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `n[i]`.	Must be at least `k[i]`.
Row major	Must be at least `k[i]`.	Must be at least `n[i]`.

beta Array of `group_count` scalar elements. `beta[i]` specifies scaling factor for matrices `C` in group `i`.

c Array of `total_batch_count` pointers for input/output matrices `C`. Size of array `C[i]` must be at least `ldc[i] * n[i]`. See Matrix Storage for more details.

ldc Array of `group_count` integers. `ldc[i]` specifies leading dimension of matrices `C` in group `i`. Must be positive.

group_count Number of groups. Must be at least zero.

group_size Array of `group_count` integers. `group_size[i]` specifies the number of `syrk` operations in group `i`. Each element in `group_size` must be at least zero.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Array of pointers to output matrices `C` overwritten by `total_batch_count` `syrk` operations of the form `alpha * op(A) * op(A)`^{T} `+ beta * C`.

Return Values

Output event to wait on to ensure computation is complete.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    A and C are matrices at offset i * stridea and i * stridec in a and c.
    C = alpha * op(A) * op(A)^T + beta * C
end for
```

where:

- op(X) is one of op(X) = X, or op(X) = X^T, or op(X) = X^H
- `alpha` and `beta` are scalars
- `A` is general matrix and `C` is symmetric matrix
- op(`A`) is n x k and `C` is n x n

For strided API, a and c arrays contain all the input matrices. The stride between matrices is given by the stride parameters. Total number of matrices in a and c arrays is given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
   sycl::event syrk_batch(sycl::queue &queue,
                          oneapi::mkl::uplo upper_lower,
                          oneapi::mkl::transpose trans,
                          std::int64_t n,
                          std::int64_t k,
                          T alpha,
                          const T *a,
                          std::int64_t lda,
                          std::int64_t stridea,
                          T beta,
                          T *c,
                          std::int64_t ldc,
                          std::int64_t stridec,
                          std::int64_t batch_size,
                          compute_mode mode = compute_mode::unset,
                          const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
   sycl::event syrk_batch(sycl::queue &queue,
                          oneapi::mkl::uplo upper_lower,
                          oneapi::mkl::transpose trans,
                          std::int64_t n,
                          std::int64_t k,
                          T alpha,
                          const T *a,
                          std::int64_t lda,
                          std::int64_t stridea,
                          T beta,
                          T *c,
                          std::int64_t ldc,
                          std::int64_t stridec,
                          std::int64_t batch_size,
                          compute_mode mode = compute_mode::unset,
                          const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

upper_lower Specifies whether matrices `C` are upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), transposition operation applied to matrices `A`. Conjugation is never performed even if `trans` = `transpose::conjtrans`. See Data Types for more details.

n Number of rows and columns of matrices `C`. Must be at least zero.

k Number of columns of matrices op(A). Must be at least zero.

alpha Scaling factor for rank-k update.

a Pointer to input matrices A. Size of the array must be at least `stridea * batch_size`.

lda Leading dimension of matrices A. Must be positive.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least n	Must be at least k
Row major	Must be at least k	Must be at least n

stridea Stride between two consecutive A matrices.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `lda * k`	Must be at least `lda * n`
Row major	Must be at least `lda * n`	Must be at least `lda * k`

beta Scaling factor for matrices C.

c Pointer to input/output matrices C. Size of the array must be at least `stridec * batch_size`.

ldc Leading dimension of matrices C. Must be positive and at least n.

stridec Stride between two consecutive C matrices. Must be least `ldc * n`.

batch_size Specifies the number of matrix multiply operations to perform.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

c Pointer to output matrices C overwritten by `batch_size` `syrk` operations of the form `alpha * op(A) * op(A)`^{T} + `beta * C`.

Return Values

Output event to wait on to ensure computation is complete.

9.4.10 trsm_batch

Computes a group of `trsm` operations.

Description

The `trsm_batch` routines are batched versions of **trsm**, performing multiple `trsm` operations in a single call. Each `trsm` solves an equation of the form op(A) * X = alpha * B or X * op(A) = alpha * B.

`trsm_batch` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

trsm_batch (Buffer Version)

Buffer version of `trsm_batch` supports only strided API.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    A and B are matrices at offset i * stridea and i * strideb in a and b.
    if (left_right == side::left) then
       compute X such that op(A) * X = alpha * B
    else
       compute X such that X * op(A) = alpha * B
    B = X
end for
```

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- `alpha` is a scalar
- A is either `m` x `m` or `n` x `n` triangular matrix
- B and X are `m` x `n` general matrices

On return, matrix B is overwritten by solution matrix X.

For strided API, a and b buffers contains all the input matrices. The stride between matrices is given by the stride parameters. Total number of matrices in a and b buffers is given by `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void trsm_batch(sycl::queue &queue,
                    oneapi::mkl::side left_right,
                    oneapi::mkl::uplo upper_lower,
                    oneapi::mkl::transpose trans,
                    oneapi::mkl::diag unit_diag,
                    std::int64_t m,
                    std::int64_t n,
                    T alpha,
                    sycl::buffer<T,1> &a,
                    std::int64_t lda,
                    std::int64_t stridea,
                    sycl::buffer<T,1> &b,
                    std::int64_t ldb,
                    std::int64_t strideb,
                    std::int64_t batch_size,
                    compute_mode mode = compute_mode::unset)
}
```

```
namespace oneapi::mkl::blas::row_major {
    void trsm_batch(sycl::queue &queue,
                    oneapi::mkl::side left_right,
                    oneapi::mkl::uplo upper_lower,
                    oneapi::mkl::transpose trans,
                    oneapi::mkl::diag unit_diag,
                    std::int64_t m,
                    std::int64_t n,
                    T alpha,
                    sycl::buffer<T,1> &a,
                    std::int64_t lda,
                    std::int64_t stridea,
                    sycl::buffer<T,1> &b,
                    std::int64_t ldb,
                    std::int64_t strideb,
                    std::int64_t batch_size,
                    compute_mode mode = compute_mode::unset)
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies whether matrices A are on the left side or right side of the multiplication. See Data Types for more details.

upper_lower Specifies whether matrices A are upper or lower triangular. See Data Types for more details.

trans Specifies op(A), transposition operation applied to matrices A. See Data Types for more details.

unit_diag Specifies whether matrices A are unit triangular or not. See Data Types for more details.

m Number of rows of matrices B. Must be at least zero.

n Number of columns of matrices B. Must be at least zero.

alpha Scaling factor for the solution.

a Buffer holding input matricees A. Size of the buffer must be at least `stridea` * `batch_size`.

lda Leading dimension of matrices A. Must be at least `m` if `left_right` = `side::left` or at least `n` if `left_right` = `side::right`. Must be positive.

stridea Stride between two consecutive A matrices.

b Buffer holding input/output matrices B. Size of the buffer must be at least `strideb` * `batch_size`.

ldb Leading dimension of matrices B. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

strideb Stride between two consecutive B matrices.

batch_size Specifies number of triangular linear systems to solve.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

Output Parameters

b Output buffer overwritten by `batch_size` solution matrices X.

Note: If `alpha` = 0, matrices B are set to zero, and A and B do not need to be initialized before calling `trsm_batch`..

---

trsm_batch (USM Version)

USM version of `trsm_batch` supports group API and strided API.

Group API

Group API operation is defined as:

```
idx = 0
for i = 0 … group_count – 1
    for j = 0 … group_size – 1
        A and B are matrices in a[idx] and b[idx]
        if (left_right == side::left) then
            compute X such that op(A) * X = alpha[i] * B
        else
            compute X such that X * op(A) = alpha[i] * B
        end if
        B = X
        idx = idx + 1
    end for
end for
```

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- alpha is a scalar
- A is either m x m or n x n triangular matrix
- B and X are m x n general matrices

On return, matrix B is overwritten by solution matrix X.

For group API, a and b arrays contain the pointers for all the input matrices. The total number of matrices in a and b are given by:

$$total\_batch\_count = \sum_{i=0}^{group\_count-1} group\_size[i]$$

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event trsm_batch(sycl::queue &queue,
                           const oneapi::mkl::side *left_right,
                           const oneapi::mkl::uplo *upper_lower,
                           const oneapi::mkl::transpose *trans,
                           const oneapi::mkl::diag *unit_diag,
                           const std::int64_t *m,
                           const std::int64_t *n,
                           const T *alpha,
                           const T **a,
                           const std::int64_t *lda,
                           T **b,
```

(continues on next page)

(continued from previous page)

```
                            const std::int64_t *ldb,
                            std::int64_t group_count,
                            const std::int64_t *group_size,
                            compute_mode mode = compute_mode::unset,
                            const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event trsm_batch(sycl::queue &queue,
                            const oneapi::mkl::side *left_right,
                            const oneapi::mkl::uplo *upper_lower,
                            const oneapi::mkl::transpose *trans,
                            const oneapi::mkl::diag *unit_diag,
                            const std::int64_t *m,
                            const std::int64_t *n,
                            const T *alpha,
                            const T **a,
                            const std::int64_t *lda,
                            T **b,
                            const std::int64_t *ldb,
                            std::int64_t group_count,
                            const std::int64_t *group_size,
                            compute_mode mode = compute_mode::unset,
                            const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Array of `group_count` `oneapi::mkl::side` values. `left_right[i]` specifies whether matrices `A` are on the left side or right side of the multiplication in group `i`. See Data Types for more details.

upper_lower Array of `group_count` `oneapi::mkl::uplo` values. `upper_lower[i]` specifies whether matrices `A` are upper or lower triangular in group `i`. See Data Types for more details.

trans Array of `group_count` `oneapi::mkl::transpose` values. `trans[i]` specifies op(`A`), transposition operation applied to matrices `A` in each group `i`. See Data Types for more details.

unit_diag Array of `group_count` `oneapi::mkl::diag` values. `unit_diag[i]` specifies whether matrices `A` are unit triangular or not. See Data Types for more details.

m Array of `group_count` integers. `m[i]` specifies number of rows of matrices `B` in group `i`. All entries must be at least zero.

n Array of `group_count` integers. `n[i]` specifies number of columns of matrices `B` in group `i`. All entries must be at least zero.

alpha Array of `group_count` scalar elements. `alpha[i]` specifies scaling factors for the solutions in group `i`.

a Array of `total_batch_count` pointers for input matrices `A`. See Matrix Storage for more details.

lda Array of `group_count` integers. `lda[i]` specifies leading dimension of matrices `A` in group `i`. Must be at least `m[i]` if `left_right[i]` = `side::left` or at least `n[i]` if `left_right[i]` = `side::right`. All entries must be positive.

b Array of `total_batch_count` pointers for input/output matrices `B`. See **Matrix Storage** for more details.

ldb Array of `group_count` integers. `ldb[i]` specifies leading dimension of matrices `B` in group `i`. Must be at least `m[i]` if column major layout or at least `n[i]` if row major layout is used. All entries must be positive.

group_count Number of groups. Must be at least zero.

group_size Array of `group_count` integers. `group_size[i]` specifies the number of `trsm` operations in group `i`. Each element in `group_size` must be at least zero.

mode **Optional.** Compute mode settings. See **Compute Modes** for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

b Array of pointers to output matrices `B` overwritten by `total_batch_count` solution matrices `X`.

Note: If `alpha` = 0, matrices `B` are set to zero, and `A` and `B` do not need to be initialized before calling `trsm_batch`..

Return Values

Output event to wait on to ensure computation is complete.

Strided API

Strided API operation is defined as:

```
for i = 0 … batch_size – 1
    A and B are matrices at offset i * stridea and i * strideb in a and b.
    if (left_right == side::left) then
       compute X such that op(A) * X = alpha * B
    else
       compute X such that X * op(A) = alpha * B
    B = X
end for
```

where:

- op(A) is one of op(A) = A, or op(A) = A^T, or op(A) = A^H
- alpha is a scalar
- A is either m x m or n x n triangular matrix
- B and X are m x n general matrices

On return, matrix B is overwritten by solution matrix X.

For strided API, a and b arrays contain all the input matrices. The stride between matrices is given by the stride parameters. Total number of matrices in a and b arrays is given by batch_size parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event trsm_batch(sycl::queue &queue,
                           oneapi::mkl::side left_right,
                           oneapi::mkl::uplo upper_lower,
                           oneapi::mkl::transpose trans,
                           oneapi::mkl::diag unit_diag,
                           std::int64_t m,
                           std::int64_t n,
                           T alpha,
                           const T *a,
                           std::int64_t lda,
                           std::int64_t stridea,
                           T *b,
                           std::int64_t ldb,
                           std::int64_t strideb,
                           std::int64_t batch_size,
                           compute_mode mode = compute_mode::unset,
                           const std::vector<sycl::event> &dependencies = {})
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event trsm_batch(sycl::queue &queue,
                           oneapi::mkl::side left_right,
                           oneapi::mkl::uplo upper_lower,
                           oneapi::mkl::transpose trans,
                           oneapi::mkl::diag unit_diag,
                           std::int64_t m,
                           std::int64_t n,
                           T alpha,
                           const T *a,
                           std::int64_t lda,
                           std::int64_t stridea,
                           T *b,
                           std::int64_t ldb,
                           std::int64_t strideb,
                           std::int64_t batch_size,
```

(continues on next page)

(continued from previous page)

```
                            compute_mode mode = compute_mode::unset,
                            const std::vector<sycl::event> &dependencies = {})
}
```

Input Parameters

queue The queue where the routine should be executed.

left_right Specifies whether matrices `A` are on the left side or right side of the multiplication. See Data Types for more details.

upper_lower Specifies whether matrices `A` are upper or lower triangular. See Data Types for more details.

trans Specifies op(`A`), transposition operation applied to matrices `A`. See Data Types for more details.

unit_diag Specifies whether matrices `A` are unit triangular or not. See Data Types for more details.

m Number of rows of matrices `B`. Must be at least zero.

n Number of columns of matrices `B`. Must be at least zero.

alpha Scaling factor for the solution.

a Pointer to input matricees `A`. Size of the array must be at least `stridea` * `batch_size`.

lda Leading dimension of matrices `A`. Must be at least `m` if `left_right` = `side::left` or at least `n` if `left_right` = `side::right`. Must be positive.

stridea Stride between two consecutive `A` matrices.

b Pointer to input/output matrices `B`. Size of the array must be at least `strideb` * `batch_size`.

ldb Leading dimension of matrices `B`. Must be at least `m` if column major layout or at least `n` if row major layout is used. Must be positive.

strideb Stride between two consecutive `B` matrices.

batch_size Specifies number of triangular linear systems to solve.

mode **Optional.** Compute mode settings. See Compute Modes for more details.

dependencies **Optional.** List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

`mode` and `dependencies` may be omitted independently; it is not necessary to specify `mode` in order to provide dependencies.

Output Parameters

b Pointer to output matrix B overwritten by `batch_size` solution matrices X.

Note: If `alpha` = 0, matrices B are set to zero, and A and B do not need to be initialized before calling `trsm_batch`..

Return Values

Output event to wait on to ensure computation is complete.

9.4.11 omatcopy

Computes an out-of-place scaled matrix transpose or copy operation using a general matrix.

Description

The `omatcopy` routine performs an out-of-place scaled matrix copy or transposition. The operation is defined as:

$$B \leftarrow alpha * op(A)$$

where:

- op(X) is one of op(X) = X, or op(X) = X^T, or op(X) = X^H
- `alpha` is a scalar
- A and B are matrices
- A is `m` x `n` matrix
- B is `m` x `n` matrix if `op` is non-transpose and an `n` x `m` matrix otherwise.

`omatcopy` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std::complex<double>`

omatcopy (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void omatcopy(sycl::queue &queue,
                  oneapi::mkl::transpose trans,
                  std::int64_t m,
                  std::int64_t n,
                  T alpha,
                  sycl::buffer<T, 1> &a,
                  std::int64_t lda,
                  sycl::buffer<T, 1> &b,
                  std::int64_t ldb);
}
```

```
namespace oneapi::mkl::blas::row_major {
    void omatcopy(sycl::queue &queue,
                  oneapi::mkl::transpose trans,
                  std::int64_t m,
                  std::int64_t n,
                  T alpha,
                  sycl::buffer<T, 1> &a,
                  std::int64_t lda,
                  sycl::buffer<T, 1> &b,
                  std::int64_t ldb);
}
```

Input Parameters

queue The queue where the routine will be executed.

trans Specifies op(A), the transposition operation applied to the matrix A. See Data Types for more details.

m Number of rows for the matrix A. Must be at least zero.

n Number of columns for the matrix A. Must be at least zero.

alpha Scaling factor for the matrix transposition or copy.

a Buffer holding the input matrix A. Must have size at least `lda` * `n` for column-major and at least `lda` * `m` for row-major.

lda Leading dimension of the matrix A. If matrices are stored using column major layout, `lda` must be at least `m`. If matrices are stored using row major layout, `lda` must be at least `n`. Must be positive.

ldb Leading dimension of the matrix B. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

Output Parameters

b Output buffer, overwritten by `alpha` * op(`A`).

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	`B` is `m` x `n` matrix. Size of array `b` must be at least `ldb` * `n`	`B` is `n` x `m` matrix. Size of array `a` must be at least `ldb` * `m`
Row major	`B` is `m` x `n` matrix. Size of array `b` must be at least `ldb` * `m`	`B` is `n` x `m` matrix. Size of array `b` must be at least `ldb` * `n`

omatcopy (USM Version)

Syntax

USM arrays:

```
namespace oneapi::mkl::blas::column_major {
    sycl::event omatcopy(sycl::queue &queue,
                         oneapi::mkl::transpose trans,
                         std::int64_t m,
                         std::int64_t n,
                         T alpha,
                         const T *a,
                         std::int64_t lda,
                         T *b,
                         std::int64_t ldb,
                         const std::vector<sycl::event> &dependencies = {});
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event omatcopy(sycl::queue &queue,
                         oneapi::mkl::transpose trans,
                         std::int64_t m,
                         std::int64_t n,
                         T alpha,
                         const T *a,
                         std::int64_t lda,
                         T *b,
                         std::int64_t ldb,
                         const std::vector<sycl::event> &dependencies = {});
}
```

Input Parameters

queue The queue where the routine will be executed.

trans Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

m Number of rows for the matrix A. Must be at least zero.

n Number of columns for the matrix A. Must be at least zero.

alpha Scaling factor for the matrix transposition or copy.

a Pointer to input matrix A. Must have size at least `lda` * n for column-major and at least `lda` * `m` for row-major.

lda Leading dimension of the matrix A. If matrices are stored using column major layout, `lda` must be at least `m`. If matrices are stored using row major layout, `lda` must be at least n. Must be positive.

ldb Leading dimension of the matrix B. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least n
Row major	Must be at least n	Must be at least `m`

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

b Pointer to output matrix B overwritten by `alpha` * op(A).

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	B is `m` x n matrix. Size of array b must be at least `ldb` * n	B is n x `m` matrix. Size of array a must be at least `ldb` * `m`
Row major	B is `m` x n matrix. Size of array b must be at least `ldb` * `m`	B is n x `m` matrix. Size of array b must be at least `ldb` * n

Return Values

Output event to wait for to ensure computation is complete.

9.4.12 imatcopy

Computes an in-place scaled matrix transpose or copy operation using a general matrix.

Description

The `imatcopy` routine performs an in-place scaled matrix copy or transposition. The operation is defined as:

$$AB \leftarrow alpha * op(AB)$$

where:

- op(X) is one of op(X) = X, or op(X) = X^T, or op(X) = X^H
- `alpha` is a scalar
- AB is a matrix
- AB is `m` x `n` on input.

`imatcopy` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

imatcopy (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void imatcopy(sycl::queue &queue,
                  oneapi::mkl::transpose trans,
                  std::int64_t m,
                  std::int64_t n,
                  T alpha,
                  sycl::buffer<T, 1> &ab,
                  std::int64_t lda,
                  std::int64_t ldb);
}
```

```
namespace oneapi::mkl::blas::row_major {
    void imatcopy(sycl::queue &queue,
                  oneapi::mkl::transpose trans,
                  std::int64_t m,
                  std::int64_t n,
                  T alpha,
                  sycl::buffer<T, 1> &ab,
                  std::int64_t lda,
                  std::int64_t ldb);
}
```

Input Parameters

queue The queue where the routine will be executed.

trans Specifies op(AB), the transposition operation applied to the matrix AB. See Data Types for more details.

m Number of rows for the matrix AB on input. Must be at least zero.

n Number of columns for the matrix AB on input. Must be at least zero.

alpha Scaling factor for the matrix transposition or copy.

ab Buffer holding the input/output matrix AB. Must have size as follows:

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Size of array `ab` must be at least `max(lda, ldb) * n`	Size of array `ab` must be at least `max(lda, ldb) * max(m, n)`
Row major	Size of array `ab` must be at least `max(lda, ldb) * m`	Size of array `ab` must be at least `max(lda, ldb) * max(m, n)`

lda Leading dimension of the matrix AB on input. If matrices are stored using column major layout, `lda` must be at least `m`. If matrices are stored using row major layout, `lda` must be at least `n`. Must be positive.

ldb Leading dimension of the matrix AB on output. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

Output Parameters

ab Output buffer, overwritten by `alpha` * op(AB).

imatcopy (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event imatcopy(sycl::queue &queue,
                         oneapi::mkl::transpose trans,
                         std::int64_t m,
                         std::int64_t n,
```

(continues on next page)

(continued from previous page)

```
                        T alpha,
                        T *ab,
                        std::int64_t lda,
                        std::int64_t ldb,
                        const std::vector<sycl::event> &dependencies = {});
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event imatcopy(sycl::queue &queue,
                        oneapi::mkl::transpose trans,
                        std::int64_t m,
                        std::int64_t n,
                        T alpha,
                        T *ab,
                        std::int64_t lda,
                        std::int64_t ldb,
                        const std::vector<sycl::event> &dependencies = {});
}
```

Input Parameters

queue The queue where the routine will be executed.

trans Specifies op(AB), the transposition operation applied to the matrix AB. See Data Types for more details.

m Number of rows for the matrix AB on input. Must be at least zero.

n Number of columns for the matrix AB on input. Must be at least zero.

alpha Scaling factor for the matrix transposition or copy.

ab Pointer to input/output matrix AB. Must have size as follows:

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Size of array ab must be at least max(lda, ldb) * n	Size of array ab must be at least max(lda, ldb) * max(m, n)
Row major	Size of array ab must be at least max(lda, ldb) * m	Size of array ab must be at least max(lda, ldb) * max(m, n)

lda Leading dimension of the matrix AB on input. If matrices are stored using column major layout, lda must be at least m. If matrices are stored using row major layout, lda must be at least n. Must be positive.

ldb Leading dimension of the matrix AB on output. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

ab Pointer to output matrix `AB` overwritten by `alpha` * op(`AB`).

Return Values

Output event to wait for to ensure computation is complete.

9.4.13 omatadd

Computes a sum of two general matrices, with optional transposes.

Description

The `omatadd` routine performs an out-of-place scaled matrix addition with optional transposes in the arguments. The operation is defined as:

$$C \leftarrow alpha * op(A) + beta * op(B)$$

where:

- op(`X`) is one of op(`X`) = `X`, or op(`X`) = X^T, or op(`X`) = X^H
- `alpha` and `beta` are scalars
- `A`, `B` and `C` are matrices
- op(`A`) is `m` x `n` matrix
- op(`B`) is `m` x `n` matrix
- `C` is `m` x `n` matrix

In general, `A`, `B`, and `C` must not overlap in memory, with the exception of the following in-place operations:

- `A` and `C` may point to the same memory if `op(A)` is non-transpose and `lda` = `ldc`;
- `B` and `C` may point to the same memory if `op(B)` is non-transpose and `ldb` = `ldc`.

`omatadd` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

omatadd (Buffer Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void omatadd(sycl::queue &queue,
                 oneapi::mkl::transpose transa,
                 oneapi::mkl::transpose transb,
                 std::int64_t m,
                 std::int64_t n,
                 T alpha,
                 sycl::buffer<T, 1> &a,
                 std::int64_t lda,
                 T beta,
                 sycl::buffer<T, 1> &b,
                 std::int64_t ldb,
                 sycl::buffer<T, 1> &c,
                 std::int64_t ldc)
 }
```

```
namespace oneapi::mkl::blas::row_major {
    void omatadd(sycl::queue &queue,
                 oneapi::mkl::transpose transa,
                 oneapi::mkl::transpose transb,
                 std::int64_t m,
                 std::int64_t n,
                 T alpha,
                 sycl::buffer<T, 1> &a,
                 std::int64_t lda,
                 T beta,
                 sycl::buffer<T, 1> &b,
                 std::int64_t ldb,
                 sycl::buffer<T, 1> &c,
                 std::int64_t ldc)
 }
```

Input Parameters

queue The queue where the routine will be executed.

transa Specifies op(A), the transposition operation applied to the matrix A.

transb Specifies op(B), the transposition operation applied to the matrix B.

m Number of rows for the result matrix C. Must be at least zero.

n Number of columns for the result matrix C. Must be at least zero.

alpha Scaling factor for the matrix A.

a Buffer holding the input matrix A.

	transa = transpose::nontrans	transa = transpose::trans **or** transa = transpose::conjtrans
Column major	A is m x n matrix. Size of array a must be at least lda * n	A is n x m matrix. Size of array a must be at least lda * m
Row major	A is m x n matrix. Size of array a must be at least lda * m	A is n x m matrix. Size of array a must be at least lda * n

lda Leading dimension of matrix A. Must be positive

	transa = transpose::nontrans	transa = transpose::trans **or** transa = transpose::conjtrans
Column major	Must be at least m	Must be at least n
Row major	Must be at least n	Must be at least m

beta Scaling factor for the matrix B.

b Buffer holding the input matrix B.

	transb = transpose::nontrans	transb = transpose::trans **or** transb = transpose::conjtrans
Column major	B is m x n matrix. Size of array b must be at least ldb * n	B is n x m matrix. Size of array b must be at least ldb * m
Row major	B is m x n matrix. Size of array b must be at least ldb * m	B is n x m matrix. Size of array b must be at least ldb * n

ldb Leading dimension of marix B. Must be positive.

	transb = transpose::nontrans	transb = transpose::trans or transb = transpose::conjtrans
Column major	Must be at least m	Must be at least n
Row major	Must be at least n	Must be at least m

ldc Leading dimension of the matrix C. If matrices are stored using column major layout, ldc must be at least m. If matrices are stored using row major layout, ldc must be at least n. Must be positive.

Output Parameters

c Output buffer overwritten by alpha * op(A) + beta * op(B).

Column major	C is m x n matrix. Size of array c must be at least ldc * n
Row major	C is m x n matrix. Size of array c must be at least ldc * m

omatadd (USM Version)

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event omatadd(sycl::queue &queue,
                        oneapi::mkl::transpose transa,
                        oneapi::mkl::transpose transb,
                        std::int64_t m,
                        std::int64_t n,
                        T alpha,
                        const T *a,
                        std::int64_t lda,
                        T beta,
                        const T *b,
                        std::int64_t ldb,
                        T *c,
                        std::int64_t ldc,
                        const std::vector<sycl::event> &dependencies = {});
 }
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event omatadd(sycl::queue &queue,
                        oneapi::mkl::transpose transa,
                        oneapi::mkl::transpose transb,
                        std::int64_t m,
                        std::int64_t n,
                        T alpha,
                        const T *a,
```

(continues on next page)

(continued from previous page)

```
                        std::int64_t lda,
                        T beta,
                        const T *b,
                        std::int64_t ldb,
                        T *c,
                        std::int64_t ldc,
                        const std::vector<sycl::event> &dependencies = {});
}
```

Input Parameters

queue The queue where the routine will be executed.

transa Specifies op(A), the transposition operation applied to matrix A. See Data Types for more details.

transb Specifies op(B), the transposition operation applied to matrix B. See Data Types for more details.

m Number of rows for the result matrix C. Must be at least zero.

n Number of columns for the result matrix C. Must be at least zero.

alpha Scaling factor for the matrix A.

a Pointer to input matrix A. See Matrix Storage for more details.

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `transa = transpose::conjtrans`
Column major	A is `m` x `n` matrix. Size of array a must be at least `lda * n`	A is `n` x `m` matrix. Size of array a must be at least `lda * m`
Row major	A is `m` x `n` matrix. Size of array a must be at least `lda * m`	A is `n` x `m` matrix. Size of array a must be at least `lda * n`

lda Leading dimension of matrix A. Must be positive

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `transa = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

beta Scaling factor for the matrix B.

b Pointer to input matrix B. See Matrix Storage for more details.

	`transb = transpose::nontrans`	`transb = transpose::trans` **or** `transb = transpose::conjtrans`
Column major	`B` is `m` x `n` matrix. Size of array `b` must be at least `ldb * n`	`B` is `n` x `m` matrix. Size of array `b` must be at least `ldb * m`
Row major	`B` is `m` x `n` matrix. Size of array `b` must be at least `ldb * m`	`B` is `n` x `m` matrix. Size of array `b` must be at least `ldb * n`

ldb Leading dimension of marix `B`. Must be positive.

	`transb = transpose::nontrans`	`transb = transpose::trans` **or** `transb = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

ldc Leading dimension of the matrix `C`. If matrices are stored using column major layout, `ldc` must be at least `m`. If matrices are stored using row major layout, `ldc` must be at least `n`. Must be positive.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

c Pointer to output matrix overwritten by `alpha` * op(`A`) + `beta` * op(`B`).

Column major	`C` is `m` x `n` matrix. Size of array `c` must be at least `ldc * n`
Row major	`C` is `m` x `n` matrix. Size of array `c` must be at least `ldc * m`

Return Values

Output event to wait for to ensure computation is complete.

9.4.14 omatcopy_batch

Computes a group of out-of-place scaled matrix transpose or copy operations using general matrices.

Description

The `omatcopy_batch` routines perform a series of out-of-place scaled matrix copies or transpositions. They are similar to the `omatcopy` routines, but the `omatcopy_batch` routines perform matrix operations with a group of matrices.

`omatcopy_batch` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std::complex<double>`

omatcopy_batch (Buffer Version)

Buffer version of `omatcopy_batch` supports only strided API.

Strided API

The operation for the strided API is defined as:

```
for i = 0 … batch_size – 1
    A and B are matrices at offset i * stride_a in a and i * stride_b in b
    B = alpha * op(A)
end for
```

where:

- `op(X)` is one of `op(X) = X`, `op(X) = X'`, or `op(X) = conjg(X')`
- `alpha` is a scalar
- `A` and `B` are matrices

For the strided API, the single input buffer contains all the input matrices, and the single output buffer contains all the output matrices. The locations of the individual matrices within the buffer are given by stride lengths, while the number of matrices is given by the `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void omatcopy_batch(sycl::queue &queue,
                        transpose trans,
                        std::int64_t m,
                        std::int64_t n,
                        T alpha,
                        sycl::buffer<T, 1> &a,
                        std::int64_t lda,
                        std::int64_t stride_a,
                        sycl::buffer<T, 1> &b,
                        std::int64_t ldb,
                        std::int64_t stride_b,
                        std::int64_t batch_size);
 }
```

```
namespace oneapi::mkl::blas::row_major {
    void omatcopy_batch(sycl::queue &queue,
                        transpose trans,
                        std::int64_t m,
                        std::int64_t n,
                        T alpha,
                        sycl::buffer<T, 1> &a,
                        std::int64_t lda,
                        std::int64_t stride_a,
                        sycl::buffer<T, 1> &b,
                        std::int64_t ldb,
                        std::int64_t stride_b,
                        std::int64_t batch_size);
 }
```

Input Parameters

queue The queue where the routine should be executed.

trans Specifies `op(A)`, the transposition operation applied to the matrices `A`.

m Number of rows for each matrix `A`. Must be at least zero.

n Number of columns for each matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix transposition or copy.

a Buffer holding the input matrices `A`. Must have size at least `stride_a*batch_size`.

lda Leading dimension of the A matrices. If matrices are stored using column major layout, `lda` must be at least `m`. If matrices are stored using row major layout, `lda` must be at least `n`. Must be positive.

stride_a Stride between the different `A` matrices. If matrices are stored using column major layout, `stride_a` must be at least `lda*n`. If matrices are stored using row major layout, `stride_a` must be at least `lda*m`.

ldb Leading dimension of the matrices B. Must be positive and satisfy:

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

stride_b Stride between the different B matrices in the buffer b. Must be positive and satisfy:

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `ldb*n`	Must be at least `ldb*m`
Row major	Must be at least `ldb*m`	Must be at least `ldb*n`

batch_size Specifies the number of matrices to transpose or copy. Must be at least zero.

Output Parameters

b Output buffer, overwritten by `batch_size` matrix transpose or copy operations of the form `alpha*op(A)`. Must have size at least `stride_b*batch_size`.

omatcopy_batch (USM Version)

USM version of `omatcopy_batch` supports group API and strided API.

Group API

The operation for the group API is defined as:

```
idx = 0
for i = 0 … group_count – 1
    m, n, alpha, lda, ldb and group_size at position i in their respective arrays
    for j = 0 … group_size – 1
        A and B are matrices at position idx in their respective arrays
        B = alpha * op(A)
        idx := idx + 1
    end for
end for
```

where:

- `op(X)` is one of `op(X) = X`, `op(X) = X'`, or `op(X) = conjg(X')`
- `alpha` is a scalar
- `A` and `B` are matrices

For the group API, the matrices are given by arrays of pointers. `A` and `B` represent matrices stored at addresses pointed to by `a` and `b` respectively. The total number of entries in `a` and `b` are given by:

$$total\_batch\_count = \sum_{i=0}^{group\_count-1} group\_size[i]$$

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event omatcopy_batch(sycl::queue &queue,
                               const transpose *trans,
                               const std::int64_t *m,
                               const std::int64_t *n,
                               const T *alpha,
                               const T **a,
                               const std::int64_t *lda,
                               T **b,
                               const std::int64_t *ldb,
                               std::int64_t group_count,
                               const std::int64_t *groupsize,
                               const std::vector<sycl::event> &dependencies = {});
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event omatcopy_batch(sycl::queue &queue,
                               const transpose *trans,
                               const std::int64_t *m,
                               const std::int64_t *n,
                               const T *alpha,
                               const T **a,
                               const std::int64_t *lda,
                               T **b,
                               const std::int64_t *ldb,
                               std::int64_t group_count,
                               const std::int64_t *groupsize,
                               const std::vector<sycl::event> &dependencies = {});
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Array of size `group_count`. Each element `i` in the array specifies `op(A)` the transposition operation applied to the matrices `A`.

m Array of `group_count` integers. `m[i]` specifies the number of rows of `A[i]`. Each entry must be at least zero.

n Array of `group_count` integers. `n[i]` specifies the number of columns of `A[i]`. Each entry must be at least zero.

alpha Array of size `group_count` containing scaling factors for the operation.

a Array of size `total_batch_count` of pointers to `A` matrices. If matrices are stored in column major layout, the array allocated for each `A` matrix of the group `i` must be of size at least `lda[i] * n[i]`. If matrices are stored in row major layout, the array allocated for each `A` matrix of the group `i` must be of size at least `lda[i]*m[i]`.

lda Array of `group_count` integers. `lda[i]` specifies the leading dimension of the `A[i]` matrix. If matrices are stored using column major layout, `lda[i]` must be at least `m[i]`. If matrices are stored using row major layout, `lda[i]` must be at least `n[i]`. Each must be positive.

ldb Array of `group_count` integers. `ldb[i]` specifies the leading dimension of the `B[i]` matrix. Each `ldb[i]` must be positive and satisfy:

	`trans[i] = transpose::nontrans`	`trans[i] = transpose::trans` **or** `trans[i] = transpose::conjtrans`
Column major	Must be at least `m[i]`	Must be at least `n[i]`
Row major	Must be at least `n[i]`	Must be at least `m[i]`

group_count Number of groups. Must be at least 0.

group_size Array of size `group_count`. The element ``group_size[i]` is the number of matrices in the group `i`. Each element in `group_size` must be at least 0.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

b Output array of pointers to `B` matrices, overwritten by `total_batch_count` matrix transpose or copy operations of the form `alpha*op(A)`. If matrices are stored using column major layout, the array allocated for each `B` matrix of the group `i` must be of size at least `ldb[i] * n[i]` if `B` is not transposed or `ldb[i]*m[i]` if `B` is transposed. If matrices are stored using row major layout, the array allocated for each `B` matrix of the group `i` must be of size at least `ldb[i] * m[i]` if `B` is not transposed or `ldb[i]*n[i]` if `B` is transposed.

Return Values

Output event to wait on to ensure computation is complete.

Strided API

The operation for the strided API is defined as:

```
for i = 0 … batch_size – 1
    A and B are matrices at offset i * stride_a in a and i * stride_b in b
    B = alpha * op(A)
end for
```

where:

- `op(X)` is one of `op(X) = X`, `op(X) = X'`, or `op(X) = conjg(X')`
- `alpha` is a scalar
- `A` and `B` are matrices

For the strided API, the single input array contains all the input matrices, and the single output array contains all the output matrices. The locations of the individual matrices within the array are given by stride lengths, while the number of matrices is given by the `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event omatcopy_batch(sycl::queue &queue,
                               transpose trans,
                               std::int64_t m,
                               std::int64_t n,
                               T alpha,
                               const T *a,
                               std::int64_t lda,
                               std::int64_t stride_a,
                               T *b,
                               std::int64_t ldb,
                               std::int64_t stride_b,
                               std::int64_t batch_size,
                               const std::vector<sycl::event> &dependencies = {});
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event omatcopy_batch(sycl::queue &queue,
                               transpose trans,
                               std::int64_t m,
                               std::int64_t n,
```

(continues on next page)

(continued from previous page)

```
                                T alpha,
                                const T *a,
                                std::int64_t lda,
                                std::int64_t stride_a,
                                T *b,
                                std::int64_t ldb,
                                std::int64_t stride_b,
                                std::int64_t batch_size,
                                const std::vector<sycl::event> &dependencies = {});
}
```

Input Parameters

trans Specifies `op(A)`, the transposition operation applied to the matrices `A`.

m Number of rows for each matrix `A`. Must be at least zero.

n Number of columns for each matrix `A`. Must be at least zero.

alpha Scaling factor for the matrix transposition or copy.

a Array holding the input matrices `A`. Must have size at least `stride_a*batch_size`.

lda Leading dimension of the `A` matrices. If matrices are stored using column major layout, `lda` must be at least `m`. If matrices are stored using row major layout, `lda` must be at least `n`. Must be positive.

stride_a Stride between the different `A` matrices. If matrices are stored using column major layout, `stride_a` must be at least `lda*n`. If matrices are stored using row major layout, `stride_a` must be at least `lda*m`.

ldb Leading dimension of the matrices `B`. Must be positive and satisfy:

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

stride_b Stride between the different `B` matrices in the array `b`. Must be positive and satisfy:

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `ldb*n`	Must be at least `ldb*m`
Row major	Must be at least `ldb*m`	Must be at least `ldb*n`

batch_size Specifies the number of matrices to transpose or copy. Must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

b Output array, overwritten by `batch_size` matrix transpose or copy operations of the form `alpha*op(A)`. Must have size at least `stride_b*batch_size`.

Return Values

Output event to wait on to ensure computation is complete.

9.4.15 imatcopy_batch

Computes a group of in-place scaled matrix transpose or copy operations using general matrices.

Description

The `imatcopy_batch` routines perform a series of in-place scaled matrix copies or transpositions. They are similar to the `imatcopy` routines, but the `imatcopy_batch` routines perform their operations with groups of matrices. The groups contain matrices with the same parameters.

`imatcopy_batch` supports the following precisions:

T
float
double
std::complex<float>
std::complex<double>

imatcopy_batch (Buffer Version)

Buffer version of `imatcopy_batch` supports only strided API.

Strided API

The operation for the strided API is defined as:

```
for i = 0 … batch_size – 1
    AB is a matrix at offset i * stride in ab
    AB = alpha * op(AB)
end for
```

where:

- `op(X)` is one of `op(X) = X`, `op(X) = X'`, or `op(X) = conjg(X')`

- `alpha` is a scalar
- `AB` is a matrix to be transformed in place

For the strided API, the single buffer `AB` contains all the matrices to be transformed in place. The locations of the individual matrices within the buffer are given by stride lengths, while the number of matrices is given by the `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void imatcopy_batch(sycl::queue &queue,
                        transpose trans,
                        std::int64_t m,
                        std::int64_t n,
                        T alpha,
                        sycl::buffer<T, 1> &ab,
                        std::int64_t lda,
                        std::int64_t ldb,
                        std::int64_t stride,
                        std::int64_t batch_size);
}
```

```
namespace oneapi::mkl::blas::row_major {
    void imatcopy_batch(sycl::queue &queue,
                        transpose trans,
                        std::int64_t m,
                        std::int64_t n,
                        T alpha,
                        sycl::buffer<T, 1> &ab,
                        std::int64_t lda,
                        std::int64_t ldb,
                        std::int64_t stride,
                        std::int64_t batch_size);
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Specifies `op(AB)`, the transposition operation applied to the matrices `AB`.

m Number of rows for each matrix `AB` on input. Must be at least 0.

n Number of columns for each matrix `AB` on input. Must be at least 0.

alpha Scaling factor for the matrix transpose or copy operation.

ab Buffer holding the matrices `AB`. Must have size at least `stride*batch_size`.

lda Leading dimension of the AB matrices on input. If matrices are stored using column major layout, `lda` must be at least `m`. If matrices are stored using row major layout, `lda` must be at least `n`. Must be positive.

ldb Leading dimension of the matrices AB on output. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

stride Stride between the different AB matrices. It must be at least `max(ldb,lda)*max(ka, kb)`, where:

- `ka` is `m` if column major layout is used or `n` if row major layout is used
- `kb` is `n` if column major layout is used and AB is not transposed, or `m` otherwise

batch_size Specifies the number of matrices to transpose or copy. Must be at least zero.

Output Parameters

ab Output buffer, overwritten by `batch_size` matrix multiply operations of the form `alpha*op(AB)`.

imatcopy_batch (USM Version)

USM version of `imatcopy_batch` supports group API and strided API.

Group API

The operation for the group API is defined as:

```
idx = 0
for i = 0 … group_count – 1
    m,n, alpha, lda, ldb and group_size at position i in their respective arrays
    for j = 0 … group_size – 1
        AB is a matrix at position idx in AB
        AB = alpha * op(AB)
        idx := idx + 1
    end for
end for
```

where:

- `op(X)` is one of `op(X) = X`, `op(X) = X'`, or `op(X) = conjg(X')`
- `alpha` is a scalar

- AB is a matrix to be transformed in place

For the group API, the matrices are given by arrays of pointers. AB represents a matrix stored at the address pointed to by ab. The total number of entries in ab is given by:

$$total\_batch\_count = \sum_{i=0}^{group\_count-1} group\_size[i]$$

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event imatcopy_batch(sycl::queue &queue,
                               const transpose *trans,
                               const std::int64_t *m,
                               const std::int64_t *n,
                               const T *alpha, T **ab,
                               const std::int64_t *lda,
                               const std::int64_t *ldb,
                               std::int64_t group_count,
                               const std::int64_t *groupsize,
                               const std::vector<sycl::event> &dependencies = {});
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event imatcopy_batch(sycl::queue &queue,
                               const transpose *trans,
                               const std::int64_t *m,
                               const std::int64_t *n,
                               const T *alpha, T **ab,
                               const std::int64_t *lda,
                               const std::int64_t *ldb,
                               std::int64_t group_count,
                               const std::int64_t *groupsize,
                               const std::vector<sycl::event> &dependencies = {});
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Array of size `group_count`. Each element `i` in the array specifies `op(AB)` the transposition operation applied to the matrices AB.

m Array of `group_count` integers. `m[i]` specifies the number of rows in `AB[i]` on input. Each entry must be at least zero.

n Array of `group_count` integers. `n[i]` specifies the number of columns in `AB[i]` on input. Each entry must be at least zero.

alpha Array of size `group_count` containing scaling factors for the matrix transpositions or copies.

ab Array of size `total_batch_count`, holding pointers to arrays used to store `AB` matrices.

lda Array of `group_count` integers. `lda[i]` specifies the leading dimension of the matrix input `AB`. If matrices are stored using column major layout, `lda[i]` must be at least `m[i]`. If matrices are stored using row major layout, `lda[i]` must be at least `n[i]`. Must be positive.

ldb Array of `group_count` integers. `ldb[i]` specifcies the leading dimension of the matrix `AB` on output. Each `ldb[i]` must be positive and satisfy:

	`trans[i] = transpose::nontrans`	`trans[i] = transpose::trans` **or** `trans[i] = transpose::conjtrans`
Column major	Must be at least `m[i]`	Must be at least `n[i]`
Row major	Must be at least `n[i]`	Must be at least `m[i]`

group_count Number of groups. Must be at least 0.

group_size Array of size `group_count`. The element `group_size[i]` is the number of matrices in the group `i`. Each element in `group_size` must be at least 0.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

ab Output array of pointers to `AB` matrices, overwritten by `total_batch_count` matrix transpose or copy operations of the form `alpha*op(AB)`.

Return Values

Output event to wait on to ensure computation is complete.

Strided API

The operation for the strided API is defined as:

```
for i = 0 … batch_size – 1
    AB is a matrix at offset i * stride in ab
    AB = alpha * op(AB)
end for
```

where:

- `op(X)` is one of `op(X) = X`, `op(X) = X'`, or `op(X) = conjg(X')`
- `alpha` is a scalar
- `AB` is a matrix to be transformed in place

For the strided API, the single array `ab` contains all the matrices `AB` to be transformed in place. The locations of the individual matrices within the array are given by stride lengths, while the number of matrices is given by the `batch_size` parameter.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event imatcopy_batch(sycl::queue &queue,
                               transpose trans,
                               std::int64_t m,
                               std::int64_t n,
                               T alpha,
                               T *ab,
                               std::int64_t lda,
                               std::int64_t ldb,
                               std::int64_t stride,
                               std::int64_t batch_size,
                               const std::vector<sycl::event> &dependencies = {});
}
```

```
namespace oneapi::mkl::blas::column_major {
    sycl::event imatcopy_batch(sycl::queue &queue,
                               transpose trans,
                               std::int64_t m,
                               std::int64_t n,
                               T alpha,
                               T *ab,
                               std::int64_t lda,
                               std::int64_t ldb,
                               std::int64_t stride,
                               std::int64_t batch_size,
                               const std::vector<sycl::event> &dependencies = {});
}
```

Input Parameters

queue The queue where the routine should be executed.

trans Specifies `op(AB)`, the transposition operation applied to the matrices `AB`.

m Number of rows for each matrix `AB` on input. Must be at least 0.

n Number of columns for each matrix `AB` on input. Must be at least 0.

alpha Scaling factor for the matrix transpose or copy operation.

ab Array holding the matrices AB. Must have size at least `stride*batch_size`.

lda Leading dimension of the AB matrices on input. If matrices are stored using column major layout, `lda` must be at least `m`. If matrices are stored using row major layout, `lda` must be at least `n`. Must be positive.

ldb Leading dimension of the matrices AB on output. Must be positive.

	`trans = transpose::nontrans`	`trans = transpose::trans` **or** `trans = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

stride Stride between the different AB matrices. It must be at least `max(ldb,lda)*max(ka, kb)`, where:

- `ka` is `m` if column major layout is used or `n` if row major layout is used
- `kb` is `n` if column major layout is used and AB is not transposed, or `m` otherwise

batch_size Specifies the number of matrices to transpose or copy. Must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

ab Output array, overwritten by `batch_size` matrix multiply operations of the form `alpha*op(AB)`.

Return Values

Output event to wait on to ensure computation is complete.

9.4.16 omatadd_batch

Computes a group of out-of-place scaled matrix additions using general matrices.

Description

The `omatadd_batch` routines perform a series of out-of-place scaled matrix additions. They are similar to the `omatadd` routines, but the `omatadd_batch` routines perform matrix operations with a group of matrices.

`omatadd_batch` supports the following precisions:

T
`float`
`double`
`std::complex<float>`
`std::complex<double>`

omatadd_batch (Buffer Version)

Strided API

The matrices are always in a strided format for `omatadd_batch`. The operation is defined as:

```
for i = 0 … batch_size – 1
    A is a matrix at offset i * stride_a in a
    B is a matrix at offset i * stride_b in b
    C is a matrix at offset i * stride_c in c
    C = alpha * op(A) + beta * op(B)
end for
```

where:

- `op(X)` is one of `op(X) = X`, `op(X) = X'`, or `op(X) = conjg(X')`
- `alpha` and `beta` are scalars
- `A`, `B` and `C` are matrices

The input buffers `a` and `b` contain all the input matrices, and the single output buffer `c` contains all the output matrices. The locations of the individual matrices within the buffer or array are given by stride lengths, while the number of matrices is given by the `batch_size` parameter.

In general, the `a`, `b`, and `c` buffers must not overlap in memory, with the exception of the following in-place operations:

- `a` and `c` may point to the same memory if `op(A)` is non-transpose and all the `A` matrices within `a` have the same parameters as all the respective `C` matrices within `c`;
- `b` and `c` may point to the same memory if `op(B)` is non-transpose and all the `B` matrices within `b` have the same parameters as all the respective `C` matrices within `c`.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    void omatadd_batch(sycl::queue &queue,
                       transpose transa,
                       transpose transb,
                       std::int64_t m,
                       std::int64_t n,
                       T alpha,
                       sycl::buffer<T, 1> &a,
                       std::int64_t lda,
                       std::int64_t stride_a,
                       T beta,
                       sycl::buffer<T, 1> &b,
                       std::int64_t ldb,
                       std::int64_t stride_b,
```

(continues on next page)

(continued from previous page)

```
                        sycl::buffer<T, 1> &c,
                        std::int64_t ldc,
                        std::int64_t stride_c,
                        std::int64_t batch_size);
}
```

```
namespace oneapi::mkl::blas::row_major {
    void omatadd_batch(sycl::queue &queue,
                        transpose transa,
                        transpose transb,
                        std::int64_t m,
                        std::int64_t n,
                        T alpha,
                        sycl::buffer<T, 1> &a,
                        std::int64_t lda,
                        std::int64_t stride_a,
                        T beta,
                        sycl::buffer<T, 1> &b,
                        std::int64_t ldb,
                        std::int64_t stride_b,
                        sycl::buffer<T, 1> &c,
                        std::int64_t ldc,
                        std::int64_t stride_c,
                        std::int64_t batch_size);
}
```

Input Parameters

queue The queue where the routine should be executed.

transa Specifies `op(A)`, the transposition operation applied to the matrices `A`.

transb Specifies `op(B)`, the transposition operation applied to the matrices `B`.

m Number of rows for the result matrix `C`. Must be at least zero.

n Number of columns for the result matrix `C`. Must be at least zero.

alpha Scaling factor for the matrices `A`.

a Buffer holding the input matrices `A`. Must have size at least `stride_a*batch_size`.

lda Leading dimension of the `A` matrices. Must be positive and satisfy:

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `transa = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

stride_a Stride between the different A matrices in the buffer a. Must be positive and satisfy:

	transa = transpose::nontrans	transa = transpose::trans **or** transa = transpose::conjtrans
Column major	Must be at least lda*n	Must be at least lda*m
Row major	Must be at least lda*m	Must be at least lda*n

beta Scaling factor for the matrices B.

b Buffer holding the input matrices B. Must have size at least stride_b*batch_size.

ldb Leading dimension of the B matrices. Must be positive and satisfy:

	transb = transpose::nontrans	transb = transpose::trans **or** transb = transpose::conjtrans
Column major	Must be at least m	Must be at least n
Row major	Must be at least n	Must be at least m

stride_b Stride between the different B matrices in the buffer b. Must be positive and satisfy:

	transb = transpose::nontrans	transb = transpose::trans **or** transb = transpose::conjtrans
Column major	Must be at least ldb*n	Must be at least ldb*m
Row major	Must be at least ldb*m	Must be at least ldb*n

ldc Leading dimension of the C matrices. If matrices are stored using column major layout, ldc must be at least m. If matrices are stored using row major layout, ldc must be at least n. Must be positive.

stride_c Stride between the different C matrices. If matrices are stored using column major layout, stride_c must be at least ldc*n. If matrices are stored using row major layout, stride_c must be at least ldc*m.

batch_size Specifies the number of input and output matrices to add. Must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

c Output buffer, overwritten by batch_size matrix addition operations of the form alpha*op(A) + beta*op(B). Must have size at least stride_c*batch_size.

omatadd_batch (USM Version)

Strided API

The matrices are always in a strided format for `omatadd`. The operation is defined as:

```
for i = 0 … batch_size – 1
    A is a matrix at offset i * stride_a in a
    B is a matrix at offset i * stride_b in b
    C is a matrix at offset i * stride_c in c
    C = alpha * op(A) + beta * op(B)
end for
```

where:

- `op(X)` is one of `op(X) = X`, `op(X) = X'`, or `op(X) = conjg(X')`
- `alpha` and `beta` are scalars
- `A`, `B` and `C` are matrices

The input buffers `a` and `b` contain all the input matrices, and the single output buffer `c` contains all the output matrices. The locations of the individual matrices within the buffer or array are given by stride lengths, while the number of matrices is given by the `batch_size` parameter.

In general, the `a`, `b`, and `c` buffers must not overlap in memory, with the exception of the following in-place operations:

- `a` and `c` may point to the same memory if `op(A)` is non-transpose and all the `A` matrices within `a` have the same parameters as all the respective `C` matrices within `c`;
- `b` and `c` may point to the same memory if `op(B)` is non-transpose and all the `B` matrices within `b` have the same parameters as all the respective `C` matrices within `c`.

Syntax

```
namespace oneapi::mkl::blas::column_major {
    sycl::event omatadd_batch(sycl::queue &queue,
                              transpose transa,
                              transpose transb,
                              std::int64_t m,
                              std::int64_t n,
                              T alpha,
                              const T *a,
                              std::int64_t lda,
                              std::int64_t stride_a,
                              T beta,
                              const T *b,
                              std::int64_t ldb,
                              std::int64_t stride_b,
```

(continues on next page)

(continued from previous page)

```
                              T *c,
                              std::int64_t ldc,
                              std::int64_t stride_c,
                              std::int64_t batch_size,
                              const std::vector<sycl::event> &dependencies = {});
}
```

```
namespace oneapi::mkl::blas::row_major {
    sycl::event omatadd_batch(sycl::queue &queue,
                              transpose transa,
                              transpose transb,
                              std::int64_t m,
                              std::int64_t n,
                              T alpha,
                              const T *a,
                              std::int64_t lda,
                              std::int64_t stride_a,
                              T beta,
                              const T *b,
                              std::int64_t ldb,
                              std::int64_t stride_b,
                              T *c,
                              std::int64_t ldc,
                              std::int64_t stride_c,
                              std::int64_t batch_size,
                              const std::vector<sycl::event> &dependencies = {});
}
```

Input Parameters

queue The queue where the routine should be executed.

transa Specifies `op(A)`, the transposition operation applied to the matrices `A`.

transb Specifies `op(B)`, the transposition operation applied to the matrices `B`.

m Number of rows for the result matrix `C`. Must be at least zero.

n Number of columns for the result matrix `C`. Must be at least zero.

alpha Scaling factor for the matrices `A`.

a Array holding the input matrices `A`. Must have size at least `stride_a*batch_size`.

lda Leading dimension of the `A` matrices. Must be positive and satisfy:

	`transa = transpose::nontrans`	`transa = transpose::trans` **or** `transa = transpose::conjtrans`
Column major	Must be at least `m`	Must be at least `n`
Row major	Must be at least `n`	Must be at least `m`

stride_a Stride between the different A matrices in the array a. Must be positive and satisfy:

	transa = transpose::nontrans	transa = transpose::trans **or** transa = transpose::conjtrans
Column major	Must be at least lda*n	Must be at least lda*m
Row major	Must be at least lda*m	Must be at least lda*n

beta Scaling factor for the matrices B.

b Array holding the input matrices B. Must have size at least stride_b*batch_size.

ldb Leading dimension of the B matrices. Must be positive and satisfy:

	transb = transpose::nontrans	transb = transpose::trans **or** transb = transpose::conjtrans
Column major	Must be at least m	Must be at least n
Row major	Must be at least n	Must be at least m

stride_b Stride between the different B matrices in the array b. Must be positive and satisfy:

	transb = transpose::nontrans	transb = transpose::trans **or** transb = transpose::conjtrans
Column major	Must be at least ldb*n	Must be at least ldb*m
Row major	Must be at least ldb*m	Must be at least ldb*n

ldc Leading dimension of the C matrices. If matrices are stored using column major layout, ldc must be at least m. If matrices are stored using row major layout, ldc must be at least n. Must be positive.

stride_c Stride between the different C matrices. If matrices are stored using column major layout, stride_c must be at least ldc*n. If matrices are stored using row major layout, stride_c must be at least ldc*m.

batch_size Specifies the number of input and output matrices to add. Must be at least zero.

dependencies List of events to wait for before starting computation, if any. If omitted, defaults to no dependencies.

Output Parameters

c Output array, overwritten by batch_size matrix addition operations of the form alpha*op(A) + beta*op(B). Must have size at least stride_c*batch_size.

Return Values

Output event to wait on to ensure computation is complete.

9.5 Compute Modes

BLAS level-3 routines and extensions support **alternate compute modes**, which can provide increased performance in exchange for different numerical properties or reduced accuracy.

A list of one or more allowed modes can be specified either at compile time, on a per-call or per-source-file basis, or at runtime, using the `MKL_BLAS_COMPUTE_MODE` environment variable. oneMKL will automatically select an appropriate implementation from this list, taking into account routine parameters and hardware characteristics. In case none of the allowed alternate modes are supported by the given routine on the selected device, or if none of the allowed alternate modes are expected to improve performance, oneMKL will automatically fall back to a standard implementation.

> **Note:** Whether a particular mode provides additional performance will depend on a number of factors, including routine type, matrix sizes, transpose parameters, and hardware configuration. In the current oneMKL release, alternate implementations are available for `gemm`, `gemmt`, `syrk`, and `syr2k` on selected hardware.

By default, oneMKL does not enable any alternate compute modes. The `MKL_BLAS_COMPUTE_MODE` environment variable is intended for quickly evaluating whether alternate compute modes provide performance benefits and acceptable accuracy for an application. After initial testing, alternate mode settings can be permanently applied within the application using the per-call or per-source-file APIs.

When running on GPU, oneMKL's verbose output indicates which mode is used for each call, whether the standard mode or one of the alternate modes. See **Checking Which Mode Is Used** for more details.

9.5.1 Mode Settings

Available alternate modes are described in the table below. Multiple modes can be combined, allowing oneMKL to choose any of the allowed modes that is expected to provide best performance.

For per-call or per-source-file mode, alternate compute modes are selected by OR'ing together one or more `oneapi::mkl::blas::compute_mode` values from the following table, e.g.:

```
using oneapi::mkl::blas;
auto mode_settings = compute_mode::float_to_bf16x2 | compute_mode::float_to_tf32; /* allow␣
↪either of these two modes */
```

When making global changes with the `MKL_BLAS_COMPUTE_MODE` environment variable, multiple modes are combined with commas:

```
set MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X2,FLOAT_TO_TF32
```

compute_mode enum value	Environment variable setting	Description
compute_mode::float_to_bf16	FLOAT_TO_BF16	Convert single-precision inputs to bfloat16 format internally; output is accumulated in single precision. Accuracy will be reduced, with possibly much higher performance.
compute_mode::float_to_bf16x2	FLOAT_TO_BF16X2	Convert each single-precision input value to a sum of two bfloat16 values internally; output is accumulated in single precision. Expected accuracy is between that of standard single precision arithmetic and bfloat16 arithmetic. **Note:** infinite inputs may produce unexpected NaNs in the output.
compute_mode::float_to_bf16x3	FLOAT_TO_BF16X3	Convert each single-precision input value to a sum of three bfloat16 values internally; output is accumulated in single precision. Expected accuracy is comparable to standard single precision arithmetic in most cases. **Note:** infinite inputs may produce unexpected NaNs in the output.
compute_mode::float_to_tf32	FLOAT_TO_TF32	Convert each single-precision input value to tf32 format internally; output is accumulated in single precision

9.5.2 Per-Call Mode Settings

BLAS level-3 routines and extensions support an optional `oneapi::mkl::blas::compute_mode` argument to specify a desired mode setting, at the end of the parameter list. For USM APIs, the `compute_mode` argument goes before the list of input dependencies, if any; either argument may be omitted. For example:

```
using oneapi::mkl::blas;

sycl::buffer<float, 1> a_buffer, c_buffer;
float *a_ptr, *c_ptr;
/* ... */

// Buffer API
syrk(my_queue, n, k, uplo, trans, alpha, a_buffer, lda, beta, c_buffer, ldc, compute_mode::
↪float_to_bf16);

// Buffer API, forcing float_to_bf16 mode
syrk(my_queue, n, k, uplo, trans, alpha, a_buffer, lda, beta, c_buffer, ldc, compute_mode::
↪float_to_bf16 | compute_mode::force_alternate);

// USM API, without dependencies
syrk(my_queue, n, k, uplo, trans, alpha, a_ptr, lda, beta, c_ptr, ldc, compute_mode::float_to_
↪bf16);

// USM API, with dependencies
syrk(my_queue, n, k, uplo, trans, alpha, a_ptr, lda, beta, c_ptr, ldc, compute_mode::float_to_
↪bf16, {event1, event2});

// USM API, dependencies but no special compute_mode settings
syrk(my_queue, n, k, uplo, trans, alpha, a_ptr, lda, beta, c_ptr, ldc, {event1, event2});
```

9.5.3 Per-Source-File Mode Settings

You can provide default mode settings for all calls within a source file by defining the `MKL_BLAS_COMPUTE_MODE` macro before including any oneMKL header files. This macro must be set to an expression of type `oneapi::mkl::blas::compute_mode`.

```
#define MKL_BLAS_COMPUTE_MODE oneapi::mkl::blas::compute_mode::complex_3m
#include <oneapi/mkl.hpp>

void my_function() {
    /* ... */

    // 3M mode will be allowed by default:
    gemm(my_queue, m, n, k, trans_a, trans_b, alpha, a, lda, b, ldb, beta, c, ldc);
}
```

`compute_mode` parameters passed to a oneMKL routine take precedence over the default setting:

```
#define MKL_BLAS_COMPUTE_MODE oneapi::mkl::blas::compute_mode::complex_3m
#include <oneapi/mkl.hpp>
```

(continues on next page)

(continued from previous page)

```
void my_function() {
    /* ... */

    // Provided compute_mode overrides the default 3M mode.
    gemm(my_queue, m, n, k, trans_a, trans_b, alpha, a, lda, b, ldb, beta, c, ldc, compute_mode:
↪:standard);
}
```

9.5.4 Runtime Mode Settings

The `MKL_BLAS_COMPUTE_MODE` environment variable allows you to set default application-wide alternate compute mode settings, as a quick method for evaluating alternate compute modes at runtime. For example, with `bash` or a similar shell:

```
// my_application.cpp
#include <oneapi/mkl.hpp>

int main() {
    /* ... */

    // Call to gemm without a compute_mode argument:
    oneapi::mkl::blas::gemm(my_queue, /* ... */);
}
```

```
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X3
my_application      # gemm may use bf16x3 arithmetic.
```

Any per-call or per-source-file mode settings take precedence over the `MKL_BLAS_COMPUTE_MODE` environment variable. One result of this is that compiling an application with `-DMKL_BLAS_COMPUTE_MODE=oneapi::mkl::blas::compute_mode::standard` effectively disables the environment variable.

9.5.5 Checking Which Mode Is Used

On GPU, when oneMKL's verbose mode is enabled, information on the compute mode(s) enabled and used for each call is provided in the verbose log. For example, in the following call, `float_to_bf16` and `float_to_tf32` were both enabled, and `float_to_bf16` was selected (some output omitted for clarity):

```
MKL_VERBOSE oneapi::mkl::blas::column_major::gemm[float](0x7ffd39046350,...,float_to_bf16|float_
↪to_tf32) mode:float_to_bf16 host:nan device:nan GPU0
```

Verbose mode can be enabled by setting the `MKL_VERBOSE` environment variable to 1, or via the `mkl_verbose` API. For more information, see "Using oneMKL Verbose Mode" in the Intel(R) oneAPI Math Kernel Library Developer Guide, available in the Intel Software Documentation Library.

10.0 Sparse BLAS Routines

The Intel® oneAPI Math Kernel Library provides a C++ with SYCL interface to some of the Sparse BLAS routines. This section describes such Sparse BLAS routines included in the Intel® oneAPI Math Kernel Library (oneMKL).

The Sparse BLAS library provides basic operations on sparse matrices and vectors. The fundamental object that encompasses the sparse matrix and is which is used in the Sparse BLAS library is the `sparse::matrix_handle_t`. See **Sparse BLAS Matrix Handle Contract between User and Library** for a discussion of the way that library APIs will interact with the handle as well as the usage contract with regards to the matrix handle for both users and the library. The currently supported sparse matrix formats in the `sparse::matrix_handle_t` object can be found in **Sparse Storage Formats**.

The routines that enable these Sparse BLAS operations can be separated into 4 groups:

1. `State management` routines
2. `Analysis` routines (also called inspector stage or optimize stage routines)
3. `Execution` routines
4. `Helper` routines.

The `state management` routines include initialization, destruction and APIs for setting data, formats and properties in the different sparse objects like the `sparse::matrix_handle_t` or `sparse::matmat_descr_t`.

In an `analysis` routine, the library inspects the matrix properties including size, sparsity pattern and available parallelism and can create new data structures or copies of the user data which applies matrix format or structure changes to enable a more optimized algorithm for the desired operation. The user data is not changed by any such analysis or optimizations. The optimizations created in the `analysis` routines may be reused by multiple `execution` routines to improve performance. For a given matrix, an `analysis` routine would typically be called a single time for each operation whereas the corresponding `execution` routines may be called multiple times.

The `execution` routines are where the actual matrix-matrix, matrix-vector operations take place and may use the data, optimizations and properties stored in the handle to perform the desired operation.

The `helper` routines operate on the data in the matrix handle and may include things like data copy/transpose into other handles, sorting of data within a handle, or eventually, when supported, changes to the sparse matrix format within the handle.

State Management Routines	Data Types	Description
sparse::init_matrix_handle	N/A	Initialize a `sparse::matrix_handle_t` object
sparse::release_matrix_handle	N/A	Release a `sparse::matrix_handle_t` object
sparse::set_csr_data	float, double	Set internal representation of `sparse::matrix_handle_t` to compressed sparse row (CSR) format with user provided CSR data arrays.

continues on next page

Table 4 – continued from previous page

State Management Routines	Data Types	Description
sparse::set_matrix_property	N/A	Set special properties of user provided marix data in `sparse::matrix_handle_t` that can provide hints for optimizations.
sparse::init_matmat_descr	N/A	Initialize a `sparse::matmat_descr_t` object. For use with sparse::matmat
sparse::set_matmat_data	N/A	Populate a `sparse::matmat_descr_t` object with desired operation description. For use with sparse::matmat
sparse::get_matmat_data	N/A	Query the operation description housed in a `sparse::matmat_descr_t` object. For use with sparse::matmat
sparse::release_matmat_descr	N/A	Release a `sparse::matmat_descr_t` object. For use with sparse::matmat

Analysis Routines	Data Types	Description
sparse::optimize_gemv	N/A	Perform internal optimizations for the sparse::gemv operation.
sparse::optimize_trmv	N/A	Perform internal optimizations for the sparse::trmv operation.
sparse::optimize_trsv	N/A	Perform internal optimizations for the sparse::trsv operation.

Execution Routines	Data Types	Description
sparse::gemv	float, double	General sparse matrix-dense vector product
sparse::gemvdot	float, double	General sparse matrix-dense vector product with fused dot product
sparse::symv	float, double	Symmetric sparse matrix-dense vector product
sparse::trmv	float, double	Triangular sparse matrix-dense vector product
sparse::trsv	float, double	Triangular solve of sparse matrix against a dense vector.
sparse::gemm	float, double	General sparse matrix-dense matrix product
sparse::matmat	float, double	General sparse matrix-sparse matrix product

Helper Routines	Data Types	Description
sparse::omatcopy	float, double	General sparse matrix out-of-place copy/transposition into a new matrix handle
sparse::sort_matrix	float, double	General sparse matrix sort of matrix format in matrix handle

10.1 Sparse BLAS Matrix Handle Contract between User and Library

The sparse matrix handle (`sparse::matrix_handle_t`) takes in data from the **User** in a call that looks like sparse::set_csr_data(q, handle, /*user data*/). Unlike most other oneMKL domains, the `sparse::matrix_handle_t` along with user data persists outside of individual calls to the oneMKL library. When the **User** subsequently makes a call to a oneMKL Sparse BLAS API with that handle, the **Library** uses that data stored in the handle internally for the various operations. Because both users and library have access to the data at the same time while the matrix handle exists, there must be some agreements about what can and cannot be done by each party. We therefore introduce an implicit contract between the **User** and the **Library** that describes each party's roles and responsibilities with respect to the Sparse BLAS SYCL APIs and the use of the `sparse::matrix_handle_t`.

10.1.1 Description of `sparse::matrix_handle_t` object

First, we note that the `sparse::matrix_handle_t` represents a generic sparse matrix object with some internal matrix format or representation. Providing device USM pointers for that matrix data in the `sparse::matrix_handle_t` object binds that handle to that particular `sycl::context` and `sycl::device` as well as any other devices in that `sycl::context` that are compatible (can read from the particular device peer to peer). It is the **Users'** responsibility to make sure they are using the matrix handle with appropriate queues and devices within the given context. Using shared and host USM pointers or `sycl::buffers` for the matrix data binds that `sparse::matrix_handle_t` object to the `sycl::context` and could reasonably be used on any device in the context.

Second, the `sparse::matrix_handle_t` can best be described as a "view of **User's** matrix data arrays with an opaque state attached to it". That is, it is a lightweight view to begin with, but through the use of `sparse::optimize_*` APIs and in some cases, the execute APIs (like sparse::matmat()), the hidden state could grow with different internal optimizations/structures that can persist through the lifetime of the handle, with the goal of enabling superior performance for the desired operations.

10.1.2 User and Library agreements with respect to `sparse::matrix_handle_t` object

Finally the following agreements describe **User** and **Library** roles with respect to the handle use:

User agreements:

1. The **User** owns any data that is provided to the `sparse::matrix_handle_t` and is responsible to create and dispose of them correctly. That is, the data is created by the **User** then passed to the matrix handle using some **Library** API like sparse::set_csr_data(). The data can only be disposed of after all uses of the matrix handle and the release of the matrix handle have finished.
2. The **User** agrees not to modify the data arrays directly while they are in a `sparse::matrix_handle_t`. Any changes should happen while outside of a matrix handle or when available, indirectly through use of a Sparse BLAS library API that states it will modify the matrix handle data in specified ways.

Library agreements:

1. The **Library** may create and store it's own data in the `sparse::matrix_handle_t` during a Sparse BLAS API library call. Such data is owned by the Library, that is, the **Library** is responsible for its disposal. The **User** cannot access this data, and the library-owned data, associated with a `sparse::matrix_handle_t`, is cleaned up at the respective handle release.

2. The **Library** agrees to not modify the user-provided data arrays in the `sparse::matrix_handle_t` unless through a library API that specifically states it may change the data (like sparse::sort_matrix() or sparse::omatcopy()).

With understanding of these agreements, the **User** can safely use the oneMKL SYCL Sparse BLAS APIs to accelerate their workloads and applications and the **Library** can rely on a consistent and known state of the user data through the lifetime of the handle.

10.1.3 Example of `sparse::matrix_handle_t` usage workflow

An example workflow to demonstrate the usage model is the following:

```
// (A) Allocate user memory for sparse matrix such as with SYCL USM device alloc or sycl::buffer
// (B) Do anything with matrix arrays, such as memcpy some data from host
// (C) Create a sparse matrix handle and pass in matrix arrays
// (D) Make some library calls of your choice (for instance with matrix handle)
// (E) Destroy the matrix handle via call to sparse::release_matrix_handle()
// (F) Do anything with user memory - modify, copy, etc
// (G) Create another sparse matrix handle using the same arrays
// (H) Make other library calls of your choice
// (I) Destroy the matrix handle via call to sparse::release_matrix_handle()
// (J) Do whatever you want with user memory
```

10.2 Sparse BLAS Supported Data and Integer Types

Data Types <fp>	Integer Types <intType>
`float`	`std::int32_t`
`double`	`std::int64_t`
`std::complex<float>`	
`std::complex<double>`	

10.3 Sparse Storage Formats

There are a variety of matrix storage formats available for representing a sparse matrix. The Intel® oneAPI Math Kernel Library (oneMKL) Sparse BLAS library provides support for the following sparse matrix formats:

Sparse Matrix Formats Supported in oneMKL Sparse BLAS
Compressed Sparse Row (CSR)

10.3.1 Compressed Sparse Row (CSR)

One of the most common sparse formats is the CSR (sometimes called 3-array CSR or CSR3) format that is represented by scalar sizes (`num_rows`, `num_cols`), as well as three data arrays: `row_ptr`, `col_inds` and `vals`, and the index_base parameter. Some versions of this format also explicitly store the number of non-zero elements (`nnz`) but this can be extracted from the `row_ptr` array as described below in the description of the `row_ptr` so we will not include it here.

CSR Matrix Format Elements	Description
`num_rows`	Number of rows in the sparse matrix.
`num_cols`	Number of columns in the sparse matrix.
`index_base`	Parameter that is used to specify whether the matrix has zero or one-based indexing.
`vals`	An array that contains the non-zero elements of the sparse matrix stored row by row.
`col_inds`	An integer array of column indices for non-zero elements stored in the `vals` array, such that `col_inds[i]` is the column number (using zero- or one-based indexing) of the element of the sparse matrix stored in `vals[i]`.
`row_ptr`	An integer array of size equal to `num_rows + 1`. Element `j` of this integer array gives the position of the element in the `vals` array that is first non-zero element in a row `j` of `A`. Note that this position is equal to `row_ptr[j] - index_base`. The last element of the `row_ptr` array (`row_ptr[num_rows]`) stores the sum of the number of non-zero elements (nnz) and `index_base`. That is, `nnz = row_ptr[num_rows] - index_base`.

Examples of CSR format

The following 3 examples show how the sparse CSR format can be used:

- CSR Case 1: sorted square matrix with zero-based indexing
- CSR Case 2: sorted rectangular matrix with one-based indexing and an empty row
- CSR Case 3: unsorted rectangular matrix with zero-based indexing

CSR Case 1: sorted square matrix with zero-based indexing

Assuming zero-based indexing and a real square matrix.

$$A = \begin{pmatrix} 1.0 & 0.0 & 2.0 \\ 0.0 & -1.0 & 4.0 \\ 3.0 & 0.0 & 0.0 \end{pmatrix}$$

num_rows	3				
num_cols	3				
index_base	0				
row_ptr	0	2	4	5	
col_inds	0	2	1	2	0
vals	1.0	2.0	-1.0	4.0	3.0

CSR Case 2: sorted rectangular matrix with one-based indexing and an empty row

Assuming one-based indexing and real rectangular matrix with an empty row.

$$A = \begin{pmatrix} 1.0 & 0.0 & 2.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 4.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.0 & 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

num_rows	4						
num_cols	5						
index_base	1						
row_ptr	1	3	6	6	8		
col_inds	1	3	2	3	5	1	4
vals	1.0	2.0	-1.0	4.0	1.0	3.0	1.0

CSR Case 3: unsorted rectangular matrix with zero-based indexing

Unsorted CSR example: Assuming zero-based indexing and a real rectangular matrix, we note that the CSR format does not require column indices to be sorted within a given row, but `vals` and `col_inds` arrays should be consistent with each other. Having the sorted property is not necessary, but can lead to better performance in actual runs due to better algorithms and data locality being enabled. See sparse::set_matrix_property() and sparse::sort_matrix for more details on how one could set the sorted property when it is applicable or reorder the matrix to achieve it when desired.

$$A = \begin{pmatrix} 1.0 & 0.0 & 2.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 4.0 & 0.0 & 1.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 0.0 \\ 3.0 & 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

num_rows	4									
num_cols	5									
index_base	0									
row_ptr	0	2	5	8	10					
col_inds	0	2	4	1	2	1	2	0	3	0
vals	1.0	2.0	1.0	-1.0	4.0	2.0	3.0	1.0	1.0	3.0

10.4 oneapi::mkl::sparse::init_matrix_handle

Initializes a `oneapi::mkl::sparse::matrix_handle_t` object to default values.

10.4.1 Description

The `oneapi::mkl::sparse::init_matrix_handle` function initializes the `oneapi::mkl::sparse::matrix_handle_t` object with default values, otherwise it throws an exception.

Note: Refer to Error Handling for a detailed description of the exceptions thrown.

10.4.2 API

Syntax

```
namespace oneapi::mkl::sparse {
    void init_matrix_handle (
        oneapi::mkl::sparse::matrix_handle_t *handle)
}
```

Include Files

- oneapi/mkl/spblas.hpp

10.5 oneapi::mkl::sparse::release_matrix_handle

Releases internal data and sets `oneapi::mkl::sparse::matrix_handle_t` object to NULL.

10.5.1 Description

The `oneapi::mkl::sparse::release_matrix_handle` routine releases (also waits for the dependencies to be finished when provided) any internal data that the `oneapi::mkl::sparse::matrix_handle_t` object holds and sets it with default values, otherwise throws an exception.

Note: Refer to Error Handling for a detailed description of the possible exceptions thrown.

10.5.2 API

Syntax

```
namespace oneapi::mkl::sparse {

    void release_matrix_handle (
        sycl::queue & queue,
        oneapi::mkl::sparse::matrix_handle_t *handle,
        const std::vector<sycl::event> &dependencies = {});

    // deprecated in 2023.0
    void release_matrix_handle (
        oneapi::mkl::sparse::matrix_handle_t *handle,
        const std::vector<sycl::event> &dependencies ={});
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

handle Handle to object containing sparse matrix and other internal data. Initialized with `oneapi::mkl::sparse::init_matrix_handle` routine and filled with user data using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines. The `oneapi::mkl::sparse::optimize_xyz` routines may have also created additional internally allocated data which would be released in this call.

Note: Currently, the only supported case for **<sparse_matrix_type>** is csr.

---

dependencies A vector of type `std::vector<sycl::event>` & containing the list of events that handle depends on before executing the release of the matrix handle.

10.6 oneapi::mkl::sparse::set_csr_data

Takes a matrix handle and the user-provided Compressed Sparse Row (CSR) matrix arrays and fills the internal CSR data structure of the matrix handle.

10.6.1 Description

The `oneapi::mkl::sparse::set_csr_data` routine takes a `sparse::matrix_handle_t` for a sparse matrix of dimensions `num_rows` -by- `num_cols` represented in the CSR format, and fills the internal state of the matrix handle with the user provided arrays in CSR format. Please familiarize yourself with the user/library contract surrounding use of the `sparse::matrix_handle_t` object.

Note: Refer to Sparse BLAS Supported Data and Integer Types for a list of supported `<fp>` and `<intType>` data and integer types and refer to Error Handling for a detailed description of the possible exceptions thrown.

10.6.2 API

Syntax

Note: Currently, complex types are not supported.

Using SYCL buffers:

```
namespace oneapi::mkl::sparse {
    void set_csr_data (
        sycl::queue &queue,
        oneapi::mkl::sparse::matrix_handle_t handle,
        const intType num_rows,
        const intType num_cols,
        oneapi::mkl::index_base index,
        sycl::buffer<intType, 1> &row_ptr,
        sycl::buffer<intType, 1> &col_ind,
        sycl::buffer<fp, 1> &val);

    //deprecated in 2023.0
    void set_csr_data (
        oneapi::mkl::sparse::matrix_handle_t handle,
        const intType num_rows,
        const intType num_cols,
        oneapi::mkl::index_base index,
        sycl::buffer<intType, 1> &row_ptr,
        sycl::buffer<intType, 1> &col_ind,
        sycl::buffer<fp, 1> &val);
 }
```

Using USM pointers:

```
namespace oneapi::mkl::sparse {

    sycl::event set_csr_data (
        sycl::queue &queue,
        oneapi::mkl::sparse::matrix_handle_t handle,
        const intType num_rows,
```

(continues on next page)

(continued from previous page)

```
        const intType num_cols,
        oneapi::mkl::index_base index,
        intType *row_ptr,
        intType *col_ind,
        fp *val,
        std::vector<sycl::event> &dependencies = {} );

    // deprecated in 2023.0
    void set_csr_data (
        oneapi::mkl::sparse::matrix_handle_t handle,
        const intType num_rows,
        const intType num_cols,
        oneapi::mkl::index_base index,
        intType *row_ptr,
        intType *col_ind,
        fp *val);
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

handle Handle to object containing sparse matrix and other internal data for subsequent Sparse BLAS operations.

num_rows Number of rows of the input matrix.

num_cols Number of columns of the input matrix.

index Indicates how input arrays are indexed.

`oneapi::mkl::index_base::zero`	Zero-based (C-style) indexing: indices start at 0.
`oneapi::mkl::index_base::one`	One-based (Fortran-style) indexing: indices start at 1.

row_ptr SYCL memory object containing an array of length `num_rows+1`. Could be a SYCL buffer or a device-accessible USM pointer. Refer to Sparse Storage Formats for a detailed description of `row_ptr`.

col_ind SYCL memory object which stores an array containing the column indices in `index`-based numbering. Could be a SYCL buffer or a device-accessible USM pointer. Refer to Sparse Storage Formats for a detailed description of `col_ind`.

val SYCL memory object which stores an array containing the non-zero elements of the input matrix. Could be a SYCL buffer or a device-accessible USM pointer. Refer to Sparse Storage Formats for a detailed description of `val`.

dependencies USM API only. A vector of type `std::vector<sycl::event>` containing the list of events that the `oneapi::mkl::sparse::set_csr_data` routine depends on.

Output Parameters

handle Handle to object containing sparse matrix and other internal data for subsequent DPC++ Sparse BLAS operations.

sycl::event USM API only. A `sycl::event` that can be used to track the completion of asynchronous events that were enqueued during the API call that continue the chain of events from the input dependencies.

10.7 oneapi::mkl::sparse::set_matrix_property

Sets matrix properties present in the user provided data provided to the `sparse::matrix_handle_t` that can serve as optimization hints for library algorithms. Properties are not verified by the library but accepted as truth from the user who specified them.

10.7.1 Description

The `oneapi::mkl::sparse::set_matrix_property` routine allows the user to set some properties of the user-provided matrix data in the `sparse::matrix_handle_t` object that can act as hints for the internal algorithms in subsequent library calls.

The `oneapi::mkl::sparse::property` enum class is defined in the `oneapi/mkl/spblas.hpp` header file

```
namespace oneapi::mkl::sparse {
    enum class property : char {
        symmetric,
        sorted
    };
}
```

where symmetric refers to the matrix being symmetric and the full pattern is present in the data arrays. The sorted property indicates that the matrix data is sorted in whatever manner is natural to the particular matrix format. See sparse::sort_matrix for more details on sorting. The library will not verify that these properties are true, but will take them as truth from the user. Setting them may affect performance as certain internal optimizations may not need to be done if they are present. The properties may also be set internally by the library when it is applicable (for instance after a call to sparse::sort_matrix.

A common usage model for setting matrix properites is the following:

```
using namespace oneapi::mkl;
sparse::matrix_handle_t handle = nullptr;

sparse::init_matrix_handle(&handle);
```

(continues on next page)

(continued from previous page)

```
sparse::set_csr_data(main_queue, handle, nrows, nrows,
                     index_base::zero, ia_buffer, ja_buffer, a_buffer);

// The csr ia/ja/a arrays are of a full square symmetric matrix that is sorted
// with increasing columns in each row, so we can set both symmetric and
// sorted property
sparse::set_matrix_property(handle, sparse::property::symmetric);
sparse::set_matrix_property(handle, sparse::property::sorted);

sparse::optimize_trsv(main_queue, uplo::lower, transpose::nontrans,
                      diag::nonunit, handle);
sparse::optimize_trsv(main_queue, uplo::upper, transpose::nontrans,
                      diag::nonunit, handle);
sparse::optimize_gemv(main_queue, transpose::nontrans, handle);

// implement some algorithm using trsv/gemv operations (like a preconditioned
// conjugate gradient algorithm with symmetric Gauss-Seidel preconditioner)

sparse::release_matrix_handle(handle);
```

10.7.2 API

Syntax

```
namespace oneapi::mkl::sparse {
    void set_matrix_property(oneapi::mkl::sparse::matrix_handle_t handle,
                             oneapi::mkl::sparse::property property_value);
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

handle Handle to object containing sparse matrix and other internal data where property will be set. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

property_value Sparse matrix property being set as in the sparse matrix handle object.

`sparse::property::symmetric`	data in matrix handle represents a symmetric matrix and has the full pattern and values provided
`sparse::property::sorted`	data in matrix handle is sorted according to natural state for the given format

10.7.3 Examples

An example of how to use `oneapi::mkl::sparse::set_matrix_property` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/sparse_blas/source/sparse_cg.cpp
```

10.8 oneapi::mkl::sparse::optimize_gemv

Performs internal optimizations for oneapi::mkl::sparse::gemv by analyzing the matrix structure.

10.8.1 Description

Note: Refer to Error Handling for a detailed description of the possible exceptions thrown.

The `oneapi::mkl::sparse::optimize_gemv` routine analyzes matrix structure and performs optimizations. Optimized data is then stored in the matrix handle.

10.8.2 API

Syntax

Note: Currently, complex types are not supported.

Using USM and SYCL buffers:

```
namespace oneapi::mkl::sparse {
    sycl::event optimize_gemv (
        sycl::queue &queue,
        oneapi::mkl::transpose transpose_flag,
        oneapi::mkl::sparse::matrix_handle_t handle,
        const std::vector<sycl::event> &dependencies = {});
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

transpose_flag Specifies operation `op()` on input matrix.

oneapi::mkl::transpose::nontrans	Non-transpose, op(A) = A.
oneapi::mkl::transpose::trans	Transpose, op(A) = A^T.
oneapi::mkl::transpose::conjtrans	Conjugate transpose, op(A) = A^H.

Note: Currently, the only supported case for operation is `oneapi::mkl::transpose::nontrans`.

handle Handle to object containing sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

Note: Currently, the only supported matrix format for `<sparse_matrix_type>` is the **csr** format.

dependencies A vector of type `std::vector<sycl::event> &` containing the list of events that the `oneapi::mkl::sparse::optimize_gemv` routine depends on.

Return Values

`sycl::event` SYCL event which can be waited upon or added as a dependency for the completion of the `optimize_gemv` routine.

```
sycl::event ev_opt = sparse::optimize_gemv(queue, trans_val, handle);              //␣
↪Allowed use in case of USM
sycl::event ev_opt = sparse::optimize_gemv(queue, trans_val, handle, {});          //␣
↪Allowed use in case of USM
sycl::event ev_opt = sparse::optimize_gemv(queue, trans_val, handle, dependencies); //␣
↪Allowed use in case of USM
sparse::optimize_gemv(queue, trans_val, handle);                                   // OK -
↪- Recommended use in case of sycl::buffer
static_cast<void>(sparse::optimize_gemv(queue, trans_val, handle));                // OK -
↪- Recommended use in case of sycl::buffer - explicitly say we aren't using the event
sparse::optimize_gemv(queue, trans_val, handle, {});                               // Not␣
↪recommended in case of sycl::buffer, but supported;
sparse::optimize_gemv(queue, trans_val, handle, dependencies);                     // Not␣
↪recommended in case of sycl::buffer, but supported;
sycl::event ev_opt = sparse::optimize_gemv(queue, trans_val, handle);              //␣
↪Allowed use in case of sycl::buffer, but most sycl::buffer oneMKL APIs do not have␣
↪ability to pass in dependencies
```

10.9 oneapi::mkl::sparse::optimize_trmv

Performs internal optimizations for oneapi::mkl::sparse::trmv by analyzing the matrix structure.

10.9.1 Description

The `oneapi::mkl::sparse::optimize_trmv` routine analyzes matrix structure and performs optimizations. Optimized data is then stored in the matrix handle.

Note: Refer to Error Handling for a detailed description of the possible exceptions thrown.

10.9.2 API

Syntax

Note: Currently, complex types are not supported.

Using USM and SYCL buffers:

```
namespace oneapi::mkl::sparse {
    sycl::event optimize_trmv (
        sycl::queue &queue,
        oneapi::mkl::uplo uplo_flag,
        oneapi::mkl::transpose transpose_flag,
        oneapi::mkl::diag diag_flag,
        oneapi::mkl::sparse::matrix_handle_t handle,
        const std::vector<sycl::event> &dependencies = {});
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

uplo_flag Specifies which part of the matrix is to be processed.

`oneapi::mkl::uplo::lower`	The lower triangular matrix part is processed.
`oneapi::mkl::uplo::upper`	The upper triangular matrix part is processed.

transpose_flag Specifies operation `op( )` on input matrix.

`oneapi::mkl::transpose::nontrans`	Non-transpose, $op(A) = A$.
`oneapi::mkl::transpose::trans`	Transpose, $op(A) = A^T$.
`oneapi::mkl::transpose::conjtrans`	Conjugate transpose, $op(A) = A^H$.

Note: Currently, the only supported case for `operation` is `oneapi::mkl::transpose::nontrans`.

diag_flag Specifies if the diagonal used for computations is unit or not.

`oneapi::mkl::diag::nonunit`	Diagonal elements might not be equal to one.
`oneapi::mkl::diag::unit`	Diagonal elements are equal to one.

Note: Currently, the only supported case for `diag_flag` is `oneapi::mkl::diag::nonunit`.

handle Handle to object containing sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

Note: Currently, the only supported matrix format for `<sparse_matrix_type>` is the **csr** format.

dependencies A vector of type `std::vector<sycl::event> &` containing the list of events that the `oneapi::mkl::sparse::optimize_trmv` routine depends on.

Return Values

sycl::event SYCL event which can be waited upon or added as a dependency for the completion of the `optimize_trmv` routine.

```
sycl::event ev_opt = sparse::optimize_trmv(queue, uplo_val, trans_val, diag_val, handle);
↪             // Allowed use in case of USM
sycl::event ev_opt = sparse::optimize_trmv(queue, uplo_val, trans_val, diag_val, handle, {}
↪);           // Allowed use in case of USM
sycl::event ev_opt = sparse::optimize_trmv(queue, uplo_val, trans_val, diag_val, handle,
↪dependencies); // Allowed use in case of USM
sparse::optimize_trmv(queue, uplo_val, trans_val, diag_val, handle);
↪             // OK Recommended use in case of sycl::buffer
static_cast<void>(sparse::optimize_trmv(queue, uplo_val, trans_val, diag_val, handle));
↪             // OK Recommended use in case of sycl::buffer (explicitly say we aren't
↪using the event)(will be used in examples)
sparse::optimize_trmv(queue, uplo_val, trans_val, diag_val, handle, {});
↪             // Not recommended in case of sycl::buffer, but supported;
sparse::optimize_trmv(queue, uplo_val, trans_val, diag_val, handle, dependencies);
↪             // Not recommended in case of sycl::buffer, but supported;
sycl::event ev_opt = sparse::optimize_trmv(queue, uplo_val, trans_val, diag_val, handle);
↪             // Allowed use in case of sycl::buffer, but most sycl::buffer oneMKL APIs
↪do not have ability to pass in dependencies
```

10.10 oneapi::mkl::sparse::optimize_trsv

Performs internal optimizations for `oneapi::mkl::sparse::trsv` by analyzing the provided matrix structure and operation parameters.

10.10.1 Description

The `oneapi::mkl::sparse::optimize_trsv` routine analyzes matrix structure and performs optimizations. Optimized data is then stored in the matrix handle.

Note: Refer to Error Handling for a detailed description of the possible exceptions thrown.

10.10.2 API

Syntax

Note: Currently, complex types are not supported.

Using USM and SYCL buffers:

```
namespace oneapi::mkl::sparse {
    sycl::event optimize_trsv (
        sycl::queue &queue,
        oneapi::mkl::uplo uplo_flag,
        oneapi::mkl::transpose  transpose_flag,
        oneapi::mkl::diag diag_flag,
        oneapi::mkl::sparse::matrix_handle_t handle,
        const std::vector<sycl::event> &dependencies = {});
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

uplo_flag Specifies which part of the matrix is to be processed.

`oneapi::mkl::uplo::lower`	The lower triangular matrix part is processed.
`oneapi::mkl::uplo::upper`	The upper triangular matrix part is processed.

transpose_flag Specifies operation `op()` on input matrix.

`oneapi::mkl::transpose::nontrans`	Non-transpose, `op(A)` = A.
`oneapi::mkl::transpose::trans`	Transpose, `op(A)` = A^T.
`oneapi::mkl::transpose::conjtrans`	Conjugate transpose, `op(A)` = A^H.

Note: Currently, the only supported case for operation is `oneapi::mkl::transpose::nontrans`.

diag_flag Specifies if the diagonal used for computations is unit or not.

`oneapi::mkl::diag::nonunit`	Diagonal elements might not be equal to one.
`oneapi::mkl::diag::unit`	Diagonal elements are equal to one.

handle Handle to object containing sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

Note: Currently, the only supported matrix format for `<sparse_matrix_type>` is the **csr** format.

dependencies A vector of type `std::vector<sycl::event>` containing the list of events that the `oneapi::mkl::sparse::optimize_trsv` routine depends on.

Return Values (USM Only)

sycl::event SYCL event which can be waited upon or added as a dependency for the completion of the `optimize_trsv` routine.

```
sycl::event ev_opt = sparse::optimize_trsv(queue, uplo_val, trans_val, diag_val, handle);
↪             // Allowed use in case of USM
sycl::event ev_opt = sparse::optimize_trsv(queue, uplo_val, trans_val, diag_val, handle, {}
↪);           // Allowed use in case of USM
sycl::event ev_opt = sparse::optimize_trsv(queue, uplo_val, trans_val, diag_val, handle,
↪dependencies); // Allowed use in case of USM
sparse::optimize_trsv(queue, uplo_val, trans_val, diag_val, handle);
↪             // OK Recommended use in case of sycl::buffer
static_cast<void>(sparse::optimize_trsv(queue, uplo_val, trans_val, diag_val, handle));
↪             // OK Recommended use in case of sycl::buffer (explicitly say we aren't
↪using the event)(will be used in examples)
sparse::optimize_trsv(queue, uplo_val, trans_val, diag_val, handle, {});
↪             // Not recommended in case of sycl::buffer, but supported;
sparse::optimize_trsv(queue, uplo_val, trans_val, diag_val, handle, dependencies);
↪             // Not recommended in case of sycl::buffer, but supported;
sycl::event ev_opt = sparse::optimize_trsv(queue, uplo_val, trans_val, diag_val, handle);
↪             // Allowed use in case of sycl::buffer, but most sycl::buffer oneMKL APIs
↪do not have ability to pass in dependencies
```

10.11 oneapi::mkl::sparse::gemv

Computes a sparse matrix-dense vector product.

10.11.1 Description

Note: Refer to Sparse BLAS Supported Data and Integer Types for a list of supported <fp> and <intType> data and integer types and refer to Error Handling for a detailed description of the possible exceptions thrown.

The `oneapi::mkl::sparse::gemv` routine computes a sparse matrix-dense vector product defined as

$$y \leftarrow \alpha \cdot \mathrm{op}(A) \cdot x + \beta \cdot y$$

where α and β are scalars, A is a general sparse matrix of dimensions `num_rows` rows and `num_cols` columns and $op()$ is a matrix modifier:

$$\mathrm{op}(A) = \begin{cases} A, & \text{oneapi::mkl::transpose::nontrans} \\ A^T, & \text{oneapi::mkl::transpose::trans} \\ A^H, & \text{oneapi::mkl::transpose::conjtrans} \end{cases}$$

The dense vectors x and y are appropriately sized based on matrix product dimensions of $\mathrm{op}(A)$.

10.11.2 API

Syntax

Note: Currently, complex types are not supported.

Using SYCL buffers:

```
namespace oneapi::mkl::sparse {
    void gemv (
        sycl::queue &queue,
        oneapi::mkl::transpose transpose_flag,
        const fp alpha,
        oneapi::mkl::sparse::matrix_handle_t handle,
        sycl::buffer<fp, 1> &x,
        const fp beta,
        sycl::buffer<fp, 1> &y)
}
```

Using USM pointers:

```
namespace oneapi::mkl::sparse {
    sycl::event gemv (
        sycl::queue &queue,
        oneapi::mkl::transpose transpose_flag,
        const fp alpha,
        oneapi::mkl::sparse::matrix_handle_t handle,
        const fp *x,
        fp beta,
        fp *y,
        const std::vector<sycl::event> &dependencies= {})
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

transpose_flag Specifies operation `op()` on input matrix.

`oneapi::mkl::transpose::nontrans`	Non-transpose, op$(A) = A$.
`oneapi::mkl::transpose::trans`	Transpose, op$(A) = A^T$.
`oneapi::mkl::transpose::conjtrans`	Conjugate transpose, op$(A) = A^H$.

Note: Currently, the only supported case for `operation` is `oneapi::mkl::transpose::nontrans`.

alpha Specifies the scalar, α.

handle Handle to object containing sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

Note: Currently, the only supported case for `<sparse_matrix_type>` is csr.

x SYCL buffer or device-accessible USM pointer of size at least equal to the number of columns of input matrix if `transpse_flag` = `oneapi::mkl::transpose::nontrans` and at least the number of rows of input matrix otherwise.

beta Specifies the scalar, β.

y SYCL buffer or device-accessible USM pointer of size at least equal to the number of rows of the input matrix if `transpose_flag` = `oneapi::mkl::transpose::nontrans` and at least the number of columns of the input matrix otherwise.

dependencies A vector of type `std::vector<sycl::event>` containing the list of events that the `oneapi::mkl::sparse::gemv` routine depends on.

Output Parameters

y Overwritten by the updated vector y.

Return Values (USM Only)

sycl::event SYCL event which can be waited upon or added as a dependency for the completion of the `gemv` routine.

10.11.3 Examples

An example of how to use `oneapi::mkl::sparse::gemv` with SYCL buffers or USM pointers can be found in the oneMKL installation directory, under:

```
examples/dpcpp/sparse_blas/source/sparse_gemv.cpp
```

```
examples/dpcpp/sparse_blas/source/sparse_gemv_usm.cpp
```

10.12 oneapi::mkl::sparse::gemvdot

Computes a sparse matrix-dense vector product with dot product.

10.12.1 Description

Note: Refer to Sparse BLAS Supported Data and Integer Types for a list of supported <fp> and <intType> data and integer types and refer to Error Handling for a detailed description of the possible exceptions thrown.

The `oneapi::mkl::sparse::gemvdot` routine computes a sparse matrix-dense vector product and dot product defined as

$$y \leftarrow \alpha \cdot \mathrm{op}(A) \cdot x + \beta \cdot y$$

$$d \leftarrow x \cdot y$$

where A is a general sparse matrix, α, β, and d are scalars, x and y are dense vectors and $op()$ is a matrix modifier:

$$\mathrm{op}(A) = \begin{cases} A, & \text{oneapi::mkl::transpose::nontrans} \\ A^T, & \text{oneapi::mkl::transpose::trans} \\ A^H, & \text{oneapi::mkl::transpose::conjtrans} \end{cases}$$

10.12.2 API

Syntax

Note: Currently, complex types are not supported.

Using SYCL buffers:

```
namespace oneapi::mkl::sparse {
    void gemvdot (
        sycl::queue &queue,
        oneapi::mkl::transpose transpose_flag,
        const fp alpha,
        oneapi::mkl::sparse::matrix_handle_t handle,
        sycl::buffer<fp, 1> &x,
        const fp beta,
        sycl::buffer<fp, 1> &y,
        sycl::buffer<fp, 1> &d)
}
```

Using USM pointers:

```
namespace oneapi::mkl::sparse {
    sycl::event gemvdot (
        sycl::queue &queue,
        oneapi::mkl::transpose transpose_flag,
        const fp alpha,
        oneapi::mkl::sparse::matrix_handle_t handle,
        fp *x,
        const fp beta,
        fp *y,
        fp *d,
        const std::vector<sycl::event> &dependencies={})
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

transpose_flag Specifies operation `op()` on input matrix.

`oneapi::mkl::transpose::nontrans`	Non-transpose, op$(A) = A$.
`oneapi::mkl::transpose::trans`	Transpose, op$(A) = A^T$.
`oneapi::mkl::transpose::conjtrans`	Conjugate transpose, op$(A) = A^H$.

Note: Currently, the only supported case for operation is `oneapi::mkl::transpose::nontrans`.

alpha Specifies the scalar, α.

handle Handle to object containing sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

Note: Currently, the only supported case for `<sparse_matrix_type>` is csr.

x SYCL buffer or device-accessible USM pointer of size at least equal to the number of columns of input matrix if `transpose_flag` == `oneapi::mkl::transpose::nontrans` and at least the number of rows of input matrix otherwise.

beta Specifies the scalar, β.

y SYCL buffer or device-accessible USM pointer of size at least equal to the number of rows of the input matrix if `transpose_flag` == `oneapi::mkl::transpose::nontrans` and at least the number of columns of the input matrix otherwise.

dependencies A vector of type `std::vector<sycl::event>` containing the list of events that the `oneapi::mkl::sparse::gemvdot` routine depends on.

Output Parameters

y Overwritten by the updated vector y.

d Overwritten by the dot product of x and y.

Return Values

`sycl::event` SYCL event which can be waited upon or added as a dependency for the completion of the `gemvdot` routine.

10.12.3 Examples

An example of how to use `oneapi::mkl::sparse::gemvdot` with SYCL buffers can be found in the oneMKL installation directory, under:

```
examples/dpcpp/sparse_blas/source/sparse_gemvdot.cpp
```

10.13 oneapi::mkl::sparse::symv

Computes a sparse matrix-dense vector product for a symmetric matrix built from the lower or upper triangular of the input matrix.

10.13.1 Description

Note: Refer to Sparse BLAS Supported Data and Integer Types for a list of supported <fp> and <intType> data and integer types and refer to Error Handling for a detailed description of the possible exceptions thrown.

The `oneapi::mkl::sparse::symv` routine computes a sparse symmetric matrix-dense vector product for a real symmetric matrix built from the lower or upper triangular portion of the input matrix A defined as

$$y \leftarrow \alpha \cdot \text{sym}(A) \cdot x + \beta \cdot y$$

where α and β are scalars, A is a real-valued square sparse matrix of dimension m rows and columns, sym() is a matrix modifier which symmetrizes the matrix according to the `oneapi::mkl::uplo` value and x and y are dense vectors.

For a given matrix decomposition into lower, diagonal and upper parts $A = L + D + U$, the `symv` routine with `oneapi::mkl::uplo::lower` selected will perform the matrix product with $\text{sym}(A) = L + D + L^T$ and for `oneapi::mkl::uplo::upper` will perform the matrix product using $\text{sym}(A) = U^T + D + U$.

10.13.2 API

Syntax

Note: Currently, complex types are not supported.

Using SYCL buffers:

```
namespace oneapi::mkl::sparse {
    void symv (
        sycl::queue &queue,
        oneapi::mkl::uplo uplo_flag,
        const fp alpha,
        oneapi::mkl::sparse::matrix_handle_t handle,
        sycl::buffer<fp, 1> &x,
        const fp beta, sycl::buffer<fp, 1> &y)
}
```

Using USM pointers:

```
namespace oneapi::mkl::sparse {
    sycl::event symv (
        sycl::queue &queue,
        oneapi::mkl::uplo uplo_flag,
        const fp alpha,
        oneapi::mkl::sparse::matrix_handle_t handle,
        fp *x,
        const fp beta,
        fp *y,
        const std::vector<sycl::event> &dependencies = {})
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

uplo_flag Specifies which part is to be processed.

`oneapi::mkl::uplo::lower`	The lower part is used for symmetric product.
`oneapi::mkl::uplo::upper`	The upper part is used for symmetric product.

alpha Specifies the scalar, α.

handle Handle to object containing sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

Note: Currently, the only supported case for `<sparse_matrix_type>` is csr.

---

x SYCL buffer or device-accessible USM pointer of size at least equal to the number of columns, m, of input matrix.

beta Specifies the scalar, β.

y SYCL buffer or device-accessible USM pointer of size at least equal to the number of rows, m, of the input matrix.

dependencies A vector of type `std::vector<sycl::event>` containing the list of events that the `oneapi::mkl::sparse::symv` routine depends on.

Output Parameters

y Overwritten by the updated vector y.

Return Values (USM Only)

sycl::event SYCL event which can be waited upon or added as a dependency for the completion of the `symv` routine.

10.13.3 Examples

An example of how to use `oneapi::mkl::sparse::symv` with SYCL buffers can be found in the oneMKL installation directory, under:

```
examples/dpcpp/sparse_blas/source/sparse_symv.cpp
```

10.14 oneapi::mkl::sparse::trmv

Computes a sparse matrix-dense vector product over upper or lower triangular parts of the matrix.

10.14.1 Description

Note: Refer to Sparse BLAS Supported Data and Integer Types for a list of supported <fp> and <intType> data and integer types and refer to Error Handling for a detailed description of the possible exceptions thrown.

The `oneapi::mkl::sparse::trmv` routine computes a sparse matrix-dense vector product over a triangular part defined as

$$y \leftarrow \alpha \cdot \mathrm{op}(A) \cdot x + \beta \cdot y$$

where α and β are scalars, A is a sparse triangular matrix of size `m` rows by `m` columns and x and y are dense vectors of size `m`. The operation $\mathrm{op}()$ is a matrix modifier:

$$\mathrm{op}(A) = \begin{cases} A, & \text{oneapi::mkl::transpose::nontrans} \\ A^T, & \text{oneapi::mkl::transpose::trans} \\ A^H, & \text{oneapi::mkl::transpose::conjtrans} \end{cases}$$

For a given matrix decomposition into lower, diagonal and upper parts $A = L + D + U$, the triangular matrix vector product with one of `oneapi::mkl::uplo::lower` or `oneapi::mkl::uplo::upper` selected will perform the appropriate matrix product using respectively $\mathrm{op}(L + D)$ or $\mathrm{op}(D + U)$ for `oneapi::mkl::diag::nonunit` or if using `oneapi::mkl::diag::unit`, will perform the appropriate matrix product for $\mathrm{op}(L + I)$ or $\mathrm{op}(I + U)$ where I is the identity matrix.

10.14.2 API

Syntax

Note: Currently, complex types are not supported.

Using SYCL buffers:

```
namespace oneapi::mkl::sparse {
    void trmv (
        sycl::queue &queue,
        oneapi::mkl::uplo uplo_flag,
        oneapi::mkl::transpose transpose_flag,
        oneapi::mkl::diag diag_flag,
        const fp alpha,
        oneapi::mkl::sparse::matrix_handle_t handle,
        sycl::buffer<fp, 1> &x,
        const fp beta, sycl::buffer<fp, 1> &y)
}
```

Using USM pointers:

```
namespace oneapi::mkl::sparse {
    sycl::event trmv(
        sycl::queue &queue,
        oneapi::mkl::uplo uplo_flag,
        oneapi::mkl::transpose transpose_flag,
        oneapi::mkl::diag
        diag_flag,
        const fp alpha,
        oneapi::mkl::sparse::matrix_handle_t handle,
        fp *x,
        const fp beta,
        fp *y,
        const std::vector<sycl::event> &dependencies = {})
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

uplo_flag Specifies which part of the matrix is to be processed.

`oneapi::mkl::uplo::lower`	The lower triangular matrix part is processed.
`oneapi::mkl::uplo::upper`	The upper triangular matrix part is processed.

transpose_flag Specifies operation `op( )` on input matrix.

`oneapi::mkl::transpose::nontrans`	Non-transpose, op$(A) = A$.
`oneapi::mkl::transpose::trans`	Transpose, op$(A) = A^T$.
`oneapi::mkl::transpose::conjtrans`	Conjugate transpose, op$(A) = A^H$.

Note: Currently, the only supported case for `operation` is `oneapi::mkl::transpose::nontrans`.

diag_flag Specifies if the diagonal used for computations is unit or not.

`oneapi::mkl::diag::nonunit`	Diagonal elements might not be equal to one.
`oneapi::mkl::diag::unit`	Diagonal elements are equal to one.

Note: Currently, the only supported case for `diag_flag` is `oneapi::mkl::diag::nonunit`.

alpha Specifies the scalar, α.

handle Handle to object containing sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

Note: Currently, the only supported case for `<sparse_matrix_type>` is csr.

x SYCL buffer or device-accessible USM pointer of size at least equal to the number of columns of input matrix if `transpose_flag` = `oneapi::mkl::transpose::nontrans` and at least the number of rows of input matrix otherwise.

beta Specifies the scalar, β.

y SYCL buffer or device-accessible USM pointer of size at least equal to the number of rows of the input matrix if `transpose_flag` = `oneapi::mkl::transpose::nontrans` and at least the number of columns of the input matrix otherwise.

dependencies A vector of type `std::vector<sycl::event>` containing the list of events that the `oneapi::mkl::sparse::trmv` routine depends on.

Output Parameters

y Overwritten by the updated vector, y.

Return Values (USM Only)

sycl::event SYCL event which can be waited upon or added as a dependency for the completion of the `trmv` routine.

10.14.3 Examples

An example of how to use `oneapi::mkl::sparse::trmv` with SYCL buffers can be found in the oneMKL installation directory, under:

```
examples/dpcpp/sparse_blas/source/sparse_trmv.cpp
```

10.15 oneapi::mkl::sparse::trsv

Solves a system of linear equations for a triangular sparse matrix.

10.15.1 Description

Note: Refer to Sparse BLAS Supported Data and Integer Types for a list of supported <fp> and <intType> data and integer types and refer to Error Handling for a detailed description of the possible exceptions thrown.

The `oneapi::mkl::sparse::trsv` routine solves the sparse triangular system

$$\text{op}(A) \cdot y = x$$

where A is a sparse triangular matrix of size m rows by m columns and op() is a matrix modifier:

$$\text{op}(A) = \begin{cases} A, & \text{oneapi::mkl::transpose::nontrans} \\ A^T, & \text{oneapi::mkl::transpose::trans} \\ A^H, & \text{oneapi::mkl::transpose::conjtrans} \end{cases}$$

The dense vectors x and y should be of length at least m. The vector x is input right hand side data and math:**y** is the resulting output vector.

For a given matrix decomposition into lower, diagonal and upper parts $A = L + D + U$, the triangular solve with one of `oneapi::mkl::uplo::lower` or `oneapi::mkl::uplo::upper` selected will perform the appropriate forward or backward substitution using respectively op$(L+D)$ or op$(D+U)$ for `oneapi::mkl::diag::nonunit` or if using `oneapi::mkl::diag::unit`, will perform the appropriate forward or backward substitution for op$(L+I)$ or op$(I+U)$ where I is the identity matrix.

10.15.2 API

Syntax

Note: Currently, complex types are not supported.

Using SYCL buffers:

```
namespace oneapi::mkl::sparse {
    void trsv (
        sycl::queue &queue,
        oneapi::mkl::uplo uplo_flag,
        oneapi::mkl::transpose transpose_flag,
        oneapi::mkl::diag diag_flag,
        oneapi::mkl::sparse::matrix_handle_t handle,
        sycl::buffer<fp, 1> &x,
        sycl::buffer<fp, 1> &y)
}
```

Using USM pointers:

```
namespace oneapi::mkl::sparse {
    sycl::event trsv(
        sycl::queue &queue,
        oneapi::mkl::uplo uplo_flag,
        oneapi::mkl::transpose transpose_flag,
        oneapi::mkl::diag diag_flag,
        oneapi::mkl::sparse::matrix_handle_t handle,
        fp *x,
        fp *y,
        const std::vector<sycl::event> &dependencies = {})
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

uplo_flag Specifies which part of the matrix is to be processed.

`oneapi::mkl::uplo::lower`	The lower triangular matrix part is processed.
`oneapi::mkl::uplo::upper`	The upper triangular matrix part is processed.

transpose_flag Specifies operation `op()` on input matrix.

`oneapi::mkl::transpose::nontrans`	Non-transpose, op$(A) = A$.
`oneapi::mkl::transpose::trans`	Transpose, op$(A) = A^T$.
`oneapi::mkl::transpose::conjtrans`	Conjugate transpose, op$(A) = A^H$.

Note: Currently, the only supported case for operation is `oneapi::mkl::transpose::nontrans`.

diag_flag Specifies if the diagonal used for computations is unit or based on provided matrix data.

`oneapi::mkl::diag::nonunit`	Diagonal elements are used as provided in the sparse matrix.
`oneapi::mkl::diag::unit`	The value of one is substituted for the diagonal elements in the triangular solve algorithm.

Note: If `oneapi::mkl::diag::nonunit` is selected, all diagonal values must be present in the sparse matrix. This is not necessary for the `oneapi::mkl::diag::unit` case.

handle Handle to object containing sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

Note: Currently, the only supported case for `<sparse_matrix_type>` is csr.

x SYCL buffer or device-accessible USM pointer of size at least equal to the number of columns of input matrix if `transpose_flag` = `oneapi::mkl::transpose::nontrans` and at least the number of rows of input matrix otherwise. It is the input vector x

dependencies A vector of type `std::vector<sycl::event>` containing the list of events that the `oneapi::mkl::sparse::trsv` routine depends on.

Output Parameters

y SYCL buffer or device-accessible USM pointer of size at least equal to the number of rows of the input matrix if `transpose_flag` = `oneapi::mkl::transpose::nontrans` and at least the number of columns of the input matrix otherwise. The solution of the triangular solve is filled into this array.

Return Values (USM Only)

sycl::event SYCL event which can be waited upon or added as a dependency for the completion of the `trsv` routine.

10.15.3 Examples

An example of how to use `oneapi::mkl::sparse::trsv` with SYCL buffers or USM can be found in the oneMKL installation directory, under:

```
examples/dpcpp/sparse_blas/source/sparse_trsv.cpp
```

```
examples/dpcpp/sparse_blas/source/sparse_trsv_usm.cpp
```

10.16 oneapi::mkl::sparse::gemm

Computes a sparse matrix-dense matrix product.

10.16.1 Description

Note: Refer to Sparse BLAS Supported Data and Integer Types for a list of supported <fp> and <intType> data and integer types and refer to Error Handling for a detailed description of the possible exceptions thrown.

The `oneapi::mkl::sparse::gemm` routine computes a sparse matrix-dense matrix product defined as

$$C \leftarrow \alpha \cdot \mathrm{op}(A) \cdot \mathrm{op}(B) + \beta \cdot C$$

where: α and β are scalars, A is a sparse matrix of size `num_rows` rows by `num_cols` columns, op() is a matrix modifier for A and B using the following description:

$$\mathrm{op}(A) = \begin{cases} A, & \text{oneapi::mkl::transpose::nontrans} \\ A^T, & \text{oneapi::mkl::transpose::trans} \\ A^H, & \text{oneapi::mkl::transpose::conjtrans.} \end{cases}$$

The dense matrix objects B and C are stored with row-major or column-major layout and have appropriately sized number of rows for the matrix product and `columns` number of columns.

10.16.2 API

Syntax

Note: Currently, complex types are not supported.

Using SYCL buffers:

```
namespace oneapi::mkl::sparse {
    void gemm(sycl::queue &queue,
      oneapi::mkl::layout dense_matrix_layout,
      oneapi::mkl::transpose transpose_A,
      oneapi::mkl::transpose transpose_B,
      const fp alpha,
      matrix_handle_t handle,
      sycl::buffer<fp, 1> &b,
      const std::int64_t columns,
      const std::int64_t ldb,
      const fp beta,
      sycl::buffer<fp, 1> &c,
      const std::int64_t ldc);
}
```

```
namespace oneapi::mkl::sparse {
   [[deprecated("Use oneapi::mkl::sparse::gemm(queue, oneapi::mkl::layout::row_major, transpose_
↪A, oneapi::mkl::transpose:nontrans, alpha, ... ) instead.")]]
    void gemm (
        sycl::queue &queue,
        oneapi::mkl::transpose transpose_flag,
        const fp alpha,
        oneapi::mkl::sparse::matrix_handle_t handle,
        sycl::buffer<fp, 1> &b,
        const std::int64_t columns,
        const std::int64_t ldb,
        const fp beta,
        sycl::buffer<fp, 1> &c,
        const std::int64_t ldc)
}
```

Using USM pointers:

```
namespace oneapi::mkl::sparse {
    sycl::event gemm(
        sycl::queue &queue,
        oneapi::mkl::layout dense_matrix_layout,
        oneapi::mkl::transpose transpose_A,
        oneapi::mkl::transpose transpose_B,
        const fp alpha,
        matrix_handle_t handle,
        fp *b,
        const std::int64_t columns,
        const std::int64_t ldb,
        const fp beta,
        fp *c,
        const std::int64_t ldc,
        const std::vector<sycl::event> &dependencies = {});
}
```

```
namespace oneapi::mkl::sparse {
   [[deprecated("Use oneapi::mkl::sparse::gemm(queue, oneapi::mkl::layout::row_major, transpose_
↪A, oneapi::mkl::transpose:nontrans,     alpha, ... ) instead.")]]
```

(continues on next page)

(continued from previous page)

```
    sycl::event gemm (
       sycl::queue &queue,
       oneapi::mkl::transpose transpose_flag,
       const fp alpha,
       oneapi::mkl::sparse::matrix_handle_t handle,
       const fp *b,
       const std::int64_t columns,
       const std::int64_t ldb,
       const fp beta,
       fp *c,
       const std::int64_t ldc,
       const std::vector<sycl::event> &dependencies = {})
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

dense_matrix_layout Specifies the storage scheme in memory for the dense matrices. Note that this layout applies to both `B` and `C` dense matrices.

transpose_A (in old API, transpose_flag) Specifies operation `op()` on input matrix `A`.

`oneapi::mkl::transpose::nontrans`	Non-transpose, op$(A) = A$.
`oneapi::mkl::transpose::trans`	Transpose, op$(A) = A^T$.
`oneapi::mkl::transpose::conjtrans`	Conjugate transpose, op$(A) = A^H$.

Note: Currently, the only supported case for operation is `oneapi::mkl::transpose::nontrans`.

transpose_B Specifies operation `op()` on input matrix `B`.

`oneapi::mkl::transpose::nontrans`	Non-transpose, op$(B) = B$.
`oneapi::mkl::transpose::trans`	Transpose, op$(B) = B^T$.
`oneapi::mkl::transpose::conjtrans`	Conjugate transpose, op$(B) = B^H$.

Note: Currently, the only supported case for operation is `oneapi::mkl::transpose::nontrans`.

alpha Specifies the scalar, α.

handle Handle to object containing sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

Note: Currently, the only supported case for <sparse_matrix_type> is csr.

---

b SYCL buffer or device-accessible USM pointer of size at least `rows*cols`, where (with the assumption of `transpose_B == oneapi::mkl::transpose::nontrans`).

	`layout=oneapi::mkl::layout::col-major`	`layout=oneapi::mkl::layout::row-major`
rows (number of rows in B)	ldb	if op$(A) = A$, number of columns in A if op$(A) = A^T$, number of rows in A
cols (number of columns in B)	columns	ldb

columns Number of columns of matrix C.

ldb Specifies the leading dimension of matrix B. Must be positive, and at least `columns` if `dense_matrix_layout=oneapi::mkl::layout::row-major` or at least number of columns in A if `dense_matrix_layout=oneapi::mkl::layout::col-major`.

beta Specifies the scalar, β.

c SYCL buffer or device-accessible USM pointer of size at least `rows*cols`, where:

	`layout=oneapi::mkl::layout::col-major`	`layout=oneapi::mkl::layout::row-major`
rows (number of rows in C)	ldc	if op$(A) = A$, number of rows in A if op$(A) = A^T$, number of columns in A
cols (number of columns in C)	columns	ldc

ldc Specifies the leading dimension of matrix C. Must be positive, and at least `columns` if `dense_matrix_layout=oneapi::mkl::layout::row-major` or at least number of rows in A if `dense_matrix_layout=oneapi::mkl::layout::col-major`.

dependencies A vector of type `std::vector<sycl::event>` containing the list of events that the `oneapi::mkl::sparse::gemm` routine depends on.

Output Parameters

c Overwritten by the updated matrix C.

Return Values (USM Only)

sycl::event SYCL event which can be waited upon or added as a dependency for the completion of the `gemm` routine.

10.16.3 Examples

An example of how to use `oneapi::mkl::sparse::gemm` with SYCL buffers or USM can be found in the oneMKL installation directory, under:

```
examples/dpcpp/sparse_blas/source/sparse_gemm_row_major.cpp
```

```
examples/dpcpp/sparse_blas/source/sparse_gemm_row_major_usm.cpp
```

```
examples/dpcpp/sparse_blas/source/sparse_gemm_col_major.cpp
```

```
examples/dpcpp/sparse_blas/source/sparse_gemm_col_major_usm.cpp
```

10.17 oneapi::mkl::sparse::init_matmat_descr

Allocates and initializes a `oneapi::mkl::sparse::matmat_descr_t` object to default values.

10.17.1 Description

The `oneapi::mkl::sparse::init_matmat_descr` routine allocates and initializes the `oneapi::mkl::sparse::matmat_descr_t` object with default values, otherwise it throws an exception.

Note: Refer to Error Handling for a detailed description of the exceptions thrown.

A common usage model for the `matmat` descriptor is the following

```
using namespace oneapi::mkl;
sparse::matmat_descr_t descr = NULL;
sparse::init_matmat_descr(descr);

// example descriptor for general
// C = A * B^T
sparse::matrix_view_descr viewA = sparse::matrix_view_descr::general;
sparse::matrix_view_descr viewB = sparse::matrix_view_descr::general;
sparse::matrix_view_descr viewC = sparse::matrix_view_descr::general;
transpose opA = transpose::nontrans;
```

(continues on next page)

(continued from previous page)

```
transpose opB = transpose::trans;

sparse::set_matmat_data(descr, viewA, opA, viewB, opB, viewC);

// use descr in sparse::matmat() api

sparse::release_matmat_descr(descr);
```

10.17.2 API

Syntax

```
namespace oneapi::mkl::sparse {
    void init_matmat_descr ( oneapi::mkl::sparse::matmat_descr_t *descr );
}
```

Include Files

- oneapi/mkl/spblas.hpp

10.18 oneapi::mkl::sparse::set_matmat_data

Sets the appropriate `oneapi::mkl::sparse::matrix_view_descr` and `oneapi::mkl::transpose` values in the `oneapi::mkl::sparse::matmat_descr_t` object reflecting the `sparse::matmat` operation to be performed: $C = \text{op}(A) \cdot \text{op}(B)$.

10.18.1 Description

The `oneapi::mkl::sparse::set_matmat_data` routine allows user to set the desired sparse matrix - sparse matrix operation in the `oneapi::mkl::sparse::matmat_descr_t` object which will be used in the sparse::matmat routine.

The `oneapi::mkl::sparse::matrix_view_descr` enum class is defined in the `oneapi/mkl/spblas.hpp` header file

```
namespace oneapi::mkl::sparse {
    enum class matrix_view_descr : std::int32_t {
        general
    };
}
```

where general view assumes all data is populated in the `sparse::matrix_handle_t` object for both lower, diagonal and upper portions of the matrix. A common usage model for the `matmat` descriptor is the following:

```
using namespace oneapi::mkl;
sparse::matmat_descr_t descr = NULL;
sparse::init_matmat_descr(descr);

// example descriptor for general
// C = A * B
sparse::matrix_view_descr viewA = sparse::matrix_view_descr::general;
sparse::matrix_view_descr viewB = sparse::matrix_view_descr::general;
sparse::matrix_view_descr viewC = sparse::matrix_view_descr::general;
transpose opA = transpose::nontrans;
transpose opB = transpose::nontrans;

sparse::set_matmat_data(descr, viewA, opA, viewB, opB, viewC);

// use descr in sparse::matmat() api

sparse::release_matmat_descr(descr);
```

10.18.2 API

Syntax

```
namespace oneapi::mkl::sparse {
    void set_matmat_descr (
        sparse::matmat_descr_t    descr,
        sparse::matrix_view_descr viewA,
        transpose                 opA,
        sparse::matrix_view_descr viewB,
        transpose                 opB,
        sparse::matrix_view_descr viewC);
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

viewA, viewB, viewC `sparse::matrix_view_descr` enum values describing how the A, B and C matrix representations should be viewed. Currently, only the **general** type is supported.

opA, opB Specifies operation `op()` on input matrix.

`oneapi::mkl::transpose::nontrans`	Non-transpose, $\text{op}(A) = A$.
`oneapi::mkl::transpose::trans`	Transpose, $\text{op}(A) = A^T$.
`oneapi::mkl::transpose::conjtrans`	Conjugate transpose, $\text{op}(A) = A^H$.

Note: Currently, the only supported case for operation is `oneapi::mkl::transpose::nontrans`.

Output Parameters

descr `sparse::matmat_descr_t` object used to define the sparse matrix - sparse matrix operation to be performed by the sparse::matmat routine.

10.19 oneapi::mkl::sparse::get_matmat_data

Queries the `oneapi::mkl::sparse::matrix_view_descr` and `oneapi::mkl::transpose` values in the `oneapi::mkl::sparse::matmat_descr_t` object reflecting the `sparse::matmat` operation to be performed: $C = \text{op}(A) \cdot \text{op}(B)$.

10.19.1 Description

The `oneapi::mkl::sparse::get_matmat_data` routine allows user to query the enum values set for the sparse matrix - sparse matrix operation in the `oneapi::mkl::sparse::matmat_descr_t` object which will be used in the sparse::matmat routine.

The `oneapi::mkl::sparse::matrix_view_descr` enum class is defined in the `oneapi/mkl/spblas.hpp` header file

```
namespace oneapi::mkl::sparse {
    enum class matrix_view_descr : std::int32_t {
        general
    };
}
```

where general view assumes all data is populated in the `sparse::matrix_handle_t` object for both lower, diagonal and upper portions of the matrix. A common usage model for the `matmat` descriptor is the following:

```
using namespace oneapi::mkl;
sparse::matmat_descr_t descr = NULL;
sparse::init_matmat_descr(descr);
// someone sets matmat descr

// query what was in descr
sparse::matrix_view_descr viewA, viewB, viewC;
transpose opA, opB;
sparse::get_matmat_data(descr, viewA, opA, viewV, opB, viewC);

sparse::release_matmat_descr(descr);
```

10.19.2 API

Syntax

```
namespace oneapi::mkl::sparse {
    void get_matmat_descr (
        sparse::matmat_descr_t    descr,
        sparse::matrix_view_descr &viewA,
        transpose                 &opA,
        sparse::matrix_view_descr &viewB,
        transpose                 &opB,
        sparse::matrix_view_descr &viewC);
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

descr `sparse::matmat_descr_t` object used to define the sparse matrix - sparse matrix operation to be performed by the sparse::matmat routine.

Output Parameters

viewA, viewB, viewC `sparse::matrix_view_descr` enum values describing how the A, B and C matrix representations should be viewed. Currently, only the **general** type is supported.

opA, opB Specifies operation `op()` on input matrix.

`oneapi::mkl::transpose::nontrans`	Non-transpose, op$(A) = A$.
`oneapi::mkl::transpose::trans`	Transpose, op$(A) = A^T$.
`oneapi::mkl::transpose::conjtrans`	Conjugate transpose, op$(A) = A^H$.

Note: Currently, the only supported case for operation is `oneapi::mkl::transpose::nontrans`.

10.20 oneapi::mkl::sparse::release_matmat_descr

Releases `oneapi::mkl::sparse::matmat_descr_t` object and sets it to NULL.

10.20.1 Description

The `oneapi::mkl::sparse::release_matmat_descr` routine releases the `oneapi::mkl::sparse::matmat_descr_t` object and sets it to NULL.

A common usage model for the `matmat` descriptor is the following

```
using namespace oneapi::mkl;
sparse::matmat_descr_t descr = NULL;
sparse::init_matmat_descr(descr);

// example descriptor for general
// C = A * B
sparse::matrix_view_descr viewA = sparse::matrix_view_descr::general;
sparse::matrix_view_descr viewB = sparse::matrix_view_descr::general;
sparse::matrix_view_descr viewC = sparse::matrix_view_descr::general;
transpose opA = transpose::nontrans;
transpose opB = transpose::nontrans;

sparse::set_matmat_data(descr, viewA, opA, viewB, opB, viewC);

// use descr in sparse::matmat() api

sparse::release_matmat_descr(descr);
```

10.20.2 API

Syntax

```
namespace oneapi::mkl::sparse {
    void release_matmat_descr ( oneapi::mkl::sparse::matmat_descr_t *descr );
}
```

Include Files

- oneapi/mkl/spblas.hpp

10.21 oneapi::mkl::sparse::matmat

Computes a sparse matrix-sparse matrix product.

10.21.1 Description

Note: Refer to Sparse BLAS Supported Data and Integer Types for a list of supported <fp> and <intType> data and integer types and refer to Error Handling for a detailed description of the possible exceptions thrown.

The `oneapi::mkl::sparse::matmat` routine computes a sparse matrix-sparse matrix product defined as

$$C = \text{op}(A) \cdot \text{op}(B)$$

where A, B and C are appropriately sized sparse matrices and op() is a matrix modifier:

$$\text{op}(A) = \begin{cases} A, & \text{oneapi::mkl::transpose::nontrans} \\ A^T, & \text{oneapi::mkl::transpose::trans} \\ A^H, & \text{oneapi::mkl::transpose::conjtrans.} \end{cases}$$

The sparse matrices are stored in the `matrix_handle_t` and currently only support the compressed sparse row (CSR) matrix format.

As the size of C and its data is generally not known before hand, the `matmat` routine is broken into several stages which allow you to query the size of the data arrays, allocate them and then pass them back into the routine to be filled. This allows you to control all the C matrix data allocations themselves. Additionally, there are cases where only the sparsity pattern of C is desired, and this routine allows you to compute C without the values array. Generally the `sparse::matmat()` algorithm is broken into three computational stages:

Stage	Description
work_estimation	do initial estimation of work and load balancing (make upper bound estimate on size of C matrix data).
compute/compute_structure	do internal products for computing the C matrix including the calculation of size of C matrix data and filling the row pointer array for C.
finalize/finalize_structure	do any remaining internal products and accumulation and transfer into final C matrix arrays.

Some additional helper stages are provided to allow you to query sizes of temporary workspace arrays or the size of the C matrix data (`nnz(C)`) to be allocated. They are set and passed to the `sparse::matmat` routine as `matmat_request` enum values:

```
namespace oneapi::mkl::sparse {
    enum class matmat_request : std::int32_t {
        get_work_estimation_buf_size,
        work_estimation,

        get_compute_structure_buf_size,
        compute_structure,
        finalize_structure,

        get_compute_buf_size,
        compute,
```

(continues on next page)

(continued from previous page)

```
        get_nnz,
        finalize
    };
}
```

A common workflow involves calling `sparse::matmat()` several times with different `matmat_request`'s:

0. **Before matmat stages**
    1. Allocate C matrix row pointer array and input into C matrix handle with dummy arguments for column and data arrays (as their sizes are not known yet).
1. **work_estimation stage**
    1. Call matmat with `matmat_request::get_work_estimation_buf_size`.
    2. Allocate the work estimation temporary workspace array.
    3. Call matmat with `matmat_request::work_estimation`.
2. **Compute stage**
    1. Call matmat with `matmat_request::get_compute_buf_size`.
    2. Allocate the compute temporary workspace array.
    3. Call matmat with `matmat_request::compute`.
3. **Finalize stage**
    1. Call matmat with `matmat_request::get_nnz`.
    2. Allocate the C matrix column and data arrays and input into `C` matrix handle.
    3. Call matmat with `matmat_request::finalize`.
4. **After matmat stages**
    1. Release or reuse the matmat descriptor for another appropriate sparse matrix product.
    2. Release any temporary workspace arrays allocated through the stages for this particular sparse matrix product.
    3. Release or use C matrix handle for subsequent operations.

Note that the `compute_structure` and `finalize_structure` and their helpers should be used if the final result desired is the sparsity pattern of C.

If you do not wish to allocate and handle the temporary workspace arrays themselves, they have the simplifying option to skip the `get_xxx_buf_size` queries for the `work_estimation` and `compute`/`compute_structure` stages and pass in null pointers for the size and `tempBuffer` arguments in the API for those stages. In this case, the library handles the allocation and memory management themselves, living until the C matrix handle is destroyed. However, you are always expected to query the size of C matrix data and allocate the C matrix arrays themselves.

This simplified workflow is reflected here:

0. **Before matmat stages**
    1. Allocate C matrix row_pointer array and input into C matrix handle with dummy arguments for column and data arrays (as their sizes are not known yet).
1. **work_estimation stage**
    1. Call matmat with the `matmat_request::work_estimation` and `nullptr` for `sizeTempBuffer` and `tempBuffer` arguments.
2. **Compute stage**
    1. Call matmat with `matmat_request::compute` and `nullptr` for the `sizeTempBuffer` and `tempBuffer` arguments.
3. **Finalize stage**
    1. Call matmat with `matmat_request::get_nnz`.
    2. Allocate the C matrix column and data arrays and input into C matrix handle.
    3. Call matmat with `matmat_request::finalize`.
4. **After matmat stages**
    1. Release or reuse the matmat descriptor for another appropriate sparse matrix product.
    2. Release or use the C matrix handle for subsequent operations.

These two workflows, and additionally, an example of computing only the sparsity pattern for C are demonstrated in the oneMKL DPC++ examples listed below.

10.21.2 API

Syntax

Note: Currently, complex types are not supported.

Using SYCL buffers:

```
namespace oneapi::mkl::sparse {
   void matmat(sycl::queue &queue,
               sparse::matrix_handle_t A,
               sparse::matrix_handle_t B,
               sparse::matrix_handle_t C,
               sparse::matmat_request req,
               sparse::matmat_descr_t descr,
               sycl::buffer<std::int64_t, 1> *sizeTempBuffer,
               sycl::buffer<std::uint8_t, 1> *tempBuffer);
}
```

Using USM pointers:

```
namespace oneapi::mkl::sparse {
   sycl::event matmat(sycl::queue &queue,
                      sparse::matrix_handle_t A,
                      sparse::matrix_handle_t B,
                      sparse::matrix_handle_t C,
                      sparse::matmat_request req,
                      sparse::matmat_descr_t descr,
                      std::int64_t *sizeTempBuffer,
                      void *tempBuffer,
                      const std::vector<sycl::event> &dependencies);
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

A The matrix handle for the first matrix in the sparse matrix - sparse matrix product.

B The matrix handle for the second matrix in the sparse matrix - sparse matrix product.

C The output matrix handle from the matmat operation. Sparse Matrix format arrays will be allocated by the user and put into the matrix handle using a **set_xxx_data** routine. The data will be filled by the library as part of the matmat operation.

request The **matmat_request** stage in the multi-stage algorithm. See descriptions of common workflows above.

descr The **matmat_descr_t** object describing the sparse matrix-sparse matrix operation to be executed. It is manipulated using the **sparse::init_matmat_descr**, **sparse::set_matmat_data** and **sparse::release_matmat_descr** routines.

sizeTempBuffer A SYCL aware container (sycl::buffer or host-accessible USM pointer) of the length of one **std::int64_t** to represent the size in bytes of the **tempBuffer**. For the **matmat_request** stages with the **get_xxx** naming convention the value is set by the library to inform the user how much memory to allocate in the temporary buffer. In the other **work_estimation** and **compute/comute_structure** stages, it is passed in along with the temporary buffer, **tempBuffer**, informing the library how much space was provided in bytes.

tempBuffer A SYCL-aware container (sycl::buffer or device-accessible USM pointer) of **sizeTempBuffer** bytes used as a temporary workspace in the algorithm. There are two stages where separate workspaces must be passed into the matmat api (**work_estimation** and **compute/compute_structure**). They should remain valid through the full matmat multi-stage algorithm as both may be used until the last **finalize/finalize_structure** request is completed.

dependencies (USM APIs only) A vector of type `std::vector<sycl::event>` containing the list of events that the current stage of `oneapi::mkl::sparse::matmat` routine depends on.

Output Parameters

C Data arrays for C will be allocated by the user and filled by the library as part of the matmat algorithm. The output sparse matrix data arrays for C are not guaranteed to be sorted, but sparse::sort_matrix() is provided in case the sorted property is desired for subsequent operations with the output sparse matrix.

Return Values (USM Only)

sycl::event SYCL event which can be waited upon or added as a dependency for the completion of the stages of the `matmat` routine.

10.21.3 Examples

Some examples of how to use `oneapi::mkl::sparse::matmat` with SYCL buffers or USM can be found in the oneMKL installation directory, under:

```
examples/dpcpp/sparse_blas/source/sparse_matmat.cpp
examples/dpcpp/sparse_blas/source/sparse_matmat_simplified.cpp
examples/dpcpp/sparse_blas/source/sparse_matmat_structure_only.cpp
```

```
examples/dpcpp/sparse_blas/source/sparse_matmat_usm.cpp
examples/dpcpp/sparse_blas/source/sparse_matmat_simplified_usm.cpp
examples/dpcpp/sparse_blas/source/sparse_matmat_structure_only_usm.cpp
```

10.22 oneapi::mkl::sparse::omatcopy

Performs an out-of-place copy of a CSR matrix handle into a new one.

10.22.1 Description

Note: Refer to Error Handling for a detailed description of the possible exceptions thrown.

Note: This routine modifies arrays provided to oneMKL through the matrix handle creation routines, `oneapi::mkl::sparse::set_<sparse_matrix_type>_data`. Arrays of only the output matrix handle are modified, and those of the input handle are not.

The `oneapi::mkl::sparse::omatcopy` copies the user-provided sparse matrix arrays stored in a given sparse matrix handle into those provided in another sparse matrix handle. The routine allows for a transpose operation and a change in array indexing. Only out-of-place copy/transpose is supported through this API.

10.22.2 API

Syntax

Note: Currently, complex types are not supported.

Using USM and SYCL buffers:

```
namespace oneapi::mkl::sparse {
    sycl::event omatcopy (
        sycl::queue &queue,
        oneapi::mkl::transpose transpose_flag,
        oneapi::mkl::sparse::matrix_handle_t from_handle,
        oneapi::mkl::sparse::matrix_handle_t to_handle,
        const std::vector<sycl::event> &dependencies = {});
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

transpose_flag Specifies operation `op()` on input matrix.

oneapi::mkl::transpose::nontrans ▪ Non-transpose, `op(A)` = `A`.
oneapi::mkl::transpose::trans ▪ Transpose, `op(A)` = A^T.
oneapi::mkl::transpose::conjtrans ▪ Conjugate transpose, `op(A)` = A^H.

Note: Currently, only operations `oneapi::mkl::transpose::nontrans` and `oneapi::mkl::transpose::trans` are supported.

from_handle Handle to object containing input sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

Note: Currently, the only supported matrix format for `<sparse_matrix_type>` is the **csr** format.

to_handle Handle to object containing output sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines. Note that the sparse matrix dimensions/array lengths of this output handle must be same as that of the input handle in case of `oneapi::mkl::transpose::nontrans` operation. The number of rows and columns of this output matrix handle for `oneapi::mkl::transpose::trans` and `oneapi::mkl::transpose::conjtrans` operations must be the number of columns and rows of the input matrix handle, respectively. Also note that in case of a transpose operation, the number of non-zeros in the sparse matrix remains unchanged.

Note: Currently, `csr` is the only supported matrix format for `<sparse_matrix_type>`.

---

dependencies A vector of type `std::vector<sycl::event> &` containing the list of events that the `oneapi::mkl::sparse::omatcopy` routine depends on.

Return Values

`sycl::event` SYCL event which can be waited upon or added as a dependency for the completion of the `omatcopy` routine.

```
// Examples of API arguments and usage
sycl::event ev_opt = sparse::omatcopy(queue, trans_val, from_handle, to_handle);
↪    // Allowed use in case of USM
sycl::event ev_opt = sparse::omatcopy(queue, trans_val, from_handle, to_handle, {});
↪    // Allowed use in case of USM
sycl::event ev_opt = sparse::omatcopy(queue, trans_val, from_handle, to_handle,
↪dependencies); // Allowed use in case of USM
sparse::omatcopy(queue, trans_val, from_handle, to_handle);
↪    // Recommended use in case of sycl::buffer
sparse::omatcopy(queue, trans_val, from_handle, to_handle, {});
↪    // Not recommended in case of sycl::buffer, but supported;
sparse::omatcopy(queue, trans_val, from_handle, to_handle, dependencies);
↪    // Not recommended in case of sycl::buffer, but supported;
sycl::event ev_opt = sparse::omatcopy(queue, trans_val, from_handle, to_handle);
↪    // Not recommended in case of sycl::buffer, but supported. Note that most sycl::
↪buffer oneMKL APIs do not have ability to pass in dependencies
```

10.23 oneapi::mkl::sparse::sort_matrix

Performs in-place sorting of user-provided matrix arrays in a matrix handle.

10.23.1 Description

Note: Refer to Error Handling for a detailed description of the possible exceptions thrown.

The `oneapi::mkl::sparse::sort_matrix` API performs in-place sorting of user-provided sparse matrix arrays stored in a given sparse matrix handle.

Note: This routine modifies user-arrays provided to oneMKL through the matrix handle creation routines, `oneapi::mkl::sparse::set_<sparse_matrix_type>_data`.

10.23.2 API

Syntax

Note: Currently, complex types are not supported.

Using USM and SYCL buffers:

```
namespace oneapi::mkl::sparse {
    sycl::event sort_matrix (
        sycl::queue &queue,
        oneapi::mkl::sparse::matrix_handle_t handle,
        const std::vector<sycl::event> &dependencies = {});
}
```

Include Files

- oneapi/mkl/spblas.hpp

Input Parameters

queue Specifies the SYCL command queue which will be used for SYCL kernels execution.

handle Handle to object containing sparse matrix and other internal data. Created using one of the `oneapi::mkl::sparse::set_<sparse_matrix_type>_data` routines.

> **Note:** Currently, the only supported matrix format for `<sparse_matrix_type>` is the **csr** format.

dependencies A vector of type `std::vector<sycl::event> &` containing the list of events that the `oneapi::mkl::sparse::sort_matrix` routine depends on.

Return Values

`sycl::event` SYCL event which can be waited upon or added as a dependency for the completion of the `sort_matrix` routine.

```
// Examples of API arguments and usage
sycl::event ev_opt = sparse::sort_matrix(queue, handle);              // Allowed use in␣
↪case of USM
sycl::event ev_opt = sparse::sort_matrix(queue, handle, {});          // Allowed use in␣
↪case of USM
sycl::event ev_opt = sparse::sort_matrix(queue, handle, dependencies); // Allowed use in␣
↪case of USM
sparse::sort_matrix(queue, handle);                                   // Recommended use␣
↪in case of sycl::buffer
sparse::sort_matrix(queue, handle, {});                               // Not recommended␣
↪in case of sycl::buffer, but supported;
sparse::sort_matrix(queue, handle, dependencies);                     // Not recommended␣
↪in case of sycl::buffer, but supported;
sycl::event ev_opt = sparse::sort_matrix(queue, handle);              // Not recommended␣
↪in case of sycl::buffer, but supported. Note that most sycl::buffer oneMKL APIs do not␣
↪have ability to pass in dependencies
```

11.0 LAPACK Routines

Intel® oneAPI Math Kernel Library (oneMKL) implements routines from the LAPACK package that are used for solving systems of linear equations, linear least squares problems, eigenvalue and singular value problems, and performing a number of related computational tasks. The library includes LAPACK routines for both real and complex data. Routines are supported for systems of equations with the following types of matrices:

- General
- Banded
- Symmetric or Hermitian positive-definite (full, packed, and rectangular full packed (RFP) storage)
- Symmetric or Hermitian positive-definite banded
- Symmetric or Hermitian indefinite (both full and packed storage)
- Symmetric or Hermitian indefinite banded
- Triangular (full, packed, and RFP storage)
- Triangular banded
- Tridiagonal
- Diagonally dominant tridiagonal.

Note: Different arrays used as parameters to oneMKL LAPACK routines must not overlap.

Warning: LAPACK routines assume that input matrices do not contain IEEE 754 special values such as INF or NaN values. Using these special values may cause LAPACK to return unexpected results or become unstable.

11.1 gebrd

Reduces a general matrix to bidiagonal form. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.1.1 Description

The routine reduces a general m-by-n matrix A to a bidiagonal matrix B by an orthogonal (unitary) transformation.

If m ≥ n, the reduction is given by

$$A = QBP^H = \begin{pmatrix} B_1 \\ 0 \end{pmatrix} P^H = Q_1 B_1 P_H$$

where B_1 is an n-by-n upper diagonal matrix, Q and P are orthogonal or, for a complex A, unitary matrices; Q_1 consists of the first n columns of Q.

If m < n, the reduction is given by

A = Q*B*PH = Q*(B10)*PH = Q1*B1*P1H,

where B_1 is an m-by-m lower diagonal matrix, Q and P are orthogonal or, for a complex A, unitary matrices; P_1 consists of the first m columns of P.

The routine does not form the matrices Q and P explicitly, but represents them as products of elementary reflectors. Routines are provided to work with the matrices Q and P in this representation:

If the matrix A is real,

- to compute Q and P explicitly, call **orgbr**.

If the matrix A is complex,

- to compute Q and P explicitly, call **ungbr**.

11.1.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void gebrd(sycl::queue &queue,
  std::int64_t m, std::int64_t n,
  sycl::buffer<T,1> &a, std::int64_t lda,
  sycl::buffer<realT,1> &d, sycl::buffer<realT,1> &e, sycl::buffer<T,1> &tauq, sycl::buffer<T,1>
↪ &taup, sycl::buffer<T,1> &scratchpad, std::int64_t scratchpad_size)
}
```

gebrd supports the following precision and devices.

T	Devices Supported
float	CPU, GPU
double	CPU, GPU
std::complex<float>	CPU, GPU
std::complex<double>	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix A (`0 ≤ m`).

n The number of columns in the matrix A (`0 ≤ n`).

a The buffer holding matrix A. The second dimension of a must be at least `max(1, m)`.

lda The leading dimension of a.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number offloating point elements of typeT.

Size should not be less then the valuereturned by the gebrd_scratchpad_sizefunction.

Output Parameters

a If `m ≥ n`, the diagonal and first super-diagonal of a are overwritten by the upper bidiagonal matrix B. The elements below the diagonal, with the buffer tauq, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the buffer taup, represent the orthogonal matrix P as a product of elementary reflectors.

If m<n, the diagonal and first sub-diagonal of a are overwritten by the lower bidiagonal matrix B. The elements below the first subdiagonal, with the buffer tauq, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the buffer taup, represent the orthogonal matrix P as a product of elementary reflectors.

d Buffer holding array of size at least `max(1, min(m,n))`. Contains the diagonal elements of B.

e Buffer holding array of size at least `max(1, min(m,n) - 1)`. Contains the off-diagonal elements of B.

tauq Buffer holding array of size at least `max(1, min(m, n))`. The scalar factors of the elementary reflectors which represent the orthogonal or unitary matrix Q.

taup Buffer holding array of size at least `max(1, min(m, n))`. The scalar factors of the elementary reflectors which represent the orthogonal or unitary matrix P.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the i-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail()i method of the exception object.

11.2 gebrd (USM Version)

Reduces a general matrix to bidiagonal form. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.2.1 Description

The routine reduces a general `m`-by-`n` matrix `A` to a bidiagonal matrix `B` by an orthogonal (unitary) transformation.

If `m ≥ n`, the reduction is given by

$$A = QBP^H = \begin{pmatrix} B_1 \\ 0 \end{pmatrix}^T P^H = Q_1 B_1 P_H$$

where B_1 is an `n`-by-`n` upper diagonal matrix, `Q` and `P` are orthogonal or, for a complex `A`, unitary matrices; Q_1 consists of the first `n` columns of `Q`.

If `m < n`, the reduction is given by

`A = Q*B*PH = Q*(B10)*PH = Q1*B1*P1H`,

where B_1 is an `m`-by-`m` lower diagonal matrix, `Q` and `P` are orthogonal or, for a complex `A`, unitary matrices; P_1 consists of the first `m` columns of `P`.

The routine does not form the matrices `Q` and `P` explicitly, but represents them as products of elementary reflectors. Routines are provided to work with the matrices `Q` and `P` in this representation:

If the matrix `A` is real,

- to compute `Q` and `P` explicitly, call orgbr (USM Version).

If the matrix `A` is complex,

- to compute `Q` and `P` explicitly, call ungbr (USM Version).

11.2.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event gebrd(sycl::queue &queue,
            std::int64_t m,
            std::int64_t n,
            T *a,
            std::int64_t lda,
            RealT *d,
            RealT *e,
            T *tauq,
            T *taup,
            T *scratchpad,
            std::int64_t scratchpad_size, const std::vector<sycl::event> &events = {})
}
```

`gebrd` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU, GPU
`double`	CPU, GPU
`std::complex<float>`	CPU, GPU
`std::complex<double>`	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

a Pointer to matrix `A`. The second dimension of a must be at least `max(1, m)`.

lda The leading dimension of a.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the gebrd_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a If `m ≥ n`, the diagonal and first super-diagonal of a are overwritten by the upper bidiagonal matrix `B`. The elements below the diagonal, with the tauq, represent the orthogonal matrix `Q` as a product of elementary reflectors, and the elements above the first superdiagonal, with the taup, represent the orthogonal matrix `P` as a product of elementary reflectors.

If `m<n`, the diagonal and first sub-diagonal of a are overwritten by the lower bidiagonal matrix `B`. The elements below the first subdiagonal, with the tauq, represent the orthogonal matrix `Q` as a product of elementary reflectors, and the elements above the diagonal, with the taup, represent the orthogonal matrix `P` as a product of elementary reflectors.

d Pointer to memory of size at least `max(1, min(m,n))`. Contains the diagonal elements of `B`.

e Pointer to memory of size at least `max(1, min(m,n) - 1)`. Contains the off-diagonal elements of `B`.

tauq Pointer to memory of size at least `max(1, min(m, n))`. The scalar factors of the elementary reflectors which represent the orthogonal or unitary matrix `Q`.

taup Pointer to memory of size at least `max(1, min(m, n))`. The scalar factors of the elementary reflectors which represent the orthogonal or unitary matrix `P`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.3 gebrd_scratchpad_size

Computes size of scratchpad memory required for `gebrd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.3.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `gebrd` (buffer or USM version) function should be able to hold.

11.3.2 API

Syntax

```
namespace oneapi::mkl::lapack {
   template<typename T>
   std::int64_t gebrd_scratchpad_size(sycl::queue &queue,
   std::int64_t m,
   std::int64_t n, std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the gebrd (buffer or USM version) function will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

lda The leading dimension of a.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `gebrd` (buffer or USM version) function should be able to hold.

11.4 gels_batch (Buffer Strided Version)

Finds the least squares solutions for a batch of overdetermined linear systems. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.4.1 Description

Finds the least squares solutions for a batch of overdetermined linear systems.

11.4.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event gels_batch(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t nrhs,
  sycl::buffer<T> &a,
```

(continues on next page)

(continued from previous page)

```
  std::int64_t lda,
  std::int64_t stride_a,
  sycl::buffer<T> &b,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	GPU
`double`	GPU
`std::complex<float>`	GPU
`std::complex<double>`	GPU

Input Parameters

queue Device queue where calculations will be performed.

trans Operation assumed to be done on input matrices A_i. Only `trans = mkl::transpose::nontrans` case is currently supported.

m The number of rows of the matrices A_i and B_i

n The number of columns of the matrices A_i

nrhs The number of right-hand sides: the number of columns in B_i

a Contains batch_size m-by-n matrices A_i

lda The leading dimension of A_i

stride_a The stride between the beginnings of matrices A_i inside the batch array a

b Contains the matrices B_i of right hand side vectors

ldb The leading dimensions of B_i

stride_b The stride between the beginnings of matrices B_i inside the batch array b.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by stride version of gels_batch_scratchpad_size (Strided Version) function.

Output Parameters

a Overwritten by the factorization data as follows: contains triangular matrix R obtained on the basis of A_i used in least squares computation

b Overwritten by least squares solutions of the batch of problems

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then A_i does not have full rank, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these matrices using the `infos()` method of the exception object.

11.5 gels_batch (USM Strided Version)

Finds the least squares solutions for a batch of overdetermined linear systems. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.5.1 Description

Finds the least squares solutions for a batch of overdetermined linear systems.

11.5.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event gels_batch(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t nrhs,
  T *a,
  std::int64_t lda,
  std::int64_t stride_a,
  T *b,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	GPU
`double`	GPU
`std::complex<float>`	GPU
`std::complex<double>`	GPU

Input Parameters

queue Device queue where calculations will be performed.

trans Operation assumed to be done on input matrices A_i. Only `trans = mkl::transpose::nontrans` case is currently supported.

m The number of rows of the matrices A_i and B_i

n The number of columns of the matrices A_i

nrhs The number of right-hand sides: the number of columns in B_i

a Contains batch_size m-by-n matrices A_i

lda The leading dimension of A_i

stride_a The stride between the beginnings of matrices A_i inside the batch array a

b Contains the matrices B_i of right hand side vectors

ldb The leading dimensions of B_i

stride_b The stride between the beginnings of matrices B_i inside the batch array b.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by stride version of gels_batch_scratchpad_size (Strided Version) function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by the factorization data as follows: contains triangular matrix R obtained on the basis of A_i used in least squares computation

b Overwritten by least squares solutions of the batch of problems

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then A_i does not have full rank, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.6 gels_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `gels_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.6.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `gels_batch` (Strided Version) function must be able to hold.

11.6.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t gels_batch_scratchpad_size(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t nrhs,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

trans Operation assumed to be done on input matrices A_i. Only `trans = mkl::transpose::nontrans` case is currently supported.

m The number of rows of the matrices A_i and B_i

n The number of columns of the matrices A_i

nrhs The number of right-hand sides: the number of columns in B_i

lda The leading dimension of A_i

stride_a The stride between the beginnings of matrices A_i inside the batch array a

ldb The leading dimensions of B_i

stride_b The stride between the beginnings of matrices B_i inside the batch array b.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `gels_batch` (Strided Version) function must be able to hold.

11.7 geqrf

Computes the QR factorization of a general m-by-n matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.7.1 Description

The routine forms the `QR` factorization of a general `m`-by-`n` matrix `A`. No pivoting is performed.

The routine does not form the matrix `Q` explicitly. Instead, `Q` is represented as a product of min(`m`, `n`) elementary reflectors. Routines are provided to work with `Q` in this representation.

11.7.2 API

Syntax

```
namespace oneapi::mkl::lapack {
   void geqrf(sycl::queue &queue,
   std::int64_t m,
   std::int64_t n,
   sycl::buffer<T> &a,
   std::int64_t lda,
   sycl::buffer<T> &tau,
   sycl::buffer<T> &scratchpad,
   std::int64_t scratchpad_size)
}
```

`geqrf` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix A (`0 ≤ m`).

n The number of columns in A (`0 ≤ n`).

a Buffer holding input matrix A. The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a; at least max(1, m).

scratchpad Buffer holding scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by the geqrf_scratchpad_size function.

Output Parameters

a Overwritten by the factorization data as follows:

The elements on and above the diagonal of the array contain the min(m,n)-by-n upper trapezoidal matrix R (R is upper triangular if m ≥ n); the elements below the diagonal, with the array tau, present the orthogonal matrix Q as a product of min(m,n) elementary reflectors.

tau Array, size at least max (1, min(m, n)). Contains scalars that define elementary reflectors for the matrix Q in its decomposition in a product of elementary reflectors.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.8 geqrf (USM Version)

Computes the QR factorization of a general m-by-n matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.8.1 Description

The routine forms the QR factorization of a general m-by-n matrix A. No pivoting is performed.

The routine does not form the matrix Q explicitly. Instead, Q is represented as a product of min(m, n) elementary reflectors. Routines are provided to work with Q in this representation.

11.8.2 API

Syntax

```
namespace oneapi::mkl::lapack {
   sycl::event geqrf(sycl::queue &queue,
   std::int64_t m,
   std::int64_t n,
   T *a,
   std::int64_t lda,
   T *tau,
   T *scratchpad,
   std::int64_t scratchpad_size,
   const std::vector<sycl::event> &events = {})
}
```

gerqf (USM version) supports the following precisions and devices:

T	Devices supported
float	CPU and GPU
double	CPU and GPU
std::complex<float>	CPU and GPU
std::complex<double>	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix A (0 ≤ m).

n The number of columns in the matrix A (0 ≤ n).

a Pointer to the memory holding input matrix A. The second dimension of a must be at least max(1, n).

lda The leading dimension of a, at least max(1, m).

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the geqrf_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by the factorization data as follows:

The elements on and above the diagonal of the array contain the `min(m,n)`-by-`n` upper trapezoidal matrix `R` (`R` is upper triangular if `m ≥ n`); the elements below the diagonal, with the array tau, present the orthogonal matrix `Q` as a product of `min(m,n)` elementary reflectors.

tau Array, size at least `min(m,n)`.

Contains scalars that define elementary reflectors for the matrix `Q` in its decomposition in a product of elementary reflectors.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.9 geqrf_batch (Buffer Strided Version)

Computes the batch of QR factorizations of a batch of general m-by-n matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.9.1 Description

The routine forms the $Q_i R_i$ factorizations of a general `m`-by-`n` matrices A_i. No pivoting is performed.

The routine does not form the matrix Q_i explicitly. Instead, Q_i is represented as a product of min(`m`, `n`) elementary reflectors. Routines are provided to work with Q_i in this representation.

11.9.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void geqrf_batch(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  std::int64_t stride_a,
  sycl::buffer<T> &tau,
  std::int64_t stride_tau,
  std::int64_t batch_size,
  sycl::buffer<T> &scratchpad, std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i ($0 \leq m$).

n The number of columns in the matrices A_i ($0 \leq n$).

a Array holding input matrices A_i.

lda The leading dimension of A_i .

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

stride_tau The stride between the beginnings of arrays tau_i inside the array `tau`.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by geqrf_batch_scratchpad_size (Strided Version).

Output Parameters

a Overwritten by the factorization data as follows:

The elements on and above the diagonal of the array contain the `min(m,n)`-by-`n` upper trapezoidal matrices R_i (R_i is upper triangular if `m` ≥ `n`); the elements below the diagonal, with the array tau_i, present the orthogonal matrix Q_i as a product of `min(m,n)` elementary reflectors.

tau Array to store batch of tau_i, each of size `min(m,n)`, containing scalars that define elementary reflectors for the matrices Q_i in its decomposition in a product of elementary reflectors.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the `info()` method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and `detail()` returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less than the value returned by the `detail()` method of the exception object.

11.10 geqrf_batch (Group Version)

Computes the batch of QR factorizations of a batch of general m-by-n matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.10.1 Description

The routine forms the Q_iR_i factorizations of a general `m`-by-`n` matrix A_i, `i`$\in${`1...batch_size`}, where `batch_size` is a sum of all parameter group sizes as provided with the `group_sizes` array. No pivoting is performed during factorization.

The routine does not form the matrix Q_i explicitly. Instead, Q_i is represented as a product of min(`m`, `n`) elementary reflectors. Routines are provided to work with Q_i in this representation.

Total number of problems to solve, `batch_size`, is a sum of sizes of all of the groups of parameters as provided by `group_sizes`array.

11.10.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event geqrf_batch(sycl::queue &queue,
  std::int64_t *m,
  std::int64_t *n,
  T **a,
  std::int64_t *lda,
  T **tau,
  std::int64_t group_count,
  std::int64_t *group_sizes,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m Array of `group_count` parameters m_g parameters.

Each of m_g specifies the number of rows in the matrices A_i from array a, belonging to group g.

n Array of `group_count` parameters n_g parameters.

Each of n_g specifies the number of columns in the matrices A_i from array a, belonging to group g.

a Array of `batch_size` pointers to input matrices A_i, each being of size $lda_g * n_g$ (g is an index of group to which A_i belongs).

lda Array of `group_count` lda_g parameters, each representing the leading dimensions of input matrices A_i, from array a, belonging to group g.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of group_count integers. Array element with index g specifies the number of problems to solve for each of the groups of parameters g. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by geqrf_batch_scratchpad_size (Group Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Matrices pointed to by array a are overwritten by the factorization data as follows:

The elements on and above the diagonal of A_i contain the `min(mg,mng)`-by-n_g upper trapezoidal matrices R_i (R_i is upper triangular if `mg` $\geq$ `ng`); the elements below the diagonal, with the array tau_i, present the orthogonal matrix Q_i as a rpoduct of `min(mg,ng)` elementary reflectors.

Here, `g` is an index of parameters group corresponding to `i`-th decomposition.

tau Array of pointers to store tau_i, each of size m_g,n_g, containing scalars that define elementary reflectors for the matrices Q_i in its decomposition in a product of elementary reflectors.

Here, `g` is an index of parameters group corresponding to `i`-th decomposition.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.11 geqrf_batch (USM Strided Version)

Computes the batch of QR factorizations of a batch of general m-by-n matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.11.1 Description

The routine forms the Q_iR_i factorizations of a general `m`-by-n matrix A_i. No pivoting is performed.

The routine does not form the matrix Q_i explicitly. Instead, Q_i is represented as a product of min(`m`, `n`) elementary reflectors. Routines are provided to work with Q_i in this representation.

11.11.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event geqrf_batch(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  std::int64_t stride_a,
  T *tau,
  std::int64_t stride_tau,
  std::int64_t batch_size,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i (`0 ≤ m`).

n The number of columns in the matrices A_i (`0 ≤ n`).

a Array holding input matrices A_i.

lda The leading dimension of A_i .

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

stride_tau The stride between the beginnings of arrays $\texttt{tau}_i$ inside the array `tau`.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by geqrf_batch_scratchpad_size (Strided Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by the factorization data as follows:

The elements on and above the diagonal of the array contain the `min(m,n)`-by-`n` upper trapezoidal matrices R_i (R_i is upper triangular if `m` $\geq$ `n`); the elements below the diagonal, with the array `taui`, present the orthogonal matrix Q_i as a product of `min(m,n)` elementary reflectors.

tau Array to store batch of tau_i, each of size `min(m,n)`, containing scalars that define elementary reflectors for the matrices Q_i in its decomposition in a product of elementary reflectors.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.12 geqrf_batch_scratchpad_size (Group Version)

Computes size of scratchpad memory required for `geqrf_batch` (Group Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.12.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the geqrf_batch (Group Version) function should be able to hold.

11.12.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t geqrf_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t *m,
  std::int64_t *n,
  std::int64_t *lda,
  std::int64_t group_count,
  std::int64_t *group_sizes)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m Array of `group_count` parameters m_g.

Each m_g specifies the number of rows in the matrices A_i belonging to group `g`.

n Array of `group_count` parameters n_g.

Each n_g specifies the number of columns in the matrices A_i belonging to group `g`.

lda Array of `group_count` parameters lda_g, each representing the leading dimension of input matrices belonging to group `g`.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of `group_count` integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the geqrf_batch (Group Version) function should be able to hold.

11.13 geqrf_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `geqrf_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.13.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `geqrf_batch` (Strided Version) function should be able to hold.

11.13.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  geqrf_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t stride_tau,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i (`0 ≤ m`).

n The number of columns in the matrices A_i (`0 ≤ n`).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

stride_tau The stride between the beginnings of arrays tau_i inside the array `tau`.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `geqrf_batch` (Strided Version) function should be able to hold.

11.14 geqrf_scratchpad_size

Computes size of scratchpad memory required for `geqrf` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.14.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `geqrf` (buffer or USM version) function should be able to hold.

11.14.2 API

Syntax

```
namespace oneapi::mkl::lapack {
   template<typename T>
   std::int64_t geqrf_scratchpad_size(sycl::queue &queue,
   std::int64_t m,
   std::int64_t n,
   std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the geqrf (buffer or USM version) function will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

lda The leading dimension of a.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `geqrf` (buffer or USM version) function should be able to hold.

11.15 gerqf

Computes the RQ factorization of a general m-by-n matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.15.1 Description

The routine forms the `RQ` factorization of a general `m`-by-`n` matrix `A` No pivoting is performed.

The routine does not form the matrix `Q` explicitly. Instead, `Q` is represented as a product of min(`m`, `n`) elementary reflectors. Routines are provided to work with `Q` in this representation.

Note: This routine supports the Progress Routine feature.

11.15.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void gerqf(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`gerqf` supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

a Buffer holding input matrix `A`. The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a, at least `max(1, m)`.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the gerqf_scratchpad_size function.

Output Parameters

a Overwritten by the factorization data as follows:

if `m ≤ n`, the upper triangle of the subarray `a(1:m, n-m+1:n )` contains the `m`-by-`m` upper triangular matrix `R`; if `m ≥ n`, the elements on and above the (`m`-`n`)-th subdiagonal contain the `m`-by-`n` upper trapezoidal matrix `R`

In both cases, the remaining elements, with the array`tau`, represent the orthogonal/unitary matrix `Q` as a product of `min(m,n)` elementary reflectors.

tau Array, size at least `min(m,n)`.

Contains scalars that define elementary reflectors for the matrix `Q` in its decomposition in a product of elementary reflectors.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.16 gerqf (USM Version)

Computes the RQ factorization of a general m-by-n matrix . This routine belongs to the `oneapi::mkl::lapack` namespace.

11.16.1 Description

The routine forms the `RQ` factorization of a general `m`-by-`n` matrix `A`. No pivoting is performed.

The routine does not form the matrix `Q` explicitly. Instead, `Q` is represented as a product of min(`m`, `n`) elementary reflectors. Routines are provided to work with `Q` in this representation.

11.16.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event gerqf(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *tau,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`gerqf` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

a Pointer to the memory holding input matrix `A`. The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a, at least `max(1, m)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the gerqf_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by the factorization data as follows:

if `m ≤ n`, the upper triangle of the subarray `a(1:m, n-m+1:n )` contains the `m`-by-`m` upper triangular matrix R; if `m ≥ n`, the elements on and above the (`m`-`n`)-th subdiagonal contain the `m`-by-`n` upper trapezoidal matrix R

In both cases, the remaining elements, with the array`tau`, represent the orthogonal/unitary matrix `Q` as a product of `min(m,n)` elementary reflectors.

tau Array, size at least `min(m,n)`.

Contains scalars that define elementary reflectors for the matrix `Q` in its decomposition in a product of elementary reflectors.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the get_info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and get_detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the get_detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.17 gerqf_scratchpad_size

Computes size of scratchpad memory required for `gerqf` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.17.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `gerqf` (buffer or USM version) function should be able to hold.

11.17.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template <typename T>
  std::int64_t gerqf_scratchpad_size(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the gerqf (buffer or USM version) function will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

lda The leading dimension of a; at least `max(1,m)`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the get_info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `gerqf` (buffer or USM version) function should be able to hold.

11.18 gesvd

Computes the singular value decomposition of a general rectangular matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.18.1 Description

The routine computes the singular value decomposition (SVD) of a real/complex `m`-by-`n` matrix `A`, optionally computing the left and/or right singular vectors. The SVD is written as:

- `A = U*Σ*VT` for real routines
- `A = U*Σ*VH` for complex routines

where `Σ` is an `m`-by-`n` diagonal matrix, `U` is an `m`-by-`m` orthogonal/unitary matrix, and `V` is an `n`-by-`n` orthogonal/unitary matrix. The diagonal elements of `Σ` are the singular values of `A`; they are real and non-negative, and are returned in descending order. The first `min(m, n)` columns of `U` and `V` are the left and right singular vectors of `A`.

11.18.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void gesvd(sycl::queue &queue,
  mkl::jobsvd jobu,
  mkl::jobsvd jobvt,
  std::int64_t m,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<RealT> &s,
  sycl::buffer<T> &u,
  std::int64_t ldu,
  sycl::buffer<T> &vt,
  std::int64_t ldvt,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`gesvd` supports the following precision and devices.

T	Devices Supported
`float`	CPU
`double`	CPU
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue Device queue where calculations will be performed.

jobu Must be `jobsvd::vectors`, `job::somevec`, `jobsvd::vectorsina`, or `job::novec`. Specifies options for computing all or part of the matrix `U`.

If `jobu = jobsvd::vectors`, all `m` columns of `U` are returned in the buffer u;

if `jobu = job::somevec`, the first `min(m, n)` columns of `U` (the left singular vectors) are returned in the buffer u;

if `jobu = jobsvd::vectorsina`, the first `min(m, n)` columns of `U` (the left singular vectors) are overwritten on the buffer a;

if `jobu = job::novec`, no columns of `U` (no left singular vectors) are computed.

jobvt Must be `jobsvd::vectors, job::somevec`, `jobsvd::vectorsina`, or `job::novec`. Specifies options for computing all or part of the matrix **VT/VH**.

If `jobvt = jobsvd::vectors`, all n columns of **VT/VH** are returned in the buffer vt;

if `jobvt = job::somevec`, the first `min(m, n)` columns of **VT/VH** (the left singular vectors) are returned in the buffer vt;

if `jobvt = jobsvd::vectorsina`, the first `min(m, n)` columns of **VT/VH** (the left singular vectors) are overwritten on the buffer a;

if `jobvt = job::novec`, no columns of **VT/VH** (no left singular vectors) are computed.

jobvt and jobu cannot both be `jobsvd::vectorsina`.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

a Buffer holding arraya, size `(lda,*)`. The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a.

ldu The leading dimension of u.

ldvt The leading dimension of vt.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the gesvd_scratchpad_size function.

Output Parameters

a On exit,

If `jobu = jobsvd::vectorsina`, a is overwritten with the first `min(m,n)` columns of `U` (the left singular vectors stored columnwise);

If `jobvt = jobsvd::vectorsina`, a is overwritten with the first `min(m, n)` rows of V^T/V^H (the right singular vectors stored rowwise);

If `jobu` ≠ `jobsvd::vectorsina` and `jobvt` ≠ `jobsvd::vectorsina`, the contents of a are destroyed.

s Array containing the singular values, size at least `max(1, min(m,n))`. Contains the singular values of `A` sorted so that `s(i)` ≥ `s(i+1)`.

u Array containing `U`; the second dimension of u must be at least `max(1, m)` if `jobu = jobsvd::vectors`, and at least `max(1, min(m, n))` if `jobu = job::somevec`.

If `jobu = jobsvd::vectors`, u contains the m-by-m orthogonal/unitary matrix `U`.

If `jobu = job::somevec`, u contains the first `min(m, n)` columns of `U` (the left singular vectors stored column-wise).

If `jobu = job::novec` or `jobsvd::vectorsina`, u is not referenced.

vt Array containing V^T; the second dimension of vt must be at least `max(1, n)`.

If `jobvt = jobsvd::vectors`, vt contains the n-by-n orthogonal/unitary matrix V^T/V^H.

If `jobvt = job::somevec`, vt contains the first `min(m, n)` rows of V^T/V^H (the right singular vectors stored row-wise).

If `jobvt = job::novec` or `jobsvd::vectorsina`, vt is not referenced.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, then if bdsqr did not converge, `i` specifies how many superdiagonals of the intermediate bidiagonal form `B` did not converge to zero, and `scratchpad(2:min(m,n))` contains the unconverged superdiagonal elements of an upper bidiagonal matrix `B` whose diagonal is in `s` (not necessarily sorted). `B` satisfies `A = U*B*VT`, so it has the same singular values as `A`, and singular vectors related by `U` and V^T. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.19 gesvd (USM Version)

Computes the singular value decomposition of a general rectangular matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.19.1 Description

The routine computes the singular value decomposition (SVD) of a real/complex `m`-by-`n` matrix `A`, optionally computing the left and/or right singular vectors. The SVD is written as:

- `A = U*Σ*VT` for real routines
- `A = U*Σ*VH` for complex routines

where Σ is an `m`-by-`n` diagonal matrix, `U` is an `m`-by-`m` orthogonal/unitary matrix, and `V` is an `n`-by-`n` orthogonal/unitary matrix. The diagonal elements of Σ are the singular values of `A`; they are real and non-negative, and are returned in descending order. The first `min(m, n)` columns of `U` and `V` are the left and right singular vectors of `A`.

11.19.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event gesvd(sycl::queue &queue,
  mkl::jobsvd jobu,
  mkl::jobsvd jobvt,
  std::int64_t m,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  RealT *s,
  T *u,
  std::int64_t ldu,
  T *vt,
  std::int64_t ldvt,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`gesvd` (USM version) supports the following precision and devices.

T	Devices Supported
`float`	CPU
`double`	CPU
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue Device queue where calculations will be performed.

jobu Must be `jobsvd::vectors`, `job::somevec`, `jobsvd::vectorsina`, or `job::novec`. Specifies options for computing all or part of the matrix `U`.

If `jobu = jobsvd::vectors`, all `m` columns of `U` are returned in the array u;

if `jobu = job::somevec`, the first `min(m, n)` columns of `U` (the left singular vectors) are returned in the array u;

if `jobu = jobsvd::vectorsina`, the first `min(m, n)` columns of `U` (the left singular vectors) are overwritten on the array a;

if `jobu = job::novec`, no columns of `U` (no left singular vectors) are computed.

jobvt Must be `jobsvd::vectors, job::somevec`, `jobsvd::vectorsina`, or `job::novec`. Specifies options for computing all or part of the matrix **VT/VH**.

If `jobvt = jobsvd::vectors`, all n columns of **VT/VH** are returned in the array vt;

if `jobvt = job::somevec`, the first `min(m, n)` columns of **VT/VH** (the left singular vectors) are returned in the array vt;

if `jobvt = jobsvd::vectorsina`, the first `min(m, n)` columns of **VT/VH** (the left singular vectors) are overwritten on the array a;

if `jobvt = job::novec`, no columns of **VT/VH** (no left singular vectors) are computed.

jobvt and jobu cannot both be `jobsvd::vectorsina`.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

a Pointer to array a, size `(lda,*)`. The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a.

ldu The leading dimension of u.

ldvt The leading dimension of vt.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the gesvd_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a On exit,

If `jobu = jobsvd::vectorsina`, a is overwritten with the first `min(m,n)` columns of `U` (the left singular vectors stored columnwise);

If `jobvt = jobsvd::vectorsina`, a is overwritten with the first `min(m, n)` rows of V^T/V^H (the right singular vectors stored rowwise);

If `jobu ≠ jobsvd::vectorsina` and `jobvt ≠ jobsvd::vectorsina`, the contents of a are destroyed.

s Array containing the singular values, size at least `max(1, min(m,n))`. Contains the singular values of `A` sorted so that `s(i) ≥ s(i+1)`.

u Array containing `U`; the second dimension of u must be at least `max(1, m)` if `jobu = jobsvd::vectors`, and at least `max(1, min(m, n))` if `jobu = job::somevec`.

If `jobu = jobsvd::vectors`, u contains the m-by-m orthogonal/unitary matrix `U`.

If `jobu = job::somevec`, u contains the first `min(m, n)` columns of `U` (the left singular vectors stored column-wise).

If `jobu = job::novec` or `jobsvd::vectorsina`, u is not referenced.

vt Array containing V^T; the second dimension of vt must be at least `max(1, n)`.

If `jobvt = jobsvd::vectors`, vt contains the n-by-n orthogonal/unitary matrix V^T/V^H.

If `jobvt = job::somevec`, vt contains the first `min(m, n)` rows of V^T/V^H (the right singular vectors stored row-wise).

If `jobvt = job::novec` or `jobsvd::vectorsina`, vt is not referenced.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the get_info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, then if bdsqr did not converge, `i` specifies how many superdiagonals of the intermediate bidiagonal form `B` did not converge to zero, and `scratchpad(2:min(m,n))` contains the unconverged superdiagonal elements of an upper bidiagonal matrix `B` whose diagonal is in `s` (not necessarily sorted). `B` satisfies `A = U*B*VT`, so it has the same singular values as `A`, and singular vectors related by `U` and V^T. If `info` is equal to the value passed as scratchpad size, and get_detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the get_detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.20 gesvd_scratchpad_size

Computes size of scratchpad memory required for `gesvd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.20.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `gesvd` (buffer or USM version) function should be able to hold.

11.20.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t gesvd_scratchpad_size(sycl::queue &queue,
  mkl::jobsvd jobu,
  mkl::jobsvd jobvt,
  std::int64_t m,
  std::int64_t n,
  std::int64_t lda,
  std::int64_t ldu,
  std::int64_t ldvt)
}
```

Input Parameters

queue Device queue where calculations by the gesvd (buffer or USM version) function will be performed.

jobu Must be `jobsvd::vectors`, `jobsvd::somevec`, `jobsvd::vectorsina`, or `jobsvd::novec`. Specifies options for computing all or part of the matrix `U`.

If `jobu = jobsvd::vectors`, all `m` columns of `U` are returned in the array u;

if `jobu = jobsvd::somevec`, the first `min(m, n)` columns of `U` (the left singular vectors) are returned in the array u;

if `jobu = jobsvd::vectorsina`, the first `min(m, n)` columns of `U` (the left singular vectors) are overwritten on the array a;

if `jobu = jobsvd::novec`, no columns of `U` (no left singular vectors) are computed.

jobvt Must be `jobsvd::vectors`, `jobsvd::somevec`, `jobsvd::vectorsina`, or `jobsvd::novec`. Specifies options for computing all or part of the matrix **VT/VH**.

If `jobvt = jobsvd::vectors`, all n columns of **VT/VH** are returned in the array vt;

if `jobvt = jobsvd::somevec`, the first `min(m, n)` columns of **VT/VH** (the left singular vectors) are returned in the array vt;

if `jobvt = jobsvd::vectorsina`, the first `min(m, n)` columns of **VT/VH** (the left singular vectors) are overwritten on the array a;

if `jobvt = jobsvd::novec`, no columns of **VT/VH** (no left singular vectors) are computed.

jobvt and jobu cannot both be `job::overwritevec`.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

lda The leading dimension of a.

ldu The leading dimension of u.

ldvt The leading dimension of vt.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the get_info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `gesvd` (buffer or USM version) function should be able to hold.

11.21 getrf

Computes the LU factorization of a general m-by-n matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.21.1 Description

The routine computes the LU factorization of a general m-by-n matrix A as

```
A = P*L*U,
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n) and U is upper triangular (upper trapezoidal if m < n). The routine uses partial pivoting, with row interchanges.

11.21.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void getrf(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &ipiv,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

getrf supports the following precisions and devices.

T	Devices supported
float	CPU and GPU
double	CPU and GPU
std::complex<float>	CPU and GPU
std::complex<double>	CPU and GPU

Input Parameters

queue	Device queue where calculations will be performed.
m	The number of rows in the matrix A (0 ≤ m).
n	The number of columns in A(0 ≤ n).
a	Buffer holding array holding input matrix A. The second dimension of a must be at least max(1, n).
lda	The leading dimension of a.
scratch-pad	Buffer holding scratchpad memory to be used by the routine for storing intermediate results.
scratch-pad_size	Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the getrf_scratchpad_size function.

Output Parameters

a	Overwritten by L and U. The unit diagonal elements of L are not stored.
ipiv	Buffer holding array of size at least `max(1,min(m, n))`. Contains the pivot indices; for `1 ≤ i ≤ min(m, n)`, row `i` was interchanged with row `ipiv(i)`.
info	Buffer containing error information. If `info=0`, execution is successful. If `info=-i`, the `i`-th parameter had an illegal value. If `info=i`, `uii` is 0. The factorization has been completed, but `U` is exactly singular. Division by 0 will occur if you use the factor `U` for solving a system of linear equations.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, `uii` is 0. The factorization has been completed, but `U` is exactly singular. Division by 0 will occur if you use the factor `U` for solving a system of linear equations. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.22 getrf (USM Version)

Computes the LU factorization of a general m-by-n matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.22.1 Description

The routine computes the LU factorization of a general `m`-by-`n` matrix `A` as

$$A = P * L * U,$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if `m > n`) and U is upper triangular (upper trapezoidal if `m < n`). The routine uses partial pivoting, with row interchanges.

11.22.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getrf(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  std::int64_t *ipiv,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`getrf` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU, GPU
`double`	CPU, GPU
`std::complex<float>`	CPU, GPU
`std::complex<double>`	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

a Pointer to the array holding input matrix `A`. The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the getrf_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by `L` and `U`. the unit diagonal elements of `L` are not stored.

ipiv Array, size at least `max(1,min(m,n))`.

Contains the pivot indices; for `1 ≤ i ≤ min(m,n)`, row `i` was interchanged with row `ipiv(i)`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, `uii` is 0. The factorization has been completed, but `U` is exactly singular. Division by 0 will occur if you use the factor `U` for solving a system of linear equations. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.23 getrf_batch (Buffer Strided Version)

Computes the batch of LU factorizations of a batch of general m-by-n matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.23.1 Description

The routine computes the LU factorizations of a batch of general `m`-by-`n` matrices A_i, as:

$$A_i = P_i \times L_i \times U_i$$

Where P_i is a permutation matrix, L_i is lower triangular with unit diagonal elements (lower trapezoidal if `m > n`) and U_i is upper triangular (upper trapezoidal if `m < n`). The routine uses partial pivoting with row interchanges.

11.23.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void getrf_batch(
     sycl::queue &queue,
     std::int64_t m,
     std::int64_t n,
     sycl::buffer<T> &a,
     std::int64_t lda,
     std::int64_t stride_a,
     sycl::buffer<std::int64_t> &ipiv,
     std::int64_t stride_ipiv,
     std::int64_t batch_size,
     sycl::buffer<T> &scratchpad,
     std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i ($0 \leq m$).

n The number of columns in the matrices A_i ($0 \leq n$).

a Array holding input matrices A_i.

lda The leading dimension of A_i .

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

stride_ipiv The stride between the beginnings of arrays $ipiv_i$ inside the array `ipiv`.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by getrf_batch_scratchpad_size (Strided Version).

Output Parameters

a Overwritten by L_i and U_i. The unit diagnonal elements of L_i are not stored.

ipiv Array containing the batch of the pivot indices $ipiv_i$ each of size at least `max(1,min(m,n))`; for $1 \le k \le$ `min(m,n)`, row k of A_i was interchanged with row $ipiv_i(k)$.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

11.24 getrf_batch (Group Version)

Computes the batch of LU factorizations of a batch of general m-by-n matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.24.1 Description

The routine computes the LU factorizations of a batch of general `m`-by-`n` matrices A_i, (`i`$\in${`1...batch_size`}) as

$$A_i = P_i \times L_i \times U_i$$

Where P_i is a permutation matrix, L_i is lower triangular with unit diagonal elements (lower trapezoidal if `m > n`) and U_i is upper triangular (upper trapezoidal if `m < n`). The routine uses partial pivoting with row interchanges.

Total number of problems to solve, `batch_size`, is a sum of sizes of all of the groups of parameters as provided by `group_sizes` array.

11.24.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getrf_batch(sycl::queue &queue,
  std::int64_t *m,
  std::int64_t *n,
  T **a,
  std::int64_t *lda,
  std::int64_t **ipiv,
  std::int64_t group_count,
  std::int64_t *group_sizes,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m Array of `group_count` parameters m_g specifying the number of rows in the matrices A_i ($0 \leq mg$) belonging to group `g`.

n Array of `group_count` parameters n_g specifying the number of columns in the matrices A_i ($0 \leq ng$) belonging to group `g`.

a Array of `batch_size` pointers to input matricesA_i.

lda Array of `group_count` parameters lda_g specifying the leading dimension of A_i belonging to group `g`.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of group_count integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by getrf_batch_scratchpad_size (Group Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Matrices pointed to by array a are overwritten by L_i and U_i. The unit diagnonal elements of L_i are not stored.

ipiv Arrays of batch_size pointers to arrays containing pivot indices $ipiv_i$ each of size at least `max(1,min(mg, ng))`; for $1 \le k \le$ `min(mg,ng)`, row k of A_i was interchanged with row $ipiv_i(k)$.

Exceptions

Exception	Description
`mkl::lapack::batch_-exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.25 getrf_batch (USM Strided Version)

Computes the batch of LU factorizations of a batch of general m-by-n matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.25.1 Description

The routine computes the LU factorizations of a batch of general `m`-by-`n` matrices A_i, as

$$A_i = P_i \times L_i \times U_i$$

Where P_i is a permutation matrix, L_i is lower triangular with unit diagonal elements (lower trapezoidal if `m > n`) and U_i is upper triangular (upper trapezoidal if `m < n`). The routine uses partial pivoting with row interchanges.

11.25.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getrf_batch(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t *ipiv,
  std::int64_t stride_ipiv,
  std::int64_t batch_size,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i (`0 ≤ m`).

n The number of columns in the matrices A_i (`0 ≤ n`).

a Array holding input matrices A_i.

lda The leading dimension of A_i .

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

stride_ipiv The stride between the beginnings of arrays $\texttt{ipiv}_i$ inside the array `ipiv`.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by getrf_batch_scratchpad_size (Strided Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by L_i and U_i. The unit diagnonal elements of L_i are not stored.

ipiv Array containing the batch of the pivot indices $ipiv_i$ each of size at least `max(1,min(m,n))`; for $1 \le k \le$ `min(m,n)`, row k of A_i was interchanged with row $ipiv_i(k)$.

Exceptions

Exception	Description
`mkl::lapack::batch_-exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.26 getrf_batch_scratchpad_size (Group Version)

Computes size of scratchpad memory required for `getrf_batch` (Group Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.26.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the getrf_batch (Group Version) function should be able to hold.

11.26.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t getrf_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t *m,
  std::int64_t *n,
  std::int64_t *lda,
  std::int64_t group_count,
  std::int64_t *group_sizes)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m Array of `group_count` parameters m_g specifying the number of rows in the matrices belonging to group `g`.

n Array of `group_count` parameters n_g specifying the number of columns in the matrices belonging to group `g`.

lda Array of `group_count` parameters lda_g specifying the leading dimension of the matrices belonging to group `g`.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of `group_count` integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the getrf_batch (Group Version) function should be able to hold.

11.27 getrf_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `getrf_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.27.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `getrf_batch` (Strided Version) function should be able to hold.

11.27.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t getrf_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t stride_ipiv,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i (`0 ≤ m`).

n The number of columns in the matrices A_i (`0 ≤ n`).

lda The leading dimension of A_i .

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

stride_ipiv The stride between the beginnings of arrays $ipiv_i$ inside the array `ipiv`.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `getrf_batch` (Strided Version) function should be able to hold.

11.28 getrf_scratchpad_size

Computes size of scratchpad memory required for `getrf` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.28.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `getrf` (buffer or USM version) function should be able to hold.

11.28.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t getrf_scratchpad_size(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the getrf (buffer or USM version) function will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

lda The leading dimension of a.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `getrf` (buffer or USM version) function should be able to hold.

11.29 getrfnp_batch (Buffer Strided Version)

Computes the batch of LU factorizations of a batch of general m-by-n matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.29.1 Description

The routine computes the LU factorizations of a batch of general `m`-by-`n` matrices A_i, as:

$$A_i = L_i \times U_i$$

Where L_i is lower triangular with unit diagonal elements (lower trapezoidal if `m > n`) and U_i is upper triangular (upper trapezoidal if `m < n`). The routine does not perform any pivoting.

11.29.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void getrfnp_batch(
     sycl::queue &queue,
     std::int64_t m,
     std::int64_t n,
     sycl::buffer<T> &a,
     std::int64_t lda,
     std::int64_t stride_a,
     std::int64_t batch_size,
     sycl::buffer<T> &scratchpad,
     std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i (`m ≥ 0`).

n The number of columns in the matrices A_i (`n ≥ 0`).

a Array holding input matrices A_i.

lda The leading dimension of A_i .

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by getrfnp_batch_scratchpad_size (Strided Version).

Output Parameters

a Overwritten by L_i and U_i. The unit diagnonal elements of L_i are not stored.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

11.30 getrfnp_batch (Group Version)

Computes the batch of LU factorizations of a batch of general m-by-n matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.30.1 Description

The routine computes the LU factorizations of a batch of general `m`-by-`n` matrices A_i, (`i`$\in${`1...batch_size`}) as

$$A_i = L_i \times U_i$$

Where L_i is lower triangular with unit diagonal elements (lower trapezoidal if `m > n`) and U_i is upper triangular (upper trapezoidal if `m < n`). The routine does not perform any pivoting.

Total number of problems to solve, `batch_size`, is a sum of sizes of all of the groups of parameters as provided by `group_sizes` array.

11.30.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getrfnp_batch(sycl::queue &queue,
  std::int64_t *m,
  std::int64_t *n,
  T **a,
  std::int64_t *lda,
  std::int64_t group_count,
  std::int64_t *group_sizes,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m Array of `group_count` parameters m_g specifying the number of rows in the matrices A_i ($0 \leq m_g$) belonging to group `g`.

n Array of `group_count` parameters n_g specifying the number of columns in the matrices A_i ($0 \leq n_g$) belonging to group `g`.

a Array of `batch_size` pointers to input matrices A_i.

lda Array of `group_count` parameters lda_g specifying the leading dimension of A_i belonging to group `g`.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of group_count integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by getrfnp_batch_scratchpad_size (Group Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Matrices pointed to by array a are overwritten by L_i and U_i. The unit diagnonal elements of L_i are not stored.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.31 getrfnp_batch (USM Strided Version)

Computes the batch of LU factorizations of a batch of general m-by-n matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.31.1 Description

The routine computes the LU factorizations of a batch of general `m`-by-`n` matrices A_i, as

$$A_i = L_i \times U_i$$

Where L_i is lower triangular with unit diagonal elements (lower trapezoidal if `m > n`) and U_i is upper triangular (upper trapezoidal if `m < n`). The routine does not perform any pivoting.

11.31.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getrfnp_batch(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t batch_size,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i (`0 ≤ m`).

n The number of columns in the matrices A_i (`0 ≤ n`).

a Array holding input matrices A_i.

lda The leading dimension of A_i .

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by getrfnp_batch_scratchpad_size (Strided Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by L_i and U_i. The unit diagnonal elements of L_i are not stored.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.32 getrfnp_batch_scratchpad_size (Group Version)

Computes size of scratchpad memory required for `getrfnp_batch` (Group Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.32.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the getrfnp_batch (Group Version) function should be able to hold.

11.32.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t getrfnp_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t *m,
  std::int64_t *n,
  std::int64_t *lda,
  std::int64_t group_count,
  std::int64_t *group_sizes)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m Array of `group_count` parameters m_g specifying the number of rows in the matrices belonging to group `g`.

n Array of `group_count` parameters n_g specifying the number of columns in the matrices belonging to group `g`.

lda Array of `group_count` parameters lda_g specifying the leading dimension of the matrices belonging to group `g`.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of `group_count` integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the getrfnp_batch (Group Version) function should be able to hold.

11.33 getrfnp_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `getrfnp_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.33.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `getrfnp_batch` (Strided Version) function should be able to hold.

11.33.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t getrfnp_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i (`0 ≤ m`).

n The number of columns in the matrices A_i (`0 ≤ n`).

lda The leading dimension of A_i .

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `getrfnp_batch` (Strided Version) function should be able to hold.

11.34 getri

Computes the inverse of an LU-factored general matrix determined by getrf. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.34.1 Description

The routine computes the inverse `inv(A)` of a general matrix `A`. Before calling this routine, call getrf to factorize `A`.

11.34.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void getri(sycl::queue &queue,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &ipiv,
  sycl::buffer<T> &scratchpad, std::int64_t scratchpad_size)
}
```

`getri` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

n The order of the matrix `A(0 ≤ n)`.

a The buffer returned by getrf. Must be of size at least `lda*max(1,n)`.

lda The leading dimension of a `(n ≤ lda)`.

ipiv Buffer holding the array as returned by getrf. The dimension of the array must be at least `max(1, n)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the getri_scratchpad_size function.

Output Parameters

a Overwritten by the n-by-n matrix A.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Known Limitations

- GPU support is for only real precisions.
- For GPU support, errors are reported through the `info` parameter, but computation does not halt for an algorithmic error.

11.35 getri (USM Version)

Computes the inverse of an LU-factored general matrix determined by `getrf` (USM Version). This routine belongs to the `oneapi::mkl::lapack` namespace.

11.35.1 Description

The routine computes the inverse `inv(A)` of a general matrix `A`. Before calling this routine, call getrf (USM Version) to factorize `A`.

11.35.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getri(sycl::queue &queue,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  std::int64_t *ipiv,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`getri` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

n The order of the matrix `A(0 ≤ n)`.

a The pointer returned by getrf (USM Version). Must be of size at least `lda*max(1,n)`.

lda The leading dimension of a `(n ≤ lda)`.

ipiv The array as returned by getrf (USM Version). The dimension of ipiv must be at least `max(1, n)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the getri_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by the n-by-n matrix `inv(A)`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.36 getri_batch (Buffer Strided Version)

Computes the batch of inverses of an LU-factored general matrices determined by `getrf_batch` (Buffer stride version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.36.1 Description

The routine computes the inverses A_i^{-1} of a general matrices A_i. Before calling this routine, call getrf_batch (Buffer Strided Version) function to factorize A_i.

11.36.2 API

Syntax

```
namespace oneapi::mkl::lapack {
 sycl::event getri_batch(sycl::queue &queue,
 std::int64_t n,
 sycl::buffer<T> &a,
 std::int64_t lda,
 std::int64_t stride_a,
 sycl::buffer<std::int64_t> &ipiv,
 std::int64_t stride_ipiv,
 std::int64_t batch_size,
 sycl::buffer<T> &scratchpad,
 std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

n The order of the matrices A_i (`0 ≤ n`).

a Result of the getrf_batch (Buffer Strided Version) function

lda The leading dimension of A_i (`lda ≥ n`).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

ipiv The array as returned by getrf_batch (Buffer Strided Version).

stride_ipiv The stride between the beginnings of arrays $ipiv_i$ inside the array `ipiv`.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by stride version of getri_batch_scratchpad_size (Strided Version) function.

Output Parameters

a Overwritten by the n-by-n matrices A_i^{-1}.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

11.37 getri_batch (Group Version)

Computes the batch of inverses of an LU-factored general matrices determined by the `getrf_batch` (Group Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.37.1 Description

The routine computes the inverses A_i^{-1} of a general matrices A_i, $i \in \{1...batch\_size\}$. Before calling this routine, call getrf_batch (Group Version) function to factorize A_i.

The total number of problems to solve, `batch_size`, is a sum of sizes of all of the groups of parameters as provided by `group_sizes`array.

11.37.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getri_batch(sycl::queue &queue,
  std::int64_t *n,
  T **a,
  std::int64_t *lda,
  std::int64_t **ipiv,
  std::int64_t group_count,
  std::int64_t *group_sizes,
```

(continues on next page)

(continued from previous page)

```
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

n Array of `group_count` parameters n_g specifying the order of the matrices A_i ($0 \le n_g$) belonging to group g.

a Result of the getrf_batch (Group Version) function

lda Array of `group_count` parameters lda_g specifying the leading dimension of A_i ($n_g \le lda_g$) belonging to group g.

ipiv The array as returned by getrf_batch (Group Version).

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of group_count integers. Array element with index g specifies the number of problems to solve for each of the groups of parameters g. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by getri_batch_scratchpad_size (Group Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by the n_g-by-n_g matrices A_i^{-1}.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.38 getri_batch (USM Strided Version)

Computes the batch of inverses of an LU-factored general matrices determined by `getrf_batch` (USM stride version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.38.1 Description

The routine computes the inverses A_i^{-1} of a general matrices A_i. Before calling this routine, call **getrf_batch (USM Strided Version)** function to factorize A_i.

11.38.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getri_batch(sycl::queue &queue,
  std::int64_t n,
  T *a,
 std::int64_t lda,
 std::int64_t stride_a,
 std::int64_t *ipiv,
 std::int64_t stride_ipiv,
 std::int64_t batch_size,
 T *scratchpad,
 std::int64_t scratchpad_size,
 const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

n The order of the matrices A_i (`0 ≤ n`).

a Result of the getrf_batch (USM Strided Version) function

lda The leading dimension of A_i (`n ≤ lda`).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

ipiv The array as returned by getrf_batch (USM Strided Version).

stride_ipiv The stride between the beginnings of arrays $ipiv_i$ inside the array `ipiv`.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by stride version of getri_batch_scratchpad_size (Strided Version) function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by the n-by-n matrices A_i^{-1}.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.39 getri_batch_scratchpad_size (Group Version)

Computes size of scratchpad memory required for `getri_batch` (Group Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.39.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the getri_batch (Group Version) function should be able to hold.

11.39.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t getri_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t *n,
  std::int64_t *lda,
  std::int64_t group_count,
  std::int64_t *group_sizes)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

n Array of `group_count` parameters n_g specifying the order of the matrices belonging to group g.

lda Array of `group_count` parameters lda_g specifying the leading dimension of the matrices belonging to group g.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of `group_count` integers. Array element with index g specifies the number of problems to solve for each of the groups of parameters g. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the getri_batch (Group Version) function must be able to hold.

11.40 getri_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `getri_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.40.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `getri_batch` (Strided Version) function should be able to hold.

11.40.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t getri_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t n,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t stride_ipiv,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

n The order of the matrices A_i ($0 \leq n$).

lda The leading dimension of A_i ($n \leq lda$).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

stride_ipiv The stride between the beginnings of arrays $ipiv_i$ inside the array ipiv.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
mkl::lapack::exception	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the getri_batch (Strided Version) function should be able to hold.

11.41 getri_batch (Out-of-place, Buffer Strided Version)

Computes the batch of inverses of an LU-factored general matrices determined by getrf_batch (Buffer stride version) function. This routine belongs to the oneapi::mkl::lapack namespace.

11.41.1 Description

The routine computes the inverses A_i^{-1} of a general matrices A_i. Before calling this routine, call getrf_batch (Buffer Strided Version) function to factorize A_i.

11.41.2 API

Syntax

```
namespace oneapi::mkl::lapack {
 sycl::event getri_batch(sycl::queue &queue,
 std::int64_t n,
 sycl::buffer<T> &a,
 std::int64_t lda,
 std::int64_t stride_a,
 sycl::buffer<std::int64_t> &ipiv,
 std::int64_t stride_ipiv,
 sycl::buffer<T> &ainv,
 std::int64_t ldainv,
 std::int64_t stride_ainv,
 std::int64_t batch_size,
 sycl::buffer<T> &scratchpad,
 std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

n The order of the matrices A_i (`0 ≤ n`).

a Result of the getrf_batch (Buffer Strided Version) function

lda The leading dimension of A_i (`lda ≥ n`).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

ipiv The array as returned by getrf_batch (Buffer Strided Version).

stride_ipiv The stride between the beginnings of arrays $ipiv_i$ inside the array `ipiv`.

ldainv The leading dimension of $Ainv_i$ (`n ≤ ldainv`).

stride_ainv The stride between the beginnings of matrices $Ainv_i$ inside the batch array `ainv`.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the out-of-place, stride version of getri_batch_scratchpad_size (Strided Version) function.

Output Parameters

ainv Overwritten by the n-by-n matrices A_i^{-1}.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

11.42 getri_batch (Out-of-place, USM Strided Version)

Computes the batch of inverses of an LU-factored general matrices determined by `getrf_batch` (USM stride version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.42.1 Description

The routine computes the inverses A_i^{-1} of a general matrices A_i. Before calling this routine, call getrf_batch (USM Strided Version) function to factorize A_i.

11.42.2 API

Syntax

```
namespace oneapi::mkl::lapack {
 sycl::event getri_batch(sycl::queue &queue,
 std::int64_t n,
 T *a,
 std::int64_t lda,
 std::int64_t stride_a,
 std::int64_t *ipiv,
 std::int64_t stride_ipiv,
 T* ainv,
 std::int64_t ldainv,
 std::int64_t stride_ainv,
```

(continues on next page)

(continued from previous page)

```
 std::int64_t batch_size,
 T *scratchpad,
 std::int64_t scratchpad_size,
 const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

n The order of the matrices A_i (`0 ≤ n`).

a Result of the getrf_batch (USM Strided Version) function

lda The leading dimension of A_i (`lda ≥ n`).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

ipiv The array as returned by getrf_batch (USM Strided Version).

stride_ipiv The stride between the beginnings of arrays $ipiv_i$ inside the array `ipiv`.

ldainv The leading dimension of $Ainv_i$ (`n ≤ ldainv`).

stride_ainv The stride between the beginnings of matrices $Ainv_i$ inside the batch array `ainv`.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the out-of-place, stride version of getri_batch_scratchpad_size (Strided Version) function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

ainv Overwritten by the n-by-n matrices A_i^{-1}.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.43 getri_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `getri_batch` (Out-of-place, Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.43.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `getri_batch` (Out-of-place, Strided Version) function should be able to hold.

11.43.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t getri_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t n,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t stride_ipiv,
  std::int64_t ldainv,
  std::int64_t stride_ainv,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

n The order of the matrices A_i (`0 ≤ n`).

lda The leading dimension of A_i (`lda ≥ n`).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

stride_ipiv The stride between the beginnings of arrays $ipiv_i$ inside the array `ipiv`.

ldainv The leading dimension of $Ainv_i$ (`n ≤ ldainv`).

stride_ainv The stride between the beginnings of matrices $Ainv_i$ inside the batch array `ainv`.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `getri_batch` (Out-of-place, Strided Version) function should be able to hold.

11.44 getri_scratchpad_size

Computes size of scratchpad memory required for `getri` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.44.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `getri` (buffer or USM version) function must be able to hold.

11.44.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t getri_scratchpad_size(sycl::queue &queue,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the getri (buffer or USM version) function will be performed.

n The order of the matrix `A` (`0 ≤ n`).

lda The leading dimension of a (`lda ≥ n`).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `getri` (buffer or USM version) function must be able to hold.

11.45 getrs

Solves a system of linear equations with an LU-factored square coefficient matrix, with multiple right-hand sides. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.45.1 Description

The routine solves for X the following systems of linear equations:

Equation	Condition
A*X = B	if trans=mkl::transpose::nontrans
AT*X = B	if trans=mkl::transpose::trans
AH*X = B	if trans=mkl::transpose::conjtrans

Before calling this routine, you must call **getrf** to compute the LU factorization of A.

11.45.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void getrs(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t n,
  std::int64_t nrhs,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &ipiv,
  sycl::buffer<T> &b,
  std::int64_t ldb,
  sycl::buffer<T> &scratchpad,
  std::int64_t   scratchpad_size)
}
```

`getrs` supports the following precisions and devices:

T	Devices supported
float	CPU and GPU
double	CPU and GPU
std::complex<float>	CPU and GPU
std::complex<double>	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

trans Indicates the form of the equations:

If `trans=mkl::transpose::nontrans`, then `A*X = B` is solved for X.

If `trans=mkl::transpose::trans`, then `AT*X = B` is solved for X.

If `trans=mkl::transpose::conjtrans`, then `AH*X = B` is solved for X.

n The order of the matrix A and the number of rows in matrix B(0 ≤ n).

nrhs The number of right-hand sides (0 ≤ nrhs).

a Buffer holding the array containing the factorization of the matrix A, as returned by **getrf**. The second dimension of a must be at least max(1, n).

lda The leading dimension of a.

ipiv Buffer holding array of size at least max(1, n) as returned by **getrf**.

b The array b contains the matrix B whose columns are the right-hand sides for the systems of equations. The second dimension of b must be at least max(1,nrhs).

ldb The leading dimension of b.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the **getrs_scratchpad_size** function.

Output Parameters

b The buffer b is overwritten by the solution matrix X.

Exceptions

Exception	Description
mkl::lapack::exception	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If info = -i, the i-th parameter had an illegal value. If info = i, the i-th diagonal element of U is zero, and the solve could not be completed. If info is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Known Limitations

For GPU support, errors are reported through the info parameter, but computation does not halt for an algorithmic error.

11.46 getrs (USM Version)

Solves a system of linear equations with an LU-factored square coefficient matrix, with multiple right-hand sides. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.46.1 Description

The routine solves for `X` the following systems of linear equations:

Equation	Condition
`A*X = B`	if `trans=mkl::transpose::nontrans`
`AT*X = B`	if `trans=mkl::transpose::trans`
`AH*X = B`	if `trans=mkl::transpose::conjtrans`

Before calling this routine, you must call the getrf (USM Version) function to compute the `LU` factorization of `A`.

11.46.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getrs(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t n,
  std::int64_t nrhs,
  T *a,
  std::int64_t lda,
  std::int64_t *ipiv,
  T *b,
  std::int64_t ldb,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`getrs` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU, GPU
`double`	CPU, GPU
`std::complex<float>`	CPU, GPU
`std::complex<double>`	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

trans Indicates the form of the equations:

If `trans=mkl::transpose::nontrans`, then `A*X = B` is solved for `X`.

If `trans=mkl::transpose::trans`, then `AT*X = B` is solved for `X`.

If `trans=mkl::transpose::conjtrans`, then `AH*X = B` is solved for `X`.

n The order of the matrix `A` and the number of rows in matrix `B(0 ≤ n)`.

nrhs The number of right-hand sides (`0 ≤ nrhs`).

a Pointer to the array containing the factorization of the matrix `A`, as returned by **getrf (USM Version)**. The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a.

ipiv Array, size at least `max(1, n)`. The ipiv array, as returned by **getrf (USM Version)**.

b The array b contains the matrix `B` whose columns are the right-hand sides for the systems of equations. The second dimension of b must be at least `max(1,nrhs)`.

ldb The leading dimension of b.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the **getrs_scratchpad_size** function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

b The buffer b is overwritten by the solution matrix `X`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, the `i`-th diagonal element of `U` is zero, and the solve could not be completed. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.47 getrs_batch (Buffer Strided Version)

Solves a batch of linear equations with an LU-factored square coefficient matrices, with multiple right-hand sides. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.47.1 Description

The routine solves for X_i the following systems of linear equations:

- $A_i * X_i = B_i$ If `trans = mkl::transpose::notrans`
- $A_i^T * X_i = B_i$ If `trans = mkl::transpose::trans`
- $A_i^H * X_i = B_i$ If `trans = mkl::transpose::conjtrans`

Before calling this routine you must call **getrf_batch (Buffer Strided Version)** to compute the LU factorization of A_i.

11.47.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void getrs_batch(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t n,
  std::int64_t nrhs,
  sycl::buffer<T> &a,
  std::int64_t lda,
  std::int64_t stride_a,
  sycl::buffer<std::int64_t> &ipiv,
  std::int64_t stride_ipiv,
  sycl::buffer<T> &b,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

trans Indicates the form of the equations:

If trans = mkl::transpose::nontrans, then $A_i*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::trans, then $A_i^T*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::conjtrans, then $A_i^H*X_i = B_i$ is solved for X_i.

n The order of the matrices A_i and the number of rows in matrices B_i (`0 ≤ n`).

nrhs The number of right hand sides (`0 ≤ nrhs`).

a Array containing the factorizations of the matrices A_i, as returned by getrf_batch (Buffer Strided Version).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

ipiv The ipiv array, as returned by getrf_batch (Buffer Strided Version).

stride_ipiv The stride between the beginnings of arrays $ipiv_i$ inside the array `ipiv`.

b The array containing the matrices B_i whose columns are the right-hand sides for the systems of equations.

ldb The leading dimensions of B_i.

stride_b The stride between the beginnings of matrices B_i inside the batch array b.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by stride version of getrs_batch_scratchpad_size (Strided Version) function.

Output Parameters

b Overwritten by the solution matrices X_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

11.48 getrs_batch (Group Version)

Solves a batch of systems of linear equations with a batch of LU-factored square coefficient matrices, with multiple right-hand sides. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.48.1 Description

The routine solves for X_i ($i \in \{1 \ldots batch\_size\}$) the following systems of linear equations:

- $A_i * X_i = B_i$ If `trans = mkl::transpose::notrans`
- $A_i^T * X_i = B_i$ If `trans = mkl::transpose::trans`
- $A_i^H * X_i = B_i$ If `trans = mkl::transpose::conjtrans`

Before calling this routine you must call getrf_batch (Group Version) to compute the LU factorization of A_i.

The total number of problems to solve, `batch_size`, is a sum of sizes of all of the groups of parameters as provided by"group_sizes" array.

11.48.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getrs_batch(sycl::queue &queue,
  mkl::transpose *trans,
  std::int64_t *n,
  std::int64_t *nrhs,
  T **a,
  std::int64_t *lda,
  std::int64_t **ipiv,
  T **b,
  std::int64_t *ldb,
  std::int64_t group_count,
  std::int64_t *group_sizes,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

trans Array of `group_count` parameters $trans_g$ indicating the form of the equations for the group `g`:

If trans = mkl::transpose::nontrans, then $A_i * X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::trans, then $A_i^T * X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::conjtrans, then $A_i^H * X_i = B_i$ is solved for X_i.

n Array of `group_count` parameters n_g specifying the order of the matrices A_i and the number of rows in matrices B_i ($0 \le n_g$) belonging to group `g`.

nrhs Array of `group_count` parameters $nrhs_g$ specifying the number of right hand sides (`0 ≤ nrhs`) for group `g`.

a Array of `batch_size` pointers to factorization of the matricesA_i, as returned by getrf_batch (Group Version).

lda Array of `group_count` parameters lda_g specifying the leading dimension of A_i from group `g`.

ipiv The ipiv array, as returned by getrf_batch (Group Version).

b The array containing @ `batch_size` pointers to the matrices B_i whose columns are the right-hand sides for the systems of equations.

ldb Array of `group_count` parameters ldb_g specifying the leading dimensions of B_i in the group `g`.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of group_count integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by getrs_batch_scratchpad_size (Group Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

b Overwritten by the solution matrices X_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.49 getrs_batch (USM Strided Version)

Solves a system of linear equations with a batch of LU-factored square coefficient matrices, with multiple right-hand sides. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.49.1 Description

The routine solves for X_i the following systems of linear equations:

- $A_i * X_i = B_i$ If `trans = mkl::transpose::notrans`
- $A_i^T * X_i = B_i$ If `trans = mkl::transpose::trans`
- $A_i^H * X_i = B_i$ If `trans = mkl::transpose::conjtrans`

Before calling this routine you must call getrf_batch (USM Strided Version) to compute the LU factorization of A_i.

11.49.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getrs_batch(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t n,
  std::int64_t nrhs,
  T *a,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t *ipiv,
  std::int64_t stride_ipiv,
  T *b,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

trans Indicates the form of the equations:

If trans = mkl::transpose::nontrans, then $A_i*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::trans, then $A_i^T*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::conjtrans, then $A_i^H*X_i = B_i$ is solved for X_i.

n The order of the matrices A_i and the number of rows in matrices B_i (`0 ≤ n`).

nrhs The number of right hand sides (`0 ≤ nrhs`).

a Array containing the factorizations of the matrices A_i, as returned by getrf_batch (USM Strided Version).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices A_i inside the batch array `a`.

ipiv The ipiv array, as returned by getrf_batch (USM Strided Version).

stride_ipiv The stride between the beginnings of arrays $\texttt{ipiv}_i$ inside the array `ipiv`.

b The array containing the matrices B_i whose columns are the right-hand sides for the systems of equations.

ldb The leading dimensions of B_i.

stride_b The stride between the beginnings of matrices B_i inside the batch array `b`.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by stride version of getrs_batch_scratchpad_size (Strided Version) function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

b Overwritten by the solution matrices X_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.50 getrs_batch_scratchpad_size (Group Version)

Computes size of scratchpad memory required for `getrs_batch` (Group Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.50.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the getrs_batch (Group Version) function must be able to hold.

11.50.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t getrs_batch_scratchpad_size(sycl::queue &queue,
  mkl::transpose *trans,
  std::int64_t *n,
  std::int64_t *nrhs,
  std::int64_t *lda,
  std::int64_t *ldb,
  std::int64_t group_count,
  std::int64_t *group_sizes)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

trans Array of `group_count` parameters $trans_g$ indicating the form of the equations for the group g:

If trans = mkl::transpose::nontrans, then $A_i*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::trans, then $A_i^T*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::conjtrans, then $A_i^H*X_i = B_i$ is solved for X_i.

n Array of `group_count` parameters n_g specifying the order of the matrices A_i and the number of rows in matrices B_i (`0 ≤ ng`) belonging to group g.

nrhs Array of `group_count` parameters $nrhs_g$ specifying the number of right hand sides (`0 ≤ nrhsg`) for group g.

lda Array of `group_count` parameters lda_g specifying the leading dimension of A_i from group g.

ldb Array of `group_count` parameters ldb_g specifying the leading dimension of B_i from group g.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of `group_count` integers. Array element with index g specifies the number of problems to solve for each of the groups of parameters g. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the getrs_batch (Group Version) function must be able to hold.

11.51 getrs_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `getrs_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.51.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `getrs_batch` (Strided Version) function must be able to hold.

11.51.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t getrs_batch_scratchpad_size(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t n,
  std::int64_t nrhs,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t stride_ipiv,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

trans Indicates the form of the equations:

If trans = mkl::transpose::nontrans, then $A_i*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::trans, then $A_i^T*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::conjtrans, then $A_i^H*X_i = B_i$ is solved for X_i.

n The order of the matrices A_i and the number of rows in matrices B_i (`0 ≤ n`).

nrhs The number of right hand sides (`0 ≤ nrhs`).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices A_i inside the batch array `a`.

stride_ipiv The stride between the beginnings of arrays $ipiv_i$ inside the array `ipiv`.

ldb The leading dimensions of B_i.

stride_b The stride between the beginnings of matrices B_i inside the batch array `b`.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `getrs_batch` (Strided Version) function must be able to hold.

11.52 getrs_scratchpad_size

Computes size of scratchpad memory required for `getrs` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.52.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `getrs` (buffer or USM version) function should be able to hold.

11.52.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t getrs_scratchpad_size(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t n,
  std::int64_t nrhs,
  std::int64_t lda,
  std::int64_t ldb)
}
```

Input Parameters

queue Device queue where calculations by the getrs (buffer or USM version) function will be performed.

trans Indicates the form of the equations:

If `trans=mkl::transpose::nontrans`, then `A*X = B` is solved for X.

If `trans=mkl::transpose::trans`, then `AT*X = B` is solved for X.

If `trans=mkl::transpose::conjtrans`, then `AH*X = B` is solved for X.

n The order of the matrix A and the number of rows in matrix `B(0 ≤ n)`.

nrhs The number of right-hand sides (`0 ≤ nrhs`).

lda The leading dimension of a.

ldb The leading dimension of b.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `getrs` (buffer or USM version) function must be able to hold.

11.53 getrsnp_batch (Buffer Strided Version)

Solves a batch of linear equations with an LU-factored square coefficient matrices, with multiple right-hand sides. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.53.1 Description

The routine solves for X_i the following systems of linear equations:

- $A_i * X_i = B_i$ If `trans = mkl::transpose::notrans`
- $A_i^T * X_i = B_i$ If `trans = mkl::transpose::trans`
- $A_i^H * X_i = B_i$ If `trans = mkl::transpose::conjtrans`

Before calling this routine you must call getrfnp_batch (Buffer Strided Version) (without pivoting) to compute the LU factorization of A_i.

11.53.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void getrsnp_batch(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t n,
  std::int64_t nrhs,
  sycl::buffer<T> &a,
  std::int64_t lda,
  std::int64_t stride_a,
  sycl::buffer<T> &b,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

trans Indicates the form of the equations:

If trans = mkl::transpose::nontrans, then $A_i*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::trans, then $A_i^T*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::conjtrans, then $A_i^H*X_i = B_i$ is solved for X_i.

n The order of the matrices A_i and the number of rows in matrices B_i (`0 ≤ n`).

nrhs The number of right hand sides (`0 ≤ nrhs`).

a Array containing the factorizations of the matrices A_i, as returned by getrfnp_batch (Buffer Strided Version).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

b The array containing the matrices B_i whose columns are the right-hand sides for the systems of equations.

ldb The leading dimensions of B_i.

stride_b The stride between the beginnings of matrices B_i inside the batch array b.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by stride version of getrsnp_batch_scratchpad_size (Strided Version) function.

Output Parameters

b Overwritten by the solution matrices X_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

11.54 getrsnp_batch (USM Strided Version)

Solves a system of linear equations with a batch of LU-factored square coefficient matrices, with multiple right-hand sides. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.54.1 Description

The routine solves for X_i the following systems of linear equations:

- $A_i * X_i = B_i$ If `trans = mkl::transpose::notrans`
- $A_i^T * X_i = B_i$ If `trans = mkl::transpose::trans`
- $A_i^H * X_i = B_i$ If `trans = mkl::transpose::conjtrans`

Before calling this routine you must call getrfnp_batch (USM Strided Version) (without pivoting) to compute the LU factorization of A_i.

11.54.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event getrsnp_batch(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t n,
  std::int64_t nrhs,
  T *a,
  std::int64_t lda,
  std::int64_t stride_a,
  T *b,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

trans Indicates the form of the equations:

If trans = mkl::transpose::nontrans, then $A_i*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::trans, then $A_i^T*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::conjtrans, then $A_i^H*X_i = B_i$ is solved for X_i.

n The order of the matrices A_i and the number of rows in matrices B_i (`0 ≤ n`).

nrhs The number of right hand sides (`0 ≤ nrhs`).

a Array containing the factorizations of the matrices A_i, as returned by getrf_batch (USM Strided Version).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

b The array containing the matrices B_i whose columns are the right-hand sides for the systems of equations.

ldb The leading dimensions of B_i.

stride_b The stride between the beginnings of matrices B_i inside the batch array b.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by stride version of getrsnp_batch_scratchpad_size (Strided Version) function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

b Overwritten by the solution matrices X_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.55 getrsnp_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `getrsnp_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.55.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `getrsnp_batch` (Strided Version) function must be able to hold.

11.55.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t getrsnp_batch_scratchpad_size(sycl::queue &queue,
  mkl::transpose trans,
  std::int64_t n,
  std::int64_t nrhs,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

trans Indicates the form of the equations:

If trans = mkl::transpose::nontrans, then $A_i*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::trans, then $A_i^T*X_i = B_i$ is solved for X_i.

If trans = mkl::transpose::conjtrans, then $A_i^H*X_i = B_i$ is solved for X_i.

n The order of the matrices A_i and the number of rows in matrices B_i `(0 ≤ n)`.

nrhs The number of right hand sides `(0 ≤ nrhs)`.

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

ldb The leading dimensions of B_i.

stride_b The stride between the beginnings of matrices B_i inside the batch array b.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `getrsnp_batch` (Strided Version) function must be able to hold.

11.56 heevd

Computes all eigenvalues and, optionally, all eigenvectors of a complex Hermitian matrix using divide and conquer algorithm. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.56.1 Description

The routine computes all the eigenvalues, and optionally all the eigenvectors, of a complex Hermitian matrix `A`. In other words, it can compute the spectral factorization of `A` as: `A = Z*Λ*ZH`.

Here Λ is a real diagonal matrix whose diagonal elements are the eigenvalues λ_i, and `Z` is the (complex) unitary matrix whose columns are the eigenvectors z_i. Thus,

`A*zi = λi*zi` for `i = 1, 2, ..., n`.

If the eigenvectors are requested, then this routine uses a divide and conquer algorithm to compute eigenvalues and eigenvectors. However, if only eigenvalues are required, then it uses the Pal-Walker-Kahan variant of the `QL` or `QR` algorithm.

11.56.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void heevd(sycl::queue &queue,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &w,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`heevd` supports the following precision and devices.

T	Devices Supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

jobz Must be `job::novec` or `job::vec`.

If `jobz = job::novec`, then only eigenvalues are computed.

If `jobz = job::vec`, then eigenvalues and eigenvectors are computed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = job::upper`, a stores the upper triangular part of `A`.

If `uplo = job::lower`, a stores the lower triangular part of `A`.

n The order of the matrix `A` (`0 ≤ n`).

a Buffer holding the array containing `A`, size (`lda,*`). The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a. Must be at least `max(1,n)`.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the heevd_scratchpad_size function.

Output Parameters

a If `jobz = job::vec`, then on exit this buffer is overwritten by the unitary matrix `Z` which contains the eigenvectors of `A`.

w Buffer holding array of size at least n. If `info = 0`, contains the eigenvalues of the matrix `A` in ascending order. See also info.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, and `jobz = job::novec`, then the algorithm failed to converge; `i` indicates the number of off-diagonal elements of an intermediate tridiagonal form which did not converge to zero. If `info = i`, and `jobz = job::vec`, then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns `info/(n+1)` through `mod(info,n+1)`. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.57 heevd (USM Version)

Computes all eigenvalues and, optionally, all eigenvectors of a complex Hermitian matrix using divide and conquer algorithm. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.57.1 Description

The routine computes all the eigenvalues, and optionally all the eigenvectors, of a complex Hermitian matrix `A`. In other words, it can compute the spectral factorization of `A` as: `A = Z*Λ*ZH`.

Here Λ is a real diagonal matrix whose diagonal elements are the eigenvalues λ_i, and `Z` is the orthogonal matrix whose columns are the eigenvectors z_i. Thus,

`A*zi = λi*zi` for `i = 1, 2, ..., n`.

If the eigenvectors are requested, then this routine uses a divide and conquer algorithm to compute eigenvalues and eigenvectors. However, if only eigenvalues are required, then it uses the Pal-Walker-Kahan variant of the `QL` or `QR` algorithm.

11.57.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event heevd(sycl::queue &queue,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *w,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

heevd (USM version) supports the following precision and devices.

T	Devices Supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

jobz Must be `job::novec` or `job::vec`.

If `jobz = job::novec`, then only eigenvalues are computed.

If `jobz = job::vec`, then eigenvalues and eigenvectors are computed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `upper_lower = job::upper`, a stores the upper triangular part of `A`.

If `upper_lower = job::lower`, a stores the lower triangular part of `A`.

n The order of the matrix `A` (`0 ≤ n`).

a The pointer to the array containing `A`, size (`lda,*`). The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a. Must be at least `max(1,n)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the heevd_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a If `jobz = job::vec`, then on exit this is overwritten by the orthogonal matrix `Z` which contains the eigenvectors of `A`.

w Pointer to array of size at least n. Contains the eigenvalues of the matrix `A` in ascending order.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, and `jobz = job::novec`, then the algorithm failed to converge; `i` indicates the number of off-diagonal elements of an intermediate tridiagonal form which did not converge to zero. If `info = i`, and `jobz = job::vec`, then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns `info/(n+1)` through `mod(info,n+1)`. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.58 heevd_scratchpad_size

Computes size of scratchpad memory required for `heevd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.58.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `heevd` (buffer or USM version) function should be able to hold.

11.58.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  heevd_scratchpad_size(sycl::queue &queue,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the heevd (buffer or USM version) function will be performed.

jobz Must be `job::novec` or `job::vec`.

If `jobz = job::novec`, then only eigenvalues are computed.

If `jobz = job::vec`, then eigenvalues and eigenvectors are computed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = job::upper`, a stores the upper triangular part of `A`.

If `uplo = job::lower`, a stores the lower triangular part of `A`.

n The order of the matrix `A` (`0 ≤ n`).

lda The leading dimension of a. Must be at least `max(1,n)`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `heevd` (buffer or USM version) function must be able to hold.

11.59 hegvd

Computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian positive-definite eigenproblem using a divide and conquer method. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.59.1 Description

The routine computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian positive-definite eigenproblem, of the form

`A*x = λ*B*x, A*B*x = λ*x`, or `B*A*x = λ*x`.

Here `A` and `B` are assumed to be Hermitian and `B` is also positive definite.

It uses a divide and conquer algorithm.

11.59.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void hegvd(sycl::queue &queue,
  std::int64_t itype,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &b,
```

(continues on next page)

(continued from previous page)

```
  std::int64_t ldb,
  sycl::buffer<T> &w,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

hegvd supports the following precision and devices.

T	Devices Supported
std::complex<float>	CPU, GPU
std::complex<double>	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

itype Must be 1 or 2 or 3. Specifies the problem type to be solved:

if itype= 1, the problem type is A*x = lambda*B*x;

if itype= 2, the problem type is A*B*x = lambda*x;

if itype= 3, the problem type is B*A*x = lambda*x.

jobz Must be job::novec or job::vec.

If jobz = job::novec, then only eigenvalues are computed.

If jobz = job::vec, then eigenvalues and eigenvectors are computed.

uplo Must be uplo::upper or uplo::lower.

If uplo = uplo::upper, a and b store the upper triangular part of A and B.

If uplo = uplo::lower, a and b stores the lower triangular part of A and B.

n The order of the matrices A and B (0 ≤ n).

a Buffer holding the array of size a(lda,*) containing the upper or lower triangle of the Hermitian matrix A, as specified by uplo.

The second dimension of a must be at least max(1, n).

lda The leading dimension of a; at least max(1,n).

b Buffer holding the array of size b(ldb,*) containing the upper or lower triangle of the Hermitian matrix B, as specified by uplo.

The second dimension of b must be at least max(1, n).

ldb The leading dimension of b; at least max(1,n).

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the hegvd_scratchpad_size function.

Output Parameters

a On exit, if `jobz = job::vec`, then if `info = 0`, a contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows:

if itype= 1 or 2, Z^H*B*Z = I;

if itype= 3, Z^H*inv(B)*Z = I;

If `jobz = job::novec`, then on exit the upper triangle (if `uplo = uplo::upper`) or the lower triangle (if `uplo = uplo::lower`) of A, including the diagonal, is destroyed.

b On exit, if `info ≤ n`, the part of b containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^H*Uor B = L*L^H.

w Buffer holding arry of size at least n. If `info = 0`, contains the eigenvalues of the matrix A in ascending order. See also info.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the i-th parameter had an illegal value. For `info ≤ n`: If `info = i`, and `jobz = job::novec`, then the algorithm failed to converge; i indicates the number of off-diagonal elements of an intermediate tridiagonal form which did not converge to zero. If `info = i`, and `jobz = job:vec`, then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns `info/(n+1)` through `mod(info,n+1)`. For `info > n`: If `info = n + i`, for `1 ≤ i ≤ n`, then the leading minor of order i of B is not positive-definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.60 hegvd (USM Version)

Computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian positive-definite eigenproblem using a divide and conquer method. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.60.1 Description

The routine computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian positive-definite eigenproblem, of the form

`A*x = λ*B*x, A*B*x = λ*x`, or `B*A*x = λ*x`.

Here `A` and `B` are assumed to be Hermitian and `B` is also positive definite.

It uses a divide and conquer algorithm.

11.60.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event hegvd(sycl::queue &queue,
  std::int64_t itype,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *b,
  std::int64_t ldb,
  T *w,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`hegvd` (USM version) supports the following precision and devices.

T	Devices Supported
`std::complex<float>`	CPU, GPU
`std::complex<double>`	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

itype Must be 1 or 2 or 3. Specifies the problem type to be solved:

if itype= 1, the problem type is `A*x = lambda*B*x;`

if itype= 2, the problem type is `A*B*x = lambda*x;`

if itype= 3, the problem type is `B*A*x = lambda*x`.

jobz Must be `job::novec` or `job::vec`.

If `jobz = job::novec`, then only eigenvalues are computed.

If `jobz = job::vec`, then eigenvalues and eigenvectors are computed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a and b store the upper triangular part of `A` and `B`.

If `uplo = uplo::lower`, a and b stores the lower triangular part of `A` and `B`.

n The order of the matrices `A` and `B` (`0 ≤ n`).

a Pointer to the array of size a (`lda,*`) containing the upper or lower triangle of the Hermitian matrix `A`, as specified by uplo.

The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a; at least `max(1,n)`.

b Pointer to the array of size `b(ldb,*)` containing the upper or lower triangle of the Hermitian matrix `B`, as specified by uplo.

The second dimension of b must be at least `max(1, n)`.

ldb The leading dimension of b; at least `max(1,n)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the **hegvd_scratchpad_size** function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a On exit, if `jobz = job::vec`, then if `info = 0`, a contains the matrix `Z` of eigenvectors. The eigenvectors are normalized as follows:

if itype= 1 or 2, Z^H`*B*Z = I`;

if itype= 3, Z^H`*inv(B)*Z = I`;

If `jobz = job::novec`, then on exit the upper triangle (if `uplo = uplo::upper`) or the lower triangle (if `uplo = uplo::lower`) of `A`, including the diagonal, is destroyed.

b On exit, if `info ≤ n`, the part of b containing the matrix is overwritten by the triangular factor `U` or `L` from the Cholesky factorization `B = U`^{H}`*U` or `B = L*L`^{H}.

w Pointer to the array of size at least n. If `info = 0`, contains the eigenvalues of the matrix `A` in ascending order. See also info.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. For `info ≤ n`: If `info = i`, and `jobz = job::novec`, then the algorithm failed to converge; `i` indicates the number of off-diagonal elements of an intermediate tridiagonal form which did not converge to zero. If `info = i`, and `jobz = job:vec`, then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns `info/(n+1)` through `mod(info,n+1)`. For `info > n`: If `info = n + i`, for `1 ≤ i ≤ n`, then the leading minor of order `i` of `B` is not positive-definite. The factorization of `B` could not be completed and no eigenvalues or eigenvectors were computed. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.61 hegvd_scratchpad_size

Computes size of scratchpad memory required for `hegvd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.61.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `hegvd` (buffer or USM version) function must be able to hold.

11.61.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t hegvd_scratchpad_size(sycl::queue &queue,
  std::int64_t itype,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda,
  std::int64_t ldb)
}
```

Input Parameters

queue Device queue where calculations by the hegvd (buffer or USM version) function will be performed.

itype Must be 1 or 2 or 3. Specifies the problem type to be solved:

if itype= 1, the problem type is `A*x = lambda*B*x;`

if itype= 2, the problem type is `A*B*x = lambda*x;`

if itype= 3, the problem type is `B*A*x = lambda*x`.

jobz Must be `job::novec` or `job::vec`.

If `jobz = job::novec`, then only eigenvalues are computed.

If `jobz = job::vec`, then eigenvalues and eigenvectors are computed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a and b store the upper triangular part of `A` and `B`.

If `uplo = uplo::lower`, a and b stores the lower triangular part of `A` and `B`.

n The order of the matrices `A` and `B` (`0 ≤ n`).

lda The leading dimension of a; at least `max(1,n)`.

ldb The leading dimension of b; at least `max(1,n)`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `hegvd` (buffer or USM version) function must be able to hold.

11.62 hetrd

Reduces a complex Hermitian matrix to tridiagonal form. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.62.1 Description

The routine reduces a complex Hermitian matrix `A` to symmetric tridiagonal form `T` by a unitary similarity transformation: `A = Q*T*QH`. The unitary matrix `Q` is not formed explicitly but is represented as a product of `n-1` elementary reflectors. Routines are provided to work with `Q` in this representation.

11.62.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event hetrd(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t   lda,
  sycl::buffer<T> &d,
  sycl::buffer<T> &e,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`hetrd` supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue The device queue where calculations will be performed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a stores the upper triangular part of `A`.

If `uplo = uplo::lower`, a stores the lower triangular part of `A`.

n The order of the matrices `A(0 ≤ n)`.

a Buffer holding the matrix `A`, size `(lda,*)`. Contains the upper or lower triangle as specified by uplo.

lda The leading dimension of a; at least `max(1, n)`

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the **hetrd_scratchpad_size** function.

Output Parameters

a On exit,

if `uplo = uplo::upper`, the diagonal and first superdiagonal of `A` are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the buffer tau, represent the unitary matrix `Q` as a product of elementary reflectors;

if `uplo = uplo::lower`, the diagonal and first subdiagonal of `A` are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array tau, represent the unitary matrix `Q` as a product of elementary reflectors.

d Buffer holding the diagonal elements of the matrix T. The dimension of d must be at least `max(1, n)`.

e Buffer holding off diagonal elements of the matrix T. The dimension of e must be at least `max(1, n-1)`.

tau Buffer holding array of size at least `max(1, n)`. Stores `(n-1)` scalars that define elementary reflectors in decomposition of the unitary matrix `Q` in a product of `n-1` elementary reflectors. tau`(n)` is used as workspace.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.63 hetrd (USM Version)

Reduces a complex Hermitian matrix to tridiagonal form. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.63.1 Description

The routine reduces a complex Hermitian matrix `A` to Hermitian tridiagonal form `T` by a unitary similarity transformation: `A = Q*T*QH`. The unitary matrix `Q` is not formed explicitly but is represented as a product of `n-1` elementary reflectors. Routines are provided to work with `Q` in this representation.

11.63.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event hetrd(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *d,
  T *e,
  T *tau,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`hetrd` (USM version) supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue The device queue where calculations will be performed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a stores the upper triangular part of `A`.

If `uplo = uplo::lower`, a stores the lower triangular part of `A`.

n The order of the matrices `A(0 ≤ n)`.

a Pointer to the matrix `A`, size `(lda,*)`. Contains the upper or lower triangle as specified by uplo.

lda The leading dimension of a; at least `max(1, n)`

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the **hetrd_scratchpad_size** function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a On exit,

if `uplo = uplo::upper`, the diagonal and first superdiagonal of `A` are overwritten by the corresponding elements of the tridiagonal matrix `T`, and the elements above the first superdiagonal, with the buffer tau, represent the unitary matrix `Q` as a product of elementary reflectors;

if `uplo = uplo::lower`, the diagonal and first subdiagonal of `A` are overwritten by the corresponding elements of the tridiagonal matrix `T`, and the elements below the first subdiagonal, with the array tau, represent the unitary matrix `Q` as a product of elementary reflectors.

d Pointer to the diagonal elements of the matrix `T`. The dimension of d must be at least `max(1, n)`.

e Pointer to off diagonal elements of the matrix `T`. The dimension of e must be at least `max(1, n-1)`.

tau Pointer to memory array of size at least `max(1, n)`. Stores `(n-1)` scalars that define elementary reflectors in decomposition of the unitary matrix `Q` in a product of `n-1` elementary reflectors. `tau(n)` is used as workspace.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.64 hetrd_scratchpad_size

Computes size of scratchpad memory required for `hetrd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.64.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `hetrd` (buffer or USM version) function must be able to hold.

11.64.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t hetrd_scratchpad_size(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the hetrd (buffer or USM version) function will be performed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a stores the upper triangular part of `A`.

If `uplo = uplo::lower`, a stores the lower triangular part of `A`.

n The order of the matrices `A(0 ≤ n)`.

lda The leading dimension of a; at least `max(1, n)`

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `hetrd` (buffer or USM version) function must be able to hold.

11.65 hetrf

Computes the Bunch-Kaufman factorization of a complex Hermitian matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.65.1 Description

The routine computes the factorization of a complex Hermitian matrix `A` using the Bunch-Kaufman diagonal pivoting method:

- if `uplo='U'`, `A = U*D*UH`
- if `uplo='L'`, `A = L*D*LH`,

where `A` is the input matrix, `U` and `L` are products of permutation and triangular matrices with unit diagonal (upper triangular for `U` and lower triangular for `L`), and `D` is a Hermitian block-diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks. `U` and `L` have 2-by-2 unit diagonal blocks corresponding to the 2-by-2 blocks of `D`.

Note: This routine supports the Progress Routine feature.

11.65.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void hetrf(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &ipiv,
  sycl::buffer<T> &scratchpad, std::int64_t scratchpad_size)
}
```

`hetrf` supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue The device queue where calculations will be performed.

uplo Indicates whether the upper or lower triangular part of `A` is stored and how `A` is factored:.

If `uplo = uplo::upper`, the arraya stores the upper triangular part of `A` and `A` is factored as $U*D*U^H$.

If `uplo = uplo::lower`, the arraya stores the lower triangular part of `A` and `A` is factored as $L*D*L^H$.

n The order of the matrix `A(0 ≤ n)`.

a Buffer holding coefficients of matrix `A`, size `max(1,lda*n)`, containing either the upper or the lower triangular part of the matrix `A` (see uplo). The second dimension of a must be at least `max(1,n)`.

lda The leading dimension of a.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the **hetrf_scratchpad_size** function.

Output Parameters

a The upper or lower triangular part of a is overwritten by details of the block-diagonal matrix `D` and the multipliers used to obtain the factor `U` (or `L`).

ipiv Buffer holding array of size at least `max(1, n)`. Contains details of the interchanges and the block structure of `D`. If `ipiv(i) = k >0`, then `dii` is a 1-by-1 block, and the `i`-th row and column of `A` was interchanged with the k-th row and column.

If `uplo` = mkl::uplo::upper and `ipiv(i)` = `ipiv(i-1)` = -`m` < 0, then `D` has a 2-by-2 block in rows/columns `i` and `i-1`, and (`i-1`)-th row and column of `A` was interchanged with the `m`-th row and column.

If `uplo` = mkl::uplo::lower and `ipiv(i)` = `ipiv(i+1)` = -`m` < 0, then `D` has a 2-by-2 block in rows/columns `i` and `i+1`, and (`i+1`)-th row and column of `A` was interchanged with the `m`-th row and column.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, d:sub:`ii` is 0. The factorization has been completed, but `D` is exactly singular. Division by 0 will occur if you use `D` for solving a system of linear equations. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.66 hetrf (USM Version)

Computes the Bunch-Kaufman factorization of a complex Hermitian matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.66.1 Description

The routine computes the factorization of a complex Hermitian matrix `A` using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is:

- if `uplo=uplo::upper`, `A = U*D*UH`
- if `uplo=uplo::lower`, `A = L*D*LH`,

where `A` is the input matrix, `U` and `L` are products of permutation and triangular matrices with unit diagonal (upper triangular for `U` and lower triangular for `L`), and `D` is a Hermitian block-diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks. `U` and `L` have 2-by-2 unit diagonal blocks corresponding to the 2-by-2 blocks of `D`.

11.66.2 API

Syntax

```
namespace oneapi::mkl::lapack {
 sycl::event hetrf(sycl::queue &queue,
 mkl::uplo uplo,
 std::int64_t n,
 T *a,
```

(continues on next page)

(continued from previous page)

```
 std::int64_t lda,
 std::int64_t *ipiv,
 T *scratchpad,
 std::int64_t scratchpad_size,
 const std::vector<sycl::event> &events = {})
}
```

`hetrf` (USM version) supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue The device queue where calculations will be performed.

uplo Indicates whether the upper or lower triangular part of `A` is stored and how `A` is factored:.

If `uplo = uplo::upper`, the arraya stores the upper triangular part of `A` and `A` is factored as $U*D*U^H$.

If `uplo = uplo::lower`, the arraya stores the lower triangular part of `A` and `A` is factored as $L*D*L^H$.

n The order of the matrix `A(0 ≤ n)`.

a The pointer to coefficients of matrix `A`, size `max(1,lda*n)`, containing either the upper or the lower triangular part of the matrix `A` (see uplo). The second dimension of a must be at least `max(1,n)`.

lda The leading dimension of a.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the **hetrf_scratchpad_size** function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a The upper or lower triangular part of a is overwritten by details of the block-diagonal matrix `D` and the multipliers used to obtain the factor `U` (or `L`).

ipiv Pointer to memory array of size at least `max(1, n)`. Contains details of the interchanges and the block structure of `D`. If `ipiv(i) = k >0`, then `dii` is a 1-by-1 block, and the `i`-th row and column of `A` was interchanged with the k-th row and column.

If `uplo` = mkl::uplo::upper and `ipiv(i)` =`ipiv(i-1)` = -`m` < 0, then `D` has a 2-by-2 block in rows/columns `i` and `i-1`, and (`i-1`)-th row and column of `A` was interchanged with the `m`-th row and column.

If `uplo` = mkl::uplo::lower and `ipiv(i)` =`ipiv(i+1)` = -`m` < 0, then `D` has a 2-by-2 block in rows/columns `i` and `i+1`, and (`i+1`)-th row and column of `A` was interchanged with the `m`-th row and column.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, d:sub:`ii` is 0. The factorization has been completed, but `D` is exactly singular. Division by 0 will occur if you use `D` for solving a system of linear equations. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.67 hetrf_scratchpad_size

Computes size of scratchpad memory required for `hetrf` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.67.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `hetrf` (buffer or USM version) function should be able to hold.

11.67.2 API

Syntax

```
namespace oneapi::mkl::lapack {
template<typename T>
std::int64_t hetrf_scratchpad_size(sycl::queue &queue,
mkl::uplo uplo,
std::int64_t n,
std::int64_t lda)

}
```

Input Parameters

queue Device queue where calculations by the hetrf (buffer or USM version) function will be performed.

uplo Indicates whether the upper or lower triangular part of `A` is stored and how `A` is factored:

If uplo = uplo::upper, the array a stores the upper triangular part of the matrix `A`, and `A` is factored as $U*D*U^H$.

If uplo = uplo::lower, the array a stores the lower triangular part of the matrix `A`, and `A` is factored as $L*D*L^H$.

n The order of matrix `A` (`0 ≤ n`).

lda The leading dimension of a.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `hetrf` (buffer or USM version) function must be able to hold.

11.68 orgbr

Generates the real orthogonal matrix `Q` or P^T determined by gebrd. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.68.1 Description

The routine generates the whole or part of the orthogonal matrices `Q` and P^T formed by the routines gebrd. Use this routine after a call to `sgebrd/dgebrd`. All valid combinations of arguments are described in **Input parameters**. In most cases you need the following:

To compute the whole `m`-by-`m` matrix `Q`:

```
orgbr(queue, generate::q, m, m, n, a, ...)
```

(note that the array a must have at least `m` columns).

To form the `n` leading columns of `Q` if `m > n`:

```
orgbr(queue, generate::q, m, n, n, a, ...)
```

To compute the whole n-by-n matrix P^T:

```
orgbr(queue, generate::p, n, n, m, a, ...)
```

(note that the array a must have at least n rows).

To form the `m` leading rows of P^T if `m < n`:

```
orgbr(queue, generate::p, m, n, m, a, ...)
```

11.68.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void orgbr(sycl::queue &queue,
  mkl::generate gen,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`orgbr` supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU

Input Parameters

queue Device queue where calculations will be performed.

gen Must be `generate::q` or `generate::p`.

If gen= `generate::q`, the routine generates the matrix `Q`.

If gen= `generate::p`, the routine generates the matrix P^T.

m The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ m)`.

If gen= `generate::q`, `m ≥ n ≥ min(m, k)`.

If gen= `generate::p`, `n ≥ m ≥ min(n, k)`.

n The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ n)`. See m for constraints.

k If gen= `generate::q`, the number of columns in the original `m`-by-k matrix reduced by **gebrd**.

If gen= `generate::p`, the number of rows in the original k-by-n matrix reduced by **gebrd**.

a Buffer holding memory returned by **gebrd**.

lda The leading dimension of a.

tau Buffer holding array of size `min (m,k)` if gen= `generate::q`, and of size `min(n,k)` if gen= `generate::p`. Scalar factor of the elementary reflectors, as returned by **gebrd** in the array tauq or taup.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the **orgbr_scratchpad_size** function.

Output Parameters

a Overwritten by n leading columns of the m-by-m orthogonal matrix `Q` or P^T (or the leading rows or columns thereof) as specified by gen, m, and n.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.69 orgbr (USM Version)

Generates the real orthogonal matrix `Q` or P^T determined by **gebrd (USM Version)**. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.69.1 Description

The routine generates the whole or part of the orthogonal matrices `Q` and P^T formed by the gebrd (USM Version) function. All valid combinations of arguments are described in **Input parameters**. In most cases you need the following:

To compute the whole `m`-by-`m` matrix `Q`:

```
orgbr(queue, generate::q, m, m, n, a, ...)
```

(note that the array a must have at least `m` columns).

To form the `n` leading columns of `Q` if `m > n`:

```
orgbr(queue, generate::q, m, n, n, a, ...)
```

To compute the whole `n`-by-`n` matrix P^T:

```
orgbr(queue, generate::p, n, n, m, a, ...)
```

(note that the array a must have at least `n` rows).

To form the `m` leading rows of P^T if `m < n`:

```
orgbr(queue, generate::p, m, n, m, a, ...)
```

11.69.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event orgbr(sycl::queue &queue,
  mkl::generate gen,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  T *a,
  std::int64_t lda,
  T *tau,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`orgbr` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU

Input Parameters

queue Device queue where calculations will be performed.

gen Must be `generate::q` or `generate::p`.

If gen= `generate::q`, the routine generates the matrix `Q`.

If gen= `generate::p`, the routine generates the matrix P^T.

m The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ m)`.

If gen= `generate::q`, `m ≥ n ≥ min(m, k)`.

If gen= `generate::p`, `n ≥ m ≥ min(n, k)`.

n The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ n)`. See m for constraints.

k If gen= `generate::q`, the number of columns in the original `m`-by-k matrix reduced by gebrd (USM Version).

If gen= `generate::p`, the number of rows in the original k-by-n matrix reduced by gebrd (USM Version).

a Pointer to memory returned by gebrd (USM Version).

lda The leading dimension of a.

tau Pointer to memory of size `min (m,k)` if gen= `generate::q`, and of size `min(n,k)` if gen= `generate::p`. Scalar factor of the elementary reflectors, as returned by gebrd (USM Version) in the array `tauq` or `taup`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the orgbr_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by n leading columns of the m-by-m orthogonal matrix `Q` or P^T (or the leading rows or columns thereof) as specified by gen, m, and n.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.70 orgbr_scratchpad_size

Computes size of scratchpad memory required for `orgbr` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.70.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `orgbr` (buffer or USM version) function must be able to hold.

11.70.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t orgbr_scratchpad_size(sycl::queue &queue,
  mkl::generate gen,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the orgbr (buffer or USM version) function will be performed.

gen Must be `generate::q` or `generate::p`.

If gen= `generate::q`, the routine generates the matrix `Q`.

If gen= `generate::p`, the routine generates the matrix P^T.

m The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ m)`.

If gen= `generate::q`, `m ≥ n ≥ min(m, k)`.

If gen= `generate::p`, `n ≥ m ≥ min(n, k)`.

n The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ n)`. See m for constraints.

k If gen= `generate::q`, the number of columns in the original `m`-by-k matrix reduced by gebrd (USM Version).

If gen= `generate::p`, the number of rows in the original k-by-n matrix reduced by gebrd (USM Version).

lda The leading dimension of a.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `orgbr` (buffer or USM version) function must be able to hold.

11.71 orgqr

Generates the real orthogonal matrix Q of the QR factorization formed by `geqrf`. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.71.1 Description

The routine generates the whole or part of `m`-by-`m` orthogonal matrix `Q` of the `QR` factorization formed by the routine geqrf.

Usually `Q` is determined from the `QR` factorization of an `m` by `p` matrix `A` with `m` ≥ `p`. To compute the whole matrix `Q`, use:

```
mkl::orgqr(queue, m, m, p, a, lda, tau, ...)
```

To compute the leading `p` columns of `Q` (which form an orthonormal basis in the space spanned by the columns of `A`):

```
mkl::orgqr(queue, m, p, p, a, lda, tau, ...)
```

To compute the matrix Q^k of the `QR` factorization of leading `k` columns of the matrix `A`:

```
mkl::orgqr(queue, m, m, k, a, lda, tau, ...)
```

To compute the leading `k` columns of Q^k (which form an orthonormal basis in the space spanned by leading `k` columns of the matrix `A`):

```
mkl::orgqr(queue, m, k, k, a, lda, tau, ...)
```

11.71.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void orgqr(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`orgqr` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a Buffer holding the result of geqrf.

lda The leading dimension of a (`lda ≥ m`).

tau Buffer holding the result of geqrf.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the orgqr_scratchpad_size function.

Output Parameters

a Overwritten by n leading columns of the m-by-m orthogonal matrix `Q`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.72 orgqr (USM Version)

Generates the real orthogonal matrix Q of the QR factorization formed by the geqrf (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.72.1 Description

The routine generates the whole or part of `m`-by-`m` orthogonal matrix `Q` of the `QR` factorization formed by the routine **geqrf (USM Version) :ref:'geqrf-usm-version** function.

Usually `Q` is determined from the `QR` factorization of an `m` by `p` matrix `A` with $m \geq p$. To compute the whole matrix `Q`, use:

```
mkl::orgqr(queue, m, m, p, a, lda, tau, ...)
```

To compute the leading `p` columns of `Q` (which form an orthonormal basis in the space spanned by the columns of `A`):

```
mkl::orgqr(queue, m, p, p, a, lda, tau, ...)
```

To compute the matrix Q^k of the `QR` factorization of leading `k` columns of the matrix `A`:

```
mkl::orgqr(queue, m, m, k, a, lda, tau, ...)
```

To compute the leading `k` columns of Q^k (which form an orthonormal basis in the space spanned by leading `k` columns of the matrix `A`):

```
mkl::orgqr(queue, m, k, k, a, lda, tau, ...)
```

11.72.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event orgqr(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  T *a,
  std::int64_t lda,
  T *tau,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`orgqr` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a Pointer to the result of geqrf (USM Version).

lda The leading dimension of a (`lda ≥ m`).

tau Pointer to the result of geqrf (USM Version).

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the orgqr_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by n leading columns of the m-by-m orthogonal matrix `Q`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, d_{ii} is 0. The factorization has been completed, but `D` is exactly singular. Division by 0 will occur if you use `D` for solving a system of linear equations. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete

11.73 orgqr_batch (Buffer Strided Version)

Generates the real orthogonal matrices Q_i of the batch QR factorizations formed by `geqrf_batch` (Buffer Strided Version). This routine belongs to the `oneapi::mkl::lapack` namespace.

11.73.1 Description

The routine generates the wholes or parts of the orthogonal matrices Q_i of the batch of QR factorizations formed by the routine geqrf_batch (Buffer Strided Version). Usually, Q_i is determined from the QR factorization of an `m`-by-`p` matrix A_i with $m \geq p$.

To compute the whole matrices Q_i, use:

```
orgqr_batch(queue, m, m, p, a, ...)
```

To compute the leading `p` columns of Q_i (which form an orthonormal basis in the space spanned by the columns of A_i):

```
orgqr_batch(queue, m, p, p, a, ...)
```

To compute the matrices Q_i^k of the QR factorizations of leading `k` columns of the matrices A_i:

```
orgqr_batch(queue, m, m, k, a, ...)
```

To compute the leading k columns of Q_i^k (which form an orthonormal basis in the space spanned by leading k columns of the matrices A_i):

```
orgqr_batch(queue, m, k, k, a, ...)
```

11.73.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void orgqr_batch(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  sycl::buffer<T> &a,
  std::int64_t lda,
  std::int64_t stride_a,
  sycl::buffer<T> &tau,
  std::int64_t stride_tau,
  std::int64_t batch_size,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i ($0 \le m$).

n The number of columns in the matrices A_i ($0 \le n$).

k the number of elementary reflectors whose product defines the matrices Q_i ($0 \le k \le n$).

a Array resulting after a call to geqrf_batch (Buffer Strided Version).

lda The leading dimension of A_i ($lda \ge m$).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

tau Array resulting after a call to geqrf_batch (Buffer Strided Version).

stride_tau The stride between the beginnings of arrays tau_i inside the array tau.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by orgqr_batch_scratchpad_size (Strided Version).

Output Parameters

a Array data is overwritten by a batch of n leading columns of the m-by-m orthogonal matrices Q_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

11.74 orgqr_batch (Group Version)

Generates the real orthogonal matrices Q_i of the batch of QR factorizations formed by `geqrf_batch` (Group Version). This routine belongs to the `oneapi::mkl::lapack` namespace.

11.74.1 Description

The routine generates the wholes or parts of the m-by-m orthogonal matrices Q_i of the batch of QR factorizations formed by the routine geqrf_batch (Group Version). Usually, Q_i is determined from the QR factorization of an m-by-p matrix A_i with $m \geq p$.

To compute the whole matrices Q_i, use:

```
orgqr_batch(queue, m, m, p, a, ...)
```

To compute the leading p columns of Q_i (which form an orthonormal basis in the space spanned by the columns of A_i):

```
orgqr_batch(queue, m, p, p, a, ...)
```

To compute the matrices Q_i^k of the QR factorizations of leading k columns of the matrices A_i:

```
orgqr_batch(queue, m, m, k, a, ...)
```

To compute the leading k columns of Q_i^k (which form an orthonormal basis in the space spanned by leading k columns of the matrices A_i):

```
orgqr_batch(queue, m, k, k, a, ...)
```

11.74.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event orgqr_batch(sycl::queue &queue,
  std::int64_t *m,
  std::int64_t *n,
  std::int64_t *k,
  T **a,
  std::int64_t *lda,
  T **tau,
  std::int64_t group_count,
  std::int64_t *group_sizes,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m Array of `group_count` parameters m_g as previously supplied to geqrf_batch (Group Version).

n Array of `group_count` parameters n_g as previously supplied to geqrf_batch (Group Version).

k Array of `group_count` parameters k_g as previously supplied to geqrf_batch (Group Version). The number of elementary reflectors whose product defines the matrices Q_i ($0 \le k_g \le n_g$).

a Array resulting after a call to **geqrf_batch (Group Version) <geqrf_batch-group-version**.

lda Array of leading dimension of A_i as previously supplied to geqrf_batch (Group Version).

tau Array resulting after a call to geqrf_batch (Group Version).

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of group_count integers. Array element with index g specifies the number of problems to solve for each of the groups of parameters g. So the total number of problems to solve, batch_size, is a sum of all parameter group sizes.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by orgqr_batch_scratchpad_size (Group Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Matrices pointed to by array a are overwritten by n_g leading columns of the m_g-by-m_g orthogonal matrices Q_i, where g is an index of group of parameters corresponding to Q_i.

Exceptions

Exception	Description
mkl::lapack::batch_exception	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If info = -n, the n-th parameter had an illegal value. If info equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.75 orgqr_batch (USM Strided Version)

Generates the orthogonal matrices Q_i of the batch QR factorizations formed by geqrf_batch (USM Strided Version). This routine belongs to the oneapi::mkl::lapack namespace.

11.75.1 Description

The routine generates the wholes or parts of the orthogonal matrices Q_i of the batch of QR factorizations formed by the routine **geqrf_batch (USM Strided Version)**. Usually, Q_i is determined from the QR factorization of an m-by-p matrix A_i with $m \geq p$.

To compute the whole matrices Q_i, use:

```
orgqr_batch(queue, m, m, p, a, ...)
```

To compute the leading p columns of Q_i (which form an orthonormal basis in the space spanned by the columns of A_i):

```
orgqr_batch(queue, m, p, p, a, ...)
```

To compute the matrices Q_i^k of the QR factorizations of leading k columns of the matrices A_i:

```
orgqr_batch(queue, m, m, k, a, ...)
```

To compute the leading k columns of Q_i^k (which form an orthonormal basis in the space spanned by leading k columns of the matrices A_i):

```
orgqr_batch(queue, m, k, k, a, ...)
```

11.75.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event orgqr_batch(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  T *a,
  std::int64_t lda,
  std::int64_t stride_a,
  T *tau,
  std::int64_t stride_tau,
  std::int64_t batch_size,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i ($0 \le m$).

n The number of columns in the matrices A_i ($0 \le n$).

k the number of elementary reflectors whose product defines the matrices Q_i ($0 \le k \le n$).

a Array resulting after a call to geqrf_batch (USM Strided Version).

lda The leading dimension of A_i ($lda \ge m$).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

tau Array resulting after a call to geqrf_batch (USM Strided Version).

stride_tau The stride between the beginnings of arrays tau_i inside the array tau.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by orgqr_batch_scratchpad_size (Strided Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Array data is overwritten by a batch of n leading columns of the m-by-m orthogonal matrices Q_i.

Exceptions

Exception	Description
mkl::lapack::batch_exception	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If info = -n, the n-th parameter had an illegal value. If info equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.76 orgqr_batch_scratchpad_size (Group Version)

Computes size of scratchpad memory required for `orgqr_batch` (Group Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.76.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the orgqr_batch (Group Version) function must be able to hold.

11.76.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t orgqr_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t *m,
  std::int64_t *n,
  std::int64_t *k,
  std::int64_t *lda,
  std::int64_t group_count,
  std::int64_t *group_sizes)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m Array of `group_count` parameters m_g as previously supplied to orgqr_batch (Group Version).

n Array of `group_count` parameters n_g as previously supplied to orgqr_batch (Group Version).

k Array of `group_count` parameters k_g as previously supplied to orgqr_batch (Group Version).

The number of elementary reflectors whose product defines the matrices Q_i ($0 \leq k_g \leq n_{gn}$).

lda The leading dimension of A_i as previously supplied to orgqr_batch (Group Version).

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of `group_count` integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the orgqr_batch (Group Version) function must be able to hold.

11.77 orgqr_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `orgqr_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.77.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `orgqr_batch` (Strided Version) function must be able to hold.

11.77.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t orgqr_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t stride_tau,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i ($0 \le m$).

n The number of columns in the matrices A_i ($0 \le n$).

k the number of elementary reflectors whose product defines the matrices Q_i ($0 \le k \le n$).

lda The leading dimension of A_i ($lda \ge m$).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

stride_tau The stride between the beginnings of arrays tau_i inside the array `tau`.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `orgqr_batch` (Strided Version) function must be able to hold.

11.78 orgqr_scratchpad_size

Computes size of scratchpad memory required for `orgqr` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.78.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `orgqr` (buffer or USM version) function must be able to hold.

11.78.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t orgqr_scratchpad_size(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the orgqr (buffer or USM version) function will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

lda The leading dimension of a (`lda ≥ m`).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `orgqr` (buffer or USM version) function must be able to hold.

11.79 orgtr

Generates the real orthogonal matrix Q determined by sytrd. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.79.1 Description

The routine explicitly generates the n-by-n orthogonal matrix `Q` formed by **sytrd** when reducing a real symmetric matrix `A` to tridiagonal form: `A = Q*T*QT`. Use this routine after a call to **sytrd**.

11.79.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void orgtr(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`orgtr` supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Must be uplo::upper or uplo::lower. Uses the same uplo as supplied to the **sytrd** function

n The order of matrix `Q` (`0 ≤ n`).

a The buffer a returned by the **sytrd** function. The second dimension of a must be at least `max(1, n)`.

tau Buffer holding `tau` returned by the **sytrd** function.

lda The leading dimension of a (`lda ≥ n`).

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the **orgtr_scratchpad_size** function.

Output Parameters

a Overwritten by the orthogonal matrix `Q`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.80 orgtr (USM Version)

Generates the real orthogonal matrix Q determined by the `sytrd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.80.1 Description

The routine explicitly generates the n-by-n orthogonal matrix `Q` formed by sytrd (USM Version) when reducing a real symmetric matrix `A` to tridiagonal form: `A = Q*T*QT`. Use this routine after a call to sytrd (USM Version).

11.80.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event orgtr(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *tau,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`orgtr` (USM version) supports the following precision and devices.

T	Devices Supported
`float`	CPU
`double`	CPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Must be uplo::upper or uplo::lower. Uses the same uplo as supplied to the sytrd (USM Version) function

n The order of matrix `Q` (`0 ≤ n`).

a The pointer a returned by the sytrd (USM Version) function. The second dimension of a must be at least `max(1, n)`.

tau The pointer to `tau` returned by the geqrf (USM Version) function.

lda The leading dimension of a (`lda ≥ n`).

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the orgtr_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by the orthogonal matrix `Q`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.81 orgtr_scratchpad_size

Computes size of scratchpad memory required for `orgtr` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.81.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `orgtr` (buffer or USM version) function must be able to hold.

11.81.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  orgtr_scratchpad_size(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the orgtr (buffer or USM version) function will be performed.

uplo Must be uplo::upper or uplo::lower. Uses the same uplo as supplied to the sytrd (USM Version) function.

n The order of matrix `Q` (`0 ≤ n`).

lda The leading dimension of a (`lda ≥ n`).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `orgtr` (buffer or USM version) function must be able to hold.

11.82 ormqr

Multiplies a real matrix by the orthogonal matrix Q of the QR factorization formed by `geqrf`. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.82.1 Description

The routine multiplies a real matrix `C` by `Q` or Q^T, where `Q` is the orthogonal matrix `Q` of the `QR` factorization formed by the routine geqrf.

Depending on the parameters `left_right` and `trans`, the routine can form one of the matrix products `Q*C`, `QT*C`, `C*Q`, or `C*QT` (overwriting the result on `C`).

11.82.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void ormqr(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &c,
  std::int64_t ldc,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`ormqr` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

side If `side` = `mkl::side::left`, `Q` or Q^T is applied to `C` from the left. If `side` = `mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a Buffer holding the result of the geqrf function. . The second dimension of a must be at least `max(1,k)`.

lda The leading dimension of a.

tau Buffer holding tau returned by the geqrf function.

c Buffer holding the matrix `C`. The second dimension of c must be at least `max(1,n)`.

ldc The leading dimension of c.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the ormqr_scratchpad_size function.

Output Parameters

c Overwritten by the product `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (as specified by left_right and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.83 ormqr (USM Version)

Multiplies a real matrix by the orthogonal matrix Q of the QR factorization formed by the `geqrf` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.83.1 Description

The routine multiplies a real matrix `C` by `Q` or Q^T, where `Q` is the orthogonal matrix `Q` of the `QR` factorization formed by the routine geqrf (USM Version).

Depending on the parameters `side` and `trans`, the routine can form one of the matrix products `Q*C`, `QT*C`, `C*Q`, or `C*QT` (overwriting the result on `C`).

11.83.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event ormqr(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  T *a,
  std::int64_t lda,
  T *tau,
  T *c,
  std::int64_t ldc,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`ormqr` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU, GPU
`double`	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

side If `side` = `mkl::side::left`, `Q` or Q^T is applied to `C` from the left. If `side` = `mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a The pointer to the result of the geqrf (USM Version) function. . The second dimension of a must be at least `max(1,k)`.

lda The leading dimension of a.

tau The pointer to tau returned by the geqrf (USM Version) function.

c The pointer to the matrix `C`. The second dimension of c must be at least `max(1,n)`.

ldc The leading dimension of c.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the ormqr_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

c Overwritten by the product `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (as specified by side and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.84 ormqr_scratchpad_size

Computes size of scratchpad memory required for `ormqr` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.84.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `ormqr` (buffer or USM version) function must be able to hold.

11.84.2 API

Syntax

```
namespace oneapi::mkl::lapack {
 template<typename T>
 std::int64_t ormqr_scratchpad_size(sycl::queue &queue,
 mkl::side side,
 mkl::transpose trans,
 std::int64_t m,
 std::int64_t n,
 std::int64_t k,
 std::int64_t lda,
 std::int64_t ldc)
}
```

Input Parameters

queue Device queue where calculations by the ormqr (buffer or USM version) function will be performed.

side If `side=mkl::side::left`, `Q` or Q^T is applied to `C` from the left.

If `side=mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

lda The leading dimension of a.

ldc The leading dimension of c.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `ormqr` (buffer or USM version) function must be able to hold.

11.85 ormrq

Multiplies a real matrix by the orthogonal matrix Q of the RQ factorization formed by `gerqf`.

11.85.1 Description

The routine multiplies a real `m`-by-`n` matrix `C` by `Q` or Q^T, where `Q` is the real orthogonal matrix defined as a product of `k` elementary reflectors H_i : `Q = H1H2 ... Hk` as returned by the `RQ` factorization routine **gerqf**.

Depending on the parameters `side` and `trans`, the routine can form one of the matrix products `Q*C`, `QT*C`, `C*Q`, or `C*QT` (overwriting the result over `C`).

11.85.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void ormrq(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &c,
  std::int64_t ldc,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`ormrq` supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU

Input Parameters

queue Device queue where calculations will be performed.

side If `side` = `mkl::side::left`, `Q` or Q^T is applied to `C` from the left. If `side` = `mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a Buffer holding the result of the gerqf function. The second dimension of a must be at least `max(1,k)`.

lda The leading dimension of a.

tau Buffer holding tau returned by the gerqf function.

c Buffer holding the matrix `C`. The second dimension of c must be at least `max(1,n)`.

ldc The leading dimension of c.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the ormrq_scratchpad_size function.

Output Parameters

c Overwritten by the product `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (as specified by side and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Application Notes

The complex counterpart of this routine is unmrq.

11.86 ormrq (USM Version)

Multiplies a real matrix by the orthogonal matrix Q of the RQ factorization formed by the `gerqf` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.86.1 Description

The routine multiplies a real matrix `C` by `Q` or Q^T, where `Q` is the real orthogonal matrix `Q` of the `RQ` factorization routine gerqf (USM Version).

Depending on the parameters `side` and `trans`, the routine can form one of the matrix products `Q*C`, `QT*C`, `C*Q`, or `C*QT` (overwriting the result on `C`).

11.86.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event ormrq(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  T *a,
  std::int64_t lda,
  T *tau,
  T *c,
  std::int64_t ldc,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`ormrq` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU

Input Parameters

queue Device queue where calculations will be performed.

side If `side` = `mkl::side::left`, `Q` or Q^T is applied to `C` from the left. If `side` = `mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a The pointer to the result of the gerqf (USM Version) function. The second dimension of a must be at least `max(1,k)`.

lda The leading dimension of a.

tau The pointer to tau returned by the gerqf (USM Version) function.

c The pointer to the matrix `C`. The second dimension of c must be at least `max(1,n)`.

ldc The leading dimension of c.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the ormrq_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

c Overwritten by the product `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (as specified by side and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.87 ormrq_scratchpad_size

Computes size of scratchpad memory required for `ormrq` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.87.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `ormrq` (buffer or USM version) function must be able to hold.

11.87.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t ormrq_scratchpad_size(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  std::int64_t lda,
   std::int64_t ldc)
}
```

Input Parameters

queue Device queue where calculations by the ormrq (buffer or USM version) function will be performed.

side If `side` = `mkl::side::left`, `Q` or Q^T is applied to `C` from the left. If `side` = `mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

lda The leading dimension of a.

ldc The leading dimension of c.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `ormrq` (buffer or USM version) function must be able to hold.

11.88 ormtr

Multiplies a real matrix by the real orthogonal matrix `Q` determined by **sytrd**. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.88.1 Description

Q^T, where `Q` is the orthogonal matrix `Q` formed by **sytrd** when reducing a real symmetric matrix `A` to tridiagonal form: `A = Q*T*QT`. Use this routine after a call to **sytrd**.

Depending on the parameters left_right and trans, the routine can form one of the matrix products `Q*C`, `QT*C`, `C*Q`, or `C*QT` (overwriting the result on `C`).

The routine multiplies a real matrix `C` by `Q` or

11.88.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void ormtr(sycl::queue &queue,
  mkl::side side,
  mkl::uplo uplo,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &c,
```

(continues on next page)

(continued from previous page)

```
  std::int64_t ldc,
  sycl::buffer<T> &scratchpad,
   std::int64_t scratchpad_size)
}
```

`ormtr` supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU

Input Parameters

In the descriptions below, `r` denotes the order of `Q`:

`r = m`	**if** `left_right = side::left`
`r = n`	if `left_right = side::right`

queue Device queue where calculations will be performed.

side Must be either `side::left` or `side::right`.

If `side = side::left`, `Q` or Q^T is applied to `C` from the left.

If `side = side::right`, `Q` or Q^T is applied to `C` from the right.

uplo Must be either `uplo::upper` or `uplo::lower`. Uses the same `uplo` as supplied to **sytrd**.

trans Must be either `transpose::nontrans` or `transpose::trans`.

If `trans = transpose::nontrans`, the routine multiplies `C` by `Q`.

If `trans = transpose::trans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `C(m ≥ 0)`.

n The number of columns in the matrix `C(n ≥ 0)`.

a Buffer holding array returned by **sytrd**.

lda The leading dimension of a `(max(1, r) ≤ lda)`.

tau Buffer holding tau returned by **sytrd**. The dimension of tau must be at least `max(1, r-1)`.

c Buffer holding the matrix `C`. The second dimension of c must be at least `max(1, n)`.

ldc The leading dimension of `c (max(1, n) ≤ ldc)`.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the **ormtr_scratchpad_size** function.

Output Parameters

c Overwritten by the product `Q*C`, `QT*C`, `C*Q`, or `C*QT` (as specified by side and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.89 ormtr (USM Version)

Multiplies a real matrix by the real orthogonal matrix `Q` determined by the `sytrd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.89.1 Description

The routine multiplies a real matrix `C` by `Q` or Q^T, where `Q` is the orthogonal matrix `Q` formed by sytrd (USM Version) when reducing a real symmetric matrix `A` to tridiagonal form: `A = Q*T*QT`. Use this routine after a call to sytrd (USM Version).

Depending on the parameters side and trans, the routine can form one of the matrix products `Q*C`, `QT*C`, `C*Q`, or `C*QT` (overwriting the result on `C`).

11.89.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event ormtr(sycl::queue &queue,
  mkl::side side,
  mkl::uplo uplo,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
```

(continues on next page)

(continued from previous page)

```
  T *a,
  std::int64_t lda,
  T *tau,
  T *c,
  std::int64_t ldc,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`ormtr` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU

Input Parameters

In the descriptions below, `r` denotes the order of `Q`:

`r = m`	if `side = side::left`
`r = n`	if `side = side::right`

queue Device queue where calculations will be performed.

side Must be either `side::left` or `side::right`.

If `side = side::left`, `Q` or Q^T is applied to `C` from the left.

If `side = side::right`, `Q` or Q^T is applied to `C` from the right.

uplo Must be either `uplo::upper` or `uplo::lower`. Uses the same `uplo` as supplied to sytrd (USM Version).

trans Must be either `transpose::nontrans` or `transpose::trans`.

If `trans = transpose::nontrans`, the routine multiplies `C` by `Q`.

If `trans = transpose::trans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `C (m ≥ 0)`.

n The number of columns in the matrix `C (n ≥ 0)`.

a The pointer to memory returned by sytrd (USM Version).

lda The leading dimension of a `(max(1, r) ≤ lda)`.

tau The pointer to tau returned by sytrd (USM Version). The dimension of tau must be at least `max(1, r-1)`.

c The pointer to memory containing the matrix `C`. The second dimension of c must be at least `max(1, n)`.

ldc The leading dimension of `c (max(1, n) ≤ ldc)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the ormtr_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

c Overwritten by `Q*C`, `QT*C`, `C*Q`, or `C*QT` (as specified by side and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.90 ormtr_scratchpad_size

Computes size of scratchpad memory required for `ormtr` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.90.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `ormtr` (buffer or USM version) function must be able to hold.

11.90.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t ormtr_scratchpad_size(sycl::queue &queue,
  mkl::side side,
  mkl::uplo uplo,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t lda,
  std::int64_t ldc)
}
```

Input Parameters

In the descriptions below, `r` denotes the order of `Q`:

`r = m`	if `side = side::left`
`r = n`	if `side = side::right`

queue Device queue where calculations by the `ormtr` (buffer or USM version) function will be performed.

side Must be either `side::left` or `side::right`.

If `side = side::left`, `Q` or Q^T is applied to `C` from the left.

If `side = side::right`, `Q` or Q^T is applied to `C` from the right.

uplo Must be either `uplo::upper` or `uplo::lower`. Uses the same `uplo` as supplied to sytrd (USM Version).

trans Must be either `transpose::nontrans` or `transpose::trans`.

If `trans = transpose::nontrans`, the routine multiplies `C` by `Q`.

If `trans = transpose::trans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `C (m ≥ 0)`.

n The number of columns in the matrix `C (n ≥ 0)`.

lda The leading dimension of a `(max(1, r) ≤ lda)`.

ldc The leading dimension of `c (max(1, n) ≤ ldc)`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the `info()` method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `ormtr` (buffer or USM version) function must be able to hold.

11.91 potrf

Computes the Cholesky factorization of a symmetric (Hermitian) positive-definite matrix.This routine belongs to the `oneapi::mkl::lapack` namespace.

11.91.1 Description

The routine forms the Cholesky factorization of a symmetric positive-definite or, for complex data, Hermitian positive-definite matrix A:

$A = U^T*U$ for real data, $A = U^H*U$ for complex data	if uplo=`mkl::uplo::upper`
$A = L*L^T$ for real data, $A = L*L^H$ for complex data	if uplo=`mkl::uplo::lower`

where L is a lower triangular matrix and U is upper triangular.

11.91.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void potrf(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`potrf` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates whether the upper or lower triangular part of A is stored and how A is factored:

If uplo=`mkl::uplo::upper`, the array a stores the upper triangular part of the matrix A, and the strictly lower triangular part of the matrix is not referenced.

If uplo=`mkl::uplo::lower`, the array a stores the lower triangular part of the matrix A, and the strictly upper triangular part of the matrix is not referenced.

n Specifies the order of the matrix A (`0 ≤ n`).

a Buffer holding input matrix A. The array holding input matrix a contains either the upper or the lower triangular part of the matrix A (see uplo). The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the potrf_scratchpad_size function.

Output Parameters

a The buffer a is overwritten by the Cholesky factor U or L, as specified by uplo.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, and detail() returns 0, the leading minor of order `i` (and therefore the matrix A itself) is not positive-definite, and the factorization could not be completed. This may indicate an error in forming the matrix A. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Known Limitations

- GPU support is for only real precisions.
- GPU support for this function does not include error reporting through the `info` parameter.

11.92 potrf (USM Version)

Computes the Cholesky factorization of a symmetric (Hermitian) positive-definite matrix.This routine belongs to the `oneapi::mkl::lapack` namespace.

11.92.1 Description

The routine forms the Cholesky factorization of a symmetric positive-definite or, for complex data, Hermitian positive-definite matrix `A`:

$A = U^T*U$ for real data, $A = U^H*U$ for complex data	if uplo=`mkl::uplo::upper`
$A = L*L^T$ for real data, $A = L*L^H$ for complex data	if uplo=`mkl::uplo::lower`

where `L` is a lower triangular matrix and `U` is upper triangular.

11.92.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event potrf(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

The USM version of `potrf` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates whether the upper or lower triangular part of `A` is stored and how `A` is factored:

If uplo=`mkl::uplo::upper`, the array a stores the upper triangular part of the matrix `A`, and the strictly lower triangular part of the matrix is not referenced.

If uplo=`mkl::uplo::lower`, the array a stores the lower triangular part of the matrix `A`, and the strictly upper triangular part of the matrix is not referenced.

n Specifies the order of the matrix `A` (`0 ≤ n`).

a Pointer to input matrix `A`. The array holding input matrix a contains either the upper or the lower triangular part of the matrix `A` (see uplo). The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the potrf_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a The memory pointed to by pointer a is overwritten by the Cholesky factor `U` or `L`, as specified by uplo.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, and detail() returns 0, the leading minor of order `i` (and therefore the matrix `A` itself) is not positive-definite, and the factorization could not be completed. This may indicate an error in forming the matrix `A`. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.93 potrf_batch (Buffer Strided Version)

Computes the Cholesky factorizations of a batch of symmetric (or Hermitian, for complex data) positive-definite matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.93.1 Description

The routine forms the Cholesky factorizations of a symmetric positive-definite or, for complex data, Hermitian positive-definite matrices A_i, `i`ϵ{`1...batch_size`}:

- $A_i = U_i^T * U_i$ for real data, $A_i = U_i^H * U_i$ for complex data. if `uplo = mkl::uplo::upper`,
- $A_i = L_i^T * L_i$ for real data, $A_i = L_i^H * L_i$ for complex data if `uplo = mkl::uplo::lower`

Where L_i is a lower triangular matrix and U_i is an upper triangular matrix.

11.93.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void potrf_batch(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t batch_size,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates whether the upper or lower triangular part of A_i is stored and how A_i is factored:

If uplo = mkl::uplo::upper, the array a stores the upper triangular parts of the matrices A_i.

If uplo = mkl::uplo::lower, the array a stores the lower triangular parts of the matrices A_i.

n Specifies the order of the matrices A_i, (`0 ≤ n`).

a Array containing a batch of input matrices A_i, each of A_i being of size `lda*n` and holding either uppoer or lower triangular parts of the matrices A_i (see uplo).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices A_i inside the batch.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by potrf_batch_scratchpad_size (Strided Version).

Output Parameters

a The batch array a is overwritten by the Cholesky factor U_i or L_i, as specified by uplo .

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

11.94 potrf_batch (Group Version)

Computes the Cholesky factorizations of a batch of symmetric (or Hermitian, for complex data) positive-definite matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.94.1 Description

The routine forms the Cholesky factorizations of a symmetric positive-definite or, for complex data, Hermitian positive-definite matrices A_i, `i`ϵ{`1...batch_size`}:

- $A_i = U_i^T * U_i$ for real data, $A_i = U_i^H * U_i$ for complex data. if `uplog = mkl::uplo::upper`,
- $A_i = L_i^T * L_i$ for real data, $A_i = L_i^H * L_i$ for complex data if `uplog = mkl::uplo::lower`

Where L_i is a lower triangular matrix and U_i is an upper triangular matrix, `g` is an index of group of parameters corresponding to A_i, and the total number of problems to solve, `batch_size`, is a sum of sizes for all of the groups of parameters as provided by the `group_sizes` array.

11.94.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event potrf_batch(sycl::queue &queue,
  mkl::uplo *uplo,
  std::int64_t *n,
  T **a,
  std::int64_t *lda,
  std::int64_t group_count,
  std::int64_t *group_sizes,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Array of `group_count`$uplo_g$ parameters.

Each of $uplo_g$ indicates whether the upper or lower triangular parts of the input matrices are provided:

If $uplo_g$=`mkl::uplo::upper`, input matrices from array a belonging to group `g` store the upper triangular parts.

If $uplo_g$=`mkl::uplo::lower`, input matrices from array a belonging to group `g` store the lower triangular parts.

n Array of `group_count` parameters n_g.

Each n_g specifies the order of the input matrices from array a belonging to group `g`.

a Array of `batch_size` pointers to input matrices A_i, each being of size `lda`$_g$*n_g (`g` is an index of the group to which A_i belongs) and holding either upper or lower triangular part as specified by $uplo_g$.

lda Array of `group_count` parameters lda_g.

Each lda_g specifies the leading dimension of matrices from a belonging to group `g`.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of group_count integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by potrf_batch_scratchpad_size (Group Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a The matrices pointed to by array a are overwritten by the Cholesky factors U_i or L_i, as specified by $uplo_g$ from the corresponding group of parameters.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.95 potrf_batch (USM Strided Version)

Computes the Cholesky factorizations of a batch of symmetric (or Hermitian, for complex data) positive-definite matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.95.1 Description

The routine forms the Cholesky factorizations of a symmetric positive-definite or, for complex data, Hermitian positive-definite matrices A_i, `i`ϵ{`1...batch_size`}:

- $A_i = U_i^T * U_i$ for real data, $A_i = U_i^H * U_i$ for complex data. if `uplo = mkl::uplo::upper`,
- $A_i = L_i^T * L_i$ for real data, $A_i = L_i^H * L_i$ for complex data if `uplo = mkl::uplo::lower`

Where L_i is a lower triangular matrix and U_i is an upper triangular matrix.

11.95.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event potrf_batch(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t batch_size,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates whether the upper or lower triangular part of A_i is stored and how A_i is factored:

If uplo = mkl::uplo::upper, the array a stores the upper triangular parts of the matrices A_i.

If uplo = mkl::uplo::lower, the array a stores the lower triangular parts of the matrices A_i.

n Specifies the order of the matrices A_i, (`0 ≤ n`).

a Array containing a batch of input matrices A_i, each of A_i being of size `lda*n` and holding either uppoer or lower triangular parts of the matrices A_i (see uplo).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices A_i inside the batch.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by potrf_batch_scratchpad_size (Strided Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a The batch array a is overwritten by the Cholesky factor U_i or L_i, as specified by uplo .

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.96 potrf_batch_scratchpad_size (Group Version)

Computes the size of scratchpad memory required for `potrf_batch` (Group Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.96.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the potrf_batch (Group Version) function must be able to hold.

11.96.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t potrf_batch_scratchpad_size(sycl::queue &queue,
  mkl::uplo *uplo,
  std::int64_t *n,
  std::int64_t *lda,
  std::int64_t group_count,
  std::int64_t *group_sizes)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Array of `group_count` parameters $\texttt{uplo}_g$.

Each of $\texttt{uplo}_g$ indicates whether the upper or lower triangular parts of the input matrices are provided.

If $\texttt{uplo}_g$=`mkl::uplo::upper`, input matrices from array a belonging to group `g` store the upper triangular parts.

If $\texttt{uplo}_g$=`mkl::uplo::lower`, input matrices from array a belonging to group `g` store the lower triangular parts.

n Array of `group_count`n_g parameters.

Each n_g specifies the order of the input matrices belonging to group `g`.

lda Array of `group_count` parameters $\texttt{lda}_g$.

Each $\texttt{lda}_g$ specifies the leading dimension of the matrices belonging to group `g`.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of `group_count` integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the potrf_batch (Group Version) function must be able to hold.

11.97 potrf_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `potrf_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.97.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `potrf_batch` (Strided Version) function must be able to hold.

11.97.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t potrf_batch_scratchpad_size(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates whether the upper or lower triangular part of A_i is stored and how A_i is factored:

If uplo = mkl::uplo::upper, the array a stores the upper triangular parts of the matrices A_i.

If uplo = mkl::uplo::lower, the array a stores the lower triangular parts of the matrices A_i.

n Specifies the order of the matrices A_i, ($0 \leq n$).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices A_i inside the batch.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `potrf_batch` (Strided Version) function must be able to hold.

11.98 potrf_scratchpad_size

Computes size of scratchpad memory required for `potrf` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.98.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `potrf` (buffer or USM version) function must be able to hold.

11.98.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t potrf_scratchpad_size(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the potrf (buffer or USM version) function will be performed.

uplo Indicates whether the upper or lower triangular part of `A` is stored and how `A` is factored:

If uplo=`mkl::uplo::upper`, the array a stores the upper triangular part of the matrix `A`, and the strictly lower triangular part of the matrix is not referenced.

If uplo=`mkl::uplo::lower`, the array a stores the lower triangular part of the matrix `A`, and the strictly upper triangular part of the matrix is not referenced.

n Specifies the order of the matrix `A` (`0 ≤ n`).

lda The leading dimension of `A`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `potrf` (buffer or USM version) function must be able to hold.

11.99 potri

Computes the inverse of a symmetric (Hermitian) positive-definite matrix using the Cholesky factorization. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.99.1 Description

The routine computes the inverse `inv(A)` of a symmetric positive definite or, for complex flavors, Hermitian positive-definite matrix `A`. Before calling this routine, call **potrf** to factorize `A`.

11.99.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void potri(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`potri` supports the following precisions and devices:

T	Devices Supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates how the input matrix A has been factored:

If `uplo = mkl::uplo::upper`, the upper triangle of `A` is stored.

If `uplo = mkl::uplo::lower`, the lower triangle of `A` is stored.

n Specifies the order of the matrix `A` (`0 ≤ n`).

a Contains the factorization of the matrix `A`, as returned by **potrf (USM Version)**. The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the **potri_scratchpad_size** function.

Output Parameters

a Overwritten by the upper or lower triangle of the inverse of `A`. Specified by `uplo`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, the `i`-th diagonal element of the Cholesky factor (and therefore the factor itself) is zero, and the inversion could not be completed. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.100 potri (USM Version)

Computes the inverse of a symmetric (Hermitian) positive-definite matrix using the Cholesky factorization. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.100.1 Description

The routine computes the inverse `inv(A)` of a symmetric positive definite or, for complex flavors, Hermitian positive-definite matrix `A`. Before calling this routine, call potrf (USM Version) to factorize `A`.

11.100.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event potri(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  T* a,
  std::int64_t lda,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`potri` (USM version) supports the following precisions and devices:

T	Devices Supported
`float`	CPU, GPU
`double`	CPU, GPU
`std::complex<float>`	CPU, GPU
`std::complex<double>`	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates how the input matrix A has been factored:

If `uplo = mkl::uplo::upper`, the upper triangle of `A` is stored.

If `uplo = mkl::uplo::lower`, the lower triangle of `A` is stored.

n Specifies the order of the matrix `A` (`0 ≤ n`).

a Contains the factorization of the matrix `A`, as returned by potrf (USM Version). The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the potri_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by the upper or lower triangle of the inverse of `A`. Specified by uplo.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, the `i`-th diagonal element of the Cholesky factor (and therefore the factor itself) is zero, and the inversion could not be completed. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.101 potri_scratchpad_size

Computes size of scratchpad memory required for `potri` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.101.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `potri` (buffer or USM version) function must be able to hold.

11.101.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t potri_scratchpad_size(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the potri (buffer or USM version) function will be performed.

uplo Indicates how the input matrix A has been factored:

If `uplo = mkl::uplo::upper`, the upper triangle of `A` is stored.

If `uplo = mkl::uplo::lower`, the lower triangle of `A` is stored.

n Specifies the order of the matrix `A` (`0 ≤ n`).

lda The leading dimension of a.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `potri` (buffer or USM version) function must be able to hold.

11.102 potrs

Solves a system of linear equations with a Cholesky-factored symmetric (Hermitian) positive-definite coefficient matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.102.1 Description

The routine solves for `X` the system of linear equations `A*X = B` with a symmetric positive-definite or, for complex data, Hermitian positive-definite matrix `A`, given the Cholesky factorization of `A`:

`A = UT*U` for real data, `A = UH*U` for complex data	if `uplo=mkl::uplo::upper`
`A = L*LT` for real data, `A = L*LH` for complex data	if `uplo=mkl::uplo::lower`

where `L` is a lower triangular matrix and `U` is upper triangular. The system is solved with multiple right-hand sides stored in the columns of the matrix `B`.

Before calling this routine, you must call **potrf** to compute the Cholesky factorization of `A`.

11.102.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void potrs(sycl::queue &queue, mkl::uplo uplo,
  std::int64_t n,
  std::int64_t nrhs,
  sycl::buffer<T> &a,
  std::int64_t lda,
```

(continues on next page)

(continued from previous page)

```
  sycl::buffer<T> &b,
  std::int64_t ldb,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`potrs` supports the following precisions and devices:

T	Devices Supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates how the input matrix has been factored:

If uplo=`mkl::uplo::upper`, the upper triangle `U` of `A` is stored, where $A = U^T*U$ for real data, $A = U^H*U$ for complex data.

If uplo=`mkl::uplo::lower`, the upper triangle `L` of `A` is stored, where $A = L*L^T$ for real data, $A = L*L^H$ for complex data.

n The order of matrix `A` ($0 \le n$).

nrhs The number of right-hand sides ($0 \le nrhs$).

a Buffer holding factorization of the matrix `A`, as returned by **potrf**. The second dimension of a must be at least max(1, n).

lda The leading dimension of a.

b Buffer holding the data of matrix `B` whose columns are the right-hand sides for the systems of equations. The second dimension of b must be at least max(1,nrhs).

ldb The leading dimension of b.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the **potrs_scratchpad_size** function.

Output Parameters

b Buffer b is overwritten by the solution matrix X.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, the `i`-th diagonal element of the Cholesky factor is zero, and the solve could not be completed. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Known Limitations

- GPU support is for only real precisions.
- GPU support for this function does not include error reporting through the `info` parameter.

11.103 potrs (USM Version)

Solves a system of linear equations with a Cholesky-factored symmetric (Hermitian) positive-definite coefficient matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.103.1 Description

The routine solves for X the system of linear equations A*X = B with a symmetric positive-definite or, for complex data, Hermitian positive-definite matrix A, given the Cholesky factorization of A:

$A = U^T*U$ for real data, $A = U^H*U$ for complex data	if uplo=`mkl::uplo::upper`
$A = L*L^T$ for real data, $A = L*L^H$ for complex data	if uplo=`mkl::uplo::lower`

where L is a lower triangular matrix and U is upper triangular. The system is solved with multiple right-hand sides stored in the columns of the matrix B.

Before calling this routine, you must call potrf (USM Version) to compute the Cholesky factorization of A.

11.103.2 API

Syntax

```
namespace oneapi::mkl::lapack {
   sycl::event potrs(
      sycl::queue &queue,
      mkl::uplo uplo,
      std::int64_t n,
      std::int64_t nrhs,
      T *a,
      std::int64_t lda,
      T *b,
      std::int64_t ldb,
      T *scratchpad,
      std::int64_t scratchpad_size,
      const std::vector<sycl::event> &events = {})
}
```

`potrs` (USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates how the input matrix has been factored:

If uplo=`mkl::uplo::upper`, the upper triangle `U` of `A` is stored, where $A = U^T*U$ for real data, $A = U^H*U$ for complex data.

If uplo=`mkl::uplo::lower`, the upper triangle `L` of `A` is stored, where $A = L*L^T$ for real data, $A = L*L^H$ for complex data.

n The order of matrix `A` ($0 \le n$).

nrhs The number of right-hand sides ($0 \le nrhs$).

a Pointer to factorization of the matrix `A`, as returned by potrf (USM Version). The second dimension of a must be at least max(1, n).

lda The leading dimension of a.

b Pointer to the data of matrix `B` whose columns are the right-hand sides for the systems of equations. The second dimension of b must be at least max(1,nrhs).

ldb The leading dimension of b.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the potrs_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

b The memory pointed to by pointer b is overwritten by the solution matrix X.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the `info()` method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, the `i`-th diagonal element of the Cholesky factor is zero, and the solve could not be completed. If `info` is equal to the value passed as scratchpad size, and `detail()` returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the `detail()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.104 potrs_batch (Buffer Strided Version)

Solves a system of linear equations with a Cholesky-factored symmetric (Hermitian) positive-definite coefficient matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.104.1 Description

The routine solves for X_i the system of linear equations $A_i*X_i = B_i$ with a symmetric positive-definite or, for complex data, Hermitian positive-definite matrices A_i, given the Cholesky factorization of A_i, `i ϵ{1...batch_size}`:

- $A_i = U_i^T*U_i$ for real data, $A_i = U_i^H*U_i$ for complex data if uplo=`mkl::uplo::upper`
- $A_i = L_i*L_i^T$ for real data, $A_i = L_i*L_i^H$ for complex data if uplo=`mkl::uplo::lower`

where L_i is a lower triangular matrix and U_i is upper triangular. The system is solved with multiple right-hand sides stored in the columns of the matrix B_i.

Before calling this routine, matrices A_i should be factorized by a call to potrf_batch (Buffer Strided Version).

11.104.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void potrs_batch(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t nrhs,
  sycl::buffer<T> &a,
  std::int64_t lda,
  std::int64_t stride_a,
  sycl::buffer<T> &b,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates how the input matrix has been factored:

If uplo=`mkl::uplo::upper`, the upper triangle U_i of A_i is stored, where $A_i = U_i^T * U_i$ for real data, $A_i = U_i^H * U_i$ for complex data.

If uplo=`mkl::uplo::lower`, the upper triangle L_i of A_i is stored, where $A_i = L_i * L_i^T$ for real data, $A_i = L_i * L_i^H$ for complex data.

n The order of the matrices A_i (`0 ≤ n`).

nrhs The number of right hand sides (`0 ≤ nrhs`).

a Array containing the batch of factorizations of the matrices A_i, as returned by potrf_batch (Buffer Strided Version).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices inside the batch array a.

b The array containing the batch of matrices B_i whose columns are the right-hand sides for the systems of equations.

ldb The leading dimensions of B_i.

stride_b The stride between the beginnings of matrices B_i inside the batch array b.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by stride version of potrs_batch_scratchpad_size (Strided Version) function.

Output Parameters

b The batch array b is overwritten by the solution matrix X_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

11.105 potrs_batch (Group Version)

Solves a batch of systems of linear equations with a Cholesky-factored symmetric (Hermitian) positive-definite coefficient matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.105.1 Description

The routine solves for X_i the systems of linear equations $A_i*X_i = B_i$ with a symmetric positive-definite or, for complex data, Hermitian positive-definite matrices A_i, given the Cholesky factorization of A_i, `i ϵ{1...batch_size}` :

- $A_i = U_i^T*U_i$ for real data, $A_i = U_i^H*U_i$ for complex data if $uplo_g$=`mkl::uplo::upper`
- $A_i = L_i*L_i^T$ for real data, $A_i = L_i*L_i^H$ for complex data if $uplo_g$=`mkl::uplo::lower`

where L_i is a lower triangular matrix and U_i is upper triangular, g is an index of group of parameters corresponding to A_i, and the total number of problems to solve, `batch_size`, is a sum of sizes for all of the groups of parameters as provided by the `group_sizes` array.

The systems are solved with multiple right-hand sides stored in the columns of the matrices B_i.

Before calling this routine, matrices A_i should be factorized by a call to potrf_batch (Group Version).

11.105.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event potrs_batch(sycl::queue &queue,
  mkl::uplo *uplo, std::int64_t *n,
  std::int64_t *nrhs,
  T **a,
  std::int64_t *lda,
  T **b,
  std::int64_t *ldb,
  std::int64_t group_count,
  std::int64_t *group_sizes,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Array of `group_count`uplo$_g$ parameters.

Each of uplo$_g$ indicates whether the upper or lower triangular parts of the input matrices are provided:

If uplo$_g$=`mkl::uplo::upper`, input matrices from array a belonging to group g store the upper triangular parts.

If uplo$_g$=`mkl::uplo::lower`, input matrices from array a belonging to group g store the lower triangular parts.

n Array of `group_count` parameters n_g.

Each n_g specifies the order of the input matrices from array a belonging to group g.

nrhs Array of `group_count` parameters nrhs$_g$ parameters.

Each nrhs$_g$ specifies the number of right-hand sides supplied for group g in corresponding part of array b.

a Array of `batch_size` pointers to Cholesky factored matrices A_i as returned by **potrf_batch (Group Version)**.

lda Array of `group_count` parameters lda$_g$.

Each lda$_g$ specifies the leading dimension of matrices from a belonging to group g.

b Array of `batch_size` pointers to right-hand side matrices B_i, each of size ldb$_g$*nrhs$_g$, where g is an index of group corresponding to B_i.

ldb Array of `group_count` parameters ldb$_g$.

Each ldb$_g$ specifies the leading dimension of matrices from b belonging to group g.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of group_count integers. Array element with index g specifies the number of problems to solve for each of the groups of parameters g. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by **potrs_batch_scratchpad_size (Group Version)**.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

b The matrices pointed to by array b are overwritten by the solution matrices X_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.106 potrs_batch (USM Strided Version)

Solves a system of linear equations with a Cholesky-factored symmetric (Hermitian) positive-definite coefficient matrices. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.106.1 Description

The routine solves for X_i the system of linear equations $A_i*X_i = B_i$ with a symmetric positive-definite or, for complex data, Hermitian positive-definite matrices A_i, given the Cholesky factorization of A_i, `i ϵ{1...batch_size}`:

- $A_i = U_i^T*U_i$ for real data, $A_i = U_i^H*U_i$ for complex data if uplo=`mkl::uplo::upper`
- $A_i = L_i*L_i^T$ for real data, $A_i = L_i*L_i^H$ for complex data if uplo=`mkl::uplo::lower`

where L_i is a lower triangular matrix and U_i is upper triangular. The system is solved with multiple right-hand sides stored in the columns of the matrix B_i.

Before calling this routine, matrices A_i should be factorized by a call to potrf_batch (USM Strided Version).

11.106.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event potrs_batch(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t nrhs,
  T *a,
  std::int64_t lda,
  std::int64_t stride_a,
  T *b,
  std::int64_t ldb,
  std::int64_t stride_b,
  std::int64_t batch_size,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates how the input matrix has been factored:

If uplo=`mkl::uplo::upper`, the upper triangle U_i of A_i is stored, where $A_i = U_i^T * U_i$ for real data, $A_i = U_i^H * U_i$ for complex data.

If uplo=`mkl::uplo::lower`, the upper triangle L_i of A_i is stored, where $A_i = L_i * L_i^T$ for real data, $A_i = L_i * L_i^H$ for complex data.

n The order of the matrices A_i (`0 ≤ n`).

nrhs The number of right hand sides (`0 ≤ nrhs`).

a Array containing the batch of factorizations of the matrices A_i, as returned by potrf_batch (USM Strided Version).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices inside the batch array a.

b The array containing the batch of matrices B_i whose columns are the right-hand sides for the systems of equations.

ldb The leading dimensions of B_i.

stride_b The stride between the beginnings of matrices B_i inside the batch array b.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by stride version of potrs_batch_scratchpad_size (Strided Version) function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

b The memory pointed to by pointer batch array b is overwritten by the solution matrix X_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object. If `info` is zero, then the diagonal element of some of U_i is zero, and the solve could not be completed. The indexes of such matrices in the batch can be obtained with the `ids()` method of the exception object. You can obtain the indexes of the first zero diagonal elements in these U_i matrices using the `infos()` method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.107 potrs_batch_scratchpad_size (Group Version)

Computes size of scratchpad memory required for `potrs_batch` (Group Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.107.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the potrs_batch (Group Version) function must be able to hold.

11.107.2 API

Syntax

```
namespace oneapi::mkl::lapack {
   std::int64_t  potrs_batch_scratchpad_size(
      sycl::queue &queue,
      mkl::uplo *uplo,
      std::int64_t *n,
      std::int64_t *nrhs,
      std::int64_t *lda,
      std::int64_t *ldb,
      std::int64_t group_count,
      std::int64_t  *group_sizes)
}
```

This function supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Array of `group_count` parameters $uplo_g$.

Each of $uplo_g$ indicates whether the upper or lower triangular parts of the input matrices are provided.

If $uplo_g$=`mkl::uplo::upper`, input matrices from array a belonging to group g store the upper triangular parts.

If $uplo_g$=`mkl::uplo::lower`, input matrices from array a belonging to group g store the lower triangular parts.

n Array of `group_count` parameters n_g.

Each n_g specifies the order of the input matrices belonging to group g.

nrhs Array of $\texttt{nrhs}_g$ parameters.

Each $\texttt{nrhs}_g$ specifies the number of right-hand sides supplied for to group `g`.

lda Array of `group_count` parameters $\texttt{lda}_g$.

Each $\texttt{lda}_g$ specifies the leading dimension of the matrices belonging to group `g`.

ldb Array of `group_count` parameters $\texttt{ldb}_g$.

Each $\texttt{ldb}_g$ specifies the leading dimension of the matrices belonging to group `g`.

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of `group_count` integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the `info()` method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the potrs_batch (Group Version) function must be able to hold.

11.108 potrs_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `potrs_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.108.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `potrs_batch` (Strided Version) function must be able to hold.

11.108.2 API

Syntax

```
namespace oneapi::mkl::lapack {
   std::int64_t potrs_batch_scratchpad_size(
      sycl::queue &queue,
      mkl::uplo uplo,
      std::int64_t n,
      std::int64_t nrhs,
      std::int64_t lda,
      std::int64_t stride_a,
      std::int64_t ldb,
      std::int64_t stride_b,
      std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates how the input matrix has been factored:

If uplo=`mkl::uplo::upper`, the upper triangle U_i of A_i is stored, where $A_i = U_i^T * U_i$ for real data, $A_i = U_i^H * U_i$ for complex data.

If uplo=`mkl::uplo::lower`, the upper triangle L_i of A_i is stored, where $A_i = L_i * L_i^T$ for real data, $A_i = L_i * L_i^H$ for complex data.

n The order of the matrices A_i (`0 ≤ n`).

nrhs The number of right hand sides (`0 ≤ nrhs`).

lda The leading dimension of A_i.

stride_a The stride between the beginnings of matrices inside the batch array a.

ldb The leading dimensions of B_i.

stride_b The stride between the beginnings of matrices B_i inside the batch array b.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the `info()` method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `potrs_batch` (Strided Version) function must be able to hold.

11.109 potrs_scratchpad_size

Computes size of scratchpad memory required for `potrs` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.109.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `potrs` (buffer or USM version) function must be able to hold.

11.109.2 API

Syntax

```
namespace oneapi::mkl::lapack {
   template<typename T>
   std::int64_t potrs_scratchpad_size(
   sycl::queue &queue,
   mkl::uplo uplo,
   std::int64_t n,
   std::int64_t nrhs,
   std::int64_t lda,
   std::int64_t ldb)
}
```

Input Parameters

queue Device queue where calculations by the `potrs` (buffer or USM version) function will be performed.

uplo Indicates how the input matrix has been factored:

If uplo=`mkl::uplo::upper`, the upper triangle `U` of `A` is stored, where `A` = U^T*`U` for real data, `A` = U^H*`U` for complex data.

If uplo=`mkl::uplo::lower`, the upper triangle `L` of `A` is stored, where `A` = `L`*L^T for real data, `A` = `L`*L^H for complex data.

n The order of matrix `A` (0 ≤ `n`).

nrhs The number of right-hand sides (0 ≤ `nrhs`).

lda The leading dimension of A.

ldb The leading dimension of B.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the `info()` method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `potrs` (buffer or USM version) function must be able to hold.

11.110 syevd

Computes all eigenvalues and, optionally, all eigenvectors of a real symmetric matrix using divide and conquer algorithm. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.110.1 Description

The routine computes all the eigenvalues, and optionally all the eigenvectors, of a real symmetric matrix `A`. In other words, it can compute the spectral factorization of `A` as: `A = Z*λ*ZT`.

Here Λ is a diagonal matrix whose diagonal elements are the eigenvalues λ_i, and `Z` is the orthogonal matrix whose columns are the eigenvectors z_i. Thus,

`A*zi = λi*zi` for `i = 1, 2, ..., n`.

If the eigenvectors are requested, then this routine uses a divide and conquer algorithm to compute eigenvalues and eigenvectors. However, if only eigenvalues are required, then it uses the Pal-Walker-Kahan variant of the `QL` or `QR` algorithm.

11.110.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void syevd(sycl::queue &queue,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
```

(continues on next page)

(continued from previous page)

```
  sycl::buffer<T> &w,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

syevd supports the following precision and devices.

T	Devices Supported
float	CPU and GPU
double	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

jobz Must be job::novec or job::vec.

If jobz = job::novec, then only eigenvalues are computed.

If jobz = job::vec, then eigenvalues and eigenvectors are computed.

uplo Must be uplo::upper or uplo::lower.

If uplo = job::upper, a stores the upper triangular part of A.

If uplo = job::lower, a stores the lower triangular part of A.

n The order of the matrix A (0 ≤ n).

a The pointer to the array containing A, size (lda,*). The second dimension of a must be at least max(1, n).

lda The leading dimension of a. Must be at least max(1,n).

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the syevd_scratchpad_size function.

Output Parameters

a If jobz = job::vec, then on exit this buffer is overwritten by the orthogonal matrix Z which contains the eigenvectors of A.

w Buffer holding array of size at least n. If info = 0, contains the eigenvalues of the matrix A in ascending order.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, and `jobz = job::novec`, then the algorithm failed to converge; `i` indicates the number of off-diagonal elements of an intermediate tridiagonal form which did not converge to zero. If `info = i`, and `jobz = job::vec`, then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns `info/(n+1)` through `mod(info,n+1)`. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.111 syevd (USM Version)

Computes all eigenvalues and, optionally, all eigenvectors of a real symmetric matrix using divide and conquer algorithm. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.111.1 Description

The routine computes all the eigenvalues, and optionally all the eigenvectors, of a real symmetric matrix `A`. In other words, it can compute the spectral factorization of `A` as: `A = Z*λ*ZT`.

Here Λ is a diagonal matrix whose diagonal elements are the eigenvalues Λ_i, and `Z` is the orthogonal matrix whose columns are the eigenvectors z_i. Thus,

`A*zi = λi*zi` for `i = 1, 2, ..., n`.

If the eigenvectors are requested, then this routine uses a divide and conquer algorithm to compute eigenvalues and eigenvectors. However, if only eigenvalues are required, then it uses the Pal-Walker-Kahan variant of the `QL` or `QR` algorithm.

11.111.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event syevd(sycl::queue &queue,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *w,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`syevd` (USM version) supports the following precision and devices.

T	Devices Supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

jobz Must be `job::novec` or `job::vec`.

If `jobz = job::novec`, then only eigenvalues are computed.

If `jobz = job::vec`, then eigenvalues and eigenvectors are computed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `upper_lower = job::upper`, a stores the upper triangular part of `A`.

If `upper_lower = job::lower`, a stores the lower triangular part of `A`.

n The order of the matrix `A` (`0 ≤ n`).

a The pointer to the array containing `A`, size (`lda,*`). The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a. Must be at least `max(1,n)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the syevd_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a If `jobz = job::vec`, then on exit this is overwritten by the orthogonal matrix `Z` which contains the eigenvectors of `A`.

w Pointer to array of size at least n. Contains the eigenvalues of the matrix `A` in ascending order.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, and `jobz = job::novec`, then the algorithm failed to converge; `i` indicates the number of off-diagonal elements of an intermediate tridiagonal form which did not converge to zero. If `info = i`, and `jobz = job::vec`, then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns `info/(n+1)` through `mod(info,n+1)`. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.112 syevd_scratchpad_size

Computes size of scratchpad memory required for `syevd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.112.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `syevd` (buffer or USM version) function must be able to hold.

11.112.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t syevd_scratchpad_size(sycl::queue &queue,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the syevd (buffer or USM version) function will be performed.

jobz Must be `job::novec` or `job::vec`.

If `jobz = job::novec`, then only eigenvalues are computed.

If `jobz = job::vec`, then eigenvalues and eigenvectors are computed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = job::upper`, a stores the upper triangular part of `A`.

If `uplo = job::lower`, a stores the lower triangular part of `A`.

n The order of the matrix `A` (`0 ≤ n`).

lda The leading dimension of a. Must be at least `max(1,n)`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Exceptions

The number of elements of type `T` the scratchpad memory to be passed to the `syevd` (buffer or USM version) function must be able to hold.

11.113 sygvd

Computes all eigenvalues and, optionally, eigenvectors of a real generalized symmetric definite eigenproblem using a divide and conquer method. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.113.1 Description

The routine computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form

`A*x = λ*B*x`, `A*B*x = λ*x`, or `B*A*x = λ*x`.

Here `A` and `B` are assumed to be symmetric and `B` is also positive definite.

It uses a divide and conquer algorithm.

11.113.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void sygvd(sycl::queue &queue,
  std::int64_t itype,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &b,
  std::int64_t ldb,
  sycl::buffer<T> &w,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`sygvd` supports the following precisions and devices:

T	Devices supported
`float`	CPU, GPU
`double`	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

itype Must be 1 or 2 or 3. Specifies the problem type to be solved:

if itype= 1, the problem type is `A*x = lambda*B*x;`

if itype= 2, the problem type is `A*B*x = lambda*x;`

if itype= 3, the problem type is `B*A*x = lambda*x`.

jobz Must be `job::novec` or `job::vec`.

If `jobz = job::novec`, then only eigenvalues are computed.

If `jobz = job::vec`, then eigenvalues and eigenvectors are computed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a and b store the upper triangular part of `A` and `B`.

If `uplo = uplo::lower`, a and b stores the lower triangular part of `A` and `B`.

n The order of the matrices `A` and `B` (`0 ≤ n`).

a Buffer holding the array of size `a(lda,*)` containing the upper or lower triangle of the symmetric matrix `A`, as specified by uplo.

The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a; at least `max(1,n)`.

b Buffer holding the array of size `b(ldb,*)` containing the upper or lower triangle of the symmetric matrix `B`, as specified by uplo.

The second dimension of b must be at least `max(1, n)`.

ldb The leading dimension of b; at least `max(1,n)`.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the sygvd_scratchpad_size function.

Output Parameters

a On exit, if `jobz = job::vec`, then if `info = 0`, a contains the matrix `Z` of eigenvectors. The eigenvectors are normalized as follows:

if itype= 1or 2 , Z^T`*B*Z = I;`

if itype= 3 , Z^T`*inv(B)*Z = I;`

If `jobz = job::novec`, then on exit the upper triangle (if `uplo = uplo::upper`) or the lower triangle (if `uplo = uplo::lower`) of `A`, including the diagonal, is destroyed.

b On exit, if `info ≤ n`, the part of b containing the matrix is overwritten by the triangular factor `U` or `L` from the Cholesky factorization `B` = U^T`*U` or `B = L*`L^T.

w Buffer, size at least `n`. If `info = 0`, contains the eigenvalues of the matrix `A` in ascending order. See also `info`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. For `info ≤ n`: If `info = i`, and `jobz = job::novec`, then the algorithm failed to converge; `i` indicates the number of off-diagonal elements of an intermediate tridiagonal form which did not converge to zero. If `info = i`, and `jobz = job:vec`, then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns `info/(n+1)` through `mod(info,n+1)`. For `info > n`: If `info = n + i`, for `1 ≤ i ≤ n`, then the leading minor of order `i` of `B` is not positive-definite. The factorization of `B` could not be completed and no eigenvalues or eigenvectors were computed. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.114 sygvd (USM Version)

Computes all eigenvalues and, optionally, eigenvectors of a real generalized symmetric definite eigenproblem using a divide and conquer method. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.114.1 Description

The routine computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form

`A*x = λ*B*x`, `A*B*x = λ*x`, or `B*A*x = λ*x`.

Here `A` and `B` are assumed to be symmetric and `B` is also positive definite.

It uses a divide and conquer algorithm.

11.114.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event sygvd(sycl::queue &queue,
  std::int64_t itype,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *b,
  std::int64_t ldb,
  T *w,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

hegvd (USM version) supports the following precision and devices.

T	Devices Supported
`float`	CPU, GPU
`double`	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

itype Must be 1 or 2 or 3. Specifies the problem type to be solved:

if itype= 1, the problem type is `A*x = lambda*B*x;`

if itype= 2, the problem type is `A*B*x = lambda*x;`

if itype= 3, the problem type is `B*A*x = lambda*x`.

jobz Must be `job::novec` or `job::vec`.

If `jobz = job::novec`, then only eigenvalues are computed.

If `jobz = job::vec`, then eigenvalues and eigenvectors are computed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a and b store the upper triangular part of A and B.

If `uplo = uplo::lower`, a and b stores the lower triangular part of A and B.

n The order of the matrices A and B (`0 ≤ n`).

a Pointer to the array of size a `(lda,*)` containing the upper or lower triangle of the symmetric matrix A, as specified by uplo.

The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a; at least `max(1,n)`.

b Pointer to the array of size b `(ldb,*)` containing the upper or lower triangle of the symmetric matrix B, as specified by uplo.

The second dimension of b must be at least `max(1, n)`.

ldb The leading dimension of b; at least `max(1,n)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the sygvd_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a On exit, if `jobz = job::vec`, then if `info = 0`, a contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows:

if itype= 1 or 2, Z^T`*B*Z = I`;

if itype= 3, Z^T`*inv(B)*Z = I`;

If `jobz = job::novec`, then on exit the upper triangle (if `uplo = uplo::upper`) or the lower triangle (if `uplo = uplo::lower`) of A, including the diagonal, is destroyed.

b On exit, if `info` ≤ `n`, the part of b containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization `B = U`^{T}`*U` or `B = L*L`^{T}.

w Pointer to the array of size at least n. If `info = 0`, contains the eigenvalues of the matrix A in ascending order. See also info.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. For `info ≤ n`: If `info = i`, and `jobz = job::novec`, then the algorithm failed to converge; `i` indicates the number of off-diagonal elements of an intermediate tridiagonal form which did not converge to zero. If `info = i`, and `jobz = job:vec`, then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns `info/(n+1)` through `mod(info,n+1)`. For `info > n`: If `info = n + i`, for `1 ≤ i ≤ n`, then the leading minor of order `i` of B is not positive-definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.115 sygvd_scratchpad_size

Computes size of scratchpad memory required for `sygvd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.115.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `sygvd` (buffer or USM version) function must be able to hold.

11.115.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t sygvd_scratchpad_size(sycl::queue &queue,
  std::int64_t itype,
  mkl::job jobz,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda,
  std::int64_t ldb)
}
```

Input Parameters

queue Device queue where calculations by the sygvd (buffer or USM version) function will be performed.

itype Must be 1 or 2 or 3. Specifies the problem type to be solved:

if itype= 1, the problem type is `A*x = lambda*B*x;`

if itype= 2, the problem type is `A*B*x = lambda*x;`

if itype= 3, the problem type is `B*A*x = lambda*x`.

jobz Must be `job::novec` or `job::vec`.

If `jobz = job::novec`, then only eigenvalues are computed.

If `jobz = job::vec`, then eigenvalues and eigenvectors are computed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a and b store the upper triangular part of `A` and `B`.

If `uplo = uplo::lower`, a and b stores the lower triangular part of `A` and `B`.

n The order of the matrices `A` and `B` (`0 ≤ n`).

lda The leading dimension of a; at least `max(1,n)`.

ldb The leading dimension of b; at least `max(1,n)`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `sygvd` (buffer or USM version) function must be able to hold.

11.116 sytrd

Reduces a real symmetric matrix to tridiagonal form. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.116.1 Description

The routine reduces a real symmetric matrix `A` to symmetric tridiagonal form `T` by an orthogonal similarity transformation: `A = Q*T*QT`. The orthogonal matrix `Q` is not formed explicitly but is represented as a product of `n-1` elementary reflectors. Routines are provided for working with `Q` in this representation .

11.116.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void sytrd(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &d,
  sycl::buffer<T> &e,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`sytrd` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a stores the upper triangular part of `A`.

If `uplo = uplo::lower`, a stores the lower triangular part of `A`.

n The order of the matrices `A(0 ≤ n)`.

a Buffer holding matrix `A`, size `(lda,*)`. Contains the upper or lower triangle as specified by uplo.

lda The leading dimension of a; at least `max(1,n)`.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the sytrd_scratchpad_size function.

Output Parameters

a On exit,

if `uplo = uplo::upper`, the diagonal and first superdiagonal of `A` are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the buffer tau, represent the orthogonal matrix `Q` as a product of elementary reflectors;

if `uplo = uplo::lower`, the diagonal and first subdiagonal of `A` are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the buffer tau, represent the orthogonal matrix `Q` as a product of elementary reflectors.

d Buffer holding the diagonal elements of the matrix T. The dimension of d must be at least `max(1, n)`.

e Buffer holding the off diagonal elements of the matrix T. The dimension of e must be at least `max(1, n-1)`.

tau Buffer holding array of size at least `max(1, n)`. Stores `(n-1)` scalars that define elementary reflectors in decomposition of the unitary matrix `Q` in a product of `n-1` elementary reflectors. tau `(n)` is used as workspace.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.117 sytrd (USM Version)

Reduces a real symmetric matrix to tridiagonal form. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.117.1 Description

The routine reduces a real symmetric matrix `A` to symmetric tridiagonal form `T` by an orthogonal similarity transformation: `A = Q*T*QT`. The orthogonal matrix `Q` is not formed explicitly but is represented as a product of n-1 elementary reflectors. Routines are provided for working with `Q` in this representation.

11.117.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event sytrd(queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *d,
  T *e,
  T *tau,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`sytrd`(USM version) supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a stores the upper triangular part of `A`.

If `uplo = uplo::lower`, a stores the lower triangular part of `A`.

n The order of the matrices `A(0 ≤ n)`.

a The pointer to matrix `A`, size `(lda,*)`. Contains the upper or lower triangle as specified by uplo.

lda The leading dimension of a; at least `max(1,n)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the **sytrd_scratchpad_size** function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a On exit,

if `uplo = uplo::upper`, the diagonal and first superdiagonal of `A` are overwritten by the corresponding elements of the tridiagonal matrix `T`, and the elements above the first superdiagonal, with the array tau, represent the orthogonal matrix `Q` as a product of elementary reflectors;

if `uplo = uplo::lower`, the diagonal and first subdiagonal of `A` are overwritten by the corresponding elements of the tridiagonal matrix `T`, and the elements below the first subdiagonal, with the array tau, represent the orthogonal matrix `Q` as a product of elementary reflectors.

d Pointer to the diagonal elements of the matrix `T`. The dimension of d must be at least `max(1, n)`.

e Pointer to the off diagonal elements of the matrix `T`. The dimension of e must be at least `max(1, n-1)`.

tau Pointer to the memory array of size at least `max(1, n)`. Stores `(n-1)` scalars that define elementary reflectors in decomposition of the unitary matrix `Q` in a product of `n-1` elementary reflectors. `tau(n)` is used as workspace.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.118 sytrd_scratchpad_size

Computes size of scratchpad memory required for `sytrd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.118.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `sytrd` (buffer or USM version) function must be able to hold.

11.118.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t sytrd_scratchpad_size(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the sytrd (buffer or USM version) function will be performed.

uplo Must be `uplo::upper` or `uplo::lower`.

If `uplo = uplo::upper`, a stores the upper triangular part of `A`.

If `uplo = uplo::lower`, a stores the lower triangular part of `A`.

n The order of the matrices `A(0 ≤ n)`.

lda The leading dimension of a; at least `max(1,n)`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `sytrd` (buffer or USM version) function must be able to hold.

11.119 sytrf

Computes the Bunch-Kaufman factorization of a symmetric matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.119.1 Description

The routine computes the factorization of a real/complex symmetric matrix `A` using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is:

- if `uplo=uplo::upper`, $A = U*D*U^T$
- if `uplo=uplo::lower`, $A = L*D*L^T$

where `A` is the input matrix, `U` and `L` are products of permutation and triangular matrices with unit diagonal (upper triangular for `U` and lower triangular for `L`), and `D` is a symmetric block-diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks. `U` and `L` have 2-by-2 unit diagonal blocks corresponding to the 2-by-2 blocks of `D`.

11.119.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void sytrf(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &ipiv,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`sytrf` supports the following precisions and devices:

T	Devices Supported
`float`	CPU
`double`	CPU
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue	Device queue where calculations will be performed.
uplo	Indicates whether the upper or lower triangular part of `A` is stored and how `A` is factored: If `uplo=uplo::upper`, the buffer a stores the upper triangular part of the matrix `A`, and `A` is factored as `U*D*UT`. If `uplo=uplo::lower`, the buffer a stores the lower triangular part of the matrix `A`, and `A` is factored as `L*D*LT`.
n	The order of matrix `A` (`0 ≤ n`).
a	Buffer holding the coefficients of matrix `A`, size `max(1,lda*n)` containing either the upper or lower triangular part of the matrix `A` (see uplo). The second dimension of a must be at least `max(1, n)`.
lda	The leading dimension of a.
scratch-pad	Buffer holding scratchpad memory to be used by the routine for storing intermediate results.
scratch-pad_-size	Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the **sytrf_scratchpad_size** function.

Output Parameters

a	The upper or lower triangular part of a is overwritten by details of the block-diagonal matrix `D` and the multipliers used to obtain the factor `U` (or `L`).
ipiv	Buffer holding array of size at least `max(1, n)`. Contains details of the interchanges and the block structure of `D`. If `ipiv(i)=k>0`, then d_{ii} is a 1-by-1 block, and the `i`-th row and column of `A` was interchanged with the k-th row and column. If `uplo=mkl::uplo::upper` and `ipiv(i)=ipiv(i-1)=-m<0`, then `D` has a 2-by-2 block in rows/columns `i` and `i-1`, and (`i-1`)-th row and column of `A` was interchanged with the `m`-th row and column. If `uplo=mkl::uplo::lower` and `ipiv(i)=ipiv(i+1)=-m<0`, then `D` has a 2-by-2 block in rows/columns `i` and `i+1`, and (`i+1`)-th row and column of `A` was interchanged with the `m`-th row and column.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, `dii` is 0. The factorization has been completed, but `D` is exactly singular. Division by 0 will occur if you use `D` for solving a system of linear equations. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.120 sytrf (USM Version)

Computes the Bunch-Kaufman factorization of a symmetric matrix. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.120.1 Description

The routine computes the factorization of a real/complex symmetric matrix `A` using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is:

- if `uplo=uplo::upper`, `A = U*D*U`^{T}
- if `uplo=uplo::lower`, `A = L*D*L`^{T}

where `A` is the input matrix, `U` and `L` are products of permutation and triangular matrices with unit diagonal (upper triangular for `U` and lower triangular for `L`), and `D` is a symmetric block-diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks. `U` and `L` have 2-by-2 unit diagonal blocks corresponding to the 2-by-2 blocks of `D`.

11.120.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event sytrf(queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
```

(continues on next page)

(continued from previous page)

```
  std::int64_t lda,
  std::int64_t *ipiv,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`sytrf` (USM version) supports the following precisions and devices:

T	Devices Supported
`float`	CPU
`double`	CPU
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates whether the upper or lower triangular part of `A` is stored and how `A` is factored:

If `uplo = uplo::upper`, the array a stores the upper triangular part of the matrix `A`, and `A` is factored as $U*D*U^T$.

If `uplo = uplo::lower`, the array a stores the lower triangular part of the matrix `A`, and `A` is factored as $L*D*L^T$.

n The order of the matrix `A` (`0 ≤ n`).

a Pointer to the coefficients of matrix `A`, size `max(1,lda*n)` containing either the upper or lower triangular part of the matrix `A` (see uplo). The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the sytrf_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a The upper or lower triangular part of a is overwritten by details of the block-diagonal matrix `D` and the multipliers used to obtain the factor `U` (or `L`).

ipiv Pointer to memory array of size at least `max(1,n)`. Contains details of the interchanges and the block structure of `D`. If `ipiv(i) = k > 0`, then `dii` is a 1-by-1 block, and the `i`-th row and column of `A` was interchanged with the k-th row and column.

If `uplo = mkl::uplo::upper` and `ipiv(i) = ipiv(i-1) = -m < 0`, then `D` has a 2-by-2 block in rows/columns `i` and `i-1`, and `(i-1)`-th row and column of `A` was interchanged with the `m`-th row and column.

If `uplo = mkl::uplo::lower` and `ipiv(i) = ipiv(i+1) = -m < 0`, then `D` has a 2-by-2 block in rows/columns `i` and `i+1`, and `(i+1)`-th row and column of `A` was interchanged with the `m`-th row and column.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info = i`, `dii` is 0. The factorization has been completed, but `D` is exactly singular. Division by 0 will occur if you use `D` for solving a system of linear equations. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.121 sytrf_scratchpad_size

Computes size of scratchpad memory required for `sytrf` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.121.1 Description

Computes the number of elements of type `T` the scratchpad memory to be passed to the `sytrf` (buffer or USM version) function must be able to hold.

11.121.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t sytrf_scratchpad_size(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the sytrf (buffer or USM version) function will be performed.

uplo Indicates whether the upper or lower triangular part of `A` is stored and how `A` is factored:

If `uplo = uplo::upper`, the array a stores the upper triangular part of the matrix `A`, and `A` is factored as $U*D*U^T$.

If `uplo = uplo::lower`, the array a stores the lower triangular part of the matrix `A`, and `A` is factored as $L*D*L^T$.

n The order of the matrix `A` (`0 ≤ n`).

lda The leading dimension of a.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `sytrf` (buffer or USM version) function must be able to hold.

11.122 trtrs

Solves a system of linear equations with a triangular coefficient matrix, with multiple right-hand sides. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.122.1 Description

The routine solves for X the following systems of linear equations with a triangular matrix A, with multiple right-hand sides stored in B:

A*X = B		if `transa = transpose::nontrans`,
AT*X = B		if `transa = transpose::trans`,
A^H*X = B		if `transa = transpose::conjtrans` (for complex matrices only).

11.122.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void trtrs(sycl::queue &queue,
  mkl::uplo uplo,
  mkl::transpose trans,
  mkl::diag diag,
  std::int64_t n,
  std::int64_t nrhs,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &b,
  std::int64_t ldb,
  sycl::buffer<T> &scratchpad,
  std::int64_t   scratchpad_size)
}
```

`trtrs` supports the following precisions and devices.

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates whether A is upper or lower triangular:

If uplo = `uplo::upper`, then A is upper triangular.

If uplo = `uplo::lower`, then A is lower triangular.

trans If transa = `transpose::nontrans`, then A*X = B is solved for X.

If transa = `transpose::trans`, then A^T*X = B is solved for X.

If transa = `transpose::conjtrans`, then A^H*X = B is solved for X.

diag If diag = `diag::nonunit`, then A is not a unit triangular matrix.

If diag = `diag::unit`, then A is unit triangular: diagonal elements of A are assumed to be 1 and not referenced in the array a.

n The order of A; the number of rows in B; `n ≥ 0`.

nrhs The number of right-hand sides; `nrhs ≥ 0`.

a Array containing the matrix A.

The second dimension of a must be at least `max(1,n)`.

lda The leading dimension of a; `lda ≥ max(1, n)`.

b Array containing the matrix B whose columns are the right-hand sides for the systems of equations.

The second dimension of b at least `max(1,nrhs)`.

ldb The leading dimension of b; `ldb ≥ max(1, n)`.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the trtrs_scratchpad_size function.

Output Parameters

b Overwritten by the solution matrix X.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

info Buffer containing error information.

If `info` = 0, the execution is successful.

If `info` = `-i`, the `i`-th parameter had an illegal value.

Known Limitations

GPU support for this function does not include error reporting through the `info` parameter.

11.123 trtrs (USM Version)

Solves a system of linear equations with a triangular coefficient matrix, with multiple right-hand sides. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.123.1 Description

The routine solves for `X` the following systems of linear equations with a triangular matrix `A`, with multiple right-hand sides stored in `B`:

A*X = B	if `transa =transpose::nontrans`,
AT*X = B	if `transa =transpose::trans`,
A^H*X = B	if `transa =transpose::conjtrans` (for complex matrices only).

11.123.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event trtrs(sycl::queue &queue,
  mkl::uplo uplo,
  mkl::transpose trans,
  mkl::diag diag,
  std::int64_t n,
  std::int64_t nrhs,
  T *a,
  std::int64_t lda,
  T *b,
  std::int64_t ldb,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`trtrs` (USM version) supports the following precisions and devices.

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Indicates whether A is upper or lower triangular:

If uplo = `uplo::upper`, then A is upper triangular.

If uplo = `uplo::lower`, then A is lower triangular.

trans If transa = `transpose::nontrans`, then A*X = B is solved for X.

If transa = `transpose::trans`, then A^T*X = B is solved for X.

If transa = `transpose::conjtrans`, then A^H*X = B is solved for X.

diag If diag = `diag::nonunit`, then A is not a unit triangular matrix.

If diag = `diag::unit`, then A is unit triangular: diagonal elements of A are assumed to be 1 and not referenced in the array a.

n The order of A; the number of rows in B; `n ≥ 0`.

nrhs The number of right-hand sides; nrhs" ≥ 0".

a Array containing the matrix A.

The second dimension of a must be at least `max(1,n)`.

lda The leading dimension of a; `lda ≥ max(1, n)`.

b Array containing the matrix B whose columns are the right-hand sides for the systems of equations.

The second dimension of b at least `max(1,nrhs)`.

ldb The leading dimension of b; ldb" ≥ max(1, n)".

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the trtrs_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

b Overwritten by the solution matrix X.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.124 trtrs_scratchpad_size

Computes size of scratchpad memory required for `trtrs` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.124.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `trtrs` (buffer or USM version) function must be able to hold.

11.124.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t trtrs_scratchpad_size(sycl::queue &queue,
  mkl::uplo uplo,
  mkl::transpose trans,
  mkl::diag diag,
  std::int64_t n,
```

(continues on next page)

(continued from previous page)

```
  std::int64_t nrhs,
  std::int64_t lda,
  std::int64_t ldb)
}
```

Input Parameters

queue Device queue where calculations by the trtrs (buffer or USM version) function will be performed.

uplo Indicates whether A is upper or lower triangular:

If uplo = `uplo::upper`, then A is upper triangular.

If uplo = `uplo::lower`, then A is lower triangular.

trans If trans = `transpose::nontrans`, then A*X = B is solved for X.

If trans = `transpose::trans`, then A^T*X = B is solved for X.

If trans = `transpose::conjtrans`, then A^H*X = B is solved for X.

diag If diag = `diag::nonunit`, then A is not a unit triangular matrix.

If diag = `diag::unit`, then A is unit triangular: diagonal elements of A are assumed to be 1 and not referenced in the array a.

n The order of A; the number of rows in B; n " ≥ 0".

nrhs The number of right-hand sides; nrhs" ≥ 0".

lda The leading dimension of a; lda" ≥ max(1, n)".

ldb The leading dimension of b; ldb" ≥ max(1, n)".

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `trtrs` (buffer or USM version) function must be able to hold.

11.125 ungbr

Generates the complex unitary matrix Q or P^t determined by gebrd. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.125.1 Description

The routine generates the whole or part of the unitary matrices `Q` and P^H formed by the routines gebrd. Use this routine after a call to `cgebrd/zgebrd`. All valid combinations of arguments are described in **Input Parameters**; in most cases you need the following:

To compute the whole `m`-by-`m` matrix `Q`, use:

```
ungbr(queue, generate::q, m, m, n, a, ...)
```

(note that the buffera must have at least `m` columns).

To form the `n` leading columns of `Q` if `m > n`, use:

```
ungbr(queue, generate::q, m, n, n, a, ...)
```

To compute the whole n-by-n matrix P^T, use:

```
ungbr(queue, generate::p, n, n, m, a, ...)
```

(note that the array a must have at least `n` rows).

To form the `m` leading rows of P^T if `m < n`, use:

```
ungbr(queue, generate::p, m, n, m, a, ...)
```

11.125.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void ungbr(sycl::queue &queue,
  mkl::generate gen,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`ungbr` supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue Device queue where calculations will be performed.

gen Must be `generate::q` or `generate::p`.

If gen= `generate::q`, the routine generates the matrix `Q`.

If gen= `generate::p`, the routine generates the matrix P^T.

m The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ m)`.

If gen= `generate::q`, `m ≥ n ≥ min(m, k)`.

If gen= `generate::p`, `n ≥ m ≥ min(n, k)`.

n The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ n)`. See m for constraints.

k If gen= `generate::q`, the number of columns in the original `m`-by-k matrix reduced by gebrd.

If gen= `generate::p`, the number of rows in the original k-by-n matrix reduced by gebrd.

a Buffer holding memory returned by gebrd.

lda The leading dimension of a.

tau For gen= `generate::q`, the array tauq is returned by the gebrd function.

For gen= `generate::p`, the array taup is returned by the gebrd function.

The dimension of tau must be at least `min(m,k)` for `gen = generate::q`, or `min(n,k)` for `gen = generate::p`.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the ungbr_scratchpad_size function.

Output Parameters

a Overwritten by n leading columns of the m-by-m unitary matrix `Q` or P^T, (or the leading rows or columns thereof) as specified by gen, m, and n.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.126 ungbr (USM Version)

Generates the complex unitary matrix Q or P^t determined by the `gebrd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.126.1 Description

The routine generates the whole or part of the unitary matrices `Q` and P^T formed by the gebrd (USM Version) function. All valid combinations of arguments are described in **Input Parameters**; in most cases you need the following:

To compute the whole `m`-by-`m` matrix `Q`, use:

```
ungbr(queue, generate::q, m, m, n, a, ...)
```

(note that the array a must have at least `m` columns).

To form the `n` leading columns of `Q` if `m > n`, use:

```
ungbr(queue, generate::q, m, n, n, a, ...)
```

To compute the whole `n`-by-`n` matrix P^T, use:

```
ungbr(queue, generate::p, n, n, m, a, ...)
```

(note that the array a must have at least `n` rows).

To form the `m` leading rows of P^T if `m < n`, use:

```
ungbr(queue, generate::p, m, n, m, a, ...)
```

11.126.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event ungbr(sycl::queue &queue,
  mkl::generate gen,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  T *a,
  std::int64_t lda,
  T *tau,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`ungbr` (USM version) supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue Device queue where calculations will be performed.

gen Must be `generate::q` or `generate::p`.

If gen= `generate::q`, the routine generates the matrix `Q`.

If gen= `generate::p`, the routine generates the matrix P^T.

m The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ m)`.

If gen= `generate::q`, `m ≥ n ≥ min(m, k)`.

If gen= `generate::p`, `n ≥ m ≥ min(n, k)`.

n The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ n)`. See m for constraints.

k If gen= `generate::q`, the number of columns in the original `m`-by-k matrix reduced by gebrd (USM Version).

If gen= `generate::p`, the number of rows in the original k-by-n matrix reduced by gebrd (USM Version).

a Pointer to memory returned by gebrd (USM Version).

lda The leading dimension of a.

tau For gen= `generate::q`, the array tauq is returned by the gebrd (USM Version) function.

For gen= `generate::p`, the array taup is returned by the gebrd (USM Version) function.

The dimension of tau must be at least `min(m,k)` for `gen = generate::q`, or `min(n,k)` for `gen = generate::p`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the ungbr_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by n leading columns of the m-by-m orthogonal matrix `Q` or P^T (or the leading rows or columns thereof) as specified by gen, m, and n.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.127 ungbr_scratchpad_size

Computes size of scratchpad memory required for `ungbr` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.127.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `ungbr` (buffer or USM version) function must be able to hold.

11.127.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t ungbr_scratchpad_size(sycl::queue &queue,
  mkl::generate gen,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the ungbr (buffer or USM version) function will be performed.

gen Must be `generate::q` or `generate::p`.

If gen= `generate::q`, the routine generates the matrix `Q`.

If gen= `generate::p`, the routine generates the matrix P^T.

m The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ m)`.

If gen= `generate::q`, `m ≥ n ≥ min(m, k)`.

If gen= `generate::p`, `n ≥ m ≥ min(n, k)`.

n The number of rows in the matrix `Q` or P^T to be returned `(0 ≤ n)`. See m for constraints.

k If gen= `generate::q`, the number of columns in the original `m`-by-k matrix reduced by gebrd (USM Version).

If gen= `generate::p`, the number of rows in the original k-by-n matrix reduced by gebrd (USM Version).

lda The leading dimension of a.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `ungbr` (buffer or USM version) function must be able to hold.

11.128 ungqr

Generates the complex unitary matrix Q of the QR factorization formed by `geqrf`. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.128.1 Description

The routine generates the whole or part of `m`-by-`m` unitary matrix `Q` of the `QR` factorization formed by the routines geqrf.

Usually `Q` is determined from the `QR` factorization of an `m` by `p` matrix `A` with `m ≥ p`. To compute the whole matrix `Q`, use:

```
mkl::ungqr(queue, m, m, p, a, lda, tau, ...)
```

To compute the leading `p` columns of `Q` (which form an orthonormal basis in the space spanned by the columns of `A`):

```
mkl::ungqr(queue, m, p, p, a, lda, tau, ...)
```

To compute the matrix Q^k of the `QR` factorization of the leading `k` columns of the matrix `A`:

```
mkl::ungqr(queue, m, m, k, a, lda, tau, ...)
```

To compute the leading `k` columns of Q^k (which form an orthonormal basis in the space spanned by the leading `k` columns of the matrix `A`):

```
mkl::ungqr(queue, m, k, k, a, lda, tau, ...)
```

11.128.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void ungqr(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
```

(continues on next page)

(continued from previous page)

```
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

ungqr" supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a Buffer holding the result of geqrf.

lda The leading dimension of `A` (`lda ≥ m`).

tau Buffer holding the result of geqrf.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the ungqr_scratchpad_size function.

Output Parameters

a Overwritten by n leading columns of the m-by-m unitary matrix `Q`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.129 ungqr (USM Version)

Generates the complex unitary matrix Q of the QR factorization formed by the `geqrf` (USM Version) function. . This routine belongs to the `oneapi::mkl::lapack` namespace.

11.129.1 Description

The routine generates the whole or part of `m`-by-`m` unitary matrix `Q` of the `QR` factorization formed by the **geqrf (USM Version)** function.

Usually `Q` is determined from the `QR` factorization of an `m` by `p` matrix `A` with `m ≥ p`. To compute the whole matrix `Q`, use:

```
ungqr(queue, m, m, p, a, lda, tau, ...)
```

To compute the leading `p` columns of `Q` (which form an orthonormal basis in the space spanned by the columns of `A`):

```
ungqr(queue, m, p, p, a, lda, tau, ...)
```

To compute the matrix Q^k of the `QR` factorization of the leading `k` columns of the matrix `A`:

```
ungqr(queue, m, m, k, a, lda, tau, ...)
```

To compute the leading `k` columns of Q^k (which form an orthonormal basis in the space spanned by the leading `k` columns of the matrix `A`):

```
ungqr(queue, m, k, k, a, lda, tau, ...)
```

11.129.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event ungqr(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  T *a,
  std::int64_t lda,
  T *tau,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`ungqr` (USM version) supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a Pointer to the result of geqrf (USM Version).

lda The leading dimension of a (`lda ≥ m`).

tau Pointer to the result of geqrf (USM Version).

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the ungqr_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by n leading columns of the m-by-m unitary matrix `Q`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.130 ungqr_batch (Buffer Strided Version)

Generates the complex unitary matrices Q_i of the batch of QR factorizations formed by `geqrf_batch` function (Buffer stride version). This routine belongs to the `oneapi::mkl::lapack` namespace.

11.130.1 Description

The routine generates the wholes or parts of `m`-by-`m` unitary matrix Q_i of the QR factorization formed by the geqrf_batch (Buffer Strided Version) function.

Usually Q_i is determined from the QR factorization of an `m` by `p` matrix A_i with $m \geq p$. To compute the whole matrices Q_i, use:

```
ungqr_batch(queue, m, m, p, a, ...)
```

To compute the leading `p` columns of Q_i (which form an orthonormal basis in the space spanned by the columns of A_i):

```
ungqr_batch(queue, m, p, p, a, ...)
```

To compute the matrix Q_i^k of the QR factorization of the leading `k` columns of the matrices A_i:

```
ungqr_batch(queue, m, m, k, a, ...)
```

To compute the leading `k` columns of Q_i^k (which form an orthonormal basis in the space spanned by the leading `k` columns of the matrices A_i):

```
ungqr_batch(queue, m, k, k, a, ...)
```

11.130.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void ungqr_batch(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  sycl::buffer<T> &a,
```

(continues on next page)

(continued from previous page)

```
  std::int64_t lda,
  std::int64_t stride_a,
  sycl::buffer<T> &tau,
  std::int64_t stride_tau,
  std::int64_t batch_size,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

This function supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i ($0 \le m$).

n The number of columns in the matrices A_i ($0 \le n$).

k the number of elementary reflectors whose product defines the matrices Q_i ($0 \le k \le n$).

a Array resulting after a call to geqrf_batch (Buffer Strided Version).

lda The leading dimension of A_i ($lda \ge m$).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

tau Array resulting after a call to geqrf_batch (Buffer Strided Version).

stride_tau The stride between the beginnings of arrays tau_i inside the array tau.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by ungqr_batch_scratchpad_size (Strided Version).

Output Parameters

a Array data is overwritten by a batch of n leading columns of the m-by-m unitary matrices Q_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

11.131 ungqr_batch (Group Version)

Generates the complex unitary matrices Q_i of the batch of QR factorizations formed by `geqrf_batch`. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.131.1 Description

The routine generates the wholes or parts of `m`-by-`m` unitary matrix Q_i of the QR factorization formed by the geqrf_batch (Group Version) function.

Usually Q_i is determined from the QR factorization of an `m` by `p` matrix A_i with `m` ≥ `p`. To compute the whole matrices Q_i, use:

```
ungqr_batch(queue, m, m, p, a, ...)
```

To compute the leading `p` columns of Q_i (which form an orthonormal basis in the space spanned by the columns of A_i):

```
ungqr_batch(queue, m, p, p, a, ...)
```

To compute the matrix Q_i^k of the QR factorization of the leading `k` columns of the matrices A_i:

```
ungqr_batch(queue, m, m, k, a, ...)
```

To compute the leading `k` columns of Q_i^k (which form an orthonormal basis in the space spanned by the leading `k` columns of the matrices A_i):

```
ungqr_batch(queue, m, k, k, a, ...)
```

11.131.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event ungqr_batch(sycl::queue &queue,
  std::int64_t *m,
  std::int64_t *n,
  std::int64_t *k,
  T **a,
  std::int64_t *lda,
  T **tau,
  std::int64_t group_count,
  std::int64_t *group_sizes,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m Array of `group_count` parameters m_g as previously supplied to geqrf_batch (Group Version).

n Array of `group_count` parameters n_g as previously supplied to geqrf_batch (Group Version).

k Array of `group_count` parameters k_g as previously supplied to geqrf_batch (Group Version). The number of elementary reflectors whose product defines the matrices Q_i ($0 \le k_g \le n_g$).

a Array resulting after a call to geqrf_batch (Group Version).

lda Array of leading dimension of A_i as previously supplied to geqrf_batch (Group Version).

tau Array resulting after a call to geqrf_batch (Group Version).

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of group_count integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less then the value returned by ungqr_batch_scratchpad_size (Group Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Matrices pointed to by array a are overwritten by n_g leading columns of the m_g-by-m_g unitary matrices Q_i, where g is an index of group of parameters corresponding to Q_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the n-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.132 ungqr_batch (USM Strided Version)

Generates the complex unitary matrices Q_i of the batch of QR factorizations formed by `geqrf_batch` function (USM stride version). This routine belongs to the `oneapi::mkl::lapack` namespace.

11.132.1 Description

The routine generates the wholes or parts of m-by-m unitary matrix Q_i of the QR factorization formed by the geqrf_batch (USM Strided Version) function.

Usually Q_i is determined from the QR factorization of an m by p matrix A_i with $m \geq p$. To compute the whole matrices Q_i, use:

```
ungqr_batch(queue, m, m, p, a, ...)
```

To compute the leading p columns of Q_i (which form an orthonormal basis in the space spanned by the columns of A_i):

```
ungqr_batch(queue, m, p, p, a, ...)
```

To compute the matrix Q_i^k of the QR factorization of the leading k columns of the matrices A_i:

```
ungqr_batch(queue, m, m, k, a, ...)
```

To compute the leading k columns of Q_i^k (which form an orthonormal basis in the space spanned by the leading k columns of the matrices A_i):

```
ungqr_batch(queue, m, k, k, a, ...)
```

11.132.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event ungqr_batch(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  T *a,
  std::int64_t lda,
  std::int64_t stride_a,
  T *tau,
  std::int64_t  stride_tau,
  std::int64_t batch_size,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

This function supports the following precisions and devices:

T	Devices supported
std::complex<float>	CPU and GPU
std::complex<double>	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i ($0 \le m$).

n The number of columns in the matrices A_i ($0 \le n$).

k the number of elementary reflectors whose product defines the matrices Q_i ($0 \le k \le n$).

a Array resulting after a call to geqrf_batch (USM Strided Version).

lda The leading dimension of A_i ($lda \ge m$).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

tau Array resulting after a call to geqrf_batch (USM Strided Version).

stride_tau The stride between the beginnings of arrays tau_i inside the array `tau`.

batch_size The number of problems in a batch.

scratchpad Scratchpad memory to be used by routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by ungqr_batch_scratchpad_size (Strided Version).

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Array data is overwritten by a batch of `n` leading columns of the `m`-by-`m` unitary matrices Q_i.

Exceptions

Exception	Description
`mkl::lapack::batch_exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -n`, the `n`-th parameter had an illegal value. If `info` equals the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad is of insufficient size, and the required size should be not less then value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.133 ungqr_batch_scratchpad_size (Group Version)

Computes size of scratchpad memory required for `ungqr_batch` (Group Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.133.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the ungqr_batch (Group Version) function must be able to hold.

11.133.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t ungqr_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t *m,
  std::int64_t *n,
  std::int64_t *k,
  std::int64_t *lda,
  std::int64_t group_count,
  std::int64_t *group_sizes)
}
```

This function supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

m Array of `group_count` parameters m_g as previously supplied to ungqr_batch (Group Version).

n Array of `group_count` parameters n_g as previously supplied to ungqr_batch (Group Version).

k Array of `group_count` parameters k_g as previously supplied to ungqr_batch (Group Version).

The number of elementary reflectors whose product defines the matrices Q_i ($0 \le k_g \le n_g$).

lda The leading dimension of A_i as previously supplied to ungqr_batch (Group Version).

group_count Specifies the number of groups of parameters. Must be at least 0.

group_sizes Array of `group_count` integers. Array element with index `g` specifies the number of problems to solve for each of the groups of parameters `g`. So the total number of problems to solve, `batch_size`, is a sum of all parameter group sizes.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the ungqr_batch (Group Version) function must be able to hold.

11.134 ungqr_batch_scratchpad_size (Strided Version)

Computes size of scratchpad memory required for `ungqr_batch` (Strided Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.134.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `ungqr_batch` (Strided Version) function must be able to hold.

11.134.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  std::int64_t ungqr_batch_scratchpad_size(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  std::int64_t lda,
  std::int64_t stride_a,
  std::int64_t stride_tau,
  std::int64_t batch_size)
}
```

Input Parameters

queue Device queue where calculations will be performed.

m The number of rows in the matrices A_i ($0 \le m$).

n The number of columns in the matrices A_i ($0 \le n$).

k the number of elementary reflectors whose product defines the matrices Q_i ($0 \le k \le n$).

lda The leading dimension of A_i ($lda \ge m$).

stride_a The stride between the beginnings of matrices A_i inside the batch array a.

stride_tau The stride between the beginnings of arrays tau_i inside the array `tau`.

batch_size The number of problems in a batch.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `ungqr_batch` (Strided Version) function must be able to hold.

11.135 ungqr_scratchpad_size

Computes size of scratchpad memory required for `ungqr` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.135.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `ungqr` (buffer or USM version) function must be able to hold.

11.135.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t ungqr_scratchpad_size(sycl::queue &queue,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the ungqr (buffer or USM version) function will be performed.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

lda The leading dimension of a (`lda ≥ m`).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `ungqr` (buffer or USM version) function must be able to hold.

11.136 ungtr

Generates the complex unitary matrix Q determined by **hetrd**. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.136.1 Description

The routine explicitly generates the `n`-by-`n` unitary matrix `Q` formed by **hetrd** when reducing a complex Hermitian matrix `A` to tridiagonal form: `A = Q*T*QH`. Use this routine after a call to **hetrd**.

11.136.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void ungtr(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`ungtr` supports the following precision and devices.

T	Devices Supported
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Must be uplo::upper or uplo::lower. Uses the same uplo as supplied to the **hetrd** function.

n The order of matrix `Q` (`0 ≤ n`).

a The buffer a returned by the **hetrd** function. The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a (`lda ≥ n`).

tau The buffer tau returned by the **hetrd** function. The dimension of tau must be at least `max(1,n-1)`.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the **ungtr_scratchpad_size** function.

Output Parameters

a Overwritten by the unitary matrix `Q`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.137 ungtr (USM Version)

Generates the complex unitary matrix Q determined by the `hetrd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.137.1 Description

The routine explicitly generates the n-by-n unitary matrix `Q` formed by the **hetrd (USM Version)** when reducing a complex Hermitian matrix `A` to tridiagonal form: `A = Q*T*QT`. Use this routine after a call to **hetrd (USM Version)** function.

11.137.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event ungtr(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *tau,
  T *tau,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`ungtr` (USM version) supports the following precision and devices.

T	Devices Supported
`float`	CPU
`double`	CPU

Input Parameters

queue Device queue where calculations will be performed.

uplo Must be uplo::upper or uplo::lower. Uses the same uplo as supplied to the **hetrd (USM Version)** function.

n The order of matrix `Q` (`0 ≤ n`).

a The pointer a returned by the **hetrd (USM Version)** function. The second dimension of a must be at least `max(1, n)`.

lda The leading dimension of a (`lda ≥ n`).

tau The pointer tau returned by the hetrd (USM Version) function. The dimension of tau must be at least `max(1,n-1)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the ungtr_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

a Overwritten by the unitary matrix `Q`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.138 ungtr_scratchpad_size

Computes size of scratchpad memory required for `ungtr` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.138.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `ungtr` (buffer or USM version) function must be able to hold.

11.138.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t ungtr_scratchpad_size(sycl::queue &queue,
  mkl::uplo uplo,
  std::int64_t n,
  std::int64_t lda)
}
```

Input Parameters

queue Device queue where calculations by the ungtr (buffer or USM version) function will be performed.

uplo Must be uplo::upper or uplo::lower. Uses the same uplo as supplied to the hetrd (USM Version) function.

n The order of matrix `Q` (`0 ≤ n`).

lda The leading dimension of a (`lda ≥ n`).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type T the scratchpad memory to be passed to the `ungtr` (buffer or USM version) function must be able to hold.

11.139 unmqr

Multiplies a complex matrix by the unitary matrix Q of the QR factorization formed by `unmqr`. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.139.1 Description

The routine multiplies a rectangular complex matrix C by Q or Q^H, where Q is the unitary matrix Q of the QR factorization formed by the routines geqrf.

Depending on the parameters `left_right` and `trans`, the routine can form one of the matrix products `Q*C`, Q^H`*C`, `C*Q`, or `C*`Q^H (overwriting the result on C).

11.139.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void unmqr(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &c,
  std::int64_t ldc,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`unmqr` supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

queue Device queue where calculations will be performed.

side If `side` = `mkl::side::left`, Q or Q^T is applied to C from the left. If `side` = `mkl::side::right`, Q or Q^T is applied to C from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies C by Q.

If `trans=mkl::transpose::nontrans`, the routine multiplies C by Q^T.

m The number of rows in the matrix A (`0 ≤ m`).

n The number of columns in the matrix A (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix Q (`0 ≤ k ≤ n`).

a Buffer holding the result of the geqrf function. The second dimension of a must be at least `max(1,k)`.

lda The leading dimension of a.

tau Buffer holding tau returned by the geqrf function.

c Buffer holding the matrix C. The second dimension of c must be at least max(1,n).

ldc The leading dimension of c.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the unmqr_scratchpad_size function.

Output Parameters

c Overwritten by the product Q*C, Q^T*C, C*Q, or C*Q^T (as specified by side and trans).

Exceptions

Exception	Description
mkl::lapack::exception	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If info = -i, the i-th parameter had an illegal value. If info is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.140 unmqr (USM Version)

Multiplies a complex matrix by the unitary matrix Q of the QR factorization formed by the geqrf (USM Version) function. This routine belongs to the oneapi::mkl::lapack namespace.

11.140.1 Description

The routine multiplies a complex matrix C by Q or Q^T, where Q is the orthogonal matrix Q of the QR factorization formed by the routine geqrf (USM Version).

Depending on the parameters side and trans, the routine can form one of the matrix products Q*C, Q^T*C, C*Q, or C*Q^T (overwriting the result on C).

11.140.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event unmqr(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  T *a,
  std::int64_t lda,
  T *tau,
  T *c,
  std::int64_t ldc,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`ormqr` (USM version) supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU, GPU
`std::complex<double>`	CPU, GPU

Input Parameters

queue Device queue where calculations will be performed.

side If `side` = `mkl::side::left`, `Q` or Q^T is applied to `C` from the left. If `side` = `mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a The pointer to the result of the geqrf (USM Version) function. The second dimension of a must be at least `max(1,k)`.

lda The leading dimension of a.

tau The pointer to tau returned by the geqrf (USM Version) function.

c The pointer to the matrix `C`. The second dimension of c must be at least `max(1,n)`.

ldc The leading dimension of c.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the unmqr_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

c Overwritten by the product `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (as specified by side and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.141 unmqr_scratchpad_size

Computes size of scratchpad memory required for `unmqr` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.141.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `unmqr` (buffer or USM version) function must be able to hold.

11.141.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t unmqr_scratchpad_size(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  std::int64_t lda,
  std::int64_t ldc)
}
```

Input Parameters

queue Device queue where calculations by the unmqr (buffer or USM version) function will be performed.

side If `side` = `mkl::side::left`, `Q` or Q^T is applied to `C` from the left. If `side` = `mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

lda The leading dimension of a.

ldc The leading dimension of c.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `unmqr` (buffer or USM version) function must be able to hold.

11.142 unmrq

Multiplies a complex matrix by the unitary matrix Q of the RQ factorization formed by `gerqf`. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.142.1 Description

The routine multiplies a complex `m`-by-`n` matrix `C` by `Q` or Q^H, where `Q` is the complex unitary matrix defined as a product of k elementary reflectors `H(i)` of order `n`: `Q = H(1)H* H(2)`^{H}`*...*H(k)`Has returned by the `RQ` factorization routine gerqf .

Depending on the parameters `side` and `trans`, the routine can form one of the matrix products `Q*C`, Q^H`*C`, `C*Q`, or `C*Q`^{H} (overwriting the result over `C`).

11.142.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void unmrq(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &c,
  std::int64_t ldc,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`unmrq` supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue Device queue where calculations will be performed.

side If `side` = `mkl::side::left`, `Q` or Q^T is applied to `C` from the left. If `side` = `mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a Buffer holding the result of the gerqf function. The second dimension of a must be at least `max(1,k)`.

lda The leading dimension of a.

tau Buffer holding tau returned by the **:ref:'gerqf** function.

c Buffer holding the matrix `C`. The second dimension of c must be at least `max(1,n)`.

ldc The leading dimension of c.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the unmrq_scratchpad_size function.

Output Parameters

c Overwritten by the product `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (as specified by side and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.143 unmrq (USM Version)

Multiplies a complex matrix by the unitary matrix Q of the RQ factorization formed by the `gerqf` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.143.1 Description

The routine multiplies a complex `m`-by-`n` matrix `C` by `Q` or Q^T, where `Q` is the orthogonal matrix Q of the `RQ` factorization formed by the routine gerqf (USM Version) .

Depending on the parameters `side` and `trans`, the routine can form one of the matrix products `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (overwriting the result over `C`).

11.143.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event unmrq(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t k,
  T *a,
  std::int64_t lda,
  T *tau,
  T *c,
  std::int64_t ldc,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`unmrq` (USM version) supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

queue Device queue where calculations will be performed.

side If `side` = `mkl::side::left`, `Q` or Q^T is applied to `C` from the left. If `side` = `mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

a The pointer to the result of the gerqf (USM Version) function. The second dimension of a must be at least `max(1,k)`.

lda The leading dimension of a.

tau The pointer to tau returned by the gerqf (USM Version) function.

c The pointer to the matrix `C`. The second dimension of c must be at least `max(1,n)`.

ldc The leading dimension of c.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the unmrq_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

c Overwritten by the product `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (as specified by side and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.144 unmrq_scratchpad_size

Computes size of scratchpad memory required for `unmrq` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.144.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `unmrq` (buffer or USM version) function must be able to hold.

11.144.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t unmrq_scratchpad_size(sycl::queue &queue,
  mkl::side side,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  std::int64_t kstd::int64_t lda,
  std::int64_t ldc)
}
```

Input Parameters

queue Device queue where calculations by the unmrq (buffer or USM version) function will be performed.

side If `side` = `mkl::side::left`, `Q` or Q^T is applied to `C` from the left. If `side` = `mkl::side::right`, `Q` or Q^T is applied to `C` from the right.

trans If `trans=mkl::transpose::trans`, the routine multiplies `C` by `Q`.

If `trans=mkl::transpose::nontrans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `A` (`0 ≤ m`).

n The number of columns in the matrix `A` (`0 ≤ n ≤ m`).

k The number of elementary reflectors whose product defines the matrix `Q` (`0 ≤ k ≤ n`).

lda The leading dimension of a.

ldc The leading dimension of c.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `unmrq` (buffer or USM version) function must be able to hold.

11.145 unmtr

ultiplies a complex matrix by the complex unitary matrix Q determined by **hetrd**. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.145.1 Description

The routine multiplies a complex matrix `C` by `Q` or Q^H, where `Q` is the unitary matrix `Q` formed by **hetrd** when reducing a complex Hermitian matrix `A` to tridiagonal form: `A = Q*T*QH`. Use this routine after a call to **hetrd**.

Depending on the parameters left_right and trans, the routine can form one of the matrix products `Q*C`, Q^H`*C`, `C*Q`, or `C*`Q^H (overwriting the result on `C`).

11.145.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  void unmtr(sycl::queue &queue,
  mkl::side side,
  mkl::uplo uplo,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  sycl::buffer<T> &a,
  std::int64_t lda,
  sycl::buffer<T> &tau,
  sycl::buffer<T> &c,
  std::int64_t ldc,
  sycl::buffer<T> &scratchpad,
  std::int64_t scratchpad_size)
}
```

`unmtr` supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

In the descriptions below, `r` denotes the order of `Q`:

`r = m`	if `side = side::left`
`r = n`	if `side = side::right`

queue Device queue where calculations will be performed.

side Must be either `side::left` or `side::right`.

If `side = side::left`, `Q` or Q^T is applied to `C` from the left.

If `side = side::right`, `Q` or Q^T is applied to `C` from the right.

uplo Must be either `uplo::upper` or `uplo::lower`. Uses the same `uplo` as supplied to hetrd (USM Version).

trans Must be either `transpose::nontrans` or `transpose::trans`.

If `trans = transpose::nontrans`, the routine multiplies `C` by `Q`.

If `trans = transpose::trans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `C (m ≥ 0)`.

n The number of columns in the matrix `C (n ≥ 0)`.

a Buffer holding array returned by hetrd (USM Version).

lda The leading dimension of `a (max(1, r) ≤ lda)`.

tau Buffer holding tau returned by hetrd (USM Version). The dimension of tau must be at least `max(1, r-1)`.

c Buffer holding the matrix `C`. The second dimension of `c` must be at least `max(1, n)`.

ldc The leading dimension of `c (max(1, n) ≤ ldc)`.

scratchpad Buffer holding scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type `T`. Size should not be less than the value returned by the unmtr_scratchpad_size function.

Output Parameters

c Overwritten by the product `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (as specified by side and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

11.146 unmtr (USM Version)

Multiplies a complex matrix by the complex unitary matrix Q determined by the `hetrd` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.146.1 Description

The routine multiplies a complex matrix `C` by `Q` or Q^H, where `Q` is the unitary matrix `Q` formed by hetrd (USM Version) when reducing a complex Hermitian matrix `A` to tridiagonal form: `A = Q*T*QT`. Use this routine after a call to hetrd (USM Version).

Depending on the parameters side and trans, the routine can form one of the matrix products `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (overwriting the result on `C`).

11.146.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  sycl::event unmtr(sycl::queue &queue,
  mkl::side side,
  mkl::uplo uplo,
  mkl::transpose trans,
  std::int64_t m,
  std::int64_t n,
  T *a,
  std::int64_t lda,
  T *tau,
  T *c,
```

(continues on next page)

(continued from previous page)

```
  std::int64_t ldc,
  T *scratchpad,
  std::int64_t scratchpad_size,
  const std::vector<sycl::event> &events = {})
}
```

`unmtr` (USM version) supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU
`std::complex<double>`	CPU

Input Parameters

In the descriptions below, `r` denotes the order of `Q`:

`r = m`	if `side = side::left`
`r = n`	if `side = side::right`

queue Device queue where calculations will be performed.

side Must be either `side::left` or `side::right`.

If `side = side::left`, `Q` or Q^T is applied to `C` from the left.

If `side = side::right`, `Q` or Q^T is applied to `C` from the right.

uplo Must be either `uplo::upper` or `uplo::lower`. Uses the same `uplo` as supplied to hetrd (USM Version).

trans Must be either `transpose::nontrans` or `transpose::trans`.

If `trans = transpose::nontrans`, the routine multiplies `C` by `Q`.

If `trans = transpose::trans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `C (m ≥ 0)`.

n The number of columns in the matrix `C (n ≥ 0)`.

a The pointer to memory returned by hetrd (USM Version).

lda The leading dimension of a `(max(1, r) ≤ lda)`.

tau The pointer to tau returned by hetrd (USM Version). The dimension of tau must be at least `max(1, r-1)`.

c The pointer to memory containing the matrix `C`. The second dimension of c must be at least `max(1, n)`.

ldc The leading dimension of `c (max(1, n) ≤ ldc)`.

scratchpad Pointer to scratchpad memory to be used by the routine for storing intermediate results.

scratchpad_size Size of scratchpad memory as a number of floating point elements of type T. Size should not be less than the value returned by the unmtr_scratchpad_size function.

events List of events to wait for before starting computation. Defaults to empty list.

Output Parameters

c Overwritten by the product `Q*C`, Q^T`*C`, `C*Q`, or `C*`Q^T (as specified by side and trans).

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when problems occur during calculations. You can obtain the info code of the problem using the info() method of the exception object: If `info = -i`, the `i`-th parameter had an illegal value. If `info` is equal to the value passed as scratchpad size, and detail() returns non-zero, then the passed scratchpad has an insufficient size, and the required size should not be less than the value returned by the detail() method of the exception object.

Return Values

Output event to wait on to ensure computation is complete.

11.147 unmtr_scratchpad_size

Computes size of scratchpad memory required for `unmtr` (USM Version) function. This routine belongs to the `oneapi::mkl::lapack` namespace.

11.147.1 Description

Computes the number of elements of type T the scratchpad memory to be passed to the `unmtr` (buffer or USM version) function must be able to hold.

11.147.2 API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename T>
  std::int64_t unmtr_scratchpad_size(sycl::queue &queue,
  mkl::side side,
  mkl::uplo uplo,
  mkl::transpose trans,
  std::int64_t n,
```

(continues on next page)

(continued from previous page)

```
  std::int64_t lda,
  std::int64_t ldc)
}
```

Input Parameters

In the descriptions below, `r` denotes the order of `Q`:

`r = m`	if `side = side::left`
`r = n`	if `side = side::right`

queue Device queue where calculations by the unmtr (buffer or USM version) function will be performed.

side Must be either `side::left` or `side::right`.

If `side = side::left`, `Q` or Q^T is applied to `C` from the left.

If `side = side::right`, `Q` or Q^T is applied to `C` from the right.

uplo Must be either `uplo::upper` or `uplo::lower`. Uses the same `uplo` as supplied to hetrd (USM Version).

trans Must be either `transpose::nontrans` or `transpose::trans`.

If `trans = transpose::nontrans`, the routine multiplies `C` by `Q`.

If `trans = transpose::trans`, the routine multiplies `C` by Q^T.

m The number of rows in the matrix `C(m ≥ 0)`.

n The number of columns in the matrix `C(n ≥ 0)`.

lda The leading dimension of a `(max(1, r) ≤ lda)`.

ldc The leading dimension of `c (max(1, n) ≤ ldc)`.

Exceptions

Exception	Description
`mkl::lapack::exception`	This exception is thrown when an incorrect argument value is supplied. You can determine the position of the incorrect argument by the info() method of the exception object.

Return Values

The number of elements of type `T` the scratchpad memory to be passed to the `unmtr` (buffer or USM version) function must be able to hold.

12.0 Vector Mathematical Functions

Intel® oneAPI Math Kernel Library (oneMKL) Vector Mathematics functions (VM) compute a mathematical function of each of the vector elements. VM includes a set of highly optimized functions (arithmetic, power, trigonometric, exponential, hyperbolic, special, and rounding) that operate on vectors of real and complex numbers.

Application programs that improve performance with VM include nonlinear programming software, computation of integrals, financial calculations, computer graphics, and many others.

VM functions fall into the following groups according to the operations they perform:

- **VM Mathematical Functions** compute values of mathematical functions, such as sine, cosine, exponential, or logarithm, on vectors stored contiguously in memory.
- **VM Service Functions** set/get the accuracy modes and the error codes,and create error handlers for mathematical functions.

The VM mathematical functions take an input vector as an argument, compute values of the respective function element-wise, and return the results in an output vector. All the VM mathematical functions can perform in-place operations, where the input and output arrays are at the same memory locations. For VM mathematical functions with positive increment indexing, in-place operations are supported only when the input and output increments have the same value.

The oneMKL interfaces are given in:

```
oneapi/mkl/vm.hpp.
```

Examples that demonstrate how to use the VM functions are located in:

```
${MKL}/examples/dpcpp/vml/source
```

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

12.1 Special Value Notations

This topic defines notations of special values for complex functions. The definitions are provided in text, tables, or formulas.

- z, z1, z2, etc. denote complex numbers.
- i, `i2=-1` is the imaginary unit.
- x, X, x1, x2, etc. denote real imaginary parts.

- y, Y, y1, y2, etc. denote imaginary parts.
- X and Y represent any finite positive IEEE-754 floating point values, if not stated otherwise.
- Quiet NaN and signaling NaN are denoted with QNAN and SNAN, respectively.
- The IEEE-754 positive infinities or floating-point numbers are denoted with a + sign before X, Y, etc.
- The IEEE-754 negative infinities or floating-point numbers are denoted with a - sign before X, Y, etc.

`CONJ(z)` and `CIS(z)` are defined as follows:

- `CONJ(x+i·y)=x-i·y`
- `CIS(y)=cos(y)+i·sin(y)`.

The special value tables show the result of the function for the z argument at the intersection of the RE(z) column and the i*IM(z) row. If the function raises an exception on the argument z, the lower part of this cell shows the raised exception and the VM Error Status. An empty cell indicates that this argument is normal and the result is defined mathematically.

12.2 VM Mathematical Functions

This section describes VM functions that compute values of mathematical functions on real and complex vector arguments.

Each function is introduced by its short name, a brief description of its purpose, and the calling sequence for each type of data, as well as a description of the input/output arguments.

The input range of parameters is equal to the mathematical range of the input data type, unless the function description specifies input threshold values, which mark off the precision overflow, as follows:

- `FLT_MAX` denotes the maximum number representable in single precision real data type
- `DBL_MAX` denotes the maximum number representable in double precision real data type

The following table lists available mathematical functions and associated data types.

Function	Data Types	Description
Arithmetic Functions		
add	s, d, c, z	Adds vector elements
sub	s, d, c, z	Subtracts vector elements
sqr	s, d	Squares vector elements
mul	s, d, c, z	Multiplies vector elements
mulbyconj	c, z	Multiplies elements of one vector by conjugated elements of the second vector
conj	c, z	Conjugates vector elements
abs	s, d, c, z	Computes the absolute value of vector elements
arg	c, z	Computes the argument of vector elements
linearfrac	s, d	Performs linear fraction transformation of vectors

continues on next page

Table 9 – continued from previous page

Function	Data Types	Description
fmod	s, d	Performs element by element computation of the modulus function of vector a with respect to vector b
remainder	s, d	Performs element by element computation of the remainder function on the elements of vector a and the corresponding elements of vector b
Power and Root Functions		
inv	s, d	Inverts vector elements
div	s, d, c, z	Divides elements of one vector by elements of the second vector
sqrt	s, d, c, z	Computes the square root of vector elements
invsqrt	s, d	Computes the inverse square root of vector elements
cbrt	s, d	Computes the cube root of vector elements
invcbrt	s, d	Computes the inverse cube root of vector elements
pow2o3	s, d	Computes the cube root of the square of each vector element
pow3o2	s, d	Computes the square root of the cube of each vector element
pow	s, d, c, z	Raises each vector element to the specified power
powx	s, d, c, z	Raises each vector element to the constant power
powr	s, d	Computes a to the power b for elements of two vectors, where the elements of vector argument a are all non-negative
hypot	s, d	Computes the square root of sum of squares
Exponential and Logarithmic Functions		
exp	s, d, c, z	Computes the base e exponential of vector elements
exp2	s, d	Computes the base 2 exponential of vector elements
exp10	s, d	Computes the base 10 exponential of vector elements
expm1	s, d	Computes the base e exponential of vector elements decreased by 1
ln	s, d, c, z	Computes the natural logarithm of vector elements
log2	s, d	Computes the base 2 logarithm of vector elements
log10	s, d, c, z	Computes the base 10 logarithm of vector elements
log1p	s, d	Computes the natural logarithm of vector elements that are increased by 1
logb	s, d	Computes the exponents of the elements of input vector a
Trigonometric Functions		

continues on next page

Table 9 – continued from previous page

Function	Data Types	Description
cos	s, d, c, z	Computes the cosine of vector elements
sin	s, d, c, z	Computes the sine of vector elements
sincos	s, d	Computes the sine and cosine of vector elements
cis	c, z	Computes the complex exponent of vector elements (cosine and sine combined to complex value)
tan	s, d, c, z	Computes the tangent of vector elements
acos	s, d, c, z	Computes the inverse cosine of vector elements
asin	s, d, c, z	Computes the inverse sine of vector elements
atan	s, d, c, z	Computes the inverse tangent of vector elements
atan2	s, d	Computes the four-quadrant inverse tangent of ratios of the elements of two vectors
cospi	s, d	Computes the cosine of vector elements multiplied by π
sinpi	s, d	Computes the sine of vector elements multiplied by π
tanpi	s, d	Computes the tangent of vector elements multiplied by π
acospi	s, d	Computes the inverse cosine of vector elements divided by π
asinpi	s, d	Computes the inverse sine of vector elements divided by π
atanpi	s, d	Computes the inverse tangent of vector elements divided by π
atan2pi	s, d	Computes the four-quadrant inverse tangent of the ratios of the corresponding elementss of two vectors divided by π
cosd	s, d	Computes the cosine of vector elements multiplied by $\pi/180$
sin	s, d	Computes the sine of vector elements multiplied by $\pi/180$
tand	s, d	Computes the tangent of vector elements multiplied by $\pi/180$
Hyperbolic Functions		
cosh	s, d, c, z	Computes the hyperbolic cosine of vector elements
sinh	s, d, c, z	Computes the hyperbolic sine of vector elements
tanh	s, d, c, z	Computes the hyperbolic tangent of vector elements
acosh	s, d, c, z	Computes the inverse hyperbolic cosine of vector elements
asinh	s, d, c, z	Computes the inverse hyperbolic sine of vector elements

continues on next page

Table 9 – continued from previous page

Function	Data Types	Description
atanh	s, d, c, z	Computes the inverse hyperbolic tangent of vector elements.
Special Functions		
erf	s, d	Computes the error function value of vector elements
erfc	s, d	Computes the complementary error function value of vector elements
cdfnorm	s, d	Computes the cumulative normal distribution function value of vector elements
erfinv	s, d	Computes the inverse error function value of vector elements
erfcinv	s, d	Computes the inverse complementary error function value of vector elements
cdfnorminv	s, d	Computes the inverse cumulative normal distribution function value of vector elements
lgamma	s, d	Computes the natural logarithm for the absolute value of the gamma function of vector elements
tgamma	s, d	Computes the gamma function of vector elements
expint1	s, d	Computes the exponential integral of vector elements
Rounding Functions		
floor	s, d	Rounds towards minus infinity
ceil	s, d	Rounds towards plus infinity
trunc	s, d	Rounds towards zero infinity
round	s, d	Rounds to nearest integer
nearbyint	s, d	Rounds according to current mode
rint	s, d	Rounds according to current mode and raising inexact result exception
modf	s, d	Computes the integer and fractional parts
frac	s, d	Computes the fractional part
Miscellaneous Functions		
copysign	s, d	Returns vector of elements of one argument with signs changed to match other argument elements
nextafter	s, d	Returns vector of elements containing the next representable floating-point values following the values from the elements of one vector in the direction of the corresponding elements of another vector
fdim	s, d	Returns vector containing the differences of the corresponding elements of the vector arguments if the first is larger and +0 otherwise
fmax	s, d	Returns the larger of each pair of elements of the two vector arguments

continues on next page

Table 9 – continued from previous page

Function	Data Types	Description
fmin	s, d	Returns the smaller of each pair of elements of the two vector arguments
maxmag	s, d	Returns the element with the larger magnitude between each pair of elements of the two vector arguments
minmag	s, d	Returns the element with the smaller magnitude between each pair of elements of the two vector arguments

12.2.1 Arithmetic Functions

Arithmetic functions perform the basic mathematical operations like addition, subtraction, multiplication or computation of the absolute value of the vector elements.

add

Computes the element-wise addition of vector a and vector b.

- Description
- API
- Examples

Description

The `add(a, b)` function performs element by element addition of vector a and vector b.

Argument 1	Argument 2	Result	Error Code
+0	+0	+0	
+0	-0	+0	
-0	+0	+0	
-0	-0	-0	
$+\infty$	$+\infty$	$+\infty$	
$+\infty$	$-\infty$	QNAN	
$-\infty$	$+\infty$	QNAN	
$-\infty$	$-\infty$	$-\infty$	
SNAN	any value	QNAN	
any value	SNAN	QNAN	

Specifications for special values of the complex functions are defined according to the following formula:

`add(x1+i*y1,x2+i*y2) = (x1+x2) + i*(y1+y2)`

Overflow in a complex function occurs (supported in the HA/LA accuracy modes only) when all RE(x), RE(y), IM(x), IM(y) arguments are finite numbers, but the real or imaginary part of the computed result is so large that it does not fit the target precision. In this case, the function returns ∞ in that part of the result, and sets the VM Error Status to `status::overflow` (overriding any possible `status::accuracy_warning` status).

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event add(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event add(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event add(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event add(sycl::queue & exec_queue,
     T const * a,
     oneapi::mkl::slice sa,
     T const * b,
     oneapi::mkl::slice sb,
     T * y,
     oneapi::mkl::slice sy,
     std::vector<sycl::event> const & depends = {},
     oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
     oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

add supports the following precisions and devices:

T	Devices supported
float	CPU and GPU
double	CPU and GPU
std::complex<float>	CPU and GPU
std::complex<double>	CPU and GPU

Description

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is mode::not_defined.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `add` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vadd.cpp
```

sub

Computes the element-wise subtraction of vector b from vector a.

Description

The `sub(a, b)` function performs element by element subtraction of vector a and vector b.

Argument 1	Argument 2	Result	Error Code
+0	+0	+0	
+0	-0	+0	
-0	+0	+0	
-0	-0	-0	
+∞	+∞	QNAN	
+∞	-∞	+∞	
-∞	+∞	-∞	
-∞	-∞	QNAN	
SNAN	any value	QNAN	
any value	SNAN	QNAN	

Specifications for special values of the complex functions are defined according to the following formula:

```
sub(x1+i*y1,x2+i*y2) = (x1-x2) + i*(y1-y2)
```

Overflow in a complex function occurs (supported in the HA/LA accuracy modes only) when all RE(x), RE(y), IM(x), IM(y) arguments are finite numbers, but the real or imaginary part of the computed result is so large that it does not fit the target precision. In this case, the function returns ∞ in that part of the result, and sets the VM Error Status to `status::overflow` (overriding any possible `status::accuracy_warning` status).

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event sub(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sub(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event sub(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sub(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`sub` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `sub` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vsub.cpp
```

sqr

Compute the square of vector elements.

Description

The `sqr()` function performs element by element squaring of the vector.

Argument	Result	Error Code
+0	+0	
-0	+0	
+∞	+∞	
-∞	+∞	
QNAN	QNAN	
SNAN	QNAN	

The `sqr` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event sqr(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event sqr(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event sqr(sycl::queue & exec_queue,
```

(continues on next page)

(continued from previous page)

```
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event sqr(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`sqr` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `sqr` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vsqr.cpp
```

mul

Computes the element-wise multiplication of vector a and vector b.

Description

The `mul(a, b)` function performs element by element multiplication of vector a and vector b.

Argument 1	Argument 2	Result	Error Code
+0	+0	+0	
+0	-0	-0	
-0	+0	-0	
-0	-0	+0	
+0	+∞	QNAN	
+0	-∞	QNAN	
-0	+∞	QNAN	
-0	-∞	QNAN	
+∞	+0	QNAN	
+∞	-0	QNAN	
-∞	+0	QNAN	
-∞	-0	QNAN	
+∞	+∞	+∞	
+∞	-∞	-∞	
-∞	+∞	-∞	
-∞	-∞	+∞	
SNAN	any value	QNAN	
any value	SNAN	QNAN	
QNAN	non-SNAN	QNAN	
non-SNAN	QNAN	QNAN	

Specifications for special values of the complex functions are defined according to the following formula:

```
mul(x1+i*y1,x2+i*y2) = (x1*x2-y1*y2) + i*(x1*y2+y1*x2)
```

Overflow in a complex function occurs (supported in the HA/LA accuracy modes only) when all RE(x), RE(y), IM(x), IM(y) arguments are finite numbers, but the real or imaginary part of the computed result is so large that it does not fit the target precision. In this case, the function returns ∞ in that part of the result, and sets the VM Error Status to `status::overflow` (overriding any possible `status::accuracy_warning` status).

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event mul(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event mul(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event mul(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event mul(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`mul` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `mul` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vmul.cpp
```

mulbyconj

Description

Computes the element-wise multiplication of vector a elements and the complex conjugate of vector b elements.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event mulbyconj(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event mulbyconj(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event mulbyconj(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event mulbyconj(sycl::queue & exec_queue,
      T const * a,
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`mulbyconj` supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `mulbyconj` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vmulbyconj.cpp
```

conj

Computes the complex conjugate of vector elements.

Description

The `conj` function performs element by element conjugation of the vector.

No special values are specified. The `conj` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event conj(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event conj(sycl::queue & exec_queue,
     sycl::buffer<T> & a,
     oneapi::mkl::slice sa,
     sycl::buffer<T> & y,
     oneapi::mkl::slice sy,
     oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event conj(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event conj(sycl::queue & exec_queue,
```

(continues on next page)

(continued from previous page)

```
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`conj` supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `conj` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vconj.cpp
```

abs

Computes absolute value of vector elements.

Description

The `abs(a)` function computes an absolute value of vector elements.

Argument	Result	Error Code
+0	+0	
-0	+0	
$+\infty$	$+\infty$	
$-\infty$	$+\infty$	
QNAN	QNAN	
SNAN	QNAN	

Specifications for special values of the complex functions are defined according to the following formula `abs(a) = hypot(RE(a),IM(a))`.

The `abs` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event abs(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event abs(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event abs(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event abs(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`abs` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use abs can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vabs.cpp
```

arg

Computes the argument of vector elements in [-pi, pi].

Description

The arg(a) function computes argument of vector elements.

See Special Value Notations for the conventions used in the table below.

RE(a) i · IM(a)	-∞	-X	-0	+0	+X	+∞	NAN
+i·∞	+3·π/4	+π/2	+π/2	+π/2	+π/2	+π/4	NAN
+i·Y	+π		+π/2	+π/2		+0	NAN
+i·0	+π	+π	+π	+0	+0	+0	NAN
-i·0	-π	-π	-π	-0	-0	-0	NAN
-i·Y	-π		-π/2	-π/2		-0	NAN
-i·∞	-3·π/4	-π/2	-π/2	-π/2	-π/2	-π/4	NAN
+i·NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN

Note: arg(a)=atan2(IM(a),RE(a))

The arg function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event arg(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event arg(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event arg(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event arg(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`arg` supports the following precisions and devices:

T	Devices supported
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `arg` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/varg.cpp
```

linearfrac

Computes the element-wise linear fractional transformation of vectors a and b with scalar parameters.

Description

The `linearfrac(a,b,scalea,shifta,scaleb,shiftb)` function performs a linear fraction transformation of vector a by vector b with scalar parameters: scaling multipliers scalea, scaleb and shifting addends shifta, shiftb:

`y[i]=(scalea·a[i]+shifta)/(scaleb·b[i]+shiftb), i=1,2 ... n`

The `linearfrac` function is implemented in the EP accuracy mode only, therefore no special values are defined for this function. If used in HA or LA mode, `linearfrac` sets the VM Error Status to `status::accuracy_warning`. Correctness is guaranteed within the threshold limitations defined for each input parameter (see the table below); otherwise, the behavior is unspecified.

Threshold Limitations on Input Parameters
2EMIN/2≤ |scalea| ≤ 2(EMAX-2)/2
2EMIN/2≤ |scaleb| ≤ 2(EMAX-2)/2
|shifta| ≤ 2EMAX-2
|shiftb| ≤ 2EMAX-2
2EMIN/2≤a[i] ≤ 2(EMAX-2)/2
2EMIN/2≤b[i] ≤ 2(EMAX-2)/2
a[i] ≠ - (shifta/scalea)*(1-δ1), |δ1| ≤ 21-(p-1)/2
b[i] ≠ - (shiftb/scaleb)*(1-δ2), |δ2| ≤ 21-(p-1)/2

EMIN and EMAX are the minimum and maximum exponents and p is the number of significant bits (precision) for the corresponding data type according to the ANSI/IEEE Standard 754-2008 [IEEE754]:

- for single precisionEMIN = -126, EMAX = 127, p = 24
- for double precisionEMIN = -1022, EMAX = 1023, p = 53

The thresholds become less strict for common cases with scalea=0 and/or scaleb=0:

- ifscalea=0, there are no limitations for the values of `a[i]` and `shifta`.
- ifscaleb=0, there are no limitations for the values of `b[i]` and `shiftb`.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event linearfrac(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      T scalea,
      T shifta,
      T scaleb,
      T shiftb,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event linearfrac(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      T scalea,
      T shifta,
      T scaleb,
      T shiftb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event linearfrac(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T scalea,
      T shifta,
      T scaleb,
```

(continues on next page)

(continued from previous page)

```
      T shiftb,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event linearfrac(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T scalea,
      T shifta,
      T scaleb,
      T shiftb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`linearfrac` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

scalea Constant value for scaling multipliers of vector a

shifta Constant value for shifting addend of vector a

scaleb Constant value for scaling multipliers of vector b

shiftb Constant value for shifting addend of vector b

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

scalea Constant value for scaling multipliers of vector a

shifta Constant value for shifting addend of vector a

scaleb Constant value for scaling multipliers of vector b

shiftb Constant value for shifting addend of vector b

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `linearfrac` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vllinearfrac.cpp
```

fmod

Computes the element-wise remainder of vector a elements divided by vector b elements that has the same sign as vector a elements.

Description

The `fmod (a,b)` function computes the modulus function of each element of vector a, with respect to the corresponding elements of vector b:

```
ai - bi*trunc(ai/bi)
```

In general, the modulus function `fmod (ai, bi)` returns the value `ai - n*bi` for some integer n such that if b_i is nonzero, the result has the same sign as a_i and a magnitude less than the magnitude of b_i.

Argument 1	Argument 2	Result	Error Code
a not NAN	±0	NAN	`status::sing`
±∞	b not NAN	NAN	`status::sing`
±0	b≠ 0, not NAN	±0	
a finite	±∞	a	
NAN	b		
a	NAN	NAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event fmod(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event fmod(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event fmod(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event fmod(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`fmod` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready).

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector of size `n`.

return value (event) Computation end event.

USM API

y Pointer to the output vector of size `n`.

return value (event) Computation end event.

Examples

An example of how to use `fmod` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vfmod.cpp
```

remainder

Computes the element-wise remainder of vector a elements divided by vector b elements, that is nearest to zero.

Description

The `remainder(a)` function computes the remainder of each element of vector a, with respect to the corresponding elements of vector b: compute the values of n such that

```
n = ai - n*bi
```

where n is the integer nearest to the exact value of a_i/b_i. If two integers are equally close to a_i/b_i, n is the even one. If n is zero, it has the same sign as a_i.

Argument 1	Argument 2	Result	VM Error Status
a not NAN	±0	NAN	status::errdom
±∞	b not NAN	NAN	
±0	b≠ 0, not NAN	±0	
a finite	±∞	a	
NAN	b	NAN	
a	NAN	NAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event remainder(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event remainder(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event remainder(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
```

(continues on next page)

(continued from previous page)

```
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event remainder(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`remainder` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `remainder` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vremainder.cpp
```

12.2.2 Power and Root Functions

inv

Computes the element-wise multiplicative inverse (or reciprocal) of vector elements.

Description

The `inv(a)` function performs element by element inversion of the vector.

Argument	Result	VM Error Status
+0	+∞	`status::sing`
-0	-∞	`status::sing`
+∞	+0	
-∞	-0	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event inv(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event inv(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event inv(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event inv(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`inv` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `inv` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vinv.cpp
```

div

Computes the element-wise division of vector a elements by vector b elements.

Description

The `div(a,b)` function performs element by element division of vector a by vector b.

Argument 1	Argument 2	Result	VM Error Status
X > +0	+0	+∞	`status::sing`
X > +0	-0	-∞	`status::sing`
X < +0	+0	-∞	`status::sing`
X < +0	-0	+∞	`status::sing`
+0	+0	QNAN	`status::sing`
-0	-0	QNAN	`status::sing`
X > +0	+∞	+0	
X > +0	-∞	-0	
+∞	+∞	QNAN	
-∞	-∞	QNAN	
QNAN	QNAN	QNAN	
SNAN	SNAN	QNAN	

Specifications for special values of the complex functions are defined according to the following formula

`Div(x1+i*y1,x2+i*y2) = (x1+i*y1)*(x2-i*y2)/(x2*x2+y2*y2).`

Overflow in a complex function occurs when `x2+i*y2` is not zero, x1, x2, y1, y2 are finite numbers, but the real or imaginary part of the exact result is so large that it does not fit the target precision. In that case, the function returns ∞ in that part of the result, and sets the VM Error Status to status::overflow.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event div(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

  }
```

```
namespace oneapi::mkl::vm {

sycl::event div(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event div(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event div(sycl::queue & exec_queue,
      T const * a,
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`div` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `div` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vdiv.cpp
```

sqrt

Computes the element-wise square root of vector elements.

Description

The `sqrt` function computes a square root of vector elements.

Argument	Result	VM Error Status
a< +0	QNAN	`status::errdom`
+0	+0	
-0	-0	
$-\infty$	QNAN	`status::errdom`
$+\infty$	$+\infty$	
QNAN	QNAN	
SNAN	QNAN	

+i·∞	+∞infty‘	+∞infty‘	+∞infty‘	+∞infty‘	+∞infty‘	+∞infty‘	+∞infty‘
+i·Y	+0+i·∞					$+\infty + i\cdot 0$	
+i·0	+0+i·∞		+0+i·0	+0+i·0		$+\infty + i\cdot 0$	
-i·0	+0-i·∞		+0-i·0	+0-i·0		+∞-i·0	
-i·Y	+0-i·∞					+∞-i·0	
-i·∞	+∞-i·∞	+∞-i·∞	+∞-i·∞	+∞-i·∞	+∞-i·∞	+∞-i·∞	+∞-i·∞
+i·NAN							

Note: `Sqrt(CONJ(z))=CONJ(Sqrt(z))`.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event sqrt(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sqrt(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event sqrt(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sqrt(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`sqrt` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `sqrt` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vsqrt.cpp
```

invsqrt

Computes the element-wise inverse square root of vector elements.

Description

The `invsqrt(a)` function computes an inverse square root of vector elements.

Argument	Result	VM Error Status
a < +0	QNAN	`status::errdom`
+0	+∞	`status::sing`
-0	-∞	`status::sing`
-∞	QNAN	`status::errdom`
+∞	+0	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event invsqrt(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event invsqrt(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event invsqrt(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event invsqrt(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`invsqrt` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `invsqrt` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vinvsqrt.cpp
```

cbrt

Computes the element-wise cube root of vector elements.

Description

The `cbrt(a)` function computes a cube root of vector elements.

Argument	Result	Error Code
+0	+0	
-0	-0	
+∞	+∞	
-∞	-∞	
QNAN	QNAN	
SNAN	QNAN	
+0	+0	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event cbrt(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event cbrt(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event cbrt(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

  }
```

```
namespace oneapi::mkl::vm {

sycl::event cbrt(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

  }
```

`cbrt` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Pointer to the input vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `cbrt` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vcbrt.cpp
```

invcbrt

Computes the element-wise inverse cube root of vector elements.

Description

The `invcbrt(a)` function computes an inverse cube root of vector elements.

Argument	Result	Error Code
+0	+∞	`status::sing`
-0	-∞	`status::sing`
+∞	+0	
-∞	-0	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event invcbrt(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event invcbrt(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event invcbrt(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event invcbrt(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`invcbrt` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `invcbrt` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vinvcbrt.cpp
```

pow2o3

Computes the element-wise cube root of the square of vector elements.

Description

The `pow2o3(a)` function computes the cube root of the square of each vector element.

Argument	Result	Error Code
+0	+0	
-0	+0	
$+\infty$	$+\infty$	
$-\infty$	$+\infty$	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event pow2o3(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event pow2o3(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event pow2o3(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event pow2o3(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

pow2o3 supports the following precisions and devices:

T	Devices supported
float	CPU and GPU
double	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `pow2o3` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vpow203.cpp
```

pow3o2

Computes the element-wise square root of the cube of vector elements.

Description

The `pow3o2(a)` function computes the square root of the cube of each vector element.

Data Type	Threshold Limitations on Input Parameters
single precision	$\|a_i\| < (FLT\_MAX)^{2/3}$
double precision	$\|a_i\| < (FLT\_MAX)^{2/3}$

Argument	Result	VM Error Status
a < +0	QNAN	`status::errdom`
+0	+0	
-0	-0	
-∞	QNAN	`status::errdom`
+∞	+∞	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event pow3o2(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event pow3o2(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event pow3o2(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event pow3o2(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

pow3o2 supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `pow3o2` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vpow3o2.cpp
```

pow

Computes the element-wise exponentiation of vector a elements raised to the power of vector b elements.

Description

The `pow(a,b)` function computes a to the power b for elements of two vectors.

The real function pow has certain limitations on the input range of a and b parameters. Specifically, if a[i] is positive, then b[i] may be arbitrary. For negative a[i], the value of b[i] must be an integer (either positive or negative).

The complex function pow has no input range limitations.

Argument 1	Argument 2	Result	Error Code
+0	neg. odd integer	$+\infty$	`status::errdom`
-0	neg. odd integer	$-\infty$	`status::errdom`
+0	neg. even integer	$+\infty$	`status::errdom`
-0	neg. even integer	$+\infty$	`status::errdom`
+0	neg. non-integer	$+\infty$	`status::errdom`
-0	neg. non-integer	$+\infty$	`status::errdom`

continues on next page

Table 11 – continued from previous page

Argument 1	Argument 2	Result	Error Code
-0	pos. odd integer	+0	
-0	pos. odd integer	-0	
+0	pos. even integer	+0	
-0	pos. even integer	+0	
+0	pos. non-integer	+0	
-0	pos. non-integer	+0	
-1	+∞	+1	
-1	-∞	+1	
+1	any value	+1	
+1	+0	+1	
+1	-0	+1	
+1	+∞	+1	
+1	-∞	+1	
+1	QNAN	+1	
any value	+0	+1	
+0	+0	+1	
-0	+0	+1	
+∞	+0	+1	
-∞	+0	+1	
QNAN	+0	+1	
any value	-0	+1	
+0	-0	+1	
-0	-0	+1	
+∞	-0	+1	
-∞	-0	+1	
QNAN	-0	+1	
a < +0	non-integer	QNAN	`status::errdom`
|a| < 1	-∞	+∞	
+0	-∞	+∞	`status::errdom`
-0	-∞	+∞	`status::errdom`
|a| > 1	-∞	+0	
+∞	-∞	+0	
-∞	-∞	+0	
|a| < 1	+∞	+0	
+0	+∞	+0	
-0	+∞	+0	
|a| > 1	+∞	+∞	
+∞	+∞	+∞	
-∞	+∞	+∞	
-∞	neg. odd integer	-0	
-∞	neg. even integer	+0	
-∞	neg. non-integer	+0	

continues on next page

Table 11 – continued from previous page

Argument 1	Argument 2	Result	Error Code
-∞	pos. odd integer	-∞	
-∞	pos. even integer	+∞	
-∞	pos. non-integer	+∞	
+∞	b < +0	+0	
+∞	b > +0	+∞	
Big finite value*	Big finite value*	+/-∞	`status::overflow`
QNAN	QNAN	QNAN	
QNAN	SNAN	QNAN	
SNAN	QNAN	QNAN	
SNAN	SNAN	QNAN	

* Overflow in a real function is supported only in the HA/LA accuracy modes. The overflow occurs when x and y are finite numbers, but the result is too large to fit the target precision. In this case, the function:

1. Returns ∞ in the result.
2. Sets the VM Error Status to status::overflow.

Overflow in a complex function occurs (supported in the HA/LA accuracy modes only) when all RE(x), RE(y), IM(x), IM(y) arguments are finite numbers, but the real or imaginary part of the computed result is so large that it does not fit the target precision. In this case, the function returns ∞ in that part of the result, and sets the VM Error Status to `status::overflow` (overriding any possible `status::accuracy_warning` status).

The complex double precision versions of this function are implemented in the EP accuracy mode only. If used in HA or LA mode, the functions set the VM Error Status to `status::accuracy_warning`.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event pow(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event pow(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event pow(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event pow(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`pow` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `pow` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vpow.cpp
```

powx

Computes the element-wise exponentiation of vector a elements raised to the power of scalar b.

Description

The `powx` function computes a to the power b for a vector a and a scalar b.

The real function `powx` has certain limitations on the input range of a and b parameters. Specifically, if a[i] is positive, then b may be arbitrary. For negative a[i], the value of b must be an integer (either positive or negative).

The complex function `powx` has no input range limitations.

Special values and VM Error Status treatment are the same as for the `pow` function.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event powx(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      T b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event powx(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      T b,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event powx(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event powx(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T b,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});
```

(continues on next page)

(continued from previous page)

```
}
```

`powx` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Fixed value of power.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Fixed value of power.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `powx` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vpowx.cpp
```

powr

Computes the element-wise exponentiation of vector a elements raised to the power of vector b elements, where the elements of vector a are all non-negative.

Description

The `powr(a,b)` function raises each element of vector a by the corresponding element of vector b. The elements of a are all nonnegative ($a_i \geq 0$).

Data Type	Threshold Limitations on Input Parameters
single precision	$a_i < (FLT\_MAX)^{1/b_i}$
double precision	$a_i < (DBL\_MAX)^{1/b_i}$

Special values and VM Error Status treatment for `v?Powr` function are the same as for `pow`, unless otherwise indicated in this table:

Argument 1	Argument 2	Result	Error Code
a < 0	any value b	NAN	`status::errdom`
0 < a < ∞	±0	1	
±0	-∞ < b < 0	+∞	
±0	-∞	+∞	
±0	b > 0	+0	
1	-∞ < b < ∞	1	
±0	±0	NAN	
+∞	±0	NAN	
1	+∞	NAN	
a≥ 0	NAN	NAN	
NAN	any value b	NAN	
0 < a <1	-∞	+∞	
a > 1	-∞	+0	
0 ≤a < 1	+∞	+0	
a > 1	+∞	+∞	
+∞	b < +0	+0	
+∞	b > +0	+∞	
QNAN	QNAN	QNAN	`status::errdom`
QNAN	SNAN	QNAN	`status::errdom`
SNAN	QNAN	QNAN	`status::errdom`
SNAN	SNAN	QNAN	`status::errdom`

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event powr(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event powr(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event powr(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event powr(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`powr` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `powr` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vpowr.cpp
```

hypot

Computes the element-wise square root of the sum of the squares of vector a elements and vector b elements.

Description

The `hypot(a,b)` function computes a square root of sum of two squared elements.

Argument 1	Argument 2	Result	Error Code
+0	+0	+0	
-0	-0	+0	
+∞	any value	+∞	
any value	+∞	+∞	
SNAN	any value	QNAN	INVALID
any value	SNAN	QNAN	INVALID
QNAN	any value	QNAN	
any value	QNAN	QNAN	

Data Type	Threshold Limitations on Input Parameters
single precision	`abs(a[i]) < sqrt(FLT_MAX) abs(b[i]) < sqrt(FLT_MAX)`
double precision	`abs(a[i]) < sqrt(DBL_MAX) abs(b[i]) < sqrt(DBL_MAX)`

The `hypot(a,b)` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event hypot(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event hypot(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event hypot(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event hypot(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`hypot` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `hypot` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vhypot.cpp
```

12.2.3 Exponential and Logarithmic Functions

exp

Computes the element-wise natural (base-e) exponential of vector elements.

Description

The `exp(a)` function computes an exponential of vector elements.

Data Type	Threshold Limitations on Input Parameters
single precision	`a[i] < Log( FLT_MAX )`
double precision	`a[i] < Log( DBL_MAX )`

Argument	Result	Error Code
+0	+1	
-0	+1	
a > overflow	+∞	`status::overflow`
a < underflow	+0	`status::overflow`
+∞	+∞	
-∞	+0	
QNAN	QNAN	
SNAN	QNAN	

+i·∞							
+i·Y							
+i·0							
-i·0							
-i·Y							
-i·∞							
+i·NAN							

Note:

- The complex exp(z) function sets the VM Error Status to status::overflow in the case of overflow, that is, when both RE(z) and IM(z) are finite non-zero numbers, but the real or imaginary part of the exact result is so large that it does not meet the target precision.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event exp(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event exp(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event exp(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event exp(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

exp supports the following precisions and devices:

T	Devices supported
float	CPU and GPU
double	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `exp` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vexp.cpp
```

exp2

Computes the element-wise base-2 exponential of vector elements.

Description

The `exp2` function computes the base 2 exponential of vector elements.

Data Type	Threshold Limitations on Input Parameters
single precision	$a_i < \log_2(\text{FLT\_MAX})$
double precision	$a_i < \log_2(\text{DBL\_MAX})$

Argument	Result	Error Code
+0	+1	
-0	+1	
a > overflow	$+\infty$	`status::overflow`
a < underflow	+0	`status::underflow`
$+\infty$	$+\infty$	
$-\infty$	+0	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event exp2(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event exp2(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event exp2(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event exp2(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

exp2 supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `exp2` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vexp2.cpp
```

exp10

Computes the element-wise base-10 exponential of vector elements.

Description

The `exp10(a)` function computes the base 10 exponential of vector elements.

Data Type	Threshold Limitations on Input Parameters
single precision	$a_i < \log_{10}(\text{FLT\_MAX})$
double precision	$a_i < \log_{10}(\text{DBL\_MAX})$

Argument	Result	VM Error Status
+0	+1	
-0	+1	
a > overflow	$+\infty$	`status::overflow`
a < underflow	+0	`status::underflow`
$+\infty$	$+\infty$	
$-\infty$	+0	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event exp10(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event exp10(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event exp10(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event exp10(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`exp10` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `exp10` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vexp10.cpp
```

expm1

Computes the element-wise subtraction of 1 from the exponential of vector elements.

Description

The `expm1(a)` function computes an exponential of vector elements decreased by 1.

Argument	Result	Error Code
+0	+1	
-0	+1	
a > overflow	+∞	`status::overflow`
+∞	+∞	
-∞	-0	
QNAN	QNAN	
SNAN	QNAN	

Data Type	Threshold Limitations on Input Parameters
single precision	`a[i] < Log( FLT_MAX )`
double precision	`a[i] < Log( DBL_MAX )`

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event expm1(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event expm1(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event expm1(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event expm1(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`expm1` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `expm1` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vexpm1.cpp
```

ln

Computes the element-wise natural logarithm of vector elements.

Description

The `ln(a)` function computes natural logarithm of vector elements.

Argument	Result	Error Code
+1	+0	
a <+0	QNAN	`status::errdom`
+0	-∞	`status::sing`
-0	-∞	`status::sing`
-∞	QNAN	`status::errdom`
+∞	+∞	
QNAN	QNAN	
SNAN	QNAN	

RE(a) i · IM(a)	$-\infty$	$-X$	-0	$+0$	$+X$	$+\infty$	NAN
$+i\cdot\infty$	$+\infty + i\cdot\frac{3\pi}{4}$	$+\infty + i\cdot\frac{\pi}{2}$	$+\infty + i\cdot\frac{\pi}{2}$	$+\infty + i\cdot\frac{\pi}{2}$	$+\infty + i\cdot\frac{\pi}{2}$	$+\infty + i\cdot\frac{\pi}{4}$	$+\infty + i\cdot QNAN$
$+i\cdot Y$	$+\infty - i\cdot\pi$					$+\infty + i\cdot 0$	$QNAN + i\cdot QNAN$
$+i\cdot 0$	$+\infty - i\cdot\pi$		$+\infty + i\cdot\pi$	$-\infty - i\cdot 0$		$+\infty + i\cdot 0$	$QNAN + i\cdot QNAN$
$-i\cdot 0$	$+\infty - i\cdot\pi$		$-\infty + i\cdot\pi$	$-\infty - i\cdot 0$		$+\infty - i\cdot 0$	$QNAN + i\cdot QNAN$
$-i\cdot Y$	$+\infty - i\cdot\pi$					$+\infty - i\cdot 0$	$QNAN + i\cdot QNAN$
$-i\cdot\infty$	$+\infty - i\cdot\frac{3\pi}{4}$	$+\infty - i\cdot\frac{\pi}{2}$	$+\infty - i\cdot\frac{\pi}{2}$	$+\infty - i\cdot\frac{\pi}{2}$	$+\infty - i\cdot\frac{\pi}{2}$	$+\infty - i\cdot\frac{\pi}{4}$	$+\infty + i\cdot QNAN$
$+i\cdot NAN$	$+\infty + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$+\infty + i\cdot QNAN$	$QNAN + i\cdot QNAN$

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event ln(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event ln(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event ln(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event ln(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`ln` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event..

Examples

An example of how to use `ln` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vln.cpp
```

log2

Computes the element-wise base-2 logarithm of vector elements.

Description

The `log2(a)` function computes the base 2 logarithm of vector elements.

Argument	Result	Error Code
+1	+0	
a < +0	QNAN	`status::errdom`
+0	-∞	`status::sing`
-0	-∞	`status::sing`
-∞	QNAN	`status::errdom`
+∞	+∞	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event log2(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event log2(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event log2(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event log2(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`log2` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event..

Examples

An example of how to use `log2` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vlog2.cpp
```

log10

Computes the element-wise base-10 logarithm of vector elements.

Description

The `log10(a)` function computes the base 10 logarithm of vector elements.

Argument	Result	Error Code
+1	+0	
a <+0	QNAN	`status::errdom`
+0	$-\infty$	`status::sing`
-0	$-\infty$	`status::sing`
$-\infty$	QNAN	`status::errdom`
$+\infty$	$+\infty$	
QNAN	QNAN	
SNAN	QNAN	

RE(a) i · IM(a)	$-\infty$	$-X$	-0	$+0$	$+X$	$+\infty$	NAN
$+i\cdot\infty$	$+\infty + i\frac{3}{4}\frac{\pi}{\ln(10)}$	$+\infty + i\frac{\pi}{2}\frac{1}{\ln(10)}$	$+\infty + i\frac{\pi}{2}\frac{1}{\ln(10)}$	$+\infty + i\frac{\pi}{2}\frac{1}{\ln(10)}$	$+\infty + i\frac{\pi}{2}\frac{1}{\ln(10)}$	$+\infty + i\frac{\pi}{4}\frac{1}{\ln(10)}$	$+\infty + i\cdot QNAN$
$+i\cdot Y$	$+\infty + i\frac{\pi}{\ln(10)}$					$+\infty + i\cdot 0$	$QNAN + i\cdot QNAN$
$+i\cdot 0$	$+\infty + i\frac{\pi}{\ln(10)}$		$-\infty + i\frac{\pi}{\ln(10)}$	$-\infty + i\cdot 0$		$+\infty + i\cdot 0$	$QNAN + i\cdot QNAN$
$-i\cdot 0$	$+\infty - i\frac{\pi}{\ln(10)}$		$-\infty - i\frac{\pi}{\ln(10)}$	$-\infty - i\cdot 0$		$+\infty - i\cdot 0$	$QNAN - i\cdot QNAN$
$-i\cdot Y$	$+\infty - i\frac{\pi}{\ln(10)}$					$+\infty - i\cdot 0$	$QNAN + i\cdot QNAN$
$-i\cdot\infty$	$+\infty + i\frac{3}{4}\frac{\pi}{\ln(10)}$	$+\infty - i\frac{\pi}{2}\frac{1}{\ln(10)}$	$+\infty - i\frac{\pi}{2}\frac{1}{\ln(10)}$	$+\infty - i\frac{\pi}{2}\frac{1}{\ln(10)}$	$+\infty - i\frac{\pi}{2}\frac{1}{\ln(10)}$	$+\infty - i\frac{\pi}{4}\frac{1}{\ln(10)}$	$+\infty + i\cdot QNAN$
$+i\cdot NAN$	$+\infty + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$+\infty + i\cdot QNAN$	$QNAN + i\cdot QNAN$

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event log10(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event log10(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event log10(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event log10(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`log10` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU
`std::complex<float>`	CPU and GPU
`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `log10` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vlog10.cpp
```

log1p

Computes the element-wise natural logarithm of 1 plus vector elements.

Description

The `log1p(a)` function computes a natural logarithm of vector elements that are increased by 1.

Argument	Result	VM Error Status
-1	-∞	`status::sing`
a <-1	QNAN	`status::errdom`
+0	+0	
-0	-0	
-∞	QNAN	`status::errdom`
+∞	+∞	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event log1p(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event log1p(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event log1p(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event log1p(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`log1p` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `log1p` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vlog1p.cpp
```

logb

Computes the element-wise exponents of vector elements.

Description

The `logb(a)` function computes the exponents of the elements of the input vector a. For each element a_i of vector a, this is the integral part of $\log_2|a_i|$. The returned value is exact and is independent of the current rounding direction mode.

Argument	Result	VM Error Status
+0	+∞	`status::errdom`
-0	-∞	`status::errdom`
-∞	+∞	
+∞	+∞	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event logb(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event logb(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event logb(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event logb(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`logb` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `logb` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vlogb.cpp
```

12.2.4 Trigonometric Functions

cos

Computes the element-wise cosine of vector elements.

Description

The `cos(a)` function computes cosine of vector elements.

Note that arguments `abs(a[i]) ≤ 213` and `abs(a[i]) ≤ 216` for single and double precisions, respectively, are called fast computational path. These are trigonometric function arguments for which VM provides the best possible performance. Avoid arguments that do not belong to the fast computational path in the VM High Accuracy (HA) and Low Accuracy (LA) functions. Alternatively, you can use VM Enhanced Performance (EP) functions that are fast on the entire function domain. However, these functions provide less accuracy.

Argument	Result	VM Error Status
+0	+1	
-0	+1	
+∞	QNAN	status::errdom
-∞	QNAN	status::errdom
QNAN	QNAN	
SNAN	QNAN	

Specifications for special values of the complex functions are defined according to the following formula

`Cos(z) = Cosh(i*z)`.

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event cos(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cos(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event cos(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cos(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`cos` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `cos` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vcos.cpp
```

sin

Computes the element-wise sine of vector elements.

Description

The `sin(a)` function computes sine of vector elements.

Note that arguments `abs(a[i]) ≤ 213` and `abs(a[i]) ≤ 216` for single and double precisions respectively are called fast computational path. These are trigonometric function arguments for which VM provides the best possible performance. Avoid arguments that do not belong to the fast computational path in the VM High Accuracy (HA) and Low Accuracy (LA) functions. Alternatively, you can use VM Enhanced Performance (EP) functions that are fast on the entire function domain. However, these functions provide less accuracy.

Argument	Result	VM Error Status
+0	+0	
-0	-0	
+∞	QNAN	`status::errdom`
-∞	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

Specifications for special values of the complex functions are defined according to the following formula:

`Sin(z) = -i*Sinh(i*z).`

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event sin(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sin(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event sin(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sin(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`sin` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event

USM API

y Pointer to the output vector.

return value (event) Computation end event

Examples

An example of how to use `sin` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vsin.cpp
```

sincos

Computes the element-wise sine and cosine of vector elements.

Description

The `sincos(a)` function computes sine and cosine of vector elements.

Note that arguments `abs(a[i]) ≤ 213` and `abs(a[i]) ≤ 216` for single and double precisions respectively are called fast computational path. These are trigonometric function arguments for which VM provides the best possible performance. Avoid arguments that do not belong to the fast computational path in the VM High Accuracy (HA) and Low Accuracy (LA) functions. Alternatively, you can use VM Enhanced Performance (EP) functions that are fast on the entire function domain. However, these functions provide less accuracy.

Argument	Result 1	Result 2	Error Code
+0	+0	+1	
-0	-0	+1	
+∞	QNAN	QNAN	`status::errdom`
-∞	QNAN	QNAN	`status::errdom`
QNAN	QNAN	QNAN	
SNAN	QNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event sincos(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      sycl::buffer<T> & z,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sincos(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      sycl::buffer<T> & z,
      oneapi::mkl::slice sz,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event sincos(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      T * z,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sincos(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      T * z,
      oneapi::mkl::slice sz,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`sincos` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

sz Slice selector for z. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

sz Slice selector for z. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output sine vector.

z The buffer containing the output cosine vector.

return value (event) Computation end event

USM API

y Pointer to the output sine vector.

z Pointer to the output cosine vector.

return value (event) Computation end event

Examples

An example of how to use `sincos` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vsincos.cpp
```

cis

Computes the element-wise complex exponential of vector elements.

Description

The `cis(a)` function computes complex exponent of real vector elements (cosine and sine of real vector elements combined to complex value).

Argument	Result	Error Code
▪ 0	+1+i·0	
▪ 0	+1-i·0	
+∞	QNAN+i·QNAN	`status::errdom`
-∞	QNAN+i·QNAN	`status::errdom`
QNAN	QNAN+i·QNAN	
SNAN	QNAN+i·QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event cis(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cis(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event cis(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cis(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`cis` supports the following precisions and devices:

T	R	Devices supported
`float`	`std::complex<float>`	CPU and GPU
`double`	`std::complex<double>`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a

The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `cis` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vcis.cpp
```

tan

Computes the element-wise tangent of vector elements.

Description

The `tan(a)` function computes tangent of vector elements.

Note that arguments `abs(a[i]) ≤ 213` and `abs(a[i]) ≤ 216` for single and double precisions respectively are called fast computational path. These are trigonometric function arguments for which VM provides the best possible performance. Avoid arguments that do not belong to the fast computational path in the VM High Accuracy (HA) and Low Accuracy (LA) functions. Alternatively, you can use VM Enhanced Performance (EP) functions that are fast on the entire function domain. However, these functions provide less accuracy.

Argument	Result	Error Code
+0	+0	
-0	-0	
+∞	QNAN	`status::errdom`
-∞	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

Specifications for special values of the complex functions are defined according to the following formula:

```
Tan(z) = -i*Tanh(i*z)
```

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event tan(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event tan(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event tan(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event tan(sycl::queue & exec_queue,
```

(continues on next page)

(continued from previous page)

```
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`tan` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event

USM API

y Pointer to the output vector.

return value (event) Computation end event

Examples

An example of how to use `tan` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vtan.cpp
```

acos

Computes the element-wise arccosine of vector elements.

Description

The `acos(a)` function computes inverse cosine of vector elements.

Argument	Result	Error Code
+0	+π/2	
-0	+π/2	
+1	+0	
-1	+π	
|a| >1	QNAN	`status::errdom`
+∞	QNAN	`status::errdom`
-∞	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

RE(a) i · IM(a)	$-\infty$	-X	-0	+0	+X	$+\infty$	NAN
$+i\cdot\infty$	+3·π/4-i·∞	+π/2-i·∞	+π/2-i·∞	+π/2-i·∞	+π/2-i·∞	+π/4-i·∞	QNAN-i·∞
+i·Y	+π-i·∞					+0-i·∞	$QNAN + i\cdot QNAN$
+i·0	+π-i·∞		+π/2-i·0	+π/2-i·0		+0-i·∞	$QNAN + i\cdot QNAN$
-i·0	+π+i·∞		+π/2+i·0	+π/2+i·0		+0+i·$infty$	$QNAN + i\cdot QNAN$
-i·Y	+π+i·∞					+0+i·∞	$QNAN + i\cdot QNAN$
-i·∞	+3π/4+i·∞	+π/2+i·∞	+π/2+i·∞	+π/2+i·∞	+π/2+i·∞	+π/4+i·∞	$QNAN + i\cdot\infty$
$+i\cdot NAN$	$QNAN + i\cdot\infty$	$QNAN + i\cdot QNAN$	$+\pi/2 + i\cdot QNAN$	$+\pi/2 + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN + i\cdot\infty$	$QNAN + i\cdot QNAN$

Note: `acos(CONJ(a))=CONJ(acos(a))`.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event acos(sycl::queue & exec_queue,
      std::int64_t n,
```

(continues on next page)

(continued from previous page)

```
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event acos(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event acos(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event acos(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`acos` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `acos` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vacos.cpp
```

asin

Computes the element-wise arcsine of vector elements.

Description

The `asin(a)` function computes inverse sine of vector elements.

Argument	Result	Error Code
+0	+0	
-0	-0	
+1	$+\pi/2$	
-1	$-\pi/2$	
|a| > 1	QNAN	`status::errdom`
$+\infty$	QNAN	`status::errdom`
$-\infty$	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

Specifications for special values of the complex functions are defined according to the following formula:

`asin(a) = -i*asinh(i*z).`

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event asin(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event asin(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event asin(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event asin(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`asin` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `asin` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vasin.cpp
```

atan

Computes the element-wise arctangent of vector elements in [-pi/2, pi/2].

Description

The `atan(a)` function computes inverse tangent of vector elements.

Argument	Result	Error Code
+0	+0	
-0	-0	
$+\infty$	$+\pi/2$	
$-\infty$	$-\pi/2$	
QNAN	QNAN	
SNAN	QNAN	

The `atan` function does not generate any errors.

Specifications for special values of the complex functions are defined according to the following formula `atan(a) = -i*atanh(i*a)`.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event atan(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);
}
```

```
namespace oneapi::mkl::vm {

sycl::event atan(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event atan(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event atan(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`atan` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `atan` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vatan.cpp
```

atan2

Computes the element-wise four-quadrant arctangent of vector a elements divided by vector b elements.

Description

The `atan2(a,b)` function computes four-quadrant inverse tangent of elements of two vectors.

The elements of the output vector y are computed as the four-quadrant arctangent of `a[i] / b[i]`.

Argument 1	Argument 2	Result	Error Code
-∞	-∞	$-3\pi/4$	
-∞	b < +0	-π/2	
-∞	-0	-π/2	
-∞	+0	-π/2	
-∞	b > +0	-π/2	
-∞	+∞	-π/4	
a < +0	-∞	-π	
a < +0	-0	-π/2	
a < +0	+0	-π/2	
a < +0	+∞	-0	
-0	-∞	-π	
-0	b < +0	-π	
-0	-0	-π	
-0	+0	-0	
-0	b > +0	-0	
-0	+∞	-0	
+0	-∞	+π	
+0	b < +0	+π	
+0	-0	+π	
+0	+0	+0	

continues on next page

Table 15 – continued from previous page

Argument 1	Argument 2	Result	Error Code
+0	b > +0	+0	
+0	+∞	+0	
a > +0	-∞	+π	
a > +0	-0	+π/2	
a > +0	+0	+π/2	
a > +0	+∞	+0	
+∞	-∞	+3*π/4	
+∞	b < +0	+π/2	
+∞	-0	+π/2	
+∞	+0	+π/2	
+∞	b > +0	+π/2	
+∞	+∞	+π/4	
a > +0	QNAN	QNAN	
a > +0	SNAN	QNAN	
QNAN	b > +0	QNAN	
SNAN	b > +0	QNAN	
QNAN	QNAN	QNAN	
QNAN	SNAN	QNAN	
SNAN	QNAN	QNAN	
SNAN	SNAN	QNAN	

The `atan2(a,b)` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event atan2(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event atan2(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event atan2(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event atan2(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`atan2` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Description

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `atan2` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vatan2.cpp
```

cospi

Computes the element-wise cosine of vector elements multiplied by pi.

Description

The `cospi(a)` function computes the cosine of vector elements multiplied by π. For an argument a, the function computes cos(π*a).

Argument	Result	Error Code
+0	+1	
-0	+1	
n + 0.5, for any integer n where n + 0.5 is representable	+0	
$+\infty$	QNAN	`status::errdom`
$-\infty$	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

If arguments abs(a_i) ≤ 2^{22} for single precision or abs(a_i) ≤ 2^{51} for double precision, they belong to the **fast computational path**: arguments for which VM provides the best possible performance. Avoid arguments which do not belong to the fast computational path in VM High Accuracy (HA) or Low Accuracy (LA) functions. For arguments which do not belong to the fast computational path you can use VM Enhanced Performance (EP) functions, which are fast on the entire function domain. However, these functions provide lower accuracy.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event cospi(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cospi(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event cospi(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cospi(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
       std::vector<sycl::event> const & depends = {},
       oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
       oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`cospi` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `cospi` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vcospi.cpp
```

sinpi

Computes the element-wise sine of vector elements multiplied by pi.

Description

The `sinpi(a)` function computes the sine of vector elements multiplied by π. For an argument a, the function computes sin(π*a).

Argument	Result	Error Code
+0	+0	
-0	-0	
+n, positive integer	+0	
-n, negative integer	-0	
+∞	QNAN	`status::errdom`
-∞	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

If arguments $abs(a_i) \le 2^{22}$ for single precision or $abs(a_i) \le 2^{51}$ for double precision, they belong to the **fast computational path**: arguments for which VM provides the best possible performance. Avoid arguments which do not belong to the fast computational path in VM High Accuracy (HA) or Low Accuracy (LA) functions. For arguments which do not belong to the fast computational path you can use VM Enhanced Performance (EP) functions, which are fast on the entire function domain. However, these functions provide lower accuracy.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event sinpi(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sinpi(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event sinpi(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sinpi(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`sinpi` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event

USM API

y Pointer to the output vector.

return value (event) Computation end event

Examples

An example of how to use `sinpi` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vsinpi.cpp
```

tanpi

Computes the element-wise tangent of vector elements multiplied by pi.

Description

The `tanpi(a)` function computes the tangent of vector elements multiplied by π. For an argument a, the function computes tan(π*a).

Argument	Result	Error Code
+0	+0	
-0	+0	
n, even integer	*copysign(0.0, n)	
n, odd integer	*copysign(0.0, -n)	
n + 0.5, for n even integer and n + 0.5 representable	+∞	
n + 0.5, for n odd integer and n + 0.5 representable	-∞	
+∞	QNAN	`status::errdom`
-∞	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

The `copysign(``x, y)“` function returns the first vector argument x with the sign changed to match that of the second argument y.

If arguments abs(a_i) ≤ 2^{13} for single precision or abs(a_i) ≤ 2^{67} for double precision, they belong to the **fast computational path**: arguments for which VM provides the best possible performance. Avoid arguments with do not belong to the fast computational path in VM High Accuracy (HA) or Low Accuracy (LA) functions. For arguments which do not belong to the fast computational path you can use VM Enhanced Performance (EP) functions, which are fast on the entire function domain. However, these functions provide lower accuracy.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event tanpi(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event tanpi(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event tanpi(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event tanpi(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`tanpi` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event

USM API

y Pointer to the output vector.

return value (event) Computation end event

Examples

An example of how to use `tanpi` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vtanpi.cpp
```

acospi

Computes the element-wise arc cosine of vector elements, divided by pi.

Description

The `acospi(a)` function computes the inverse cosine of vector elements divided by π. For an argument a, the function computes acos(a)/π.

Argument	Result	Error Code
+0	+1/2	
-0	+1/2	
+1	+0	
-1	+1	
|a| > 1	QNAN	`status::errdom`
$+\infty$	QNAN	`status::errdom`
$-\infty$	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event acospi(sycl::queue & exec_queue,
      std::int64_t n,
```

(continues on next page)

(continued from previous page)

```
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event acospi(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event acospi(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event acospi(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`acospi` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer y to the output vector of size n.

return value (event) Computation end event.

Examples

An example of how to use `acospi` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vacospi.cpp
```

asinpi

Computes the element-wise arcsine of vector elements, divided by pi.

Description

The `asinpi(a)` function computes the inverse sine of vector elements divided by π. For an argument a, the function computes asinpi(a)/π.

Argument	Result	Error Code
+0	+0	
-0	-0	
+1	+1/2	
-1	-1/2	
|a| > 1	QNAN	`status::errdom`
$+\infty$	QNAN	`status::errdom`
$-\infty$	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event asinpi(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event asinpi(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event asinpi(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event asinpi(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`asinpi` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `asinpi` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vasinpi.cpp
```

atanpi

Computes the element-wise arctangent of vector elements, divided by pi.

Description

The `atanpi(a)` function computes the inverse tangent of vector elements divided by π. For an argument a, the function computes atan(a)/π.

Argument	Result	Error Code
+0	+0	
-0	-0	
$+\infty$	+1/2	
$-\infty$	-1/2	
QNAN	QNAN	
SNAN	QNAN	

The `atanpi` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event atanpi(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event atanpi(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event atanpi(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event atanpi(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`atanpi` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `atanpi` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vatanpi.cpp
```

atan2pi

Computes the element-wise four-quadrant arctangent of vector a elements divided by vector b elements, divided by pi.

Description

The `atan2pi(a,b)` function computes the four-quadrant inverse tangent of the ratios of the corresponding elements of two vectors divided by π.

For the elements of the output vector y, the function computers the four-quadrant arctangent of a_i/b_i, with the result divided by π.

Argument 1	Argument 2	Result	Error Code
-∞	-∞	-3/4	
-∞	b < +0	-1/2	
-∞	-0	+1/2	
-∞	+0	-1/2	
-∞	x > +0	-1/2	
-∞	+∞	-1/4	
a < +0	-∞	-1	
a < +0	-0	-1/2	
a < +0	+0	-1/2	
a < +0	+∞	-0	
-0	-∞	-1	
-0	b < +0	-1	
-0	-0	-1	
-0	+0	-0	
-0	b > +0	-0	
-0	+∞	-0	
+0	-∞	+1	
+0	b < +0	+1	

continues on next page

Table 17 – continued from previous page

Argument 1	Argument 2	Result	Error Code
+0	-0	+1	
+0	+0	+0	
+0	b > +0	+0	
+0	+∞	+0	
a > +0	-∞	+1	
a > +0	-0	+1/2	
x > +0	+0	+1/2	
a > +0	+∞	+1/4	
+∞	-∞	+3/4	
+∞	b < +0	+1/2	
+∞	-0	+1/2	
+∞	+0	+1/2	
+∞	b > +0	+1/2	
+∞	+∞	+1/4	
a > +0	QNAN	QNAN	
a > +0	SNAN	QNAN	
QNAN	b > +0	QNAN	
SNAN	x > +0	QNAN	
QNAN	QNAN	QNAN	
QNAN	SNAN	QNAN	
SNAN	QNAN	QNAN	
SNAN	SNAN	QNAN	

The `atan2pi(a,b)` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event atan2pi(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event atan2pi(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event atan2pi(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event atan2pi(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`atan2pi` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `atan2pi` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vatan2pi.cpp
```

cosd

Computes the element-wise cosine of vector elements expressed in degrees.

Description

The `cosd(a)` function is a degree argument trigonometric function. It computes the cosine of vector elements multiplied by $\pi/180$. For an argument a, the function computes cos(π*a/180).

Note that arguments abs(a_i) ≤ 2^{24} for single precision or abs(a_i) ≤ 2^{52} for double precision, they belong to the **fast computational path**: trigonometric function arguments for which VM provides the best possible performance. Avoid arguments with do not belong to the fast computational path in VM High Accuracy (HA) or Low Accuracy (LA) functions. For arguments which do not belong to the fast computational path you can use VM Enhanced Performance (EP) functions, which are fast on the entire function domain. However, these functions provide lower accuracy.

Argument	Result	Error Code
+0	+1	
-0	+1	
+∞	QNAN	`status::errdom`
-∞	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event cosd(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cosd(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event cosd(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cosd(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`cosd` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `cosd` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vcosd.cpp
```

sind

Computes the element-wise sine of vector elements expressed in degrees.

Description

The `sind(a)` function is a degree argument trigonometric function. It computes the sine of vector elements multiplied by $\pi/180$. For an argument a, the function computes sin(π*a/180).

Note that arguments abs(a_i) ≤ 2^{24} for single precision or abs(a_i) ≤ 2^{52} for double precision, they belong to the **fast computational path**: trigonometric function arguments for which VM provides the best possible performance. Avoid arguments with do not belong to the fast computational path in VM High Accuracy (HA) or Low Accuracy (LA) functions. For arguments which do not belong to the fast computational path you can use VM Enhanced Performance (EP) functions, which are fast on the entire function domain. However, these functions provide lower accuracy.

Argument	Result	Error Code
+0	+0	
-0	-0	
+∞	QNAN	`status::errdom`
-∞	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event sind(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sind(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event sind(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sind(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`sind` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event

USM API

y Pointer to the output vector.

return value (event) Computation end event

Examples

An example of how to use `sind` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vsind.cpp
```

tand

Computes the element-wise tangent of vector elements expressed in degrees.

Description

The `tand(a)` function computes the tangent of vector elements multiplied by $\pi/180$. For an argument x, the function computes tan(π*x/180).

Note that arguments abs(a_i) ≤ 2^{38} for single precision or abs(a_i) ≤ 2^{67} for double precision, they belong to the **fast computational path**: trigonometric function arguments for which VM provides the best possible performance. Avoid arguments with do not belong to the fast computational path in VM High Accuracy (HA) or Low Accuracy (LA) functions. For arguments which do not belong to the fast computational path you can use VM Enhanced Performance (EP) functions, which are fast on the entire function domain. However, these functions provide lower accuracy.

Argument	Result	Error Code
+0	+1	
-0	+1	
±∞	QNAN	`status::errdom`
±∞	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event tand(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event tand(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event tand(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event tand(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`tand` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event

USM API

y Pointer to the output vector.

return value (event) Computation end event

Examples

An example of how to use `tand` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vtand.cpp
```

12.2.5 Hyperbolic Functions

cosh

Computes the element-wise hyperbolic cosine of vector elements.

Description

The `cosh(a)` function computes hyperbolic cosine of vector elements.

Data Type	Threshold Limitations on Input Parameters
single precision	`-Log(FLT_MAX)-Log2 <a[i] < Log(FLT_MAX)+Log2`
double precision	`-Log(DBL_MAX)-Log2 <a[i] < Log(DBL_MAX)+Log2`

Argument	Result	Error Code
+0	+1	
-0	+1	
X > overflow	$+\infty$	`status::overflow`
X < -overflow	$+\infty$	`status::overflow`
$+\infty$	$+\infty$	
$-\infty$	$+\infty$	
QNAN	QNAN	
SNAN	QNAN	

$+i\cdot\infty$	$+\infty + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN - i\cdot 0$	$QNAN + i\cdot 0$	$QNAN + i\cdot QNAN$	$+\infty + i\cdot QNAN$	$QNAN + i\cdot QNAN$
+i·Y	$+\infty$·Cos(Y)- i·∞·Sin(Y)					$+\infty\cdot CIS(Y)$	$QNAN + i\cdot QNAN$
$+i\cdot 0$	$+\infty - i\cdot 0$		$+1 - i\cdot 0$	$+1 + i\cdot 0$		$+\infty + i\cdot 0$	$QNAN + i\cdot 0$
$-i\cdot 0$	$+\infty + i\cdot 0$		$+1 + i\cdot 0$	$+1 - i\cdot 0$		$+\infty - i\cdot 0$	$QNAN - i\cdot 0$
$-i\cdot Y$	$+\infty$·Cos(Y)- i·∞·Sin(Y)					$+\infty\cdot CIS(Y)$	$QNAN + i\cdot QNAN$
$-i\cdot\infty$	$+\infty + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN + i\cdot 0$	$QNAN - i\cdot 0$	$QNAN + i\cdot QNAN$	$+\infty + i\cdot QNAN$	$QNAN + i\cdot QNAN$
$+i\cdot NAN$	$+\infty + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN + i\cdot QNAN$	$QNAN - i\cdot QNAN$	$QNAN + i\cdot QNAN$	$+\infty + i\cdot QNAN$	$QNAN + i\cdot QNAN$

Note:

- The complex `cosh(a)` function sets the VM Error Status to status::overflow in the case of overflow, that is, when RE(a), IM(a) are finite non-zero numbers, but the real or imaginary part of the exact result is so large that it does not meet the target precision.
- `cosh(CONJ(a))=CONJ(cosh(a))`
- `cosh(-a)=cosh(a)`.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event cosh(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cosh(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event cosh(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cosh(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`cosh` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `cosh` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vcosh.cpp
```

sinh

Computes the element-wise hyperbolic sine of vector elements.

Description

The `sinh(a)` function computes hyperbolic sine of vector elements.

Data Type	Threshold Limitations on Input Parameters
single precision	`-Log(FLT_MAX)-Log(2) <a[i] < Log(FLT_MAX)+Log(2)`
double precision	`-Log(DBL_MAX)-Log(2) <a[i] < Log(DBL_MAX)+Log(2)`

Argument	Result	Error Code
+0	+0	
-0	-0	
a > overflow	+∞	`status::overflow`
a < -overflow	-∞	`status::overflow`
+∞	+∞	
-∞	-∞	
QNAN	QNAN	
SNAN	QNAN	

$+i{\cdot}\infty$	$-\infty + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$	$-0 + i{\cdot}QNAN$	$+0 + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$	$+\infty + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$
$+i{\cdot}Y$	$-\infty$·Cos(Y)+ i·∞·Sin(Y)					$+\infty{\cdot}CIS(Y)$	$QNAN + i{\cdot}QNAN$
$+i{\cdot}0$	$-\infty + i{\cdot}0$		$-0 + i{\cdot}0$	$+0 + i{\cdot}0$		$+\infty + i{\cdot}0$	$QNAN + i{\cdot}0$
$-i{\cdot}0$	$-\infty - i{\cdot}0$		$-0 - i{\cdot}0$	$+0 - i{\cdot}0$		$+\infty - i{\cdot}0$	$QNAN - i{\cdot}0$
$-i{\cdot}Y$	$-\infty$·Cos(Y)+ i· infty ·Sin(Y)'					$+\infty{\cdot}CIS(Y)$	$QNAN + i{\cdot}QNAN$
$-i{\cdot}\infty$	$-\infty + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$	$-0 + i{\cdot}QNAN$	$+0 + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$	$+\infty + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$
$+i{\cdot}NAN$	$-\infty + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$	$-0 + i{\cdot}QNAN$	$+0 + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$	$+\infty + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$

Note:

- The complex `sinh(a)` function sets the VM Error Status to `status::overflow` in the case of overflow, that is, when RE(a), IM(a) are finite non-zero numbers, but the real or imaginary part of the exact result is so large that it does not meet the target precision.
- `sinh(CONJ(a))=CONJ(sinh(a))`
- `sinh(-a)=-sinh(a)`.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event sinh(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sinh(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event sinh(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event sinh(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`sinh` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `sinh` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vsinh.cpp
```

tanh

Computes the element-wise hyperbolic tangent of vector elements.

Description

The `tanh(a)` function computes hyperbolic tangent of vector elements.

Argument	Result	Erro Code
+0	+0	
-0	-0	
$+\infty$	+1	
$-\infty$	-1	
QNAN	QNAN	
SNAN	QNAN	

$+i{\cdot}\infty$	$-1+i{\cdot}0$	$QNAN+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$+1+i{\cdot}0$	$QNAN+i{\cdot}QNAN$
$+i{\cdot}Y$	$-1+i{\cdot}0{\cdot}Tan(Y)$					$+1+i{\cdot}0{\cdot}Tan(Y)$	$QNAN+i{\cdot}QNAN$
$+i{\cdot}0$	$-1+i{\cdot}0$		$-0+i{\cdot}0$	$+0+i{\cdot}0$		$+1+i{\cdot}0$	$QNAN+i{\cdot}0$
$-i{\cdot}0$	$-1-i{\cdot}0$		$-0-i{\cdot}0$	$+0-i{\cdot}0$		$+1-i{\cdot}0$	$QNAN-i{\cdot}0$
$-i{\cdot}Y$	$-1+i{\cdot}0{\cdot}Tan(Y)$					$+1+i{\cdot}0{\cdot}Tan(Y)$	$QNAN+i{\cdot}QNAN$
$-i{\cdot}\infty$	$-1-i{\cdot}0$	$QNAN+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$+1-i{\cdot}0$	$QNAN+i{\cdot}QNAN$
$+i{\cdot}NAN$	$-1+i{\cdot}0$	$QNAN+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$+1+i{\cdot}0$	$QNAN+i{\cdot}QNAN$

Note:

- `tanh(CONJ(a))=CONJ(tanh(a))`
- `tanh(-a)=-tanh(a)`.

The `tanh(a)` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event tanh(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event tanh(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy, oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event tanh(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event tanh(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`tanh` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `tanh` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vtanh.cpp
```

acosh

Computes the element-wise inverse hyperbolic cosine of vector elements.

Description

The `acosh(a)` function computes inverse hyperbolic cosine (nonnegative) of vector elements.

Argument	Result	Error Code
+1	+0	
a < +1	QNAN	`status::errdom`
$-\infty$	QNAN	`status::errdom`
$+\infty$	$+\infty$	
QNAN	QNAN	
SNAN	QNAN	

RE(a) i · IM(a)	$-\infty$	$-X$	-0	$+0$	$+X$	$+\infty$	NAN
$+i\cdot\infty$	$+\infty+i\cdot\frac{3\pi}{4}$	$+\infty+i\cdot\frac{\pi}{2}$	$+\infty+i\cdot\frac{\pi}{2}$	$+\infty+i\cdot\frac{\pi}{2}$	$+\infty+i\cdot\frac{\pi}{2}$	$+\infty+i\cdot\frac{\pi}{4}$	$+\infty+i\cdot QNAN$
$+i\cdot Y$	$+\infty+i\cdot\pi$					$+\infty+i\cdot 0$	$QNAN+i\cdot QNAN$
$+i\cdot 0$	$+\infty+i\cdot\pi$		$+0+i\cdot\frac{\pi}{2}$	$+0+i\cdot\frac{\pi}{2}$		$+\infty+i\cdot 0$	$QNAN+i\cdot QNAN$
$-i\cdot 0$	$+\infty+i\cdot\pi$		$+0+i\cdot\frac{\pi}{2}$	$+0+i\cdot\frac{\pi}{2}$		$+\infty+i\cdot 0$	$QNAN+i\cdot QNAN$
$-i\cdot Y$	$+\infty+i\cdot\pi$					$+\infty+i\cdot 0$	$QNAN+i\cdot QNAN$
$-i\cdot\infty$	$+\infty-i\cdot\frac{3\pi}{4}$	$+\infty-i\cdot\frac{\pi}{2}$	$+\infty-i\cdot\frac{\pi}{2}$	$+\infty-i\cdot\frac{\pi}{2}$	$+\infty-i\cdot\frac{\pi}{2}$	$+\infty-i\cdot\frac{\pi}{4}$	$+\infty+i\cdot QNAN$
$+i\cdot NAN$	$+\infty+i\cdot QNAN$	$QNAN+i\cdot QNAN$	$QNAN+i\cdot QNAN$	$QNAN+i\cdot QNAN$	$QNAN+i\cdot QNAN$	$+\infty+i\cdot QNAN$	$QNAN+i\cdot QNAN$

Note: `acosh(CONJ(a))=CONJ(acosh(a))`.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event acosh(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event acosh(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event acosh(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event acosh(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`acosh` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `acosh` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vacosh.cpp
```

asinh

Computes the element-wise inverse hyperbolic sine of vector elements.

Description

The `asinh(a)` function computes inverse hyperbolic sine of vector elements.

Argument	Result	Error Code
+0	+0	
-0	-0	
$+\infty$	$+\infty$	
$-\infty$	$-\infty$	
QNAN	QNAN	
SNAN	QNAN	

RE(a) i · IM(a)	$-\infty$	-X	-0	+0	+X	$+\infty$	NAN
$+i{\cdot}\infty$	$-\infty$	$-\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty + i{\cdot}QNAN$
$+i{\cdot}Y$	$-\infty + i{\cdot}0$					$+\infty + i{\cdot}0$	$QNAN + i{\cdot}QNAN$
$+i{\cdot}0$	$+\infty + i{\cdot}0$		$+0 + i{\cdot}0$	$+0 + i{\cdot}0$		$+\infty + i{\cdot}0$	$QNAN + i{\cdot}QNAN$
$-i{\cdot}0$	$-\infty - i{\cdot}0$		$-0 - i{\cdot}0$	$+0 - i{\cdot}0$		$+\infty - i{\cdot}0$	$QNAN - i{\cdot}QNAN$
$-i{\cdot}Y$	$-\infty - i{\cdot}0$					$+\infty - i{\cdot}0$	$QNAN + i{\cdot}QNAN$
$-i{\cdot}\infty$	$-\infty$-i· pi /4′	$-\infty$-i· pi /2′	$-\infty$-i· pi /2′	$+\infty$-i· pi /2′	$+\infty$-i· pi /2′	$+\infty$-i· pi /4′	$+\infty + i{\cdot}QNAN$
$+i{\cdot}NAN$	$-\infty + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$	$+\infty + i{\cdot}QNAN$	$QNAN + i{\cdot}QNAN$

The `asinh(a)` function does not generate any errors.

Note:

- asinh(CONJ(a))=CONJ(asinh(a))
- asinh(-a)=-asinh(a).

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event asinh(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event asinh(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event asinh(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event asinh(sycl::queue & exec_queue,
      T const * a,
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`asinh` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `asinh` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vasinh.cpp
```

atanh

Computes the element-wise inverse hyperbolic tangent of vector elements.

Description

The `atanh(a)` function computes inverse hyperbolic tangent of vector elements.

Argument	Result	Error Code
+1	$+\infty$	`status::sing`
-1	$-\infty$	`status::sing`
|a| > 1	QNAN	`status::errdom`
$-\infty$	QNAN	`status::errdom`
$+\infty$	$+\infty$	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

RE(a) i · IM(a)	$-\infty$	-X	-0	+0	+X	$+\infty$	NAN
$+i{\cdot}\infty$	$-0+i{\cdot}\pi/2$	$-0+i{\cdot}\pi/2$	$-0+i{\cdot}\pi/2$	$+0+i{\cdot}\pi/2$	$+0+i{\cdot}\pi/2$	$+0+i{\cdot}\pi/2$	$+0+i{\cdot}\pi/2$
$+i{\cdot}Y$	$-0+i{\cdot}\pi/2$					$+0+i{\cdot}\pi/2$	$QNAN+i{\cdot}QNAN$
$+i{\cdot}0$	$-0+i{\cdot}\pi/2$		$-0+i{\cdot}0$	$+0+i{\cdot}0$		$+0+i{\cdot}\pi/2$	$QNAN+i{\cdot}QNAN$
$-i{\cdot}0$	$-0-i{\cdot}\pi/2$		$-0-i{\cdot}0$	$+0-i{\cdot}0$		$+0-i{\cdot}\pi/2$	$QNAN-i{\cdot}QNAN$
$-i{\cdot}Y$	$-0-i{\cdot}\pi/2$					$+0-i{\cdot}\pi/2$	$QNAN+i{\cdot}QNAN$
$-i{\cdot}\infty$	$-0-i{\cdot}\pi/2$	$-0-i{\cdot}\pi/2$	$-0-i{\cdot}\pi/2$	$+0-i{\cdot}\pi/2$	$+0-i{\cdot}\pi/2$	$+0-i{\cdot}\pi/2$	$+0-i{\cdot}\pi/2$
$+i{\cdot}NAN$	$-0+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$-0+i{\cdot}QNAN$	$+0+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$	$+0+i{\cdot}QNAN$	$QNAN+i{\cdot}QNAN$

Note:

- `atanh(±1±i*0)=±:math:`\infty`±i*0`, and status::sing error is generated
- `atanh(CONJ(a))=CONJ(atanh(a))`
- `atanh(-a)=-atanh(a)`.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event atanh(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event atanh(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event atanh(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event atanh(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`atanh` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `atanh` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vatanh.cpp
```

12.2.6 Special Functions

erf

Computes the element-wise error function of vector elements.

Description

The `erf` function computes the error function values for elements of the input vector a and writes them to the output vector y.

The error function is defined as given by:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

Useful relations:

$$\text{erfc}(x) = 1 - \text{erf}(x)$$

where `erfc` is the complementary error function.

$$\Phi(x) = \frac{1}{2}\text{erf}(\frac{x}{\sqrt{2}})$$

where

$$\Phi(x) = -\frac{1}{\sqrt{2\pi}} \int_0^x \text{exp}(-fract^2 2) dt$$

is the cumulative normal distribution function.

$$\Phi^{-1}(x) = \sqrt{2}\text{erf}^{-1}(2x - 1)$$

where $\phi - 1(x)$ and `erf-1(x)` are the inverses to $\phi(x)$ and `erf(x)`, respectively.

The following figure illustrates the relationships among `erf` family functions (`erf`, `erfc`, `cdfnorm`).

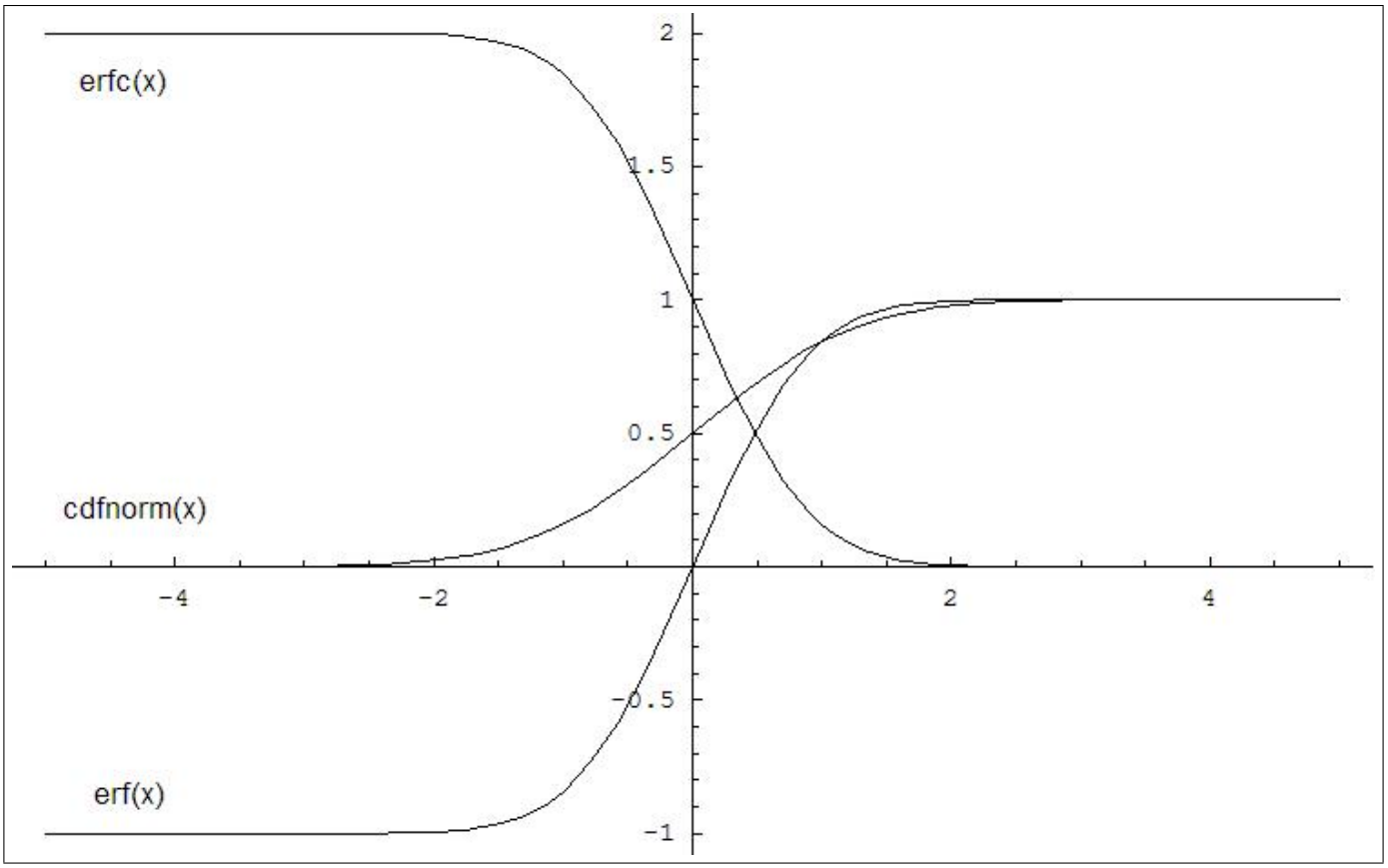


Fig. 2: `erf` Family Functions Relationship

Useful relations for these functions:

$$\text{erf}(x) + \text{erfc} = 1$$

$$\text{cdfnorm}(x) = \frac{1}{2}(1 + \text{erf}(\frac{x}{\sqrt{2}})) = 1 - frac12\text{erfc}(\frac{x}{\sqrt{2}})$$

Argument	Result	Error Code
+∞	+1	
-∞	-1	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event erf(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event erf(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event erf(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event erf(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`erf` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `erf` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/verf.cpp
```

erfc

Computes the element-wise complementary error function of vector elements.

Description

The `erfc` function computes the complementary error function values for elements of the input vector a and writes them to the output vector y.

The complementary error function is defined as follows:

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$$

Useful relations:

$$\text{erfc}(x) = 1 - \text{erf}(x)$$

$$\Phi(x) = \frac{1}{2}\text{erf}(\frac{x}{\sqrt{2}})$$

where

$$\Phi(x) = -\frac{1}{\sqrt{2\pi}} \int_0^x \text{exp}(-fract^2 2) dt$$

is the cumulative normal distribution function.

$$\Phi^{-1}(x) = \sqrt{2}\text{erf}^{-1}(2x - 1)$$

where $\phi - 1(x)$ and `erf-1(x)` are the inverses to $\phi(x)$ and `erf(x)`, respectively.

The following figure illustrates the relationships among `erf` family functions (`erf`, `erfc`, `cdfnorm`).

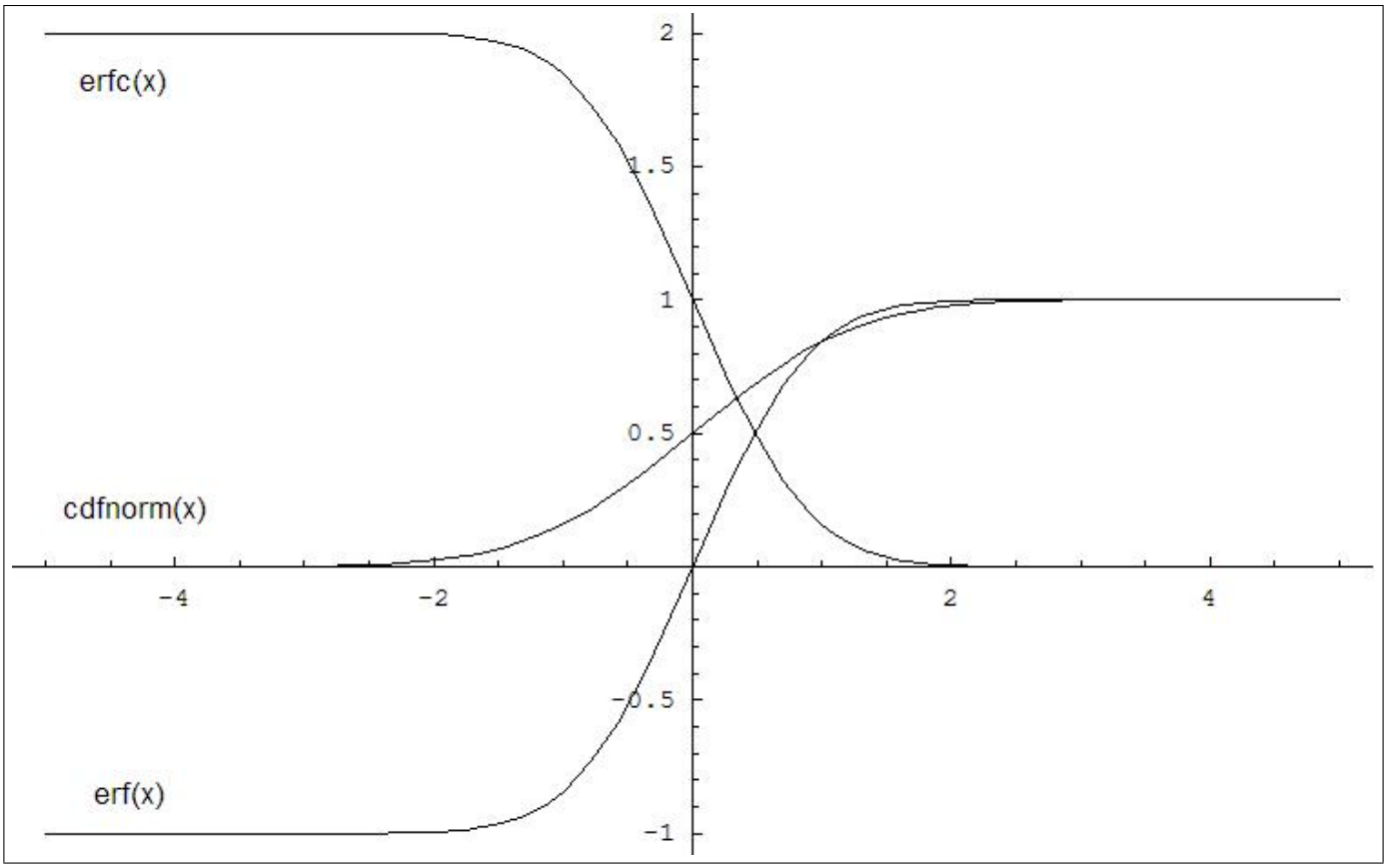


Fig. 3: `erfc` Family Functions Relationship

Useful relations for these functions:

$$\text{erf}(x) + \text{erfc} = 1$$

$$\text{cdfnorm}(x) = \frac{1}{2}(1 + \text{erf}(\frac{x}{\sqrt{2}})) = 1 - frac12\text{erfc}(\frac{x}{\sqrt{2}})$$

Argument	Result	Error Code
a > underflow	+0	`status::underflow`
$+\infty$	+0	
$-\infty$	+2	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event erfc(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event erfc(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event erfc(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event erfc(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`erfc` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the ref:**create_error_handler** function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `erfc` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/verfc.cpp
```

cdfnorm

Computes the element-wise cumulative normal distribution function of vector elements.

Description

The `cdfnorm` function computes the cumulative normal distribution function values for elements of the input vector a and writes them to the output vector y.

The cumulative normal distribution function is defined as given by:

$$\text{CdfNorm}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2} dt$$

Useful relations:

$$\text{cdfnorm}(x) = \frac{1}{2}(1 + \text{erf}(\frac{x}{\sqrt{2}})) = 1 - frac12\text{erfc}(\frac{x}{\sqrt{2}})$$

where `erf` and `erfc` are the error and complementary error functions.

The following figure illustrates the relationships among `erf` family functions (`erf`, `erfc`, `cdfnorm`).

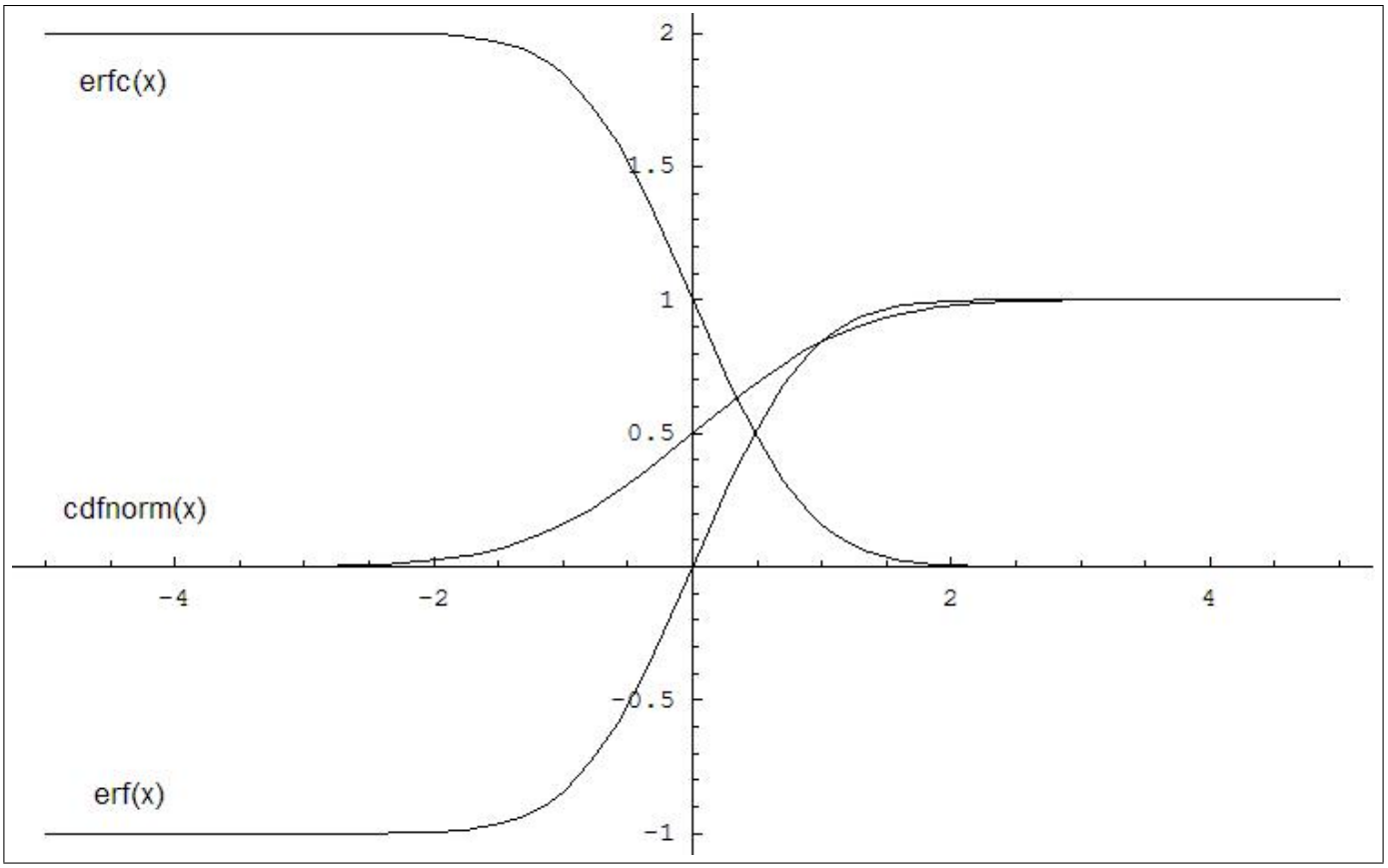


Fig. 4: `cdfnorm` Family Functions Relationship

Useful relations for these functions:

$$\text{erf}(x) + \text{erfc} = 1$$

$$\text{cdfnorm}(x) = \frac{1}{2}(1 + \text{erf}(\frac{x}{\sqrt{2}})) = 1 - frac12\text{erfc}(\frac{x}{\sqrt{2}})$$

Argument	Result	Error Code
a < underflow	+0	`status::underflow`
$+\infty$	+1	
$-\infty$	+0	
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event cdfnorm(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cdfnorm(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event cdfnorm(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cdfnorm(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`cdfnorm` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `cdfnorm` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vcdfnorm.cpp
```

erfinv

Computes the element-wise inverse error function of vector elements.

Description

The `erfinv(a)` function computes the inverse error function values for elements of the input vector a and writes them to the output vector y

`y = erf-1(a),`

where `erf(x)` is the error function defined as given by:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

Useful relations:

$$\text{erf}^{-1}(x) = \text{erfc}^{-1}(1 - x)$$

where `erfc` is the complementary error function.

$$\Phi = \frac{1}{2}\text{erf}(\frac{x}{\sqrt{2}})$$

where

$$\Phi(x) = -\frac{1}{\sqrt{2\pi}} \int_0^x \exp(-fract^2 2)dt$$

is the cumulative normal distribution function.

$$\Phi^{-1}(x) = \sqrt{2}\text{erf}^{-1}(2x-1)$$

where $\phi - 1(x)$ and `erf-1(x)` are the inverses to $\phi(x)$ and `erf(x)`, respectively.

The following figure illustrates the relationships among `erfinv` family functions (`erfinv`, `erfcinv`, `cdfnorminv`).

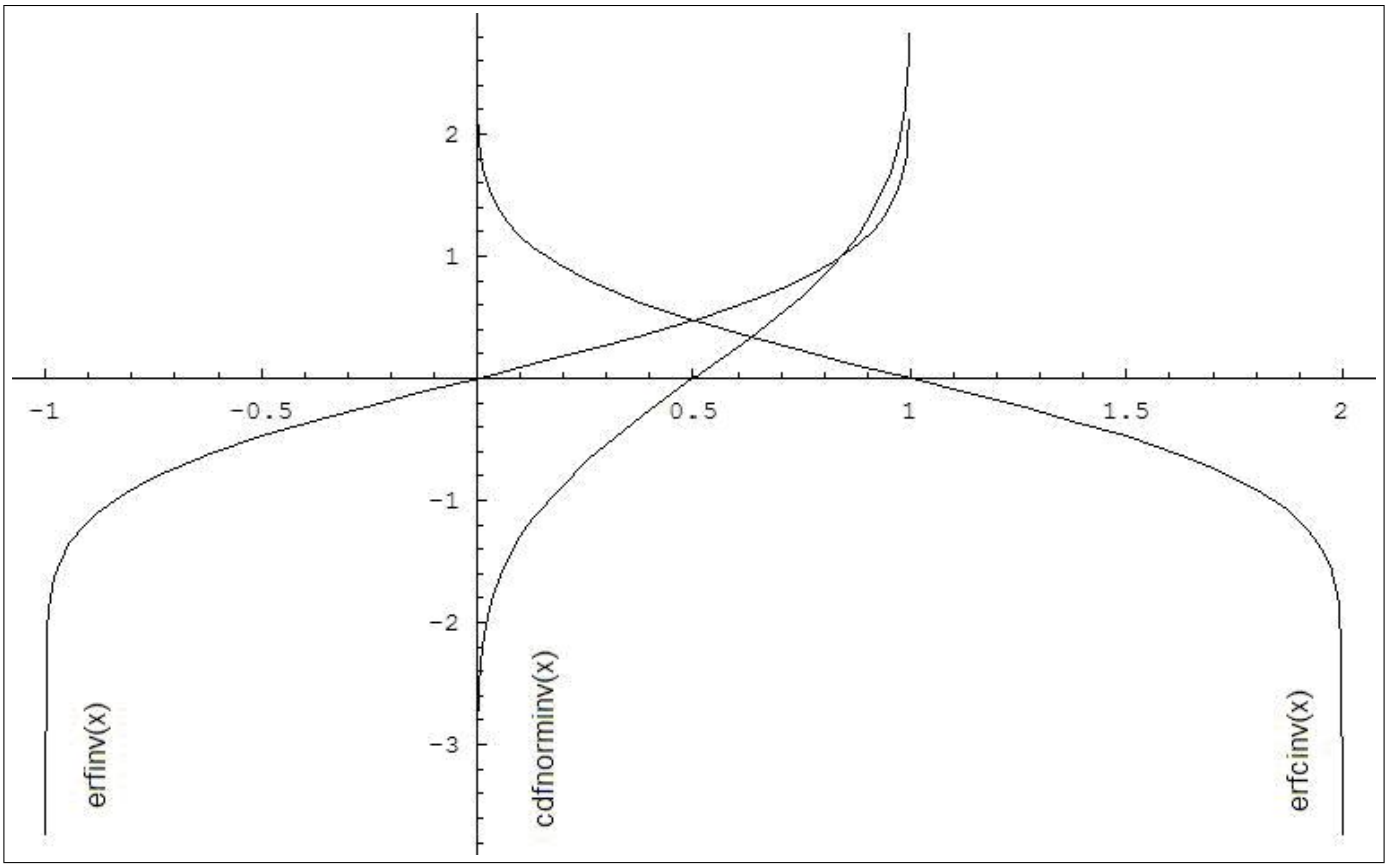


Fig. 5: `erfinv` Family Functions Relationship

Useful relations for these functions:

$$\text{erfcinv}(x) = \text{erfinv}(1-x)$$

$$\text{cdfnorminv}(x) = \sqrt{2}\text{erfinv}(2x-1) = \sqrt{2}\text{erfcinv}(2-2x)$$

Argument	Result	Error Code
+0	+0	
-0	-0	
+1	+∞	status::sing
-1	-∞	status::sing
|a| >1	QNAN	status::errdom
+∞	QNAN	status::errdom
-∞	QNAN	status::errdom
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event erfinv(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event erfinv(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});
}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event erfinv(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
```

(continues on next page)

(continued from previous page)

```
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event erfinv(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`erfinv` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `erfinv` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/verfinv.cpp
```

erfcinv

Computes the element-wise inverse complementary error function of vector elements.

Description

The `erfcinv(a)` function computes the inverse complimentary error function values for elements of the input vector a and writes them to the output vector y.

The inverse complementary error function is defined as given by:

$$\text{erfcinv}(x) = \text{erfinv}(1 - x)$$

$$\text{erfinv}(x) = \text{erf}^{-1}(x)$$

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

where `erf(x)` denotes the error function and `erfinv(x)` denotes the inverse error function.

The following figure illustrates the relationships among `erfinv` family functions (`erfinv`, `erfcinv`, `cdfnorminv`).

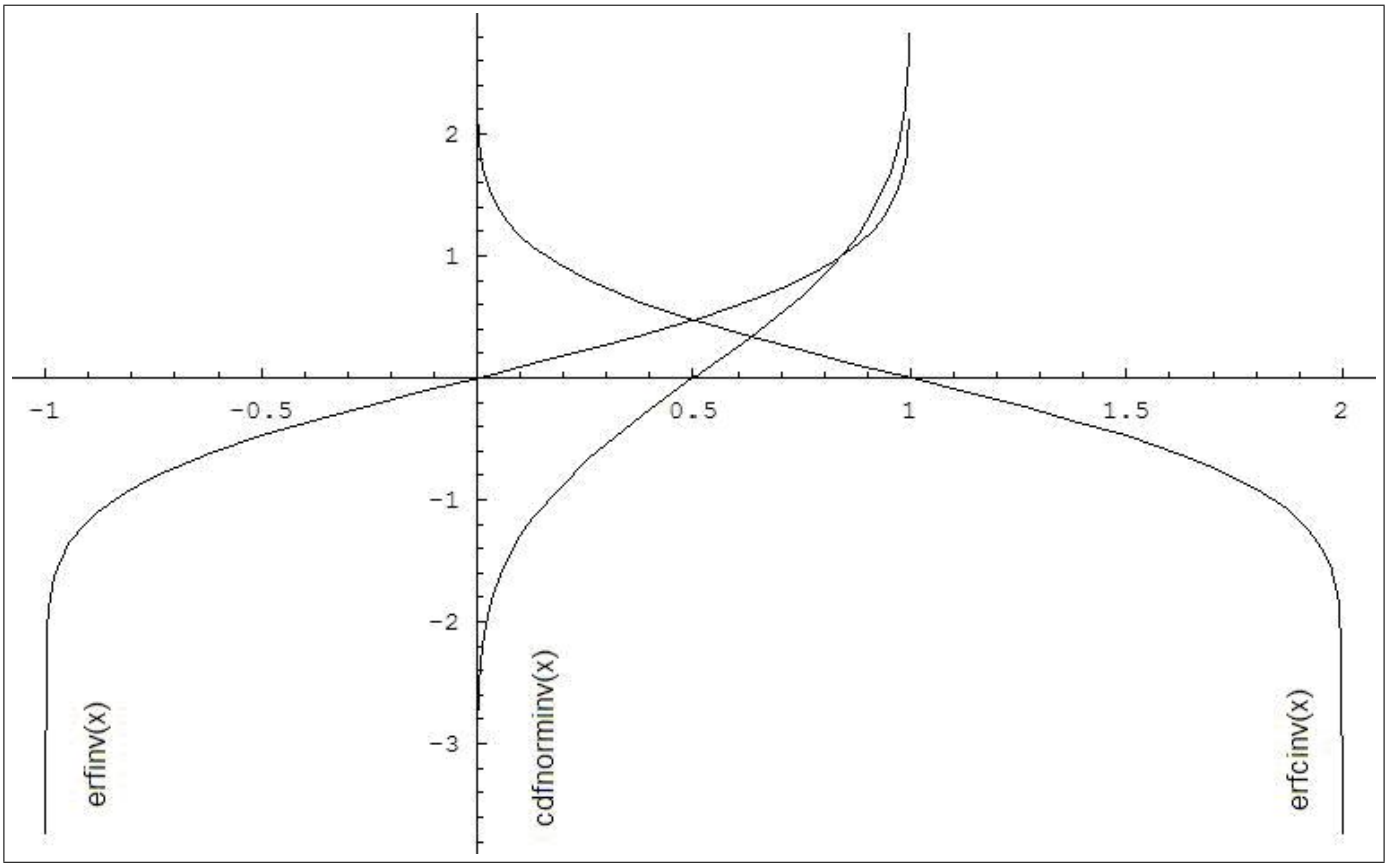


Fig. 6: `erfcinv` Family Functions Relationship

Useful relations for these functions:

$$\text{erfcinv}(x) = \text{erfinv}(1 - x)$$

$$\text{cdfnorminv}(x) = \sqrt{2}\text{erfinv}(2x - 1) = \sqrt{2}\text{erfcinv}(2 - 2x)$$

Argument	Result	Error Code
+1	+0	
+2	-∞	status::sing
-0	+∞	status::sing
+0	+∞	status::sing
a < -0	QNAN	status::errdom
a > +2	QNAN	status::errdom
+∞	QNAN	status::errdom
-∞	QNAN	status::errdom
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event erfcinv(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event erfcinv(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event erfcinv(sycl::queue & exec_queue,
```

(continues on next page)

(continued from previous page)

```
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event erfcinv(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`erfcinv` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the ref:**create_error_handler** function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer y containing the output vector of size n.

return value (event) Computation end event.

USM API

y Pointer y to the output vector of size n.

return value (event) Computation end event.

Examples

An example of how to use `erfcinv` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/verfcinv.cpp
```

cdfnorminv

Computes the element-wise inverse cumulative normal distribution function of vector elements.

Description

The `cdfnorminv(a)` function computes the inverse cumulative normal distribution function values for elements of the input vector a and writes them to the output vector y.

The inverse cumulative normal distribution function is defined as given by:

$$\text{CdfNormInv}(x) = \text{CdfNorm}^{-1}(x),$$

where `cdfnorm(x)` denotes the cumulative normal distribution function.

Useful relations:

$$\text{cdfnorminv}(x) = \sqrt{2}\text{erfinv}(2x-1) = \sqrt{2}\text{erfcinv}(2-2x)$$

where `erfinv(x)` denotes the inverse error function and `erfcinv(x)` denotes the inverse complementary error functions.

The following figure illustrates the relationships among `erfinv` family functions (`erfinv`, `erfcinv`, `cdfnorminv`).

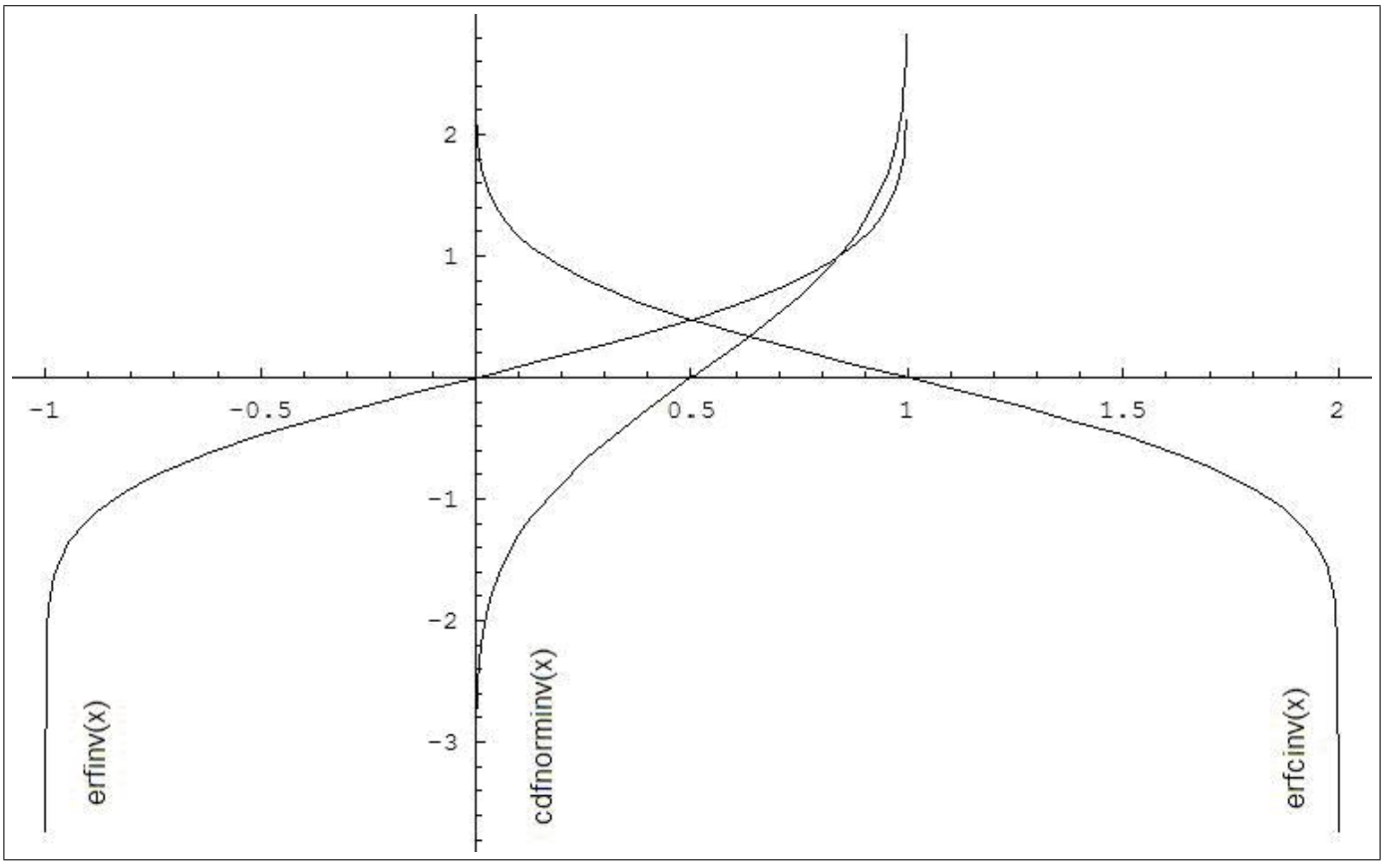


Fig. 7: `cdfnorminv` Family Functions Relationship

Useful relations for these functions:

$$\text{erfcinv}(x) = \text{erfinv}(1-x)$$

$$\text{cdfnorminv}(x) = \sqrt{2}\text{erfinv}(2x-1) = \sqrt{2}\text{erfcinv}(2-2x)$$

Argument	Result	Error Code
+0.5	+0	
+1	+∞	status::sing
-0	-∞	status::sing
+0	-∞	status::sing
a < -0	QNAN	status::errdom
a > +1	QNAN	status::errdom
+∞	QNAN	status::errdom
-∞	QNAN	status::errdom
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event cdfnorminv(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cdfnorminv(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event cdfnorminv(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event cdfnorminv(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`cdfnorminv` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `cdfnorminv` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vcdfnorminv.cpp
```

lgamma

Computes the element-wise natural logarithm of the absolute value of the gamma function of vector elements.

Description

The `lgamma(a)` function computes the natural logarithm of the absolute value of gamma function for elements of the input vector a and writes them to the output vector y. Precision overflow thresholds for the lgamma function are beyond the scope of this document. If the result does not meet the target precision, the function sets the VM Error Status to `status::overflow`.

Argument	Result	Error Code
+1	+0	
+2	+0	
+0	+∞	`status::sing`
-0	+∞	`status::sing`
negative integer	+∞	`status::sing`
-∞	+∞	
+∞	+∞	
a > overflow	+∞	`status::overflow`
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event lgamma(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event lgamma(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event lgamma(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event lgamma(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`lgamma` supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `lgamma` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vlgamma.cpp
```

tgamma

Computes the element-wise gamma function of vector elements.

Description

The `tgamma(a)` function computes the gamma function for elements of the input vector a and writes them to the output vector y. Precision overflow thresholds for the `tgamma` function are beyond the scope of this document. If the result does not meet the target precision, the function raises sets the VM Error Status to `status::sing`.

Argument	Result	Error Code
+0	+∞	`status::sing`
-0	-∞	`status::sing`
negative integer	QNAN	`status::errdom`
-∞	QNAN	`status::errdom`
+∞	+∞	
a > overflow	+∞	`status::sing`
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event tgamma(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event tgamma(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
```

(continues on next page)

(continued from previous page)

```
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event tgamma(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event tgamma(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`tgamma` supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for more details.

sy Slice selector for y. See Data Types for more details.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `tgamma` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vtgamma.cpp
```

expint1

Computes the element-wise exponential integral of vector elements.

Description

The `expint1(a)` function computes the exponential integral of vector elements of the input vector a and writes them to the output vector y.

For positive real values x, this can be written as:

$$E_1(x) = \int_x^\infty \frac{e^{-t}}{t} dt = \int_1^\infty \frac{e^{-xt}}{t} dt$$

For negative real values x, the result is defined as NAN.

Argument	Result	Error Code
x < +0	QNAN	`status::errdom`
+0	$+\infty$	`status::sing`
-0	$+\infty$	`status::sing`
$+\infty$	+0	
$-\infty$	QNAN	`status::errdom`
QNAN	QNAN	
SNAN	QNAN	

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event expint1(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event expint1(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event expint1(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event expint1(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`expint1` supports the following precisions and devices:

T	Devices supported
`float`	CPU
`double`	CPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `expint1` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vexpint1.cpp
```

12.2.7 Rounding Functions

floor

Computes the element-wise rounding of vector elements to the nearest integer towards minus infinity.

Description

The `floor(a)` function computes an integer value rounded towards minus infinity for each vector element.

$$y_i = \lfloor a_i \rfloor$$

Argument	Result	Error Code
+0	+0	
-0	-0	
$+\infty$	$+\infty$	
$-\infty$	$-\infty$	
QNAN	QNAN	
SNAN	QNAN	

The `floor` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event floor(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event floor(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event floor(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event floor(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`floor` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `floor` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vfloor.cpp
```

ceil

Computes the element-wise rounding of vector elements to the nearest integer towards plus infinity.

Description

The `ceil(a)` function computes an integer value rounded towards plus infinity for each vector element.

$$y_i = \lceil a_i \rceil$$

Argument	Result	Error Code
+0	+0	
-0	-0	
$+\infty$	$+\infty$	
$-\infty$	$-\infty$	
QNAN	QNAN	
SNAN	QNAN	

The `ceil` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event ceil(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event ceil(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event ceil(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event ceil(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`ceil` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `ceil` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vceil.cpp
```

trunc

Computes the element-wise rounding of vector elements to the nearest integer towards zero.

Description

The `trunc(a)` function computes an integer value rounded towards zero for each vector element.

$$a_i \geq 0, y_i = \lfloor a_i \rfloor$$
$$a < 0, y_i = \lceil a_i \rceil$$

Argument	Result	Error Code
+0	+0	
-0	-0	
$+\infty$	$+\infty$	
$-\infty$	$-\infty$	
QNAN	QNAN	
SNAN	QNAN	

The `trunc` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event trunc(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event trunc(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event trunc(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event trunc(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`trunc` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `trunc` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vtrunc.cpp
```

round

Computes the element-wise rounding of vector elements to the nearest integer, breaking ties away from zero.

Description

The `round(a)` function computes a value rounded to the nearest integer for each vector element. Input elements that are halfway between two consecutive integers are always rounded away from zero regardless of the rounding mode.

Argument	Result	Error Code
+0	+0	
-0	-0	
$+\infty$	$+\infty$	
$-\infty$	$-\infty$	
QNAN	QNAN	
SNAN	QNAN	

The `round(a)` function does not generate any errors.

API

Syntax

Buffer API

```
sycl::event round(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm{

sycl::event round(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm{

sycl::event round(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm{

sycl::event round(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`round` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `round` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vround.cpp
```

nearbyint

Computes the element-wise rounding of vector elements to the nearest integer, according to the current rounding mode.

Description

The `nearbyint(a)` function computes a rounded integer value in a current rounding mode for each vector element.

Argument	Result	Error Code
+0	+0	
-0	-0	
$+\infty$	$+\infty$	
$-\infty$	$-\infty$	
QNAN	QNAN	
SNAN	QNAN	

The `nearbyint` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event nearbyint(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event nearbyint(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event nearbyint(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event nearbyint(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`nearbyint` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `nearbyint` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vnearbyint.cpp
```

rint

Computes the element-wise rounding of vector elements to the nearest integer, according to the current rounding mode.

Description

The `rint(a)` function computes a rounded floating-point integer value using the current rounding mode for each vector element.

The rounding mode affects the results computed for inputs that fall between consecutive integers. For example:

- `f(0.5) = 0`, for rounding modes set to round to nearest round toward zero or to minus infinity.
- `f(0.5) = 1`, for rounding modes set to plus infinity.
- `f(-1.5) = -2`, for rounding modes set to round to nearest or to minus infinity.
- `f(-1.5) = -1`, for rounding modes set to round toward zero or to plus infinity.

Argument	Result	Error Code
+0	+0	
-0	-0	
+∞	+∞	
-∞	-∞	
QNAN	QNAN	
SNAN	QNAN	

The `rint` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event rint(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event rint(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event rint(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event rint(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`rint` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `rint` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vrint.cpp
```

modf

Computes the element-wise truncated integral and remaining fractional parts of vector elements.

Description

The `modf(a)` function computes a truncated integer value and the remaining fraction part for each vector element.

$$a_i \geq 0, \begin{cases} y_i = \lfloor a_i \rfloor \\ z_i = a_i - \lfloor a_i \rfloor \end{cases}$$

$$a_i < 0, \begin{cases} y_i = \lceil a_i \rceil \\ z_i = a_i - \lceil a_i \rceil \end{cases}$$

Argument	Result 1	Result 2	Error Code
+0	+0	+0	
-0	-0	-0	
$+\infty$	$+\infty$	+0	
$-\infty$	$-\infty$	-0	
SNAN	QNAN	QNAN	
QNAN	QNAN	QNAN	

The `modf` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event modf(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      sycl::buffer<T> & z,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event modf(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      sycl::buffer<T> & z,
      oneapi::mkl::slice sz,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event modf(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      T * z,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event modf(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      T * z,
      oneapi::mkl::slice sz,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`modf` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

sz Slice selector for z. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

sz Slice selector for z. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector for truncated integer values.

z The buffer containing the output vector for remaining fraction parts.

return value (event) Computation end event.

USM API

y Pointer to the output vector for truncated integer values.

z Pointer to the output vector for remaining fraction parts.

return value (event) Computation end event.

Examples

An example of how to use `modf` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vmodf.cpp
```

frac

Computes the element-wise signed fractional part of vector elements.

Description

The `frac(a)` function computes a signed fractional part for each vector element.

$$y_i =$$

$$\begin{cases} \\ a_i - \lfloor a_i \rfloor a_i, & \geq 0 \\ a_i - \lceil a_i \rceil, & a < 0 \end{cases}$$

$$y_i =$$

$$\begin{cases} \\ a_i - \lfloor a_i \rfloor a_i, & \geq 0 \\ a_i - \lceil a_i \rceil, & a < 0 \end{cases}$$

Argument	Result	Error Code
+0	+0	
-0	-0	
$+\infty$	+0	
$-\infty$	-0	
QNAN	QNAN	
SNAN	QNAN	

The `frac` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event frac(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event frac(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event frac(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T * y,
      std::vector<sycl::event> const & depends = {},
```

(continues on next page)

(continued from previous page)

```
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event frac(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`frac` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the input vector.

sa Slice selector for a. See Data Types for a description of the |O-MKL| slice type.

sy Slice selector for y. See Data Types for a description of the |O-MKL| slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the input vector.

sa Slice selector for a. See Data Types for a description of the |O-MKL| slice type.

sy Slice selector for y. See Data Types for a description of the |O-MKL| slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `frac` can be found in the |O-MKL| installation directory, under:

```
examples/dpcpp/vml/source/vfrac.cpp
```

12.3 VM Service Functions

The VM Service functions enable you to set/get the accuracy mode and error code. These functions are available both in the Fortran and C interfaces. The table below lists available VM Service functions and their short description.

Function Short Name	Description
set_mode	Sets the VM mode
get_mode	Gets the VM mode
set_status	Sets the VM Error Status
get_status	Gets the VM Error Status
clear_status	Clears the VM Error Status
create_error_handler	Creates the local VM error handler for a function

12.3.1 set_mode

Sets a new mode for VM functions according to the `mode` parameter and returns the previous VM mode.

Description

The `set_mode` function sets a new mode for VM functions according to the `new_mode` parameter and returns the previous VM mode. The mode change has a global effect on all the VM functions within a queue.

oneMKL VM mode parameters provide control on the accuracy of mathematical functions, and on oneMKL VM Strided API behavior. The mode set for a given queue can be overridden locally by the local mode parameter in a VM function call.

The mode value is a bitwise OR (|) combination of the values described in the following table.

Value	Description
Accuracy Control	
`oneapi::mkl::vm::mode::ha`	High accuracy versions of VM functions. (DEFAULT)
`oneapi::mkl::vm::mode::la`	Low accuracy versions of VM functions.
`oneapi::mkl::vm::mode::ep`	Enhanced performance accuracy versions of VM functions.
Slice Argument Checking	
`oneapi::mkl::vm::mode::badarg_exception`	Throw a `oneapi::mkl::invalid_argument` exception on invalid arguments. (DEFAULT)
`oneapi::mkl::vm::mode::badarg_quiet`	Invalid arguments quietly make the call a “no-op”. The VM status is set to `vm::status::empty_computation`.
Slice Indexing Controls	
`oneapi::mkl::vm::mode::slice_normal`	Non-equal slice sizes are considered invalid. (DEFAULT)
`oneapi::mkl::vm::mode::slice_minimum`	The minimum of all slice sizes defines the number of evaluations.
`oneapi::mkl::vm::mode::slice_cyclic`	The output slice(s) size defines the number of evaluations. Input slices wrap around from the start.
Default Local Mode	
`oneapi::mkl::vm::mode::not_defined`	VM mode not defined. This has no effect.

The default value if no VM mode is defined or if the VM mode value is set to `mode::not_defined` is (`mode::badarg_exception` | `mode::slice_normal` | `mode::ha`).

API

Syntax

```
uint64_t set_mode(queue& exec_queue, uint64_t   new_mode )
```

`set_mode` supports the following devices: CPU and GPU.

Input Parameters

exec_queue The queue where the routine should be executed.

new_mode Specifies the VM mode to be set.

Output Parameters

return value (old_mode) Specifies the former VM mode.

Examples

```
oldmode = set_mode (exec_queue , mode::la);
oldmode = set_mode (exec_queue , mode::ep | mode::ftzdazon);
```

12.3.2 get_mode

Gets the VM mode for a queue.

Description

The `get_mode` function returns the global VM `mode` parameter that controls accuracy and VM Strided API behavior for a given queue. The variable value is a combination by bitwise OR (|) of the values listed in the following table.

Value of mode	Description
Accuracy Control	
`oneapi::mkl::vm::mode::ha`	High accuracy versions of VM functions. (DEFAULT)
`oneapi::mkl::vm::mode::la`	Low accuracy versions of VM functions.

continues on next page

Table 22 – continued from previous page

Value of mode	Description
`oneapi::mkl::vm::mode::ep`	Enhanced performance accuracy versions of VM functions.
Slice Argument Checking	
`oneapi::mkl::vm::mode::badarg_exception`	Throw a oneapi::mkl::invalid_argument exception on invalid arguments. (DEFAULT)
`oneapi::mkl::vm::mode::badarg_quiet`	Invalid arguments quietly make the call a "no-op". The VM status is set to `vm::status::empty_computation`.
Slice Indexing Controls	
`oneapi::mkl::vm::mode::slice_normal`	Non-equal slice sizes are considered invalid. (DEFAULT)
`oneapi::mkl::vm::mode::slice_minimum`	The minimum of all slice sizes defines the number of evaluations.
`oneapi::mkl::vm::mode::slice_cyclic`	The output slice(s) size defines the number of evaluations. Input slices wrap around from the start.
Default Local Mode	
`oneapi::mkl::vm::mode::not_defined`	VM mode not defined. This has no effect.

See the example below

API

Syntax

```
uint64_t get_mode( queue& exec_queue )
```

`get_mode` supports the following devices: CPU and GPU.

Input Parameters

exec_queue The queue where the routine should be executed.

Output Parameters

return value (old_mode) Specifies the global VM mode.

Examples

```
accm = get_mode (exec_queue) & mode::accuracy_mask;
denm = get_mode (exec_queue) & mode::ftzdaz_mask;
```

12.3.3 set_status

Sets the global VM Status according to new values and returns the previous VM Status.

Description

The `set_status` function sets the global VM Status to new value and returns the previous VM Status.

The global VM Status is a single value and it accumulates via bitwise OR (|) all errors that happen inside VM functions. The following table lists the possible error values.

Status	Description
Successful Execution	
`status::success`	VM function execution completed successfully
`status::not_defined`	VM status not defined
Warnings	
`status::accuracy_-warning`	VM function execution completed successfully in a different accuracy mode
Computational Errors	
`status::errdom`	Values are out of a range of definition producing invalid (QNaN) result
`status::sing`	Values cause divide-by-zero (singularity) errors and produce and invalid (QNaN or Inf) result
`status::overflow`	An overflow happened during the calculation process
`status::underflow`	An underflow happened during the calculation process

API

Syntax

```
uint8_t set_status (queue& exec_queue,uint_8   new_status )
```

`set_status` supports the following devices: CPU and GPU.

Input Parameters

exec_queue The queue where the routine should be executed.

new_status Specifies the VM status to be set.

Output Parameters

return value (old_status) Specifies the former VM status.

Examples

```
uint8_t olderr = set_status (exec_queue, status::success);


if (olderr & status::errdom)
{
    std::cout << ”Errdom status returned” << std::endl;
}


if (olderr & status::sing)
{
    std::cout << ”Singularity status returned” << std::endl;
}
```

12.3.4 get_status

Gets the VM Status.

Description

The `get_status` function gets the VM status.

The global VM Status is a single value and it accumulates via bitwise OR (|) all computational errors that happen inside VM functions. The following table lists the possible error values.

Status	Description
Successful Execution	
`status::success`	VM function execution completed successfully

continues on next page

Table 23 – continued from previous page

Status	Description
`status::not_defined`	VM status not defined
Warnings	
`status::accuracy_warning`	VM function execution completed successfully in a different accuracy mode
Computational Errors	
`status::errdom`	Values are out of a range of definition producing invalid (QNaN) result
`status::sing`	Values cause divide-by-zero (singularity) errors and produce and invalid (QNaN or Inf) result
`status::overflow`	An overflow happened during the calculation process
`status::underflow`	An underflow happened during the calculation process

API

Syntax

```
uint8_t get_status (queue& exec_queue )
```

`get_status` supports the following devices: CPU and GPU.

Input Parameters

exec_queue The queue where the routine should be executed.

Output Parameters

return value (status) Specifies the VM status.

Examples

```
uint8_t err = get_status (exec_queue);

if (err & status::errdom)
{
    std::cout << ”Errdom status returned” << std::endl;
}


if (err & status::sing)
{
    std::cout << ”Singularity status returned” << std::endl;
}
```

12.3.5 clear_status

Sets the VM Status according to `status::success` and returns the previous VM Status.

Description

The `clear_status` function sets the VM status to `status::success` and returns the previous VM status.

The global VM Status is a single value and it accumulates all errors that happen inside VM functions. The following table lists the possible error values.

Status	Description
Successful Execution	
`status::success`	VM function execution completed successfully
`status::not_defined`	VM status not defined
Warnings	
`status::accuracy_warning`	VM function execution completed successfully in a different accuracy mode
Computational Errors	
`status::errdom`	Values are out of a range of definition producing invalid (QNaN) result
`status::sing`	Values cause divide-by-zero (singularity) errors and produce and invalid (QNaN or Inf) result
`status::overflow`	An overflow happened during the calculation process
`status::underflow`	An underflow happened during the calculation process

API

Syntax

```
namespace oneapi::mkl::vm {
  uint8_t clear_status
  (queue& exec_queue )
}
```

clear_status supports the following devices: CPU and GPU.

Input Parameters

exec_queue The queue where the routine should be executed.

Output Parameters

return value (old_status) Specifies the former VM status.

Examples

```
uint8_t olderr = clear_status (exec_queue);
```

12.3.6 create_error_handler

Creates the local VM Error Handler for a function.

Syntax

Buffer API:

```
namespace oneapi::mkl::vm {
  error_handler<T> create_error_handler(
  buffer<uint8_t,
  1> & errarray,
  int64_t length = 1,
  uint8_t errstatus = status::not_defined,
  T fixup = 0.0,
  bool copysign = false )
}
```

USM API:

```
namespace oneapi::mkl::vm {
  error_handler<T> create_error_handler(
  uint8_t* errarray,
  int64_t length = 1,
  uint8_t errstatus = status::not_defined,
  T fixup = 0.0,
  bool copysign = false )
}
```

`create_error_handler` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Description

The create_error_handler creates the local VM Error handler to be passed to VM functions which support error handling.

The local VM Error Handler supports three modes:

- **Single status mode**: all errors happened during function execution are being written into one status value.

  At the execution end the single value is either un-changed if no errors happened or contained accumulated (merged by bitwise OR) error statuses happened in function execution.

  Set the array pointer to any `status` object and the length equals 1 to enable this mode.

- **Multiple status mode**: error statuses are saved as an array by indices where they happen.

  Notice that only error statuses are being written into the array, the success statuses are not to be written.

  That means the array needs to be allocated and initialized by user before function execution.

  To enable this mode allocate `status` array with the same size as argument and result vectors, set the errarray pointer to it and the length to the vector size.

- **Fixup mode**: for all arguments which caused specific error status results to be overwritten by a user-defined value.

  To enable this mode set desirable errstatus and fixup values. The fixup value is written to results for each argument which caused the errstatus error.

  If the copysign is set to true then fixup value's sign set to the same sign of the argument which caused the errstatus – a suitable option for symmetric math functions.

The following table lists the possible computational error values.

Status	Description
Successful Execution	
`status::success`	VM function execution completed successfully
`status::not_defined`	VM status not defined
Warnings	
`status::accuracy_warning`	VM function execution completed successfully in a different accuracy mode
Computational Errors	
`status::errdom`	Values are out of a range of definition producing invalid (QNaN) result
`status::sing`	Values cause divide-by-zero (singularity) errors and produce and invalid (QNaN or Inf) result
`status::overflow`	An overflow happened during the calculation process
`status::underflow`	An underflow happened during the calculation process

Notes:

- You must allocate and initialize array errarray before calling VM functions in multiple status error handling mode.

  The array should be large enough to contain n error codes, where n is the same as inputoutput vector size for the VM function.

- If no arguments passed to the create_error_handler function, then the empty object is created with all of three error handling modes disabled.

  In this case, the VM math functions set the global error status only.

Input Parameters

errarray Array to store error statuses (should be a buffer for buffer API).

length Length of the errarray. This is an optional argument, default value is `1`.

errcode Error status to fixup results. This is an optional argument, default value is `status::not_defined`.

fixup Fixup value for results. This is an optional argument, default value is `0.0`.

copysign Flag for setting the fixup value's sign the same as the argument's. This is an optional argument, default value `false`.

Output Parameters

return value Specifies the error handler object to be created.

Examples

The following examples are possible usage models (USM API).

Single status mode with create_error_handler():

```
error_handler<float>  handler = vm::create_error_handler (st);
    vm::sin(exec_queue, 1000, a, r, handler);
    if ( st[0] & status::errdom)
    {
        std::cout << ”Errdom status returned” << std::endl;
    }
```

Single status mode without create_error_handler():

```
vm::sin(exec_queue, 1000, a, r, {st });
    std::cout << status << std::endl;
    if ( st[0] & status::errdom)
    {
        std::cout << ”Errdom status returned” << std::endl;
    }
```

The `st` contains either `status::success` or accumulated error statuses if computational errors occured in `vm::erfinv`.

Multiple status mode with `create_error_handler()`:

```
error_handler<float>  handler   = vm::create_error_handler (st, 1000);
    vm::inv(exec_queue, 1000, a, r, handler);
    for(int i=0; i<1000; i++)
        std::cout << st[i] << std::endl;
```

Multiple status mode without `create_error_handler()`:

```
vm::inv(exec_queue, 1000, a, r, {st, 1000});
    for(int i=0; i<1000; i++)
        std::cout << st[i] << std::endl;
```

The `st` array contains all codes for computational errors that occur at the same vector indices `i` as the arguments that caused the errors.

Fixup status mode with `create_error_handler()`:

```
float fixup = 1.0;
    error_handler<float>  handler = vm::create_error_handler (nullptr, 0, status::errdom, fixup,
↪ true);
    vm::erfinv(exec_queue, 1000, a, r, handler);
```

Fixup status mode without create_error_handler():

```
float fixup = 1.0;
    vm::erfinv(exec_queue, 1000, a, r, { nullptr, 0, status::errdom, fixup, true });
```

All results in `r` which computation generated `status::errdom` are replaced by `fixup` values.

In the example above all the `erfinv` function's NAN results caused by greater than |1| arguments are replaced by 1.0 value with the same sign as the corresponding argument.

Mixed (Single and Fixup) status mode with create_error_handler():

```
float   fixup = 1e38;
    error_handler<float> handler = vm::create_error_handler (st, 1, status::overflow, fixup);
    vm::exp(exec_queue, 1000, a, r, handler);
    if ( st & status::underflow)
    {
        std::cout << "Underflow status returned" << std::endl;
    }
```

Mixed (Single and Fixup) status mode without `create_error_handler()`:

```
float   fixup = 1e38;
    vm::exp(exec_queue, 1000, a, r, {st, 1, status::overflow, fixup});
    if ( st & status::underflow)
    {
        std::cout << "Underflow status returned" << std::endl;
    }
```

Mixed (Multiple and Fixup) status mode with `create_error_handler()`:

```
namespace oneapi::mkl::vm {

float   fixup     = 1.0;
    error_handler<float>  handler   = vm::create_error_handler (st, 1000, status::errdom,␣
↪fixup);
    vm::acospi(exec_queue, 1000, a, r, handler);
    for(int i=0; i<1000; i++)
        std::cout << st[i] << std::endl;
```

Mixed (Multiple and Fixup) status mode without `create_error_handler()`:

```
float   fixup     = 1.0;
    vm::acospi(exec_queue, 1000, a, r,{ st, 1000, status::errdom, fixup});
    for(int i=0; i<1000; i++)
        std::cout << st[i] << std::endl;
```

The `st` array contains all codes for computational errors that occur at the same vector indices `i` as the arguments that caused the errors. Additionally, all results in `r` which computation generated `status::errdom` are replaced by `fixup` values.

No local error handling mode:

```
vm::pow(exec_queue, n, a, b, r);
    uint8_t err = vm::get_status (exec_queue);

    if (err & status::errdom)
    {
        std::cout << "Errdom status returned" << std::endl;
    }


    if (err & status::sing)
    {
        std::cout << "Singularity status returned" << std::endl;
    }
```

Only global accumulated error status `err` is set.

12.4 Miscellaneous VM Functions

12.4.1 copysign

Computes the element-wise copy of vector a elements with the sign of vector b elements.

- Description
- API
- Examples

Description

The `copysign(a,b)` function returns the first vector argument elements with the sign changed to match the sign of the second vector argument's corresponding elements.

Argument 1	Argument 2	Result	Error Code
any value	positive value	+any value	
any value	negative value	-any value	

The `copysign(a,b)` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event copysign(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event copysign(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event copysign(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
```

(continues on next page)

(continued from previous page)

```
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event copysign(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`copysign` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `copysign` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vcopysign.cpp
```

12.4.2 nextafter

Computes the element-wise next representable floating-point value of vector a elements in the direction of vector b elements.

- Description
- API
- Examples

Description

The `nextafter(a,b)` function returns a vector containing the next representable floating-point values following the first vector argument elements in the direction of the second vector argument's corresponding elements.

Arguments/Results	Error Code
Input vector argument element is finite and the corresponding result vector element value is infinite	`status::overflow`
Result vector element value is subnormal or zero, and different from the corresponding input vector argument element	`status::underflow`

Even though underflow or overflow can occur, the returned value is independent of the current rounding direction mode.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event nextafter(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event nextafter(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event nextafter(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

```
namespace oneapi::mkl::vm {

sycl::event nextafter(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined,
      oneapi::mkl::vm::error_handler<T> errhandler = {});

}
```

`nextafter` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

errhandler Sets local error handling mode for this function call. See the create_error_handler function for arguments and their descriptions. This is an optional parameter. The local error handler is disabled by default.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `nextafter` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vnextafter.cpp
```

12.4.3 fdim

Computes the element-wise positive difference between vector a elements and vector b elements.

Description

The `fdim(a,b)` function returns a vector containing the differences of the corresponding elements of the first and second vector arguments if the first element is larger, and +0 otherwise.

Argument 1	Argument 2	Result	Error Code
any	QNAN	QNAN	
any	SNAN	QNAN	
QNAN	any	QNAN	
SNAN	any	QNAN	

The `fdim(a,b)` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event fdim(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event fdim(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event fdim(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event fdim(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`fdim` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `fdim` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vfdim.cpp
```

12.4.4 fmax

Computes the element-wise maximum of each pair of vector a elements and vector b elements.

Description

The `fmax(a,b)` function returns a vector with element values equal to the larger value from each pair of corresponding elements of the two vectors a and b: if a < b fmax(a,b) returns b, otherwise fmax(a,b) returns a.

Argument 1	Argument 2	Result	Error Code
a not NAN	NAN	a	
NAN	b not NAN	b	
NAN	NAN	NAN	

The `fmax(a,b)` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event fmax(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event fmax(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event fmax(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event fmax(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

fmax supports the following precisions and devices:

T	Devices supported
float	CPU and GPU
double	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `fmax` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vfmax.cpp
```

12.4.5 fmin

Computes the element-wise minimum of each pair of vector a elements and vector b elements.

Description

The `fmin(a,b)` function returns a vector with element values equal to the smaller value from each pair of corresponding elements of the two vectors a and b: if a > b fmin(a,b) returns b, otherwise fmin(a,b) returns a.

Argument 1	Argument 2	Result	Error Code
a not NAN	NAN	a	
NAN	b not NAN	b	
NAN	NAN	NAN	

The `fmin(a,b)` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event fmin(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event fmin(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);
}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event fmin(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event fmin(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`fmin` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `fmin` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vfmin.cpp
```

12.4.6 maxmag

Computes the element-wise maximum of each pair of vector a elements and vector b elements in terms of magnitude.

Description

The `maxmag(a,b)` function returns a vector with element values equal to the element with the larger magnitude from each pair of corresponding elements of the two vectors a and b:

- If |a| > |b| maxmag(a,b) returns a, otherwise maxmag(a,b) returns b.
- If |b| > |a| maxmag(a,b) returns b, otherwise maxmag(a,b) returns a.
- Otherwise maxmag(a,b) behaves like fmax.

Argument 1	Argument 2	Result	Error Code
a not NAN	NAN	a	
NAN	b not NAN	b	
NAN	NAN	NAN	

The `maxmag(a,b)` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event maxmag(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
```

(continues on next page)

(continued from previous page)

```
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event maxmag(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event maxmag(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event maxmag(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`maxmag` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `maxmag` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vmaxmag.cpp
```

12.4.7 minmag

Computes the element-wise minimum of each pair of vector a elements and vector b elements in terms of magnitude.

Description

The `minmag(a,b)` function returns a vector with element values equal to the element with the smaller magnitude from each pair of corresponding elements of the two vectors a and b:

- If |a| < |b| minmag(a,b) returns a, otherwise minmag(a,b) returns b.
- If |b| < |a| minmag(a,b) returns b, otherwise minmag(a,b) returns a.
- Otherwise minmag behaves like fmin.

Argument 1	Argument 2	Result	Error Code
a not NAN	NAN	a	
NAN	b not NAN	b	
NAN	NAN	NAN	

The `minmag(a,b)` function does not generate any errors.

API

Syntax

Buffer API

```
namespace oneapi::mkl::vm {

sycl::event minmag(sycl::queue & exec_queue,
      std::int64_t n,
      sycl::buffer<T> & a,
      sycl::buffer<T> & b,
      sycl::buffer<T> & y,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event minmag(sycl::queue & exec_queue,
      sycl::buffer<T> & a,
      oneapi::mkl::slice sa,
      sycl::buffer<T> & b,
      oneapi::mkl::slice sb,
      sycl::buffer<T> & y,
      oneapi::mkl::slice sy,
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

USM API

```
namespace oneapi::mkl::vm {

sycl::event minmag(sycl::queue & exec_queue,
      std::int64_t n,
      T const * a,
      T const * b,
      T * y,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

```
namespace oneapi::mkl::vm {

sycl::event minmag(sycl::queue & exec_queue,
      T const * a,
      oneapi::mkl::slice sa,
      T const * b,
      oneapi::mkl::slice sb,
```

(continues on next page)

(continued from previous page)

```
      T * y,
      oneapi::mkl::slice sy,
      std::vector<sycl::event> const & depends = {},
      oneapi::mkl::vm::mode mode = oneapi::mkl::vm::mode::not_defined);

}
```

`minmag` supports the following precisions and devices:

T	Devices supported
`float`	CPU and GPU
`double`	CPU and GPU

Input Parameters

Buffer API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a The buffer containing the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b The buffer containing the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

mode Overrides the global VM mode setting for this function call. See set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

USM API

exec_queue The queue where the routine should be executed.

n Specifies the number of elements to be calculated.

a Pointer to the 1st input vector.

sa Slice selector for a. See Data Types for a description of the oneMKL slice type.

b Pointer to the 2nd input vector.

sb Slice selector for b. See Data Types for a description of the oneMKL slice type.

sy Slice selector for y. See Data Types for a description of the oneMKL slice type.

depends Vector of dependent events (to wait for input data to be ready). This is an optional parameter. The default is an empty vector.

mode Overrides the global VM mode setting for this function call. See the set_mode function for possible values and their description. This is an optional parameter. The default value is `mode::not_defined`.

Output Parameters

Buffer API

y The buffer containing the output vector.

return value (event) Computation end event.

USM API

y Pointer to the output vector.

return value (event) Computation end event.

Examples

An example of how to use `minmag` can be found in the oneMKL installation directory, under:

```
examples/dpcpp/vml/source/vminmag.cpp
```

13.0 Random Number Generators

Intel® oneAPI Math Kernel Library (oneMKL) provides Data Parallel C++ interfaces for the Vector Statistics Random Number Generators (RNG) routines implementing commonly used pseudorandom, quasi-random, and non-deterministic generators with continuous and discrete distributions.

oneMKL RNG includes manual offload functionality similar to other domains (**Random Number Generators Routines**) and also device functionality — a set of functions callable directly from DPC++ kernels (refer to **Random Number Generators Device Routines** for detailed descriptions).

13.1 Definitions

Pseudo-random number generator is defined by a structure(S, μ, f, U, g), where:

- S is a finite set of states (the state space)
- μ is a probability distribution on S for the initial state (or seed) s_0
- f : S→S is the transition function
- U - a finite set of output symbols
- g : S→U an output function

The generation of random numbers is as follows:

1. Generate the initial state (called the seed) s_0 according to μ and compute u_0=g(s_0).
2. Iterate for $i = 1, \cdots, s_i = f(s_{i-1})$ and $u_i = g(s_i)$. Output values u_i are the so-called random numbers produced by the PRNG.

In computational statistics, random variate generation is usually made in two steps:

1. Generating imitations of independent and identically distributed (i.i.d.) random variables having the uniform distribution over the interval (0, 1)
2. Applying transformations to these i.i.d. U(0, 1) random variates in order to generate (or imitate) random variates and random vectors from arbitrary distributions.

All RNG routines can be classified into several categories:

- Engines (Basic random number generators) classes, which hold the state of the generator and is a source of i.i.d. random variables.
- Transformation classes for different types of statistical distributions, for example, uniform, normal (Gaussian), binomial, etc. These classes contain all of the distribution's parameters (including generation method).
- Generate function. The current routine is used to obtain random numbers from a given engine with proper statistics defined by a given distribution.
- Service routines to modify the engine state: skip ahead and leapfrog functions.

13.1.1 Manual Offload RNG Routines

This section represents manual offload RNG routines, including engines and distributions classes templates, generate function, and service routines.

RNG Manual Offload Structure

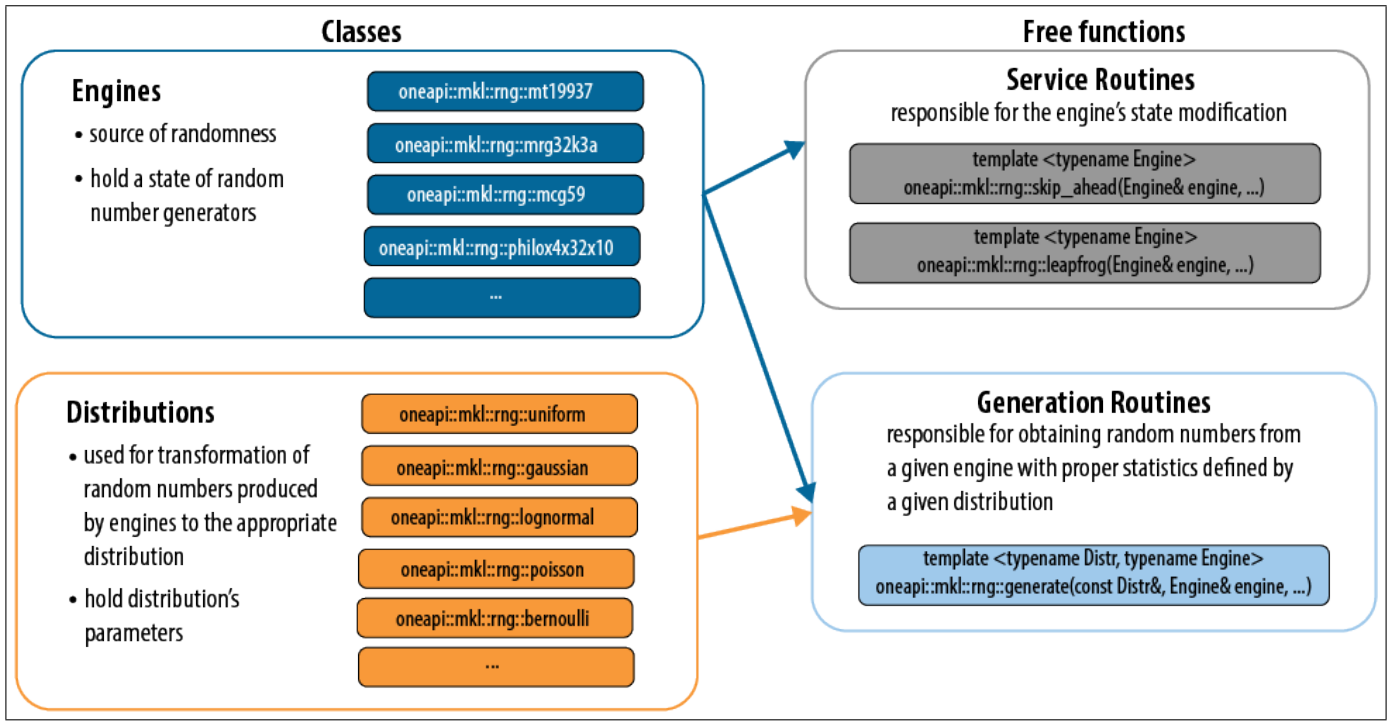


Fig. 8: RNG Manual Offload Structure

Additionally, examples that demonstrate usage of random number generators functionality are available in:

```
${MKL}/examples/dpcpp/rng/source
```

Random Number Generators Routines

This section represents manual offload RNG routines, including engines and distributions classes templates, generate function, and service routines.

oneMKL RNG Usage Model

- Example of RNG Usage

A typical algorithm for random number generators is as follows:

1. Create and initialize the object for basic random number generator.
   - Use the `skip_ahead` or `leapfrog` function if it is required (used in parallel with random number generation for CPU devices).
2. Create and initialize the object for distribution generator.
3. Call the generate routine to get random numbers with appropriate statistical distribution.

The following example demonstrates generation of random numbers that is output from basic generator (engine) PHILOX4X32X10. The seed is equal to 777. The generator is used to generate 10,000 normally distributed random numbers with parameters a = 5 and `sigma`= 2. The purpose of the example is to calculate the sample mean for normal distribution with the given parameters.

Example of RNG Usage

Buffer API

```
#include <iostream>
#include <vector>


#include “CL/sycl.hpp”
#include “oneapi/mkl/rng.hpp”
#define SEED 777


int main() {
    sycl::queue queue;


    const size_t n = 10000;
    std::vector<double> r(n);


    // create basic random number generator object
    oneapi::mkl::rng::philox4x32x10 engine(queue, SEED);
    // create distribution object
    oneapi::mkl::rng::gaussian<double, oneapi::mkl::rng::gaussian_method::icdf> distr(5.0, 2.0);


    {
        // buffer for random numbers
```

(continues on next page)

(continued from previous page)

```
        sycl::buffer<double, 1> r_buf(r.data(), r.size());
        // perform generation
        oneapi::mkl::rng::generate(distr, engine, n, r_buf);


    }


    double s = 0.0;
    for(int i = 0; i < n; i++) {
       s += r[i];
    }
    s /= n;


    std::cout << “Average = ” << s << std::endl;


    return 0;
}
```

USM API

```
#include <iostream>
#include <vector>
#include “CL/sycl.hpp”
#include “oneapi/mkl/rng.hpp”
#define SEED 777


int main() {
  sycl::queue queue;


  const size_t n = 10000;


  // create USM allocator
  sycl::usm_allocator<double, sycl::usm::alloc::shared> allocator(queue);


  // create vector with USM allocator
  std::vector<double, decltype(allocator)> r(n, allocator);


  // create basic random number generator object
  oneapi::mkl::rng::philox4x32x10 engine(queue, SEED);
  // create distribution object
  oneapi::mkl::rng::gaussian<double, oneapi::mkl::rng::gaussian_method::icdf> distr(5.0, 2.0);


```

(continues on next page)

(continued from previous page)

```
  // perform generation
  auto event = oneapi::mkl::rng::generate(distr, engine, n, r.data());


  // sycl::event object is returned by generate function for synchronization
  event.wait();  // synchronization can be also done by queue.wait()


  double s = 0.0;
  for(int i = 0; i < n; i++) {
      s += r[i];
  }
  s /= n;


  std::cout << “Average = ” << s << std::endl;


  return 0;
  }
```

You can also use USM with raw pointers by using the `sycl::malloc_shared` / `sycl::malloc_device` function.

Additionally, examples that demonstrate usage of random number generators functionality are available in:

```
${MKL}/examples/dpcpp/rng/source
```

Device Support

Data Parallel C++ supports several types of devices:

- Host device: Performs computations directly on the current CPU
- CPU device: Performs computations on a CPU using OpenCL™
- GPU device: Performs computations on a GPU

All Data Parallel C++ routines of oneMKL RNG support at least the CPU devices. GPU devices are supported for the following engines:

- `oneapi::mkl::rng::mcg31m1`
- `oneapi::mkl::rng::mcg59`
- `oneapi::mkl::rng::mrg32k3a`
- `oneapi::mkl::rng::mt19937`
- `oneapi::mkl::rng::mt2203`
- `oneapi::mkl::rng::philox4x32x10`

- `oneapi::mkl::rng::sobol`

GPU devices are supported for all distributions.

Note: Some of the distributions use double precision inside the implementations, so not all GPUs are supported for them.

Refer to Engines and Distributions for more detailed descriptions of each routine.

Generate Routine

Use the `oneapi::mkl::rng::generate` routine to obtain random numbers from a given engine with proper statistics of a given distribution.

oneapi::mkl::rng::generate

Description

Entry point to obtain random numbers from a given engine with proper statistics of a given distribution.

API

Syntax

Buffer API

```
template<typename Distr, typename Engine>
  void generate (const Distr& distr,
  Engine& engine,
  std::int64_t n,
  sycl::buffer<typename Distr::result_type, 1>& r)
```

USM API

```
template<typename Distr, typename Engine>
  sycl::event generate (const Distr& distr,
  Engine& engine,
  std::int64_t n,
  typename Distr::result_type* r,
  const std::vector<sycl::event> & dependencies)
```

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
distr	const Distr&	Distribution object. See Distributions for details.
engine	Engine&	Engine object. See Engines for details.
n	`std::int64_t`	Number of random values to be generated.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	const `std::vector <sycl::event>` &	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
r	`sycl::buffer<typename Distr::result_type, 1>`	`sycl::buffer` r to the output vector.

USM API

Name	Type	Description
r	typename `Distr::result_type`	Pointer r to the output vector.
event	`sycl::event`	Function return event after submitting task in `sycl::queue` from the engine.

`oneapi::mkl::rng::generate` submits a kernel into a queue that is held by the engine and fills `sycl::buffer`/typename `Distr::result_type` * vector with n random numbers.

Engines (Basic Random Number Generators)

oneMKL RNG provides pseudorandom, quasi-random, and non-deterministic random number generators for Data Parallel C++:

Routine	Description
oneapi::mkl::rng::mrg32k3a	The combined multiple recursive pseudorandom number generator `MRG32k3a` [L'Ecuyer99a]
oneapi::mkl::rng::philox4x32x10	Philox4x32-10 counter-based pseudorandom number generator with a period of 2^{128} `PHILOX4X32X10` [Salmon11]
oneapi::mkl::rng::mcg31m1	The 31-bit multiplicative congruential pseudorandom number generator MCG(1132489760, 2^{32} -1) [L'Ecuyer99a]
oneapi::mkl::rng::r250	The 32-bit generalized feedback shift register pseudorandom number generator `GFSR(250,103)` [Kirkpatrick81]
oneapi::mkl::rng::mcg59	The 59-bit multiplicative congruential pseudorandom number generator `MCG(`13^{13}`,` 2^{59}`)` from NAG Numerical Libraries [NAG]
oneapi::mkl::rng::wichmann_hill	Wichmann-Hill pseudorandom number generator (a set of 273 basic generators) from NAG Numerical Libraries [NAG]
oneapi::mkl::rng::mt19937	Mersenne Twister pseudorandom number generator `MT19937`[Matsumoto98] with period length 2^{19937}-1 of the produced sequence
oneapi::mkl::rng::mt2203	Set of 6024 Mersenne Twister pseudorandom number generators `MT2203` [Matsumoto98], [Matsumoto00]. Each of them generates a sequence of period length equal to 2^{2203}-1. Parameters of the generators provide mutual independence of the corresponding sequences.
oneapi::mkl::rng::sfmt19937	SIMD-oriented Fast Mersenne Twister pseudorandom number generator `SFMT19937` [Saito08] with a period length equal to 2^{19937}-1 of the produced sequence.
oneapi::mkl::rng::sobol	Sobol quasi-random number generator [Sobol76], [Bratley88], which works in arbitrary dimension.
oneapi::mkl::rng::niederreiter	Niederreiter quasi-random number generator [Bratley92], which works in arbitrary dimension.
oneapi::mkl::rng::ars5	ARS-5 counter-based pseudorandom number generator with a period of 2^{128}, which uses instructions from the AES-NI set `ARS5` [Salmon11].
oneapi::mkl::rng::nondeterministic	Non-deterministic random number generator (RDRAND-based) [AVX], [IntelSWMan]

For some basic generators, oneMKL RNG provides two methods of creating independent states in multiprocessor computations, which are the leapfrog method and the block-splitting method. These sequence splitting methods are also useful in sequential Monte Carlo. The description of these functions can be found in the Service Routines section.

In addition, MT2203 pseudorandom number generator is a set of 6024 generators designed to create up to

6024 independent random sequences, which might be used in parallel Monte Carlo simulations. Another generator that has the same feature is Wichmann-Hill. It allows creating up to 273 independent random streams. The properties of the generators designed for parallel computations are discussed in detail in [Coddington94].

See VS Notes for the detailed description.

oneapi::mkl::rng::mrg32k3a

Description

The combined multiple recursive pseudorandom number generator MRG32k3a [L'Ecuyer99a].

API

Syntax

```
class mrg32k3a {
public:
    static constexpr std::uint32_t default_seed = 1;
    mrg32k3a(sycl::queue queue, std::uint32_t seed = default_seed);
    mrg32k3a(sycl::queue queue, std::initializer_list<std::uint32_t> seed);
    mrg32k3a(const mrg32k3a& other);
    mrg32k3a(mrg32k3a&& other);
    mrg32k3a& operator=(const mrg32k3a& other);
    mrg32k3a& operator=(mrg32k3a&& other);
    ~mrg32k3a();
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of the `oneapi::mkl::rng::generate()` routine submits kernels in this queue.
seed	`std::uint32_t` `std::initializer_list<std::uint32_t>`	Initial conditions of the generator state or engine state.

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::philox4x32x10

Description

A Philox4x32-10 counter-based pseudorandom number generator [Salmon11].

API

Syntax

```
class philox4x32x10 {
public:
    static constexpr std::uint64_t default_seed = 0;
    philox4x32x10(sycl::queue queue, std::uint64_t seed = default_seed);
    philox4x32x10(sycl::queue queue, std::initializer_list<std::uint64_t> seed);
    philox4x32x10(const philox4x32x10& other);
    philox4x32x10(philox4x32x10&& other);
    philox4x32x10& operator=(const philox4x32x10& other);
    philox4x32x10& operator=(philox4x32x10&& other);
    ~philox4x32x10();
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of the `oneapi::mkl::rng::generate()` routine submits kernels in this queue.
seed	`std::uint64_t` `std::initializer_list<std::uint64_t>`	Initial conditions of the generator state or engine state.

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::mcg31m1

Description

The 31-bit multiplicative congruential pseudorandom number generator MCG(1132489760, 2^{32}-1) [L'Ecuyer99a].

API

Syntax

```
class mcg31m1 {
public:
    static constexpr std::uint32_t default_seed = 1;
    mcg31m1(sycl::queue queue, std::uint32_t seed = default_seed);
    mcg31m1(const mcg31m1& other);
    mcg31m1(mcg31m1&& other);
    mcg31m1& operator=(const mcg31m1& other);
    mcg31m1& operator=(mcg31m1&& other);
    ~mcg31m1();
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of the `oneapi::mkl::rng::generate()` routine submits kernels in this queue.
seed	`std::uint32_t`	Initial conditions of the engine.

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::mcg59

Description

The 59-bit multiplicative congruential pseudorandom number generator MCG(13^{13}, 2^{59}) from NAG Numerical Libraries [NAG].

API

Syntax

```
class mcg59 {
public:
    static constexpr std::uint64_t default_seed = 1;
    mcg59(sycl::queue queue, std::uint64_t seed = default_seed);
    mcg59(const mcg59& other);
    mcg59(mcg59&& other);
    mcg59& operator=(const mcg59& other);
    mcg59& operator=(mcg59&& other);
    ~mcg59();
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of `oneapi::mkl::rng::generate()` routine submit kernels in this queue.
seed	`std::uint64_t`	Initial conditions of the engine.

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::r250

Description

The 32-bit generalized feedback shift register pseudorandom number generator GFSR(250,103) [Kirkpatrick81].

API

Syntax

```
class r250 {
public:
    static constexpr std::uint32_t default_seed = 1;
    r250(sycl::queue queue, std::uint32_t seed = default_seed);
    r250(sycl::queue queue, std::vector<std::uint32_t> seed);
    r250(const r250& other);
    r250(r250&& other);
    r250& operator=(const r250& other);
    r250& operator=(r250&& other);
    ~r250();
};
```

Devices supported: CPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of `oneapi::mkl::rng::generate()` routine submit kernels in this queue.
seed	`std::uint32_t` `std::initializer_list<std::uint32_t>`	Initial conditions of the engine.

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::wichmann_hill

Description

Wichmann-Hill pseudorandom number generator (a set of 273 basic generators) from NAG Numerical Libraries [NAG].

API

Syntax

```
class wichmann_hill {
public:
    static constexpr std::uint32_t default_seed = 1;
    wichmann_hill(sycl::queue queue, std::uint32_t seed = default_seed);
    wichmann_hill(sycl::queue queue, std::uint32_t seed, std::uint32_t engine_idx);
    wichmann_hill(sycl::queue queue, std::initializer_list<std::uint32_t> seed);
    wichmann_hill(sycl::queue queue, std::initializer_list<std::uint32_t> seed, std::uint32_t␣
↪engine_idx);
    wichmann_hill(const wichmann_hill& other);
```

(continues on next page)

(continued from previous page)

```
    wichmann_hill(wichmann_hill&& other);
    wichmann_hill& operator=(const wichmann_hill& other);
    wichmann_hill& operator=(wichmann_hill&& other);
    ~wichmann_hill();
};
```

Devices supported: CPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of `oneapi::mkl::rng::generate()` routine submit kernels in this queue.
seed	`std::uint32_t` `std::initializer_list<std::uint32_t>`	Initial conditions of the engine.
engine_idx	`std::uint32_t`	Index of the engine from the set (set contains 273 basic generators)

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::mt19937

Description

Mersenne Twister pseudorandom number generator MT19937 [Matsumoto98] with period length 2^{19937}-1 of the produced sequence.

API

Syntax

```
class mt19937 {
public:
    static constexpr std::uint32_t default_seed = 1;
    mt19937(sycl::queue queue, std::uint32_t seed = default_seed);
    mt19937(sycl::queue queue, std::initializer_list<std::uint32_t> seed);
    mt19937(const mt19937& other);
    mt19937(mt19937&& other);
    mt19937& operator=(const mt19937& other);
    mt19937& operator=(mt19937&& other);
    ~mt19937();
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of `oneapi::mkl::rng::generate()` routine submit kernels in this queue.
seed	`std::uint32_t` `std::initializer_list<std::uint32_t>`	Initial conditions of the engine.

See VS Notes for detailed descriptions.

oneapi::mkl::rng::sfmt19937

- Description
- API

Description

SIMD-oriented Fast Mersenne Twister pseudorandom number generator SFMT19937 [Saito08] with a period length equal to 2^{19937}-1 of the produced sequence.

API

Syntax

```
class sfmt19937 {
public:
    static constexpr std::uint32_t default_seed = 1;
    sfmt19937(sycl::queue queue, std::uint32_t seed = default_seed);
    sfmt19937(sycl::queue queue, std::initializer_list<std::uint32_t> seed);
    sfmt19937(const sfmt19937& other);
    sfmt19937(sfmt19937&& other);
    sfmt19937& operator=(const sfmt19937& other);
    sfmt19937& operator=(sfmt19937&& other);
    ~sfmt19937();
};
```

Devices supported: CPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl queue`, calls of `oneapi::mkl::rng::generate()` routine submit kernels in this queue.
seed	`std::uint32_t` `std::initializer_list<std::uint32_t>`	Initial conditions of the engine.

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::mt2203

Description

Set of 6024 Mersenne Twister pseudorandom number generators MT2203 [Matsumoto98], [Matsumoto00]. Each of them generates a sequence of period length equal to 2^{2203}-1. Parameters of the generators provide mutual independence of the corresponding sequences.

API

Syntax

```
class mt2203 {
public:
    static constexpr std::uint32_t default_seed = 1;
    mt2203(sycl::queue queue, std::uint32_t seed = default_seed);
    mt2203(sycl::queue queue, std::uint32_t seed, std::uint32_t engine_idx);
    mt2203(sycl::queue queue, std::initializer_list<std::uint32_t> seed);
    mt2203(sycl::queue queue, std::initializer_list<std::uint32_t> seed, std::uint32_t engine_
↪idx);
    mt2203(const mt2203& other);
    mt2203(mt2203&& other);
    mt2203& operator=(const mt2203& other);
    mt2203& operator=(mt2203&& other);
    ~mt2203()
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of `oneapi::mkl::rng::generate()` routine submit kernels in this queue.
seed	`std::uint32_t std::initializer_list<std::uint32_t>`	Initial conditions of the engine.
engine_idx	`std::uint32_t`	Index of the engine from the set (set contains 6024 basic generators).

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::ars5

Description

ARS-5 counter-based pseudorandom number generator with a period of 2^{128}, which uses instructions from the AES-NI set ARS5 [Salmon11].

API

Syntax

```
class ars5 {
public:
    static constexpr std::uint64_t default_seed = 0;
    ars5(sycl::queue queue, std::uint64_t seed = default_seed);
    ars5(sycl::queue queue, std::initializer_list<std::uint64_t> seed);
    ars5(const ars5& other);
    ars5(ars5&& other);
    ars5& operator=(const ars5& other);
    ars5& operator=(ars5&& other);
    ~ars5();
};
```

Devices supported: CPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of `oneapi::mkl::rng::generate()` routine submit kernels in this queue.
seed	`std::uint64_t` `std::initializer_list<std::uint64_t>`	Initial conditions of the engine.

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::sobol

- Description
- API

Description

Sobol quasi-random number generator [Sobol76], [Bratley88], which works in arbitrary dimension.

API

Syntax

```
class sobol {
public:
    static constexpr std::uint32_t default_dimensions_number = 1;
    sobol(sycl::queue queue, std::uint32_t dimensions = default_dimensions_number);
    sobol(sycl::queue queue, std::vector<std::uint32_t>& direction_numbers);
    sobol(const sobol& other);
    sobol(sobol&& other);
    sobol& operator=(const sobol& other);
    sobol& operator=(sobol&& other);
    ~sobol();
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of `oneapi::mkl::rng::generate()` routine submit kernels in this queue.
dimensions	`std::uint32_t`	Number of dimensions.
direction_-numbers	`std::vector<std::uint32_t>`	User-defined direction numbers.

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::niederreiter

Description

Niederreiter quasi-random number generator [Bratley92], which works in arbitrary dimension.

API

Syntax

```
class niederreiter {
public:
    static constexpr std::uint32_t default_dimensions_number = 1;
    niederreiter(sycl::queue queue, std::uint32_t dimensions = default_dimensions_number);
    niederreiter(sycl::queue queue, std::vector<std::uint32_t>& irred_polynomials);
    niederreiter(const niederreiter& other);
    niederreiter(niederreiter&& other);
    niederreiter& operator=(const niederreiter& other);
    niederreiter& operator=(niederreiter&& other);
    ~niederreiter();
};
```

Devices supported: CPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of `oneapi::mkl::rng::generate()` routine submit kernels in this queue.
dimensions	`std::uint32_t`	Number of dimensions.
irred_poly-nomials	`std::vector<std::uint32_t>`	User-defined direction numbers.

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::nondeterministic

Description

Non-deterministic random number generator (RDRAND-based) [AVX], [IntelSWMan].

API

Syntax

```
class nondeterministic {
public:
    nondeterministic(sycl::queue queue);
    nondeterministic(const nondeterministic& other);
    nondeterministic(nondeterministic&& other);
    nondeterministic& operator=(const nondeterministic& other);
    nondeterministic& operator=(nondeterministic&& other);
    ~nondeterministic();
};
```

Devices supported: CPU.

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
queue	`sycl::queue`	Valid `sycl::queue`, calls of `oneapi::mkl::rng::generate()` routine submits kernels in this queue.

See VS Notes for the detailed descriptions.

Service Routines

Routine	Description
oneapi::mkl::rng::leapfrog	Proceed state of engine by the leapfrog method to generate a subsequence of the original sequence
oneapi::mkl::rng::skip_ahead	Proceed state of engine by the skip-ahead method to skip a given number of elements from the original sequence
oneapi::mkl::rng::save_state	Functionality to save the engine state in binary format to the file or memory buffer
oneapi::mkl::rng::load_state	Functionality to load the state of the random number engine from the provided memory buffer or file, then creates new engine object
oneapi::mkl::rng::get_state_size	Function to get the size in bytes which is needed to store the state of random number engine

oneapi::mkl::rng::leapfrog

Proceed state of engine using the leapfrog method.

- Description
- API

Description

The `oneapi::mkl::rng::leapfrog` function generates random numbers in an engine with non-unit stride. This feature is particularly useful in distributing random numbers from the original stream across the stride buffers without generating the original random sequence with subsequent manual distribution.

One of the important applications of the leapfrog method is splitting the original sequence into non-overlapping subsequences across stride computational nodes. The function initializes the original random stream (see the following figure) to generate random numbers for the computational node `idx, 0 ≤idx < stride`, where `stride` is the largest number of computational nodes used.

Leapfrog Method

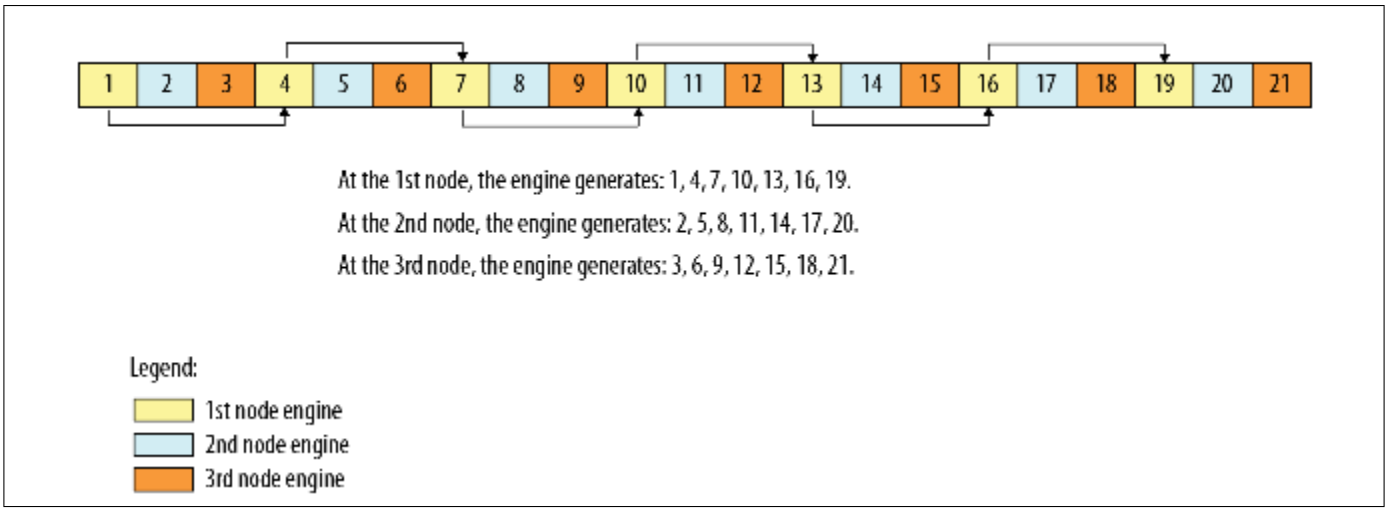


Fig. 9: Leapfrog Method

The leapfrog method is supported only for those basic generators that allow splitting elements by the leapfrog method, which is more efficient than simply generating them by a generator with subsequent manual distribution across computational nodes. See VS Notes for details.

The following code illustrates the initialization of three independent streams using the leapfrog method:

Code for Leapfrog Method

```
1   // Creating 3 identical engines
2   mkl::rng::mcg31m1 engine_1(queue, seed);
3
4
5   mkl::rng::mcg31m1 engine_2(engine_1);
6   mkl::rng::mcg31m1 engine_3(engine_1);
7
8
9   // Leapfrogging the states of engines
10  mkl::rng::leapfrog(engine_1, 0 , 3);
```

(continues on next page)

(continued from previous page)

```
   mkl::rng::leapfrog(engine_2, 1 , 3);
   mkl::rng::leapfrog(engine_3, 2 , 3);
   // Generating random numbers
   ...
```

API

Syntax

```
template<typename Engine>
void leapfrog (Engine& engine,
  std::uint64_t idx,
  std::uint64_t stride)
```

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
engine	`Engine`	Object of engine class, which supports leapfrog.
idx	`std::uint64_t`	Index of the computational node.
stride	`std::uint64_t`	Largest number of computational nodes, or stride.

oneapi::mkl::rng::skip_ahead

Proceeds the state of the engine using the skip-ahead method. The `oneapi::mkl::rng::skip_ahead` function supports the following interfaces to apply the skip-ahead method:

- Common interface
- Interface with a partitioned number of skipped elements

- Description
- API

Description

The `oneapi::mkl::rng::skip_ahead` function skips a given number of elements in a random sequence provided by engine. This feature is particularly useful in distributing random numbers from original engine across different computational nodes. If the largest number of random numbers used by a computational node is num_to_skip, then the original random sequence may be split by `oneapi::mkl::rng::skip_ahead` into non-overlapping blocks of `num_to_skip` size so that each block corresponds to the respective computational node. The number of computational nodes is unlimited. This method is known as the block-splitting method or as the skip-ahead method (see the following figure).

Block-Splitting Method

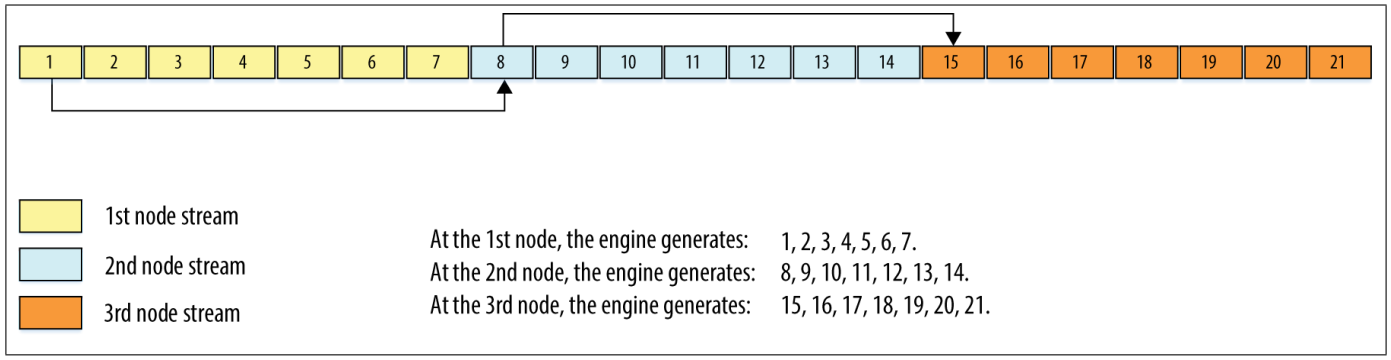


Fig. 10: Block-Splitting Method

The skip-ahead method is supported only for those basic generators that allow skipping elements by the skip-ahead method, which is more efficient than simply generating them by generator with subsequent manual skipping. See VS Notes for details.

Please note that for quasi-random basic generators the skip-ahead method works with components of quasi-random vectors rather than with whole quasi-random vectors. Therefore, to skip NS quasi-random vectors, set the `num_to_skip` parameter equal to the `num_to_skip` *dim, where `dim` is the dimension of the quasi-random vector.

When the number of skipped elements is greater than 2^{63} the interface with the partitioned number of skipped elements is used.

Prior calls to the function represent the number of skipped elements with the list of size n as shown below:

`num_to_skip`[0]+ `num_to_skip`[1]*2^{64}+ `num_to_skip`[2]* 2^{128}+ … +`num_to_skip`[n-1]*$2^{64*(n-1)}$;

When the number of skipped elements is less than 2^{63} both interfaces can be used.

The following code illustrates how to initialize three independent streams using the `oneapi::mkl::rng::skip_ahead` function:

Code for Block-Splitting Method

```
...
// Creating 3 identical engines
oneapi::mkl::rng::mcg31m1 engine_1(queue, seed);
oneapi::mkl::rng::mcg31m1 engine_2(engine_1);
oneapi::mkl::rng::mcg31m1 engine_3(engine_2);


// Skipping ahead by 7 elements the 2nd engine
oneapi::mkl::rng::skip_ahead(engine_2, 7);


// Skipping ahead by 14 elements the 3nd engine
oneapi::mkl::rng::skip_ahead(engine_3, 14);
...
```

Code for Block-Splitting Method with Partitioned Number of Elements

```
// Creating first engine
oneapi::mkl::rng::mrg32k3a engine_1(queue, seed);


// To skip 2^64 elements in the random stream number of skipped elements should be
/represented as num_to_skip = 2^64 = 0 + 1 * 2^64
std::initializer_list<std::uint64_t> num_to_skip = {0, 1};




// Creating the 2nd engine based on 1st. Skipping by 2^64
oneapi::mkl::rng::mrg32k3a engine_2(engine_1);
oneapi::mkl::rng::skip_ahead(engine_2, num_to_skip);


// Creating the 3rd engine based on 2nd. Skipping by 2^64
oneapi::mkl::rng::mrg32k3a engine_3(engine_2);
oneapi::mkl::rng::skip_ahead(engine_3, num_to_skip);
...
```

API

Syntax

Common Interface

```
template<typename Engine>
  void skip_ahead (Engine& engine, std::uint64_t num_to_skip)
```

Interface with Partitioned Number of Skipped Elements

```
template<typename Engine>
  void skip_ahead (Engine& engine,   std::initializer_list<std::uint64_t> num_to_skip)
```

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Common Interface

Name	Type	Description
engine	`Engine`	Object of engine class, which supports the block-splitting method.
num_to_skip	`std::uint64_t`	Number of skipped elements.

Interface with Partitioned Number of Skipped Elements

Name	Type	Description
engine	`Engine`	Object of engine class, which supports the block-splitting method.
num_to_skip	`std::initializer_list<std::uint64_t>`	Partitioned number of skipped elements.

oneapi::mkl::rng::save_state

Writes state of the random number engine to the file or memory buffer.

Description

The `oneapi::mkl::rng::save_state` function allows you to store the state of the random number engine in the binary format in a file or memory buffer.

API

Syntax

Save to Memory Interface

```
template<typename Engine>
void save_state (Engine& engine,
  std::uint8_t* mem);
```

Save to File Interface

```
template<typename Engine>
void save_state (Engine& engine,
 const std::string& filename);
```

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Save to Memory Interface

Name	Type	Description
engine	`Engine&`	Object of engine class, which state would be saved.
mem	`std::uint8_t*`	Memory, which you allocate to store the engine's state. To check the size of memory in bytes needed for the particular engine, use the `oneapi::mkl::rng::get_state_size` function.

Save to File Interface

Name	Type	Description
engine	`Engine&`	Object of engine class, which state would be saved.
filename	`const std::string&`	Name of the file where the engine's state would be written.

oneapi::mkl::rng::load_state

Loads the state of the random number engine from the provided memory buffer or file, then creates new engine object.

Description

The `oneapi::mkl::rng::load_state` function allows you to create a new random number engine object from the binary state of another engine, which was written in a memory buffer or file. You can use different `sycl::queue` objects for the new engine.

API

Syntax

Load from Memory Interface

```
template<typename Engine>
Engine load_state (const sycl::queue& queue,
  const std::uint8_t* mem);
```

Load from File Interface

```
template<typename Engine>
Engine load_state (const sycl::queue& queue,
 const std::string& filename);
```

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Load from Memory Interface

Name	Type	Description
queue	`const sycl::queue& queue`	`sycl::queue` object, which will be used for the newly-created engine.
mem	`const std::uint8_t*`	Memory, where engine's state was stored.

Load from File Interface

Name	Type	Description
queue	`const sycl::queue& queue`	`sycl::queue` object, which will be used for the newly-created engine.
file-name	`const std::string&`	Name of the file where engine's state was written.

Output Parameters

Type	Description
Engine	New random number engine object, which would be created from the given `sycl::queue` and loaded engine's state.

The following code illustrates how to store the engine's state to the `engine_state.dat` file and load it back:

Code for Save/Load state functionality

```
   // Creating GPU engine
   oneapi::mkl::rng::default_engine engine(gpu_queue);

   // Saving state of engine in the file
   oneapi::mkl::rng::save_state(engine, "engine_state.dat");

   // Generating random numbers from the GPU engine
   oneapi::mkl::rng::generate(oneapi::mkl::rng::uniform{}, engine, n, r1);

   // Loading state for the CPU queue
   auto engine_2 = oneapi::mkl::rng::load_state<oneapi::mkl::rng::default_engine>(cpu_queue,
↪"engine_state.dat");

   // Generating random numbers from the CPU engine
   oneapi::mkl::rng::generate(oneapi::mkl::rng::uniform{}, engine_2, n, r2);
```

oneapi::mkl::rng::get_state_size

Returns the size in bytes which is needed to store the state of a given random number engine.

Description

The `oneapi::mkl::rng::get_state_size` function allows you to know the amount of memory needed for storing the engine's state in memory.

API

Syntax

```
template<typename Engine>
std::int64_t get_state_size (Engine& engine);
```

Include Files

- `oneapi/mkl/rng.hpp`

Input Parameters

Name	Type	Description
engine	`Engine&`	Random number engine object, which state size would be calculated.

Output Parameters

Type	Description
`std::int64_t`	Size of the given engine's state in bytes.

The following code illustrates how to use the `oneapi::mkl::rng::get_state_size` function to store the engine's state to memory:

Code for Save/Load state functionality

```
  // Creating GPU engine
  oneapi::mkl::rng::default_engine engine(gpu_queue);

  // Checking how much memory is required to save
  auto mem_size = oneapi::mkl::rng::get_state_size(engine);

  // Allocating memory buffer
  std::uint8_t* mem_buf = new std::uint8_t[mem_size];

  // Saving state of engine in the file
  oneapi::mkl::rng::save_state(engine, mem_buf);

  // Generating random numbers from the GPU engine
  oneapi::mkl::rng::generate(oneapi::mkl::rng::uniform{}, engine, n, r1);

  // Loading state for the CPU queue
  auto engine_2 = oneapi::mkl::rng::load_state<oneapi::mkl::rng::default_engine>(cpu_queue,␣
↪mem_buf);

  // Generating random numbers from the CPU engine
  oneapi::mkl::rng::generate(oneapi::mkl::rng::uniform{}, engine_2, n, r2);

  // Cleaning up memory buffer
  delete[] mem_buf;
```

Distributions

- Modes of Random Number Generation

oneMKL RNG routines are used to generate random numbers with different types of distribution. Each function group is introduced below by the type of underlying distribution and contains a short description of its functionality, as well as specifications of the call sequence and the explanation of input and output parameters. The following tables list the random number generator routines with data types and output distributions, and sets correspondence between data types of the generator routines and the basic random number generators.

Type of Distribution	Data Types	BRNG Data Type	Description
oneapi::mkl::rng::uniform (Continuous)	s, d	s, d	Uniform continuous distribution on the interval [a , b)
oneapi::mkl::rng::gaussian	s, d	s, d	Normal (Gaussian) distribution
oneapi::mkl::rng::gaussian_mv	s, d	s, d	Normal (Gaussian) multivariate distribution

continues on next page

Table 27 – continued from previous page

Type of Distribution	Data Types	BRNG Data Type	Description
oneapi::mkl::rng::exponential	s, d	s, d	Exponential distribution
oneapi::mkl::rng::laplace	s, d	s, d	Laplace distribution (double exponential distribution)
oneapi::mkl::rng::weibull	s, d	s, d	Weibull distribution
oneapi::mkl::rng::cauchy	s, d	s, d	Cauchy distribution
oneapi::mkl::rng::rayleigh	s, d	s, d	Rayleigh distribution
oneapi::mkl::rng::lognormal	s, d	s, d	Lognormal distribution
oneapi::mkl::rng::gumbel	s, d	s, d	Gumbel (extreme value) distribution
oneapi::mkl::rng::gamma	s, d	s, d	Gamma distribution
oneapi::mkl::rng::gamma	s, d	s, d	Beta distribution
oneapi::mkl::rng::chi_square	s, d	s, d	Chi-Square distribution

Type of Distribution	Data Types	BRNG Data Type	Description
oneapi::mkl::rng::uniform (Discrete)	i	**GPU:** s for `oneapi::mkl::rng::uniform_method::standard`, d for `oneapi::mkl::rng::method::accurate` **CPU:** d for both methods	Uniform discrete distribution on the interval `[a,b)`
oneapi::mkl::rng::uniform_bits	i	i	Uniformly distributed bits in 32-bit chunks
	i	i	Uniformly distributed bits in 64-bit chunks
oneapi::mkl::rng::bits	i	i	Bits of underlying BRNG integer recurrence
oneapi::mkl::rng::bernoulli	i	s	Bernoulli distribution
oneapi::mkl::rng::geometric	i	s	Geometric distribution
oneapi::mkl::rng::binomial	i	d	Binomial distribution
oneapi::mkl::rng::hypergeometric	i	d	Hypergeometric distribution

continues on next page

Table 28 – continued from previous page

Type of Distribution	Data Types	BRNG Data Type	Description
oneapi::mkl::rng::poisson	i	s (for) oneapi::mkl::rng::poisson_method::gaussian_icdf_based s (for distribution parameter $\lambda \geq 27$) and d (for $\lambda < 27$) (for oneapi::mkl::rng::poisson_method::ptpe)	Poisson distribution
oneapi::mkl::rng::poisson_v	i	d	Poisson distribution with varying mean
oneapi::mkl::rng::negative_binomial	i	d	Negative binomial distribution, or Pascal distribution
oneapi::mkl::rng::multinomial	i	CPU - s GPU - d	Multinomial distribution

Modes of Random Number Generation

The library provides two modes of random number generation, accurate and fast. Accurate generation mode is intended for the applications that are highly demanding to accuracy of calculations. When used in this mode, the generators produce random numbers lying completely within definitional domain for all values of the distribution parameters. For example, random numbers obtained from the generator of continuous distribution that is uniform on interval [a,b] belong to this interval irrespective of what a and b values may be. Fast mode provides high performance of generation and also guarantees that generated random numbers belong to the definitional domain except for some specific values of distribution parameters. The generation mode is set by specifying relevant value of the method parameter in generator routines. List of distributions that support accurate mode of generation is given in the table below.

Distribution	Distribution Method
oneapi::mkl::rng::uniform (Discrete)	oneapi::mkl::rng::unform_method::accurate
oneapi::mkl::rng::exponential	oneapi::mkl::rng::exponential_method::icdf_accurate
oneapi::mkl::rng::weibull	oneapi::mkl::rng::weibull_method::icdf_accurate
oneapi::mkl::rng::rayleigh	oneapi::mkl::rng::rayleigh_method::icdf_accurate
oneapi::mkl::rng::lognormal	oneapi::mkl::rng::lognormal_method::icdf_accurate oneapi::mkl::rng::lognormal_method::box_muller2_accurate
oneapi::mkl::rng::gamma	oneapi::mkl::rng::gamma_method::marsaglia_accurate
oneapi::mkl::rng::gamma	oneapi::mkl::rng::beta_method::cja_accurate

Distributions Template Parameter Method

Method Type	Distributions	Math Description
`uniform_method::standard` `uniform_method::accurate`	`uniform`	Standard method. Currently there is only one method for these functions. `uniform_method::accurate` checks for additional `s` and `d` data types. For `integer` data types, it uses `d` as a `BRNG` data type (`s` `BRNG` data type is used in `uniform_method::standard` method on GPU).
`gaussian_method::box_muller`	`gaussian`	Generates normally distributed random number x thru the pair of uniformly distributed numbers u1 and u2 according to the formula: $x = \sqrt{-2 \ln u_1} \sin 2\pi u_2$
`gaussian_method::box_muller2`	`gaussian`	Generates normally distributed random numbers x1 and x2 thru the pair of uniformly distributed numbers u1 and u2 according to the formulas: $x_1 = \sqrt{-2 \ln u_1} \sin 2\pi u_2$ $x_2 = \sqrt{-2 \ln u_1} \cos 2\pi u_2$ Lognormal distribution: generated normally distributed random numbers x1 and x2 are converted to lognormal distribution.
`gaussian_method::icdf` `geometric_-method::icdf`	`gaussian` `geometric`	Inverse cumulative distribution function (ICDF) method.
`exponential_method::icdf` `exponential_method::icdf_-accurate`	`exponential`	Inverse cumulative distribution function (ICDF) method.
`weibull_method::icdf` `weibull_method::icdf_accurate`	`weibull`	Inverse cumulative distribution function (ICDF) method.
`cauchy_method::icdf`	`cauchy`	Inverse cumulative distribution function (ICDF) method.
`rayleigh_method::icdf` `rayleigh_method::icdf_accurate`	`rayleigh`	Inverse cumulative distribution function (ICDF) method.
`lognormal_method::icdf` `lognormal_method::icdf_accurate`	`lognormal`	Inverse cumulative distribution function (ICDF) method.
`lognormal_method::box_muller2` `lognormal_method::box_muller2_-accurate`	`lognormal`	Normally distributed random numbers x1 and x2 are produced through the pair of uniformly distributed numbers u1 and u2 according to the formulas: $x_1 = -2 \ln u_1 \sin 2\pi u_2$ $x_2 = -2 \ln u_1 \cos 2\pi u_2$ Then x1 and x2 are converted to lognormal distribution.

continues on next page

Table 30 – continued from previous page

Method Type	Distributions	Math Description
gumbel_method::icdf	gumbel	Inverse cumulative distribution function (ICDF) method.
bernoulli_method::icdf	bernoulli	Inverse cumulative distribution function (ICDF) method.
gamma_method::marsaglia gamma_method::marsaglia_accurate	gamma	For α > 1, a gamma distributed random number is generated as a cube of properly scaled normal random number; for 0.6 ≤α < 1, a gamma distributed random number is generated using rejection from Weibull distribution; for α < 0.6, a gamma distributed random number is obtained using transformation of exponential power distribution; for α = 1, gamma distribution is reduced to exponential distribution.
beta_method::cja beta_method::cja_accurate	beta	Cheng-Johnk-Atkinson method. For min(p, q) > 1, Cheng method is used; for min(p, q) < 1, Johnk method is used, if q + K·p2+ C≤ 0 (K = 0.852..., C=-0.956...) otherwise, Atkinson switching algorithm is used; for max(p, q) < 1, method of Johnk is used; for min(p, q) < 1, max(p, q)> 1, Atkinson switching algorithm is used (CJA stands for Cheng, Johnk, Atkinson); for p = 1or q = 1, inverse cumulative distribution function method is used; for p = 1 and q = 1, beta distribution is reduced to uniform distribution.
chi_square_method::gamma_based	chi_square	(most common): If ν ≥ 17 or ν is odd and 5 ≤ ν ≤ 15, a chi-square distribution is reduced to a Gamma distribution with these parameters: Shape α = ν / 2 Offset a = 0 Scale factor β = 2. The random numbers of the Gamma distribution are generated.
gaussian_mv_method::box_muller gaussian_mv_method::box_muller2 gaussian_mv_method::icdf	gaussian_mv	BoxMuller method for multivariate Gaussian distribution. BoxMuller_2 method for multivariate Gaussian distribution. Inverse cumulative distribution function (ICDF) method.

continues on next page

Table 30 – continued from previous page

Method Type	Distributions	Math Description
`binomial_method::btpe`	`binomial`	Acceptance/rejection method for `ntrial·min(p, 1p) ≥ 30` with decomposition into four regions: Two parallelograms Triangle Left exponential tail Right exponenetial tail
`poisson_method::ptpe`	`poisson`	Acceptance/rejection method for λ≥ 27 with decomposition into four regions: Two parallelograms Triangle Left exponential tail Right exponenetial tail
`poisson_method::gaussian_icdf_based` `poisson_v_method::gaussian_icdf_based`	`poisson` `poisson_v`	for λ≥ 1, method based on Poisson inverse CDF approximation by Gaussian inverse CDF; for λ < 1, table lookup method is used.
`hypergeometric_method::h2pe`	`hypergeometric`	Acceptance/rejection method for large mode of distribution with decomposition into three regions: Rectangular Left exponential tail Right exponential tail
`negative_binomial_method::nbar`	`negative_binomial`	Acceptance/rejection method for: $\frac{(a-1)\cdot(1-p)}{p} \geq 100$with decomposition into five regions: Rectangular (2) trapezoid Left exponential tail Right exponential tail
`multinomial_method::poisson_icdf_based`	`multinomial`	Multinomial distribution with parameters `m`, `k`, and a probability vector `p`. Random numbers of the multinomial distribution are generated by Poisson Approximation method.

oneapi::mkl::rng::uniform (Continuous)

Generates random numbers with uniform distribution.

Description

The class object is used in `oneapi::mkl::rng::generate` function to provide random numbers uniformly distributed over the interval [a, b), where a, b are the left and right bounds of the interval, respectively, and a, b∈R ; a < b.

The probability density function is given by:

$$F_{a,b}(x) = \begin{cases} \frac{1}{b-a}, & x \in [a,b) \\ 1, & x \notin [a,b) \end{cases}, -\infty < x < +\infty$$

The cumulative distribution function is as follows:

$$F_{a,b}(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \le x < b \\ 1, & x \ge b \end{cases}, -\infty < x < +\infty$$

API

Syntax

```
template<typename Type = float, typename Method = uniform_method::by_default>
class uniform {public:
using method_type = Method;
using result_type = Type;
uniform(): uniform(static_cast<Type>(0.0),
static_cast<Type>(1.0));
explicit uniform(Type a, Type b);
explicit uniform(const param_type& pt);
Type a() const;
Type b() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename Type = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneapi::mkl::rng::uniform_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::uniform_method::standard` `oneapi::mkl::rng::uniform_method::accurate` See brief descriptions of the methods in Distributions Template Parameter Method

Input Parameters

Name	Type	Description
a	`Type (float, double)`	Left bound a
b	`Type (float, double)`	Right bound b

oneapi::mkl::rng::gaussian

Generates normally distributed random numbers.

Description

The class object is used in `oneapi::mkl::rng::generate` function to provide random numbers with normal (Gaussian) distribution with mean (a) and standard deviation (`stddev, σ`), where a, $\sigma \in R; \sigma > 0$.

The probability density function is given by:

$$f_{a,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-a)^2}{2\sigma^2}\right) dy, -\infty < x < +\infty$$

The cumulative distribution function is as follows:

$$F_{a,\sigma}(x) = \int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-a)^2}{2\sigma^2}\right) dy, -\infty < x < +\infty$$

The cumulative distribution function $Fa, \sigma(x)$ can be expressed in terms of standard normal distribution $\phi(x)$ as $Fa, \sigma(x) = \phi((x-a)/\sigma)$

API

Syntax

```
template<typename RealType = float, typename Method = gaussian_method::by_default>
class gaussian {
public:
using method_type = Method;
using result_type = RealType;
gaussian(): gaussian(static_cast<RealType>(0.0), static_cast<RealType>(1.0));
explicit gaussian(RealType mean, RealType stddev);
explicit gaussian(const param_type& pt);
RealType mean() const;
RealType stddev() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

Name	Description
`typename RealType = float`	Type of the produced values. The specific values are as follows: `float` `double`
`typename Method = ``oneapi::mkl::rng::gaussian_-method::by_-default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::gaussian_method::box_muller` `oneapi::mkl::rng::gaussian_method::box_muller2` `oneapi::mkl::rng::gaussian_method::icdf` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
mean	`RealType (float, double)`	Mean value a.
stddev	`RealType (float, double)`	Standard deviation σ.

oneapi::mkl::rng::exponential

Generates exponentially distributed random numbers.

Description

The `oneapi::mkl::rng::exponential` class object is used in `oneapi::mkl::rng::generate` function to provide random numbers with exponential distribution that has displacement a and scalefactor β, where $a, \beta \in R; \beta > 0$.

The probability density function is given by:

$$F_{a,\beta}(x) = \begin{cases} \frac{1}{\beta}\exp(-\frac{(x-a)}{\beta}), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

The cumulative distribution function is as follows:

$$F_{a,\beta}(x) = \begin{cases} 1 - \exp(-\frac{(x-a)}{\beta}), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

API

Syntax

```
template<typename RealType = float, typename Method = exponential_method::by_default>
class exponential {
public:
  using method_type = Method;
  using result_type = RealType;
  exponential(): exponential((RealType)0.0, (RealType)1.0){}
  explicit exponential(RealType mean, RealType stddev);
  explicit exponential(const param_type& pt);
  RealType a() const;
  RealType beta() const;
  param_type param() const;
  void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneapi::mkl::rng::exponential_method::by_default`	Generation method. The specific values are as follows:
	`oneapi::mkl::rng::exponential_method::icdf`
	`oneapi::mkl::rng::exponential_method::icdf_accurate`
	See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
a	`RealType (float, double)`	Displacement a.
beta	`RealType (float, double)`	Scalefactor β.

oneapi::mkl::rng::laplace

Generates random numbers with Laplace distribution.

Description

The `oneapi::mkl::rng::laplace` class object is used in the `oneapi::mkl::rng::generate` function to provide random numbers with Laplace distribution with mean value (or average) a and scalefactor (b, β), where $a, \beta \in R; \beta > 0$. The scalefactor value determines the standard deviation as

$$\sigma = \beta\sqrt{2}$$

The probability density function is given by:

$$F_{a,\beta}(x) = \frac{1}{\sqrt{2\beta}} \exp\left(-\frac{|x-a|}{\beta}\right), -\infty < x < +\infty$$

The cumulative distribution function is as follows:

$$F_{\alpha,\beta}(x) = \begin{cases} \frac{1}{2}\exp\left(-\frac{|x-\alpha|}{\beta}\right), & x \geq \alpha \\ 1 - \frac{1}{2}\exp\left(-\frac{|x-\alpha|}{\beta}\right), & x < \alpha \end{cases}, -\infty < x < +\infty$$

API

Syntax

```
template<typename RealType = float, typename Method = laplace_method::by_default>
class laplace {
public:
using method_type = Method;
using result_type = RealType;
laplace(): laplace((RealType)0.0, (RealType)1.0){}
explicit laplace(RealType a, RealType b);
explicit laplace(const param_type& pt);
RealType a() const;
```

(continues on next page)

(continued from previous page)

```
RealType b() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneapi::mkl::rng::laplace_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::laplace_method::icdf` See brief descriptions of the methods in Distributions Template Parameter Method

Input Parameters

Name	Type	Description
a	`RealType (float, double)`	Mean value a.
b	`RealType (float, double)`	Scalefactor b.

oneapi::mkl::rng::weibull

Generates Weibull distributed random numbers.

- Description
- API

Description

The `oneapi::mkl::rng::weibull` class object is used in the `oneapi::mkl::rng::generate` function to provide Weibull distributed random numbers with displacement a, scalefactor β, and shape α, where $\alpha, \beta, a \in R; \alpha > 0, \beta > 0$.

The probability density function is given by:

$$F_{a,\alpha,\beta}(x) = \begin{cases} \frac{\alpha}{\beta^{\alpha}}(x-a)^{\alpha-1} \exp\left(-\left(\frac{x-a}{\beta}\right)^{\alpha}\right), & x \geq a \\ 0, & x < a \end{cases}$$

The cumulative distribution function is as follows:

$$F_{a,\alpha,\beta}(x) = \begin{cases} 1 - \exp\left(-\left(\frac{x-a}{\beta}\right)^{\alpha}\right), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
namespace oneapi::mkl::rng {
template<typename RealType = float, typename Method = weibull_method::by_default>
class weibull {
public:
    using method_type = Method;
    using result_type = RealType;
    weibull(): weibull((Real_Type)1.0, (RealType)0.0, (RealType)1.0){}
    explicit weibull(RealType alpha, RealType a, RealType beta);
    explicit weibull(const param_type& pt);
    RealType alpha() const;
    RealType a() const;
    RealType beta() const;
    param_type param() const;
    void param(const param_type& pt);
};
}
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneapi::mkl::rng::weibull_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::weibull_method::icdf` `oneapi::mkl::rng::weibull_method::icdf_accurate` See brief descriptions of the methods in Distributions Template Parameter Method

Input Parameters

Name	Type	Description
alpha	`RealType (float, double)`	Shape .
a	`RealType (float, double)`	Displacement a.
beta	`RealType (float, double)`	Scalefactor .

oneapi::mkl::rng::cauchy

Generates Cauchy distributed random values.

- Description
- API

Description

The `oneapi::mkl::rng::cauchy` class object is used in the `oneapi::mkl::rng::generate` function to provide Cauchy distributed random numbers with displacement (a) and scalefactor (b, β), where $a, \beta \in R; \beta > 0$.

The probability density function is given by:

$$F_{a,\beta}(x) = \frac{1}{\pi\beta\left(1+\left(\frac{x-a}{\beta}\right)^2\right)}, -\infty < x < +\infty$$

The cumulative distribution function is as follows:

$$F_{a,\beta}(x) = \frac{1}{2} + \frac{1}{\pi}\arctan(\frac{x-a}{\beta}), -\infty < x < +\infty$$

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
template<typename RealType = float,
  typename Method = cauchy_method::by_default>
  class cauchy {
  public:
  using method_type = Method;
  using result_type = RealType;
  cauchy(): cauchy((RealType)0.0, (RealType)1.0){}
  explicit cauchy(RealType a, RealType b);
  explicit cauchy(const param_type& pt);
  RealType a() const;
  RealType b() const;
  param_type param() const;
  void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneapi::mkl::rng::cauchy_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::cauchy_method::icdf` See brief descriptions of the methods in Distributions Template Parameter Method

Input Parameters

Name	Type	Description
a	`RealType (float, double)`	Displacement a.
b	`RealType (float, double)`	Scalefactor b.

oneapi::mkl::rng::rayleigh

Generates Rayleigh distributed random values.

Description

The `oneapi::mkl::rng::rayleigh` class object is used by the `oneapi::mkl::rng::generate` function to provide Rayleigh distributed random numbers with displacement (a) and scalefactor (b, β), where $a, \beta \in R; \beta > 0$.

The Rayleigh distribution is a special case of the **Weibull** distribution, where the shape parameter α = 2.

The probability density function is given by:

$$f_{a,\beta}(x) = \begin{cases} \frac{2(x-a)}{\beta^2} \exp(-\frac{(x-a)^2}{\beta^2}), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

The cumulative distribution function is as follows:

$$F_{\alpha,\beta}(x) = \begin{cases} 1 - \exp\left(-\frac{(x-\alpha)^2}{\beta^2}\right), & x \geq \alpha) \\ 0, & x < \alpha \end{cases}, -\infty < x < +\infty$$

API

Syntax

```
template<typename RealType = float, typename Method = rayleigh_method::by_default>
class rayleigh {
public:
using method_type = Method;
using result_type = RealType;
rayleigh(): rayleigh((RealType)0.0, (RealType)1.0){}
explicit rayleigh(RealType a, RealType b);
explicit rayleigh(const param_type& pt);
RealType a() const;
RealType b() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneapi::mkl::rng::rayleigh_method::by_default`	Generation method. The specific values are as follows:
	`oneapi::mkl::rng::rayleigh_method::icdf`
	`oneapi::mkl::rng::rayleigh_method::icdf_accurate`
	See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
a	`RealType (float, double)`	Displacement a.
b	`RealType (float, double)`	Scalefactor b.

oneapi::mkl::rng::lognormal

Generates lognormally distributed random numbers.

Description

The `oneapi::mkl::rng::lognormal` class object is used in the `oneapi::mkl::rng::generate` function to provide random numbers with average of distribution (`m`, a) and standard deviation (`s`, σ) of subject normal distribution, displacement (`displ`, b), and scalefactor (`scale`, β), where `a, σ, b, β ∈ R ; σ > 0 , β > 0`.

The probability density function is given by:

$$F_{a,\sigma,b,\beta}(x) = \begin{cases} \frac{1}{\sigma(x-b)\sqrt{2\pi}} \exp\left(-\frac{\ln(\frac{x-b}{\beta})-a)^2}{2\sigma^2}\right), & x > b \\ 0, & x \le b \end{cases}$$

The cumulative distribution function is as follows:

$$F_{a,\sigma,b,\beta}(x) = \begin{cases} \Phi\left(\frac{\ln(\frac{x-b}{\beta})-a}{\sigma}\right), & x > b \\ 0, & x \le b \end{cases}$$

API

Syntax

```
template<typename RealType = float, typename Method = lognormal_method::by_default>
class lognormal {
public:
using method_type = Method;
using result_type = RealType;
lognormal(): lognormal((RealType)0.0, (RealType)1.0, (RealType) 0.0,
(RealType)1.0){} explicit lognormal(RealType m, RealType s, RealType displ
= RealType) 0.0, RealType scale = (RealType)1.0);
explicit lognormal(const param_type& pt);
RealType m() const;
RealType s() const;
RealType displ() const;
RealType scale() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneapi::mkl::rng::lognormal_method::by_default`	Generation method. The specific values are as follows:
	`oneapi::mkl::rng::lognormal_method::box_muller2`
	`oneapi::mkl::rng::lognormal_method::box_muller2_accurate`
	`oneapi::mkl::rng::lognormal_method::icdf`
	`oneapi::mkl::rng::lognormal_method::icdf_accurate`
	See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
m	`RealType (float, double)`	Average a of the subject normal distribution.
s	`RealType (float, double)`	Standard deviation σ of the subject normal distribution.
displ	`RealType (float, double)`	Displacement `displ`.
scale	`RealType (float, double)`	Scalefactor `scale`.

oneapi::mkl::rng::gumbel

Generates Gumbel distributed random values.

Description

The `oneapi::mkl::rng::gumbel` class object is used in the `oneapi::mkl::rng::generate` function to provide Gumbel distributed random numbers with displacement (a) and scalefactor (b, β), where $a, \beta \in R; \beta > 0$.

The probability density function is given by:

$$f_{a,\beta}(x) = \frac{1}{\beta} \exp\left(\frac{x-a}{\beta}\right) \exp(-\exp\left(\frac{x-a}{\beta}\right)), -\infty < x < +\infty$$

The cumulative distribution function is as follows:

$$F_{\alpha,\beta}(x) = 1 - \exp(-\exp(\frac{x-\alpha}{\beta})), -\infty < x < +\infty$$

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
template<typename RealType = float, typename Method = gumbel_method::by_default>
class gumbel {
public:
using method_type = Method; using result_type =
RealType; gumbel(): gumbel((RealType)0.0,
(RealType)1.0){} explicit gumbel(RealType a,
RealType b);
explicit gumbel(const param_type& pt);
RealType a() const;
RealType b() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

Name	Description

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneapi::mkl::rng::gumbel_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::gumbel_method::icdf` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
a	`RealType (float, double)`	Displacement a.
b	`RealType (float, double)`	Scalefactor b.

oneapi::mkl::rng::gamma

Generates gamma distributed random values.

- Description
- API

Description

The `oneapi::mkl::rng::gamma` class object is used in the `oneapi::mkl::rng::generate` function to provide random numbers with gamma distribution that has shape parameter [illegible], displacement a, and scale parameter β, where $\alpha, \beta, and a \in R; \alpha > 0, \beta > 0$.

The probability density function is given by:

$$F_{\alpha,a,\beta}(x) = \begin{cases} \frac{1}{\Gamma(\alpha)\beta^{\alpha}}(x-a)^{\alpha-1}e^{-\frac{(x-a)}{\beta}}, & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

where Γ(α) is the complete gamma function.

The cumulative distribution function is as follows:

$$F_{\alpha,a,\beta}(x) = \begin{cases} \int_a^x \frac{1}{\Gamma(\alpha)\beta^{\alpha}}(y-a)^{\alpha-1}e^{-\frac{(y-a)}{\beta}}dy, & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
template<typename RealType = float, typename Method = gamma_method::by_default>
class gamma {
public:
using method_type = Method; using result_type = RealType;
gamma(): gamma((RealType)1.0, (RealType)0.0, (RealType)1.0){}
explicit gamma(RealType alpha, RealType a, RealType beta);
explicit gamma(const param_type& pt);
RealType alpha() const;
RealType a() const;
RealType beta() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneapi::mkl::rng::gamma_method::by_default`	Generation method. The specific values are as follows:
	`oneapi::mkl::rng::gamma_method::marsaglia`
	`oneapi::mkl::rng::gamma_method::marsaglia_accurate`
	See brief descriptions of the methods in **Distributions Template Parameter Method**.

Input Parameters

Name	Type	Description
alpha	`RealType (float, double)`	Shape
a	`RealType (float, double)`	Displacement a.
beta	`RealType (float, double)`	Scalefactor β.

oneapi::mkl::rng::beta

Generates beta distributed random values.

Description

The `oneapi::mkl::rng::beta` class object is used in the `oneapi::mkl::rng::generate` function to provide random numbers with beta distribution that has shape parameters p and q, displacement a, and scale parameter (b, β), where p, q, a, and $\beta \in R; p > 0, q > 0, \beta > 0$.

The probability density function is given by:

$$f_{p,q,a,\beta}(x) = \begin{cases} \frac{1}{B(p,q)\beta^{p+q-1}}(x-a)^{p-1}(\beta+a-x)^{q-1}, & a \le x < a+\beta \\ 0, & x < a, x \ge a+\beta \end{cases}, -\infty < x < \infty$$

where B(p, q) is the complete beta function.

The cumulative distribution function is as follows:

$$f_{p,q,a,\beta}(x) = \begin{cases} 0, & x < a \\ \int_a^x \frac{1}{B(p,q)\beta^{p+q-1}}(y-a)^{p-1}(\beta+a-y)^{q-1}dy, & a \le x < a+\beta \\ 1, & x \ge a+\beta \end{cases}, -\infty < x < \infty$$

API

Syntax

```
template<typename RealType = float,
  typename Method = beta_method::by_default>
  class beta {
  public:
  using method_type = Method; using result_type = RealType;
  beta(): beta((RealType)1.0, (RealType)1.0, (RealType)(0.0),
  (RealType)(1.0)){} explicit beta(RealType p, RealType q, RealType a,
  RealType b);
  explicit beta(const param_type& pt);
  RealType p() const;
  RealType q() const;
  RealType a() const;
  RealType b() const;
  param_type param() const;
  void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneapi::mkl::rng::beta_method::by_default`	Generation method. The specific values are as follows:
	`oneapi::mkl::rng::beta_method::cja`
	`oneapi::mkl::rng::beta_method::cja_accurate`
	See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
p	`RealType (float, double)`	Shape `p`
q	`RealType (float, double)`	Shape `q`
a	`RealType (float, double)`	Displacement `a`.
b	`RealType (float, double)`	Scalefactor `b`.

oneapi::mkl::rng::chi_square

Generates chi-square distributed random values.

Description

The `oneapi::mkl::rng::chi_square` class object is used in the `oneapi::mkl::rng::generate` function to provide random numbers with chi-square distribution and ν degrees of freedom, n ∈ **N**, n > 0.

The probability density function is:

$$F_v(x) = \begin{cases} \frac{x^{\frac{n-2}{2}} e^{-\frac{x}{2}}}{2^{n/2}\Gamma(\frac{n}{2})} dy, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

The cumulative distribution function is:

$$F_v(x) = \begin{cases} \int_0^x \frac{y^{\frac{n-2}{2}} e^{-\frac{n}{2}}}{2^{n/2}\Gamma(\frac{n}{2})} dy, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

API

Syntax

```
template<typename RealType = float, typename Method = chi_square_method::by_default>
class chi_square {
public:
using method_type = Method;
using result_type = RealType;
```

(continues on next page)

(continued from previous page)

```
chi_square(): chi_square(5){}
explicit chi_square(std::int32_t n);
explicit chi_square(const param_type& pt);
std::int32_t n() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method = oneamkl::rng::chi_square_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::chi_square_method::gamma_based` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
n	`std::int32_t`	Degrees of freedom.

oneapi::mkl::rng::gaussian_mv

Generates random numbers from multivariate normal distribution.

Description

The class object is used in the `oneapi::mkl::rng::generate` function to provide random numbers with d-variate normal (Gaussian) distribution with mean (a) and variance-covariance matrix C, where a $\in R^d$; C is `d x d` symmetric positive-definite matrix.

The probability density function is given by:

$$f_{a,C}(x) = \frac{1}{\sqrt{\det(2\pi C)}} \exp(-\frac{1}{2}(x-a)^T C^{-1}(x-a))$$

where x``∈ ``R^d.

Matrix C can be represented as `C = TTT`, where T is a lower triangular matrix - Cholesky factor of C.

API

Syntax

```
template<typename RealType = float, layout Layout = layout::packed,
         typename Method = gaussian_mv_method::by_default>
class gaussian_mv {
public:
  using method_type = Method;
  using result_type = RealType;
  static constexpr layout layout_type = Layout;
  explicit gaussian_mv(std::uint32_t dimen, std::vector<RealType> mean,
                       std::vector<RealType> matrix); // deprecated since oneMKL 2023.0
  explicit gaussian_mv(std::uint32_t dimen, sycl::span<RealType> mean,
                       sycl::span<RealType> matrix);
  explicit gaussian_mv(const param_type& pt);
  std::uint32_t dimen() const;
  std::vector<RealType> mean() const;
  std::vector<RealType> matrix() const;
  param_type param() const;
  void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

Name	Description

`typename RealType = float`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`layout Layout = layout::packed`	Type of the matrix storage. The specific values are as follows:
	`layout::packed`
	`layout::full`
	`layout::diagonal`
	See brief descriptions of the methods in Distributions Template Parameter Method.

`typename Method = oneapi::mkl::rng::gaussian_mv_method::by_default`	Generation method. The specific values are as follows:
	`oneapi::mkl::rng::gaussian_mv_method::box_muller`
	`oneapi::mkl::rng::gaussian_mv_method::box_muller2`
	`oneapi::mkl::rng::gaussian_mv_method::icdf`
	See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
dimen	`std::uint32_t`	Dimension of output random vectors
mean	`sycl::span< RealType> (float, double`	Mean span a of dimension `d`.
matrix	`sycl::span< RealType> (float, double`	Variance-covariance matrix `C`.

Note:

- `explicit gaussian_mv(std::uint32_t dimen, std::vector<RealType> mean, std::vector<RealType> matrix);` is deprecated and will be removed in one of the next releases. Use `explicit gaussian_mv(std::uint32_t dimen, sycl::span<RealType> mean, sycl::span<RealType> matrix);` instead.
- When passing a `sycl::span` that is constructed over a user's memory to the constructor, users must manage the memory under sycl::span by themselves. They must not destroy the memory while data are processed.

oneapi::mkl::rng::uniform (Discrete)

Generates random numbers uniformly distributed over the interval `[a, b)`.

Description

The `oneapi::mkl::rng::uniform` class object is used in `oneapi::mkl::rng::generate` functions to provide random numbers uniformly distributed over the interval `[a, b)`, where `a, b` are the left and right bounds of the interval respectively, and `a, b∈Z; a < b`.

The probability distribution is given by:

$$P(X = k) = \frac{1}{b-a}$$
$$k \in \{a, a+1, \ldots, b-1\}$$

The cumulative distribution function is as follows:

$$F_{a,b}(x) = \begin{cases} 0, & x < a \\ \frac{x-a+1}{b-a}, & a \le x < b \\ 1, & x \ge b \end{cases}, x \in R$$

API

Syntax

```
template<typename Method>
class uniform<std::(u)int32_t, Method> {
public:
using method_type = Method;
using result_type =  std::(u)int32_t;
uniform(): uniform((std::(u)int32_t )(0),
              std::is_same<Method,
              uniform_method::standard>::value ? (1 << 23)
              :std::numeric_limits<Type>::max()){};
explicit uniform(std::(u)int32_t a,  std::(u)int32_t b);
explicit uniform(const param_type& pt);
std::(u)int32_t a() const;
std::(u)int32_t b() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename Method = oneapi::mkl::rng::uniform_method::by_default`	Generation method. The specific values are as follows:
	`oneapi::mkl::rng::uniform_method::standard`
	`oneapi::mkl::rng::uniform_method::accurate`
	See brief descriptions of the methods in Distributions Template Parameter Method.

Note: The `oneapi::mkl::rng::uniform_method::standard` uses the s BRNG type on GPU devices. This might cause the produced numbers to have incorrect statistics (due to rounding error) when $(abs(b–a) > 2^{23} || abs(b) > 2^{23} || abs(a) > 2^{23})$. To get proper statistics for this case, use the `oneapi::mkl::rng::uniform_method::accurate` method instead.

Input Parameters

Name	Type	Description
a	`std::int32_t` `std::uint32_t`	Left bound a
b	`std::int32_t` `std::uint32_t`	Right bound b

oneapi::mkl::rng::uniform_bits

Generates uniformly distributed bits in 32/64-bit chunks.

- Description
- API

Description

The `oneapi::mkl::rng::uniform_bits` class object is used to generate uniformly distributed bits in 32/64-bit chunks. It is designed to ensure each bit in the 32/64-bit chunk is uniformly distributed. It is not supported not for all engines.

`IUntType` denotes the chunk size and can be `std::uint32_t`, `std::uint64_t`. See VS Notes for details.

API

Syntax

```
template<typename UIntType = std::uint32_t>
   class uniform_bits {
   using result_type = UIntType
   }
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename UIntType = std::uint32_t`	Type of the produced values. The specific values are as follows:
	`std::uint32_t`
	`std::uint64_t`

oneapi::mkl::rng::bits

Generates bits of underlying engine (BRNG) integer reccurence.

Description

The `oneapi::mkl::rng::bits` class object is used to generate integer random values. Each integer can be treated as a vector of several bits. In a truly random generator, these bits are random, while in pseudorandom generators this randomness can be violated.

For example, a drawback of linear congruential generators is that lower bits are less random than higher bits (for example, see [Knuth81]. For this reason, exercise care when using this function. Typically, in a 32-bit LCG only 24 higher bits of an integer value can be considered random. See VS Notes for details.

API

Syntax

```
template<typename UIntType = std::uint32_t>
   class bits {
   using result_type = UIntType
   }
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

Name	Description
`typename UIntType = std::uint32_t`	Type of the produced values. The specific values are as follows: `std::uint32_t`

oneapi::mkl::rng::bernoulli

Generates Bernoulli distributed random values.

- Description
- API

Description

The `oneapi::mkl::rng::bernoulli` class object is used in the `oneapi::mkl::rng::generate` function to provide Bernoulli distributed random numbers with probability p of a single trial success, where $p \epsilon R; 0 \leq p \leq 1$.

A variate is called Bernoulli distributed if after a trial it is equal to 1 with probability of success `p` and to `0` with probability `1 - p`.

The probability distribution is given by:

`P(X=1) = p`

`P(X=0) = 1 - p`

The cumulative distribution function is as follows:

$$F_p(x) = \begin{cases} 0, & x < 0 \\ 1-p, & 0 \leq x < 1, x \in R \\ 1, & x \geq 1 \end{cases}$$

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
template<typename IntType = std::int32_t,
         typename Method = bernoulli_method::by_default>
class bernoulli {
public:
  using method_type = Method;
  using result_type = IntType;
  bernoulli(): bernoulli(0.5f){}
  explicit bernoulli(float p);
  explicit bernoulli(const param_type& pt);
  float p() const;
  param_type param() const;
  void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename _IntType = std::int32_t`	Type of the produced values. The specific values are as follows:
	`std::int32_t`
	`std::uint32_t`

`typename Method = oneapi::mkl::rng::bernoulli_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::bernoulli_method::icdf` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
p	`float`	Success probability p of a trial.

oneapi::mkl::rng::geometric

Generates geometrically distributed random values.

Description

The `oneapi::mkl::rng::geometric` class object is used in the `oneapi::mkl::rng::generate` function to provide geometrically distributed random numbers with probability p of a single trial success, where `p∈R; 0 < p < 1`.

A geometrically distributed variate represents the number of independent Bernoulli trials preceding the first success. The probability of a single Bernoulli trial success is p.

The probability distribution is given by:

$P(X = k) = p \cdot (1 - p)^k$, $k \in \{0,1,2, \ldots\}$.

The cumulative distribution function is as follows:

$$F_p(x) = \begin{cases} 0, & x < 0 \\ 1 - (1-p)^{\lfloor x+1 \rfloor}, & x \geq 0 \end{cases}, x \in R$$

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
template<typename IntType = std::int32_t, typename Method = geometric_method::by_default>
class geometric {
public:
using method_type = Method;
using result_type = IntType;
geometric(): geometric(0.5){} explicit geometric(float p);
explicit geometric(const param_type& pt);
float p() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

Name	Description

`typename IntType = std::int32_t`	Type of the produced values. The specific values are as follows:
	`std::int32_t`
	`std::uint32_t`

`typename Method = oneapi::mkl::rng::geometric_method:: by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::geometric_method::icdf` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
p	`float`	Success probability p of a trial.

oneapi::mkl::rng::binomial

Generates binomially distributed random numbers.

Description

The `oneapi::mkl::rng::binomial` class object is used in the `oneapi::mkl::rng::generate` function to provide binomially distributed random numbers with number of independent Bernoulli trials `m`, and with probability `p` of a single trial success, where `p∈R; 0 ≤p≤ 1, m∈N`.

A binomially distributed variate represents the number of successes in `m` independent Bernoulli trials with probability of a single trial success `p`.

The probability distribution is given by:

$$P(X = k) = C_m^k p^k (1-p)^{m-k}, k \in \{0, 1, \ldots, m\}$$

The cumulative distribution function is as follows:

$$F_{m,p}(x) = \begin{cases} 0, & x < 0 \\ \sum_{k=0}^{\lfloor x \rfloor} C_m^k p^k (1-p)^{m-k}, & 0 \le x < m) \\ 1, & x > m \end{cases}, x \in R$$

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
template<typename IntType = std::int32_t,
  typename Method = binomial_method::by_default>
  class binomial {
  public:
  using method_type = Method;
  using result_type = IntType;
  binomial(): binomial(5, 0.5){}
  explicit binomial(std::int32_t ntrial, double p);
  explicit binomial(const param_type& pt);
  std::int32_t ntrial() const;
  double p() const;
  param_type param() const;
  void param(const param_type& pt);
};
```

Devices supported: CPU and GPU.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

Name	Description

`typename IntType = std::int32_t`	Type of the produced values. The specific values are as follows:
	`std::int32_t`
	`std::uint32_t`

`typename Method = oneapi::mkl::rng::binomial_method:: by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::binomial_method::btpe` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
ntrials	`std::int32_t`	Number of independent trials.
p	`double`	Success probability p of a single trial.

oneapi::mkl::rng::hypergeometric

Generates hypergeometrically distributed random values.

Description

The `oneapi::mkl::rng::hypergeometric` class object is used in the `oneapi::mkl::rng::generate` function to provide hypergeometrically distributed random values with lot size `l`, size of sampling `s`, and number of marked elements in the lot `m`, where `l, m, s∈Nu{0}; l≥ max(s, m)`.

Consider a lot of `l` elements comprising `m` "marked" and `l-m` "unmarked" elements. A trial sampling without replacement of exactly `s` elements from this lot helps to define the hypergeometric distribution, which is the probability that the group of `s` elements contains exactly `k` marked elements.

The probability distribution is given by:

$$P(X = k) = \frac{C_m^k C_{l-m}^{s-k}}{C_l^s}$$

, k∈ {max(0, s + m - l), ..., min(s, m)}

The cumulative distribution function is as follows:

$$F_{l,s,m}(x) = \begin{cases} 0, & x < max(0, s + m - l) \\ \sum_{k=\max(0,s+m-l)}^{\lfloor x \rfloor} \frac{C_m^k C_{l-m}^{s-k}}{C_l^s}, & max(0, s + m - l) \le x \le \min(s, m) \\ 1, & x > \min(s, m) \end{cases}$$

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
template<typename IntType = std::int32_t, typename Method = hypergeometric_method::by_default>
class hypergeometric {
public:
using method_type = Method;
using result_type = IntType;
hypergeometric(): hypergeometric(1, 1, 1){}
explicit hypergeometric(std::int32_t l, std::int32_T s, std::int32_T m);
explicit hypergeometric(const param_type& pt);
std::int32_t s() const;
std::int32_t m() const;
std::int32_t l() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename IntType = std::int32_t`	Type of the produced values. The specific values are as follows:
	`std::int32_t`
	`std::uint32_t`

`typename Method = oneapi::mkl::rng::hypergeometric_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::hypergeometric_method::h2pe` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
l	`std::int32_t`	Lot size of `l`.
s	`std::int32_t`	Size of sampling without replacement.
m	`std::int32_t`	Number of marked elements `m`.

oneapi::mkl::rng::poisson

Generates Poisson distributed random values.

Description

The `oneapi::mkl::rng::poisson` class object is used in the `oneapi::mkl::rng::generate` function to provide Poisson distributed random numbers with distribution parameter λ, where `λ∈R; λ > 0`.

The probability distribution is given by:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

$k \in \{0, 1, 2, \ldots\}$. The cumulative distribution function is as follows:

$$F_\lambda(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} \frac{\lambda^k e^{-\lambda}}{k!}, & x \geq 0 \\ 0, & x < 0 \end{cases}, x \in R$$

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
template<typename IntType = std::int32_t, typename Method = poisson_method::by_default>
class poisson {
public:
using method_type = Method;
using result_type = IntType;
poisson(): poisson(0.5){} explicit poisson(double lambda);
explicit poisson(const param_type& pt);
double lambda() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename IntType = std::int32_t`	Type of the produced values. The specific values are as follows:
	`std::int32_t`
	`std::uint32_t`

`typename Method = oneapi::mkl::rng::poisson_method::by_default`	Generation method. The specific values are as follows:
	`oneapi::mkl::rng::poisson_method::ptpe`
	`oneapi::mkl::rng::poisson_method::gaussian_icdf_based`
	See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
lambda	`double`	Distribution parameter λ.

oneapi::mkl::rng::poisson_v

Generates Poisson distributed random values with varying mean.

- Description
- API

Description

The `oneapi::mkl::rng::poisson_v` class object is used in the `oneapi::mkl::rng::generate` function to provide n Poisson distributed random numbers x_i (`i` = 1, ..., n) with distribution parameter λ_i, where $\lambda_i \in R$; $\lambda_i >$ `0`.

The probability distribution is given by:

$$P(X_i = k) = \frac{\lambda_i^k \exp(-\lambda_i)}{k!}, k \in \{0, 1, 2, \ldots\}$$

The cumulative distribution function is as follows:

$$F_{\lambda_i}(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} \frac{\lambda_i^k e^{-\lambda_i}}{k!}, & x \geq 0 \\ 0, & x < 0 \end{cases}, x \in R$$

Product and Performance Information
Performance varies by use, configuration, and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
template<typename IntType = std::int32_t, typename Method = poisson_v_method::by_default>
class poisson_v {
public:
  using method_type = Method;
  using result_type = IntType;
  explicit poisson_v(std::vector<double> lambda); // deprecated since oneMKL 2023.0
  explicit poisson_v(sycl::span<double> lambda);
  explicit poisson_v(const param_type& pt);
  std::vector<double> lambda() const;
  param_type param() const;
  void param(const param_type& pt);
};
```

Devices supported: CPU and GPU

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename IntType = std::int32_t`	Type of the produced values. The specific values are as follows:
	`std::int32_t`
	`std::uint32_t`

`typename Method = oneapi::mkl::rng::poisson_v_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::poisson_v_method::gaussian_icdf_based` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
lambda	`sycl::span<double>`	Array of n distribution parameters λ.

Note:

- `explicit poisson_v(std::vector<double> lambda);` is deprecated and will be removed in one of the next releases. Use `explicit poisson_v(sycl::span<double> lambda);` instead.
- When passing a `sycl::span` that is constructed over a user's memory to the constructor, users must manage the memory under sycl::span by themselves. They must not destroy the memory while data are processed.

oneapi::mkl::rng::negative_binomial

Generates random numbers with negative binomial distribution.

Description

The `oneapi::mkl::rng::negative_binomial` class object is used in the `oneapi::mkl::rng::generate` function to provide random numbers with negative binomial distribution and distribution parameters a and p, where p, a $\in$ R; 0 < p < 1; a > 0.

If the first distribution parameter a $\in$ N, this distribution is the same as Pascal distribution. If a $\in$ N, the distribution can be interpreted as the expected time of a-th success in a sequence of Bernoulli trials, when the probability of success is p.

The probability distribution is given by:

$$P(X = k) = C_{a+k-1}^{k} p^{a} (1-p)^{k}, k \in \{0, 1, 2, \ldots\}$$

The cumulative distribution function is as follows:

$$F_{a,p}(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} C_{a+k-1}^{k} p^{a} (1-p)^{k}, & x \geq 0 \\ 0, & x < 0 \end{cases}, x \in R$$

API

Syntax

```
template<typename IntType = std::int32_t, typename Method = negative_binomial_method::by_
↪default>
class negative_binomial {
public:
using method_type = Method;
using result_type = IntType;
negative_binomial(): negative_binomial(0.1, 0.5){}
explicit negative_binomial(double a, double p);
explicit negative_binomial(const param_type& pt);
double a() const;
double p() const;
param_type param() const;
void param(const param_type& pt);
};
```

Devices supported: CPU and GPU

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename IntType = std::int32_t`	Type of the produced values. The specific values are as follows:
	`std::int32_t`
	`std::uint32_t`

`typename Method = oneapi::mkl::rng::negative_binomial_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::negative_binomial_method::nbar` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
a	`double`	The first distribution parameter a.
p	`double`	The second distribution parameter p.

oneapi::mkl::rng::multinomial

Generates multinomially distributed random numbers.

Description

The `oneapi::mkl::rng::multinomial` class object is used in the `oneapi::mkl::rng::generate` function to provide multinomially distributed random numbers with `ntrial` independent trials and k possible mutually exclusive outcomes, with corresponding probabilities p_i, where $p_i \in R$; $0 \le p_i \le 1$, $m \in N$, $k \in N$.

The probability distribution is given by:

$$P(X_1 = x_1, \ldots, X_k = x_k) = \frac{m!}{\prod_{i=1}^{k} x_i!} \prod_{i=1}^{k} {p_i}^{x_i}, 0 \le x_i \le m, \sum_{i=1}^{k} x_i = m$$

Product and Performance Information
Performance varies by use, configuration, and other factors. Learn more at https://www.intel.com/PerformanceIndex. Notice revision #20201201

API

Syntax

```
template<typename IntType = std::int32_t, typename Method = multinomial_method::by_default>
class multinomial {
public:
  using method_type = Method;
  using result_type = IntType;
  explicit multinomial(double ntrial, std::vector<double> p); // deprecated since oneMKL 2023.0
  explicit multinomial(double ntrial, sycl::span<double> p);
  explicit multinomial(const param_type& pt);
  std::int32_t ntrial() const;
  std::vector<double> p() const;
  param_type param() const;
  void param(const param_type& pt);
};
```

Devices supported: CPU and GPU

Note: This distribution supports only input parameters (on GPU) that follow the condition $(k \ge ntrial * 16 \wedge ntrial \le 16)$, where k is the probability vector length.

Include Files

- `oneapi/mkl/rng.hpp`

Template Parameters

`typename IntType = std::int32_t`	Type of the produced values. The specific values are as follows:
	`std::int32_t`
	`std::uint32_t`

`typename Method = oneapi::mkl::rng::multinomial_method::by_default`	Generation method. The specific values are as follows: `oneapi::mkl::rng::multinomial_method::poisson_icdf_based` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
ntrial	`std::int32_t`	Number of independent trials `m`.
p	`sycl::span<double>`	Probability vector of possible outcomes (length is k).

Note:

- `explicit multinomial(double ntrial, std::vector<double> p);` is deprecated and will be removed in one of the next releases. Use `explicit multinomial(double ntrial, sycl::span<double> p);` instead.
- When passing a `sycl::span` that is constructed over a user's memory to the constructor, users must manage the memory under sycl::span by themselves. They must not destroy the memory while data are processed.

13.1.2 Random Number Generators Device Routines

The main purpose of Device routines is to make them callable from your DPC++ kernels; however, there are no limitations to be called from the Host. For example:

```
sycl::queue queue;

queue.submit([&](sycl::handler& cgh) {
  cgh.parallel_for(range,[=](...) {
     mkl::rng::device::routine(...); // calling routine from user's kernel code
  });
```

(continues on next page)

(continued from previous page)

```
});

mkl::rng::device::routine(...); // calling routine from host
```

RNG Device APIs Structure

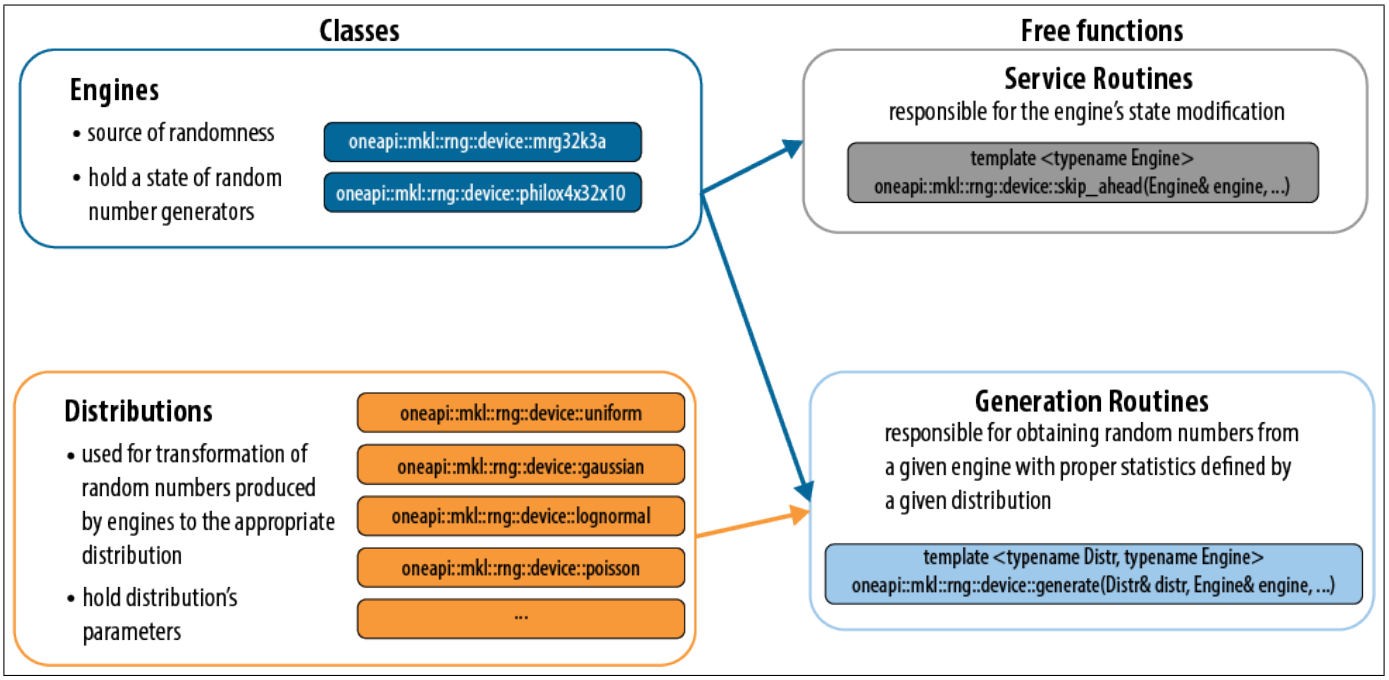


Fig. 11: RNG Device API Structure

oneMKL RNG Device Usage Model

- Example of Scalar Random Numbers Generation
- Example of Vector Random Numbers Generation
- Example of Random Numbers Generation by Engines Stored in `sycl::buffer`
- Example of Random Numbers Generation with Host-side Helpers Usage

A typical usage model for device routines is the same as described in oneMKL RNG Usage Model:

1. Create and initialize the object for basic random number generator.
2. Create and initialize the object for distribution generator.
3. Call the generate routine to get random numbers with appropriate statistical distribution.

Example of Scalar Random Numbers Generation

```
#include<iostream>
#include <CL/sycl.hpp>
#include "oneapi/mkl/rng/device.hpp"


int main() {
    sycl::queue queue;
    const int n = 1000;
    const int seed = 1;
    // Prepare an array for random numbers
    std::vector<float> r(n);


    sycl::buffer<float, 1> r_buf(r.data(), r.size());
    // Submit a kernel to generate on device
    queue.submit([&](sycl::handler& cgh) {
        auto r_acc = r_buf.template get_access<sycl::access::mode::write>(cgh);
        cgh.parallel_for(sycl::range<1>(n), [=](sycl::item<1> item) {
            // Create an engine object
            oneapi::mkl::rng::device::philox4x32x10<> engine(seed, item.get_id(0));
            // Create a distribution object
            oneapi::mkl::rng::device::uniform<> distr;
            // Call generate function to obtain scalar random number
            float res = oneapi::mkl::rng::device::generate(distr, engine);


            r_acc[item.get_id(0)] = res;
        });
    });


    auto r_acc = r_buf.template get_access<sycl::access::mode::read>();
    std::cout << "Samples of uniform distribution" << std::endl;
    for(int i = 0; i < 10; i++) {
        std::cout << r_acc[i] << std::endl;
    }


    return 0;
}
```

Example of Vector Random Numbers Generation

```
#include<iostream>
#include <CL/sycl.hpp>


#include "oneapi/mkl/rng/device.hpp"


int main() {
    sycl::queue queue;
    const int n = 1000;
    const int seed = 1;
    const int vec_size = 4;
    // Prepare an array for random numbers
    std::vector<float> r(n);


    sycl::buffer<float, 1> r_buf(r.data(), r.size());
    // Submit a kernel to generate on device
    sycl::queue{}.submit([&](sycl::handler& cgh) {
        auto r_acc = r_buf.template get_access<sycl::access::mode::write>(cgh);
        cgh.parallel_for(sycl::range<1>(n / vec_size), [=](sycl::item<1> item) {
            // Create an engine object
            oneapi::mkl::rng::device::philox4x32x10<vec_size> engine(seed, item.get_id(0) * vec_
↪size);
            // Create a distribution object
            oneapi::mkl::rng::device::uniform<> distr;
            // Call generate function to obtain sycl::vec<float, 4> with random numbers
            auto res = oneapi::mkl::rng::device::generate(distr, engine);


            res.store(ite.get_id(0), r_acc);
        });
    });


    auto r_acc = r_buf.template get_access<sycl::access::mode::read>();
    std::cout << "Samples of uniform distribution" << std::endl;
    for(int i = 0; i < 10; i++) {
        std::cout << r_acc[i] << std::endl;
    }


    return 0;
}
```

There is an opportunity to store engines between kernels manually via `sycl::buffer` / USM pointers or by using a specific host-side helper class called, engine descriptor.

Example of Random Numbers Generation by Engines Stored in `sycl::buffer`

Engines are initialized in the first kernel. Random number generation is performed in the second kernel.

```
#include<iostream>
#include <CL/sycl.hpp>


#include "oneapi/mkl/rng/device.hpp"


int main() {
    sycl::queue queue;
    const int n = 1000;
    const int seed = 1;
    const int vec_size = 4;
    // Prepare an array for random numbers
    std::vector<float> r(n);
    sycl::buffer<float, 1> r_buf(r.data(), r.size());
    sycl::range<1> range(n / vec_size);
    sycl::buffer<oneapi::mkl::rng::device::mrg32k3a<vec_size>, 1> engine_buf(range);


    sycl::queue queue;


    // Kernel with initialization of engines
    queue.submit([&](sycl::handler& cgh) {
        // Create an accessor to sycl::buffer with engines to write initialized states
        auto engine_acc = engine_buf.template get_access<sycl::access::mode::write>(cgh);
        cgh.parallel_for(range, [=](sycl::item<1> item) {
            size_t id = item.get_id(0);
            // Create an engine object with offset id * 2^64
            oneapi::mkl::rng::device::mrg32k3a<vec_size> engine(seed, {0, id});
            engine_acc[id] = engine;
        });
    });
    // Kernel for random numbers generation
    queue.submit([&](sycl::handler& cgh) {
        auto r_acc = r_buf.template get_access<sycl::access::mode::write>(cgh);
        // Create an accessor to sycl::buffer with engines to read initialized states
        auto engine_acc = engine_buf.template get_access<sycl::access::mode::read>(cgh);
        cgh.parallel_for(range, [=](sycl::item<1> item) {
            size_t id = item.get_id(0);


            auto engine = engine_acc[id];
            oneapi::mkl::rng::device::uniform distr;
            auto res = oneapi::mkl::rng::device::generate(distr, engine);


            res.store(id, r_acc);
```

(continues on next page)

(continued from previous page)

```
        });
    });


    auto r_acc = r_buf.template get_access<sycl::access::mode::read>();
    std::cout << "Samples of uniform distribution" << std::endl;
    for(int i = 0; i < 10; i++) {
        std::cout << r_acc[i] << std::endl;
    }


    return 0;
}
```

Example of Random Numbers Generation with Host-side Helpers Usage

```
#include<iostream>
#include <CL/sycl.hpp>


#include "oneapi/mkl/rng/device.hpp"


int main() {
    sycl::queue queue;
    const int n = 1000;
    const int seed = 1;
    const int vec_size = 4;
    // prepare array for random numbers
    std::vector<float> r(n);
    sycl::buffer<float, 1> r_buf(r.data(), r.size());
    sycl::range<1> range(n / vec_size);


    // offset of each engine in engine_descriptor
    int offset = vec_size;
    // each engine would be created in enqueued task as of specified range
    // as oneapi::mkl::rng::device::mrg32k3a<vec_size>(seed, id * offset);
    oneapi::mkl::rng::device::engine_descriptor<oneapi::mkl::rng::device::mrg32k3a<vec_size>>
    descr(queue, range, seed, offset);
    queue.submit([&](sycl::handler& cgh) {
        auto r_acc = r_buf.template get_access<sycl::access::mode::write>(cgh);
        // create engine_accessor
        auto engine_acc = descr.get_access(cgh);
        cgh.parallel_for(range, [=](sycl::item<1> item) {
            size_t id = item.get_id(0);
            // load engine from engine_accessor
            auto engine = engine_acc.load(id);
            oneapi::mkl::rng::device::uniform<Type> distr;
```

(continues on next page)

(continued from previous page)

```
34
35
36              auto res = oneapi::mkl::rng::device::generate(distr, engine);
37
38
39              res.store(id, r_acc);
40              // store engine for furter calculations if needed
41              engine_acc.store(engine, id);
42          });
43      });
44
45
46      auto r_acc = r_buf.template get_access<sycl::access::mode::read>();
47      std::cout << "Samples of uniform distribution" << std::endl;
48      for(int i = 0; i < 10; i++) {
49          std::cout << r_acc[i] << std::endl;
50      }
51
52
53      return 0;
54  }
```

Additionally, examples that demonstrate usage of random number generators device routines are available in:

```
${MKL}/examples/dpcpp_device/rng/source
```

Device Generate Routines

Use the `oneapi::mkl::rng::device::generate` or `oneapi::mkl::rng::device::generate_single` routines to obtain random numbers from a given engine with proper statistics of a given distribution.

oneapi::mkl::rng::device::generate

Entry point to obtain random numbers from a given engine with proper statistics of a given distribution.

API

Syntax

```
template<typename Distr, typename Engine>
  auto generate(Distr& distr, Engine& engine) ->
  typename std::conditional<Engine::vec_size == 1,      typename Distr::result_type,
  sycl::vec<typename Distr::result_type,      Engine::vec_size>>::type
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Input Parameters

Name	Type	Description
distr	Distr&	Distribution object. See Device Distributions for des.
engine	Engine&	Engine object. See Device Engines (Basic Random Number Generators) for details.

Output Parameters

Name	Type	Description
result	`sycl::vec<typename Distr::result_type, Engine::vec_size>` or `typename Distr::result_type`	Function returns `sycl::vec` of type specified by the Distribution object and vector size specified by the Engine object filled with random numbers or a scalar random number in case `vec_size` =1.

oneapi::mkl::rng::device::generate_single

Entry point to obtain a single random number from a given vector engine with proper statistics of a given distribution.

- API

API

Syntax

```
template<typename Distr, typename Engine>
  typename Distr::result_type generate_single(Distr& distr, Engine& engine)
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Input Parameters

Name	Type	Description
distr	Distr&	Distribution object. See Device Distributions for details.
engine	Engine&	Engine object. See Device Engines (Basic Random Number Generators) for details.

Output Parameters

Name	Type	Description
result	typename `Distr::result_type`	Function returns a scalar random number, may be used for engines with `vec_size` > 1.

Device Engines (Basic Random Number Generators)

Intel® oneAPI Math Kernel Library (oneMKL) RNG provides two device pseudorandom number generators:

Routine	Description
oneapi::mkl::rng::device::mrg32k3a	The combined multiple recursive pseudorandom number generator `MRG32k3a` [L'Ecuyer99]
oneapi::mkl::rng::device::philox4x32x10	Philox4x32-10 counter-based pseudorandom number generator with a period of 2^{128} `PHILOX4X32X10` [Salmon11]
oneapi::mkl::rng::device::mcg31m1	The 31-bit multiplicative congruential pseudorandom number generator MCG(1132489760, 2^{32}-1) [L'Ecuyer99a].
oneapi::mkl::rng::device::mcg59	The 59-bit multiplicative congruential pseudorandom number generator MCG(13^{13}, 2^{59}) from NAG Numerical Libraries [NAG].

oneapi::mkl::rng::device::mrg32k3a

Description

The combined multiple recursive pseudorandom number generator MRG32k3a [L'Ecuyer99a].

API

Syntax

```
template<std::int32_t VecSize = 1>
class mrg32k3a {
public:
  static constexpr std::uint32_t default_seed = 1;
  static constexpr std::int32_t vec_size = VecSize;

  mrg32k3a() : mrg32k3a(default_seed) {}
  mrg32k3a(std::uint32_t seed, std::uint64_t offset = 0);
  mrg32k3a(std::initializer_list<std::uint32_t> seed, std::uint64_t offset = 0);
  mrg32k3a(std::uint32_t seed, std::initializer_list<std::uint64_t> offset);
  mrg32k3a(std::initializer_list<std::uint32_t> seed, std::initializer_list<std::uint64_t>␣
↪offset);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

Name	Type	Description
VecSize	`std::int32_t`	Describes the size of vector which will be produced by generate function by this engine. VecSize values may be 1, 2, 3, 4, 8, 16 as `sycl::vec` class size. By default VecSize = 1, for this case, a single random number is returned by the `oneapi::mkl::rng::device::generate` function.

Constructors Input Parameters

Name	Type	Description
seed	`std::uint32_t std::initializer_list<std::uint32_t>`	Initial conditions of the engine state.
offset	`std::uint64_t std::initializer_list<std::uint64_t>`	number of skipped elements, for `initializer_list` offset is calculated as: `num_to_skip` [0]+ `num_to_skip` [1]*2^{64} + `num_to_skip` [2]*2^{128} + ... + `num_to_skip` [n-1]*2^{64} *(n-1).

See VS Notes for the detailed descriptions.

oneapi::mkl::rng::device::philox4x32x10

Description

A Philox4x32-10 counter-based pseudorandom number generator [Salmon11].

API

Syntax

```
template<std::int32_t VecSize = 1>
  class philox4x32x10 {
  public:
  static constexpr std::uint64_t default_seed = 1;
  static constexpr std::int32_t vec_size = VecSize;
  philox4x32x10() : philox4x32x10(default_seed) {}
  philox4x32x10(std::uint64_t seed, std::uint64_t offset = 0);
  philox4x32x10(std::initializer_list<std::uint64_t> seed, std::uint64_t offset = 0);␣
↪philox4x32x10(std::uint64_t seed, std::initializer_list<std::uint64_t> offset);
  philox4x32x10(std::initializer_list<std::uint64_t> seed, std::initializer_list<std::uint64_t>␣
↪offset);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

Name	Type	Description
VecSize	`std::int32_t`	Describes the size of vector which will be produced by generate function by this engine. VecSize values may be 1, 2, 3, 4, 8, 16 as `sycl::vec` class size. By default VecSize = 1, for this case single random number is returned by the `oneapi::mkl::rng::device::generate` function.

Constructors Input Parameters

Name	Type	Description
seed	`std::uint64_t` `std::initializer_list<std::uint64_t>`	Initial conditions of the engine state.
offset	`std::uint64_t` `std::initializer_list<std::uint64_t>`	Number of skipped elements, for `initializer_list` offset is calculated as: `num_to_skip`[0]+ `num_to_skip` [1]*2^{64}+ `num_to_skip` [2] * 2^{128}+ ... +`num_to_skip`[n-1]*2^{64} * (n-1).

See VS Notes for detailed descriptions.

oneapi::mkl::rng::device::mcg31m1

Description

The 31-bit multiplicative congruential pseudorandom number generator MCG(1132489760, 2^{32}-1) [L'Ecuyer99a].

API

Syntax

```
template<std::int32_t VecSize = 1>
class mcg31m1 {
public:
  static constexpr std::uint32_t default_seed = 1;
  static constexpr std::int32_t vec_size = VecSize;
  mcg31m1() : mcg31m1(default_seed) {}
  mcg31m1(std::uint32_t seed, std::uint64_t offset = 0);
  mcg31m1(std::initializer_list<std::uint32_t> seed, std::uint64_t offset = 0);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

Name	Type	Description
VecSize	`std::int32_t`	Describes the size of vector that will be produced by generate function by this engine. VecSize values may be 1, 2, 3, 4, 8, 16 as `sycl::vec` class size. By default VecSize = 1, for this case, a single random number is returned by the `oneapi::mkl::rng::device::generate` function.

Constructors Input Parameters

Name	Type	Description
seed	`std::uint32_t` `std::initializer_list<std::uint32_t>`	Initial conditions of the engine state.
offset	`std::uint64_t`	Number of skipped elements.

See VS Notes for detailed descriptions.

oneapi::mkl::rng::device::mcg59

Description

The 59-bit multiplicative congruential pseudorandom number generator MCG(13^{13}, 2^{59}) from NAG Numerical Libraries [NAG].

API

Syntax

```
template<std::int32_t VecSize = 1>
class mcg59 {
public:
  static constexpr std::uint32_t default_seed = 1;
  static constexpr std::int32_t vec_size = VecSize;
  mcg59() : mcg59(default_seed) {}
  mcg59(std::uint32_t seed, std::uint64_t offset = 0);
  mcg59(std::initializer_list<std::uint32_t> seed, std::uint64_t offset = 0);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

Name	Type	Description
VecSize	`std::int32_t`	Describes the size of vector that will be produced by generate function by this engine. VecSize values may be 1, 2, 3, 4, 8, 16 as `sycl::vec` class size. By default VecSize = 1, for this case, a single random number is returned by the `oneapi::mkl::rng::device::generate` function.

Constructors Input Parameters

Name	Type	Description
seed	`std::uint32_t` `std::initializer_list<std::uint32_t>`	Initial conditions of the engine state.
offset	`std::uint64_t`	Number of skipped elements.

See VS Notes for detailed descriptions.

Host-Side Helpers

Intel® oneAPI Math Kernel Library (oneMKL) RNG provides host-side helper-class templates for initializing engines.

Routine	Description
oneapi::mkl::rng::device::engine_descriptor	Provide an abstraction over `sycl::buffer` to initialize and store engines' states between DPC++ kernels.
oneapi::mkl::rng::device::engine_accessor	Provide an abstraction over `sycl::accessor` to access engines' which are hold by engine_descriptor in kernels.

oneapi::mkl::rng::device::engine_descriptor

Host side helper to provide an abstraction over `sycl::buffer` to initialize and store engines' states between DPC++ kernels.

Description

`engine_descriptor` provides an abstraction over `sycl::buffer` to initialize and store engines' states between SYCL kernels.

Note: The constructor of the engine descriptor submits a parallel_for task to a given queue to initialize engines' states: Each engine is initialized as `Engine {seed, id * offset}` for scalar offset case, where id is a value from 0 to range.size().

API

Syntax

```
template<Engine>
  class engine_descriptor {
  public:
  engine_descriptor(sycl::queue& queue, sycl::range<1> range, std::uint64_t seed, std::uint64_t␣
↪offset);
  template<typename InitEngineFunc>
  engine_descriptor(sycl::queue& queue, sycl::range<1> range, InitEngineFunc func);
  engine_accessor<Engine> get_access(sycl::handler& cgh);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

Type	Description
`Engine`	Specify an engine which state is hold by engine descriptor.

Constructors Input Parameters

Name	Type	Description
queue	`sycl::queue&`	`sycl::queue` object. Task is submitted to this queue to inlize engines' states directly on the device which is associated wiueue.
range	`sycl::range<1>`	Describes number of engines which are hold by this en_descriptor object.
seed	`std::uint64_t std::initializer_list<std::uint64_t>`	Initial conditions of the engine state.
offset	`std::uint64_t std::initializer_list<std::uint64_t>`	Number of skipped elements, for `initializer_list` offset is calculated as `num_to_skip` [0]+ `num_to_skip` [1]*2^{64} + `num_to_skip` [2]* 2^{128} + ... + `num_to_skip` [n-1]*2^{64} *(n-1).
func	InitEngineFunc	Functor which would be used to initialize engines. This functor should take `sycl::item<1>` as an argument and return object of type `Engine`.

oneapi::mkl::rng::device::engine_accessor

Host side helper to provide an abstraction over `sycl::accessor` to access engines' which are held by `engine_descriptor` in kernels.

Description

`engine_accessor` provides an abstraction over `sycl::accessor` to access engines which are held by `engine_descriptor` in kernels by the `load()` and `store()` functions.

API

Syntax

```
template<Engine>
  class engine_descriptor {
  public:
  Engine load(size_t id) const
  void store(Engine engine, size_t id) const
  }
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

Type	Description
`Engine`	Specify an engine which state is hold by `engine_accessor`.

Device Service Routines

Routine	Description
oneapi::mkl::rng::device::skip_ahead	Proceed state of engine by the skip-ahead method to skip a given number of elements from the original sequence.

oneapi::mkl::rng::device::skip_ahead

Description

Proceed state of engine by the skip-ahead method.

The `oneapi::mkl::rng::device::skip_ahead` function supports the following interfaces to apply the skip-ahead method:

- Common interface
- Interface with a partitioned number of skipped elements

API

Syntax

Common Interface

```
template<typename_Engine>
  void skip_ahead (Engine& engine, std::uint64_t      num_to_skip)
```

Interface with a partitioned number of skipped elements

```
template<typename_Engine>
  void skip_ahead (Engine& engine,      std::initializer_list<std::uint64_t> num_to_skip)
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

Common Interface

Name	Type	Description
`engine`	`Engine&`	Object of `engine` class, which supports the block-splitting method.
`num_to_skip`	`std::uint64_t`	Number of skipped elements.

Interface with a Partitioned Number of Skipped Elements

Name	Type	Description
`engine`	`Engine&`	Object of `engine` class, which supports the block-splitting method.
`num_to_skip`	`std::initializer_list<std::uint64_t>`	Partitioned number of skipped elements.

Device Distributions

oneMKL RNG routines are used to generate random numbers with different types of distribution. Each function group is introduced below by the type of underlying distribution and contains a short description of its functionality, as well as specifications of the call sequence and the explanation of input and output parameters. The Device Continuous Distribution Generators table and Device Discrete Distribution Generators table list the random number generator routines with data types and output distributions, and sets correspondence between data types of the generator routines and the basic random number generators.

Device Continuous Distribution Generators

Type of Distribution	Data Types	BRNG Data Type	Description
oneapi::mkl::rng::device::uniform (Continuous)	s, d	s, d	Uniform continuous distribution on the interval [a, b)
oneapi::mkl::rng::device::gaussian	s, d	s, d	Normal (Gaussian) distribution
oneapi::mkl::rng::device::lognormal	s, d	s, d	Lognormal distribution

Device Discrete Distribution Generators

Type of Distribution	Data Types	BRNG Data Type	Description
oneapi::mkl::rng::device::uniform (Discrete)		s	Uniform discrete distribution on the interval [a, b)
oneapi::mkl::rng::device::bits	i	i	Bits of underlying BRNG integer recurrence
oneapi::mkl::rng::device::bernoulli	i	i	Bernoulli distribution

oneapi::mkl::rng::device::uniform (Continuous)

Generates random numbers with uniform distribution.

Description

The `oneapi::mkl::rng::device::uniform` class object is used in the `oneapi::mkl::rng::device::generate` function to provide random numbers uniformly distributed over the interval [a, b), where a, b are the left and right bounds of the interval, respectively, and a, b∈R ; a < b.

The probability density function is given by:

$$F_{a,b}(x) = \begin{cases} \frac{1}{b-a}, & x \in [a,b) \\ 1, & x \notin [a,b) \end{cases}, -\infty < x < +\infty$$

The cumulative distribution function is as follows:

$$F_{a,b}(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \le x < b \\ 1, & x \ge b \end{cases}, -\infty < x < +\infty$$

API

Syntax

```
namespace oneapi::mkl::lapack {
  template<typename Type, typename Method>
  class uniform {public:
  using method_type = Method;
  using result_type = Type;
  uniform(): uniform((Type)0.0,
  (Type)1.0){};
  explicit uniform(Type a, Type b);
  explicit uniform(const param_type& pt);
  Type a() const;
  Type b() const;
  param_type param() const;
  void param(const param_type& pt);
  };
}
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

`typename Type`	**Type of the produced values. The specific values are as follows:** `float double`
`typename Method`	Generation method. The specific values are as follows: `oneapi::mkl::rng::device::uniform_method::standard` `oneapi::mkl::rng::device::uniform_method::accurate`

See brief descriptions of the methods in Distributions Template Parameter Method

Input Parameters

Name	Type	Description
a	`Type (float, double)`	Left bound a
b	`Type (float, double)`	Right bound b

oneapi::mkl::rng::device::gaussian

Generates normally distributed random numbers.

Description

The `oneapi::mkl::rng::device::gaussian` class object is used in the `oneapi::mkl::rng::device::generate` function to provide random numbers with normal (Gaussian) distribution with mean (a) and standard deviation (`stddev`, σ), where a, σ ∈ R; σ > 0.

The probability density function is given by:

$$f_{a,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-a)^2}{2\sigma^2}\right) dy, -\infty < x < +\infty$$

The cumulative distribution function is as follows:

$$F_{a,\sigma}(x) = \int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-a)^2}{2\sigma^2}\right) dy, -\infty < x < +\infty$$

The cumulative distribution function $Fa, \sigma(x)$ can be expressed in terms of standard normal distribution $\phi(x)$ as

$$F$$
$$a, \sigma(x) = \phi((x-a)/\sigma)$$

API

Syntax

```
template<typename RealType, typename Method>
class gaussian {
public:
using method_type = Method;
using result_type = RealType;
gaussian(): gaussian((RealType)0.0, (RealType)1.0){}
explicit gaussian(RealType mean, RealType stddev);
explicit gaussian(const param_type& pt);
RealType mean() const;
RealType stddev() const;
param_type param() const;
void param(const param_type& pt);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

`typename RealType`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method`	Generation method. The specific values are as follows: `oneapi::mkl::rng::device::gaussian_method::box_muller2`

Input Parameters

Name	Type	Description
mean	`RealType (float, double)`	Mean value a.
stddev	`RealType (float, double)`	Standard deviation σ.

oneapi::mkl::rng::device::lognormal

Generates lognormally distributed random numbers.

Description

The `oneapi::mkl::rng::device::lognormal` class object is used in the `oneapi::mkl::rng::device::generate` function to provide random numbers with average of distribution (`m`, a) and standard deviation (`s`, σ) of subject normal distribution, displacement (`displ`, b), and scalefactor (`scale`, β), where `a, σ, b, β ∈ R ; σ > 0 , β > 0`.

The probability density function is given by:

$$F_{a,\sigma,b,\beta}(x) = \begin{cases} \frac{1}{\sigma(x-b)\sqrt{2\pi}} \exp\left(-\frac{\ln(\frac{x-b}{\beta})-a)^2}{2\sigma^2}\right), & x > b \\ 0, & x \le b \end{cases}$$

The cumulative distribution function is as follows:

$$F_{a,\sigma,b,\beta}(x) = \begin{cases} \Phi\left(\frac{\ln(\frac{x-b}{\beta})-a}{\sigma}\right), & x > b \\ 0, & x \le b \end{cases}$$

API

Syntax

```
template<typename RealType, typename Method>
class lognormal { public:
using method_type = Method; using result_type = RealType;
lognormal(): lognormal((RealType)0.0, (RealType)1.0, (RealType) 0.0,
(RealType)1.0){} explicit lognormal(RealType m, RealType s, RealType displ =
(RealType)0.0, RealType scale = (RealType)1.0);

explicit lognormal(const param_type& pt);
RealType m() const;
RealType s() const;
RealType displ() const;
RealType scale() const;
param_type param() const;
void param(const param_type& pt);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

`typename RealType`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method`	Generation method. The specific values are as follows: `oneapi::mkl::rng::device::lognormal_method::box_muller2`

Input Parameters

Name	Type	Description
m	`RealType (float, double)`	Average a of the subject normal distribution.
s	`RealType (float, double)`	Standard deviation σ of the subject normal distribution.
displ	`RealType (float, double)`	Displacement `displ`.
scale	`RealType (float, double)`	Scalefactor `scale`.

oneapi::mkl::rng::device::exponential

Generates exponentially distributed random numbers.

Description

The `oneapi::mkl::rng::device::exponential` class object is used in `oneapi::mkl::rng::device::generate` function to provide random numbers with exponential distribution that has displacement a and scale-factor β, where $a, \beta \in R; \beta > 0$.

The probability density function is given by:

$$F_{a,\beta}(x) = \begin{cases} \frac{1}{\beta}\exp(-\frac{(x-a)}{\beta}), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

The cumulative distribution function is as follows:

$$F_{a,\beta}(x) = \begin{cases} 1 - \exp(-\frac{(x-a)}{\beta}), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

API

Syntax

```
template<typename RealType, typename Method>
class exponential {
public:
  using method_type = Method;
  using result_type = RealType;
  exponential(): exponential((RealType)0.0, (RealType)1.0){}
  explicit exponential(RealType mean, RealType stddev);
  explicit exponential(const param_type& pt);
  RealType a() const;
  RealType beta() const;
  param_type param() const;
  void param(const param_type& pt);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

`typename RealType`	Type of the produced values. The specific values are as follows:
	`float`
	`double`

`typename Method`	Generation method. The specific values are as follows:
	`oneapi::mkl::rng::device::exponential_method::icdf`
	`oneapi::mkl::rng::device::exponential_method::icdf_accurate`
	See brief descriptions of the methods in Distributions Template Parameter Method

Input Parameters

Name	Type	Description
a	`RealType (float, double)`	Displacement a.
beta	`RealType (float, double)`	Scalefactor β.

oneapi::mkl::rng::device::uniform (Discrete)

Generates random numbers uniformly distributed over the interval `[a, b)`.

Description

The `oneapi::mkl::rng::device::uniform` class object is used in `oneapi::mkl::rng::device::generate` functions to provide random numbers uniformly distributed over the interval `[a, b)`, where `a, b` are the left and right bounds of the interval respectively, and `a, b∈Z; a < b`.

The probability distribution is given by:

$$P(X = k) = \frac{1}{b-a}, k \in \{a, a+1, \ldots, b-1\}$$

The cumulative distribution function is as follows:

$$F_{a,b}(x) = \begin{cases} 0, & x < a \\ \frac{x-a+1}{b-a}, & a \le x < b \\ 1, & x \ge b \end{cases}, x \in R$$

API

Syntax

```
template<typename Type, typename Method>
class uniform<std::(u)int32_t, Method> {
public:
using method_type = Method;
using result_type = Type;
uniform(): uniform((Type)0,
              std::is_same<Method,
              uniform_method::standard>::value ? (1 << 23) :
              std::numeric_limits<Type>::max()){};
explicit uniform(Type a, Type b);
explicit uniform(const param_type& pt);
Type a() const;
Type b() const;
param_type param() const;
void param(const param_type& pt);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

`typename Method`	Generation method. The specific values are as follows:
	`oneapi::mkl::rng::device::uniform_method::standard`
	`oneapi::mkl::rng::device::uniform_method::accurate`
	See brief descriptions of the methods in Distributions Template Parameter Method.

Note: The `oneapi::mkl::rng::device::uniform_method::standard` uses the `s` BRNG type. This might cause the produced numbers to have incorrect statistics (due to rounding error) when $(abs(b–a) > 2^{23} || abs(b) > 2^{23} || abs(a) > 2^{23})$. To get proper statistics for this case, use the `oneapi::mkl::rng::device::uniform_method::accurate` method instead.

Input Parameters

Name	Type	Description
c	`std::int32_t` `std::uint32_t`	Left bound a
d	`std::int32_t` `std::uint32_t`	Right bound b

oneapi::mkl::rng::device::bits

Generates bits of underlying engine (BRNG) integer recurrence.

Description

The `oneapi::mkl::rng::device::bits` class object is used to generate integer random values. Each integer can be treated as a vector of several bits. In a truly random generator, these bits are random, while in pseudo-random generators this randomness can be violated. See VS Notes for details.

API

Syntax

```
template<typename UIntType = std::uint32_t>
  class bits {
  using result_type = UIntType;
  };
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

`typename UIntType = std::uint32_t`	Type of the produced values. The specific values are as follows: `std::uint32_t`

oneapi::mkl::rng::device::poisson

Generates Poisson distributed random values.

Description

The `oneapi::mkl::rng::device::poisson` class object is used in the `oneapi::mkl::rng::device::generate` function to provide Poisson distributed random numbers with distribution parameter λ, where $\lambda \in R$; $\lambda > 0$.

The probability distribution is given by:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

$k \in \{0, 1, 2, \ldots\}$.

The cumulative distribution function is as follows:

$$F_\lambda(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} \frac{\lambda^k e^{-\lambda}}{k!}, & x \geq 0 \\ 0, & x < 0 \end{cases}, x \in R$$

API

Syntax

```
template<typename IntType, typename Method>
class poisson {
public:
using method_type = Method;
using result_type = IntType;
poisson(): poisson(0.5){} explicit poisson(double lambda);
explicit poisson(const param_type& pt);
double lambda() const;
param_type param() const;
void param(const param_type& pt);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

Name	Description

`typename IntType`	Type of the produced values. The specific values are as follows:
	`std::int32_t`
	`std::uint32_t`

`typename Method`	Generation method. The specific values are as follows: `oneapi::mkl::rng::device::poisson_method::devroye`

Note: The Poisson `oneapi::mkl::rng::device::poisson_method::devroye` method uses the following underlying methods depending on the lambda parameter:

- table-lookup method for small lambdas ($\lambda < 60$)
- Devroye's method [Devroye] for medium lambdas ($60 \leq \lambda < 1000$)
- Gaussian approximation for huge lambdas ($\lambda \geq 1000$)

Input Parameters

Name	Type	Description
lambda	`double`	Distribution parameter λ.

oneapi::mkl::rng::device::bernoulli

Generates Bernoulli distributed random values.

Description

The `oneapi::mkl::rng::device::bernoulli` class object is used in the `oneapi::mkl::rng::device::generate` function to provide Bernoulli distributed random numbers with probability p of a single trial success, where $p \epsilon R; 0 \leq p \leq 1$.

A variate is called Bernoulli distributed if after a trial it is equal to 1 with probability of success p and to `0` with probability `1 - p`.

The probability distribution is given by:

`P(X=1) = p`

`P(X=0) = 1 - p`

The cumulative distribution function is as follows:

$$F_p(x) = \begin{cases} 0, & x < 0 \\ 1-p, & 0 \leq x < 1, x \in R \\ 1, & x \geq 1 \end{cases}$$

API

Syntax

```
template<typename IntType, typename Method>
class bernoulli {
public:
  using method_type = Method;
  using result_type = IntType;
  bernoulli(): bernoulli(0.5f){}
  explicit bernoulli(float p);
  explicit bernoulli(const param_type& pt);
  float p() const;
  param_type param() const;
  void param(const param_type& pt);
};
```

Include Files

- `oneapi/mkl/rng/device.hpp`

Template Parameters

`typename IntType`	Type of the produced values. The specific values are as follows:
	`std::int32_t`
	`std::uint32_t`

`typename Method`	Generation method. The specific values are as follows: `oneapi::mkl::rng::device::bernoulli_method::icdf` See brief descriptions of the methods in Distributions Template Parameter Method.

Input Parameters

Name	Type	Description
p	`float`	Success probability p of a trial.

14.0 Summary Statistics

Intel® oneAPI Math Kernel Library (oneMKL) provides Data Parallel C++ interfaces for the Summary Statistics routines that compute basic statistical estimates for single and double precision multi-dimensional datasets.

14.1 Definitions

Struct dataset consolidates the information of a multi-dimensional dataset (see detailed description in Dataset). All computational routines are represented as free functions (see detailed description for each routine in Summary Statistics Routines):

- Raw and central sums/moments up to the fourth order
- Variation coefficient
- Skewness and excess kurtosis (further referred to as a kurtosis)
- Minimum and maximum

14.2 Routines

14.2.1 oneMKL Summary Statistics Usage Model

Description

A typical algorithm for random number generators is as follows:

1. Create and initialize the object for the dataset.
2. Call the summary statistics routine to calculate the appropriate estimate.

The following example demonstrates how to calculate mean values for a 3-dimentional dataset filled with random numbers. For dataset creation, the `make_dataset` helper function is used.

Example of Summary Statistics Usage

Buffer API

```
#include <iostream>
#include <vector>


#include “CL/sycl.hpp”
#include “oneapi/mkl/stats.hpp”


int main() {
    sycl::queue queue;


    const size_t n_observations = 1000;
    const size_t n_dims = 3;
    std::vector<float> x(n_observations * n_dims);
    // fill x storage with random numbers
    for(int i = 0; i < n_dims, i++) {
        for(int j = 0; j < n_observations; j++) {
            x[j + i * n_observations] = float(std::rand()) / float(RAND_MAX);
        }
    }
    //create buffer for dataset
    sycl::buffer<float, 1> x_buf(x.data(), x.size());
    // create buffer for mean values
    sycl::buffer<float, 1> mean_buf(n_dims);
    // create mkl::stats::dataset
    auto dataset = oneapi::mkl::stats::make_dataset<mkl::stats::layout::row_major>(n_dims, n_
↪observations, x_buf);


    oneapi::mkl::stats::mean(queue, dataset, mean_buf);


    // create host accessor for mean_buf to print results
    auto acc = mean_buf.template get_access<sycl::access::mode::read>();


    for(int i = 0; i < n_dims; i++) {
        std::cout << “Mean value for dimension ” << i << “: ”<< acc[i]<<
    std::endl;
    }
    return 0;
}
```

USM API

```
#include <iostream>
#include <vector>


#include “CL/sycl.hpp”
#include “oneapi/mkl/stats.hpp”


int main() {
    sycl::queue queue;


    const size_t n_observations = 1000;
    const size_t n_dims = 3;


    sycl::usm_allocator<float, sycl::usm::alloc::shared> allocator(queue);


    std::vector<float, decltype(allocator)> x(n_observations * n_dims, allocator);
    // fill x storage with random numbers
    for(int i = 0; i < n_dims, i++) {
        for(int j = 0; j < n_observations; j++) {
            x[j + i * n_observations] = float(std::rand()) / float(RAND_MAX);
        }
    }
    std::vector<float, decltype(allocator)> mean_buf(n_dims, allocator);
    // create mkl::stats::dataset
    auto dataset = oneapi::mkl::stats::make_dataset<mkl::stats::layout::row_major>(n_dims,  n_
↪observations, x);


    sycl::event event = oneapi::mkl::stats::mean(queue, dataset, mean);
    event.wait();
   for(int i = 0; i < n_dims; i++) {
         std::cout << “Mean value for dimension ” << i << “: ”<< mean[i]<<
      std::endl;
   }
   return 0;
}
```

You can also use USM with raw pointers by using the `sycl::malloc_shared/malloc_device` functions. Additionally, examples that demonstrate usage of summary statistics functionality are available in:

```
${MKL}/examples/dpcpp/stats/source
```

14.2.2 Summary Statistics Device Support

Data Parallel C++ supports several types of devices:

- CPU device: Performs computations on a CPU using OpenCL™
- GPU device: Performs computations on a GPU

All Data Parallel C++ routines of oneMKL summary statistics support the CPU and GPU devices.

14.2.3 Dataset

The `oneapi::mkl::stats::dataset` structure consolidates information of a multi-dimensional dataset.

- API

API

Syntax

```
namespace oneapi::mkl::stats {
  template<layout ObservationsLayout = layout::row_major,
  typename Type = float*>
```

```
namespace oneapi::mkl::stats {
  struct dataset {}
}
```

Buffer API specialization

```
namespace oneapi::mkl::stats {

  template<layout ObservationsLayout, typename Type>struct dataset<ObservationsLayout, sycl::
↪buffer<Type, 1>> {

     explicit dataset(std::int64_t n_dims_, std::int64_t n_observations_,
           sycl::buffer<Type, 1> observations_, sycl::buffer<Type, 1> weights_ = {0},
           sycl::buffer<std::int64_t, 1> indices_ = {0}) : n_dims(n_dims_), n_observations(n_
↪observations_), observations(observations_),
           weights(weights_), indices(indices_);

 std::int64_t n_dims;
 std::int64_t n_observations;
 sycl::buffer<Type, 1> observations;
 sycl::buffer<Type, 1> weights = {0};
 sycl::buffer<std::int64_t, 1> indices = {0};
 static constexpr layout layout = ObservationsLayout;
 };
}
```

USM API specialization

```
namespace oneapi::mkl::stats {

 template< layout ObservationsLayout, typename Type>
 struct dataset<ObservationsLayout, Type*> {
      explicit dataset(std::int64_t n_dims_, std::int64_t n_observations_, Type* observations_,
           Type* weights_ = nullptr, std::int64_t* indices_ = nullptr) :
           n_dims(n_dims_), n_observations(n_observations_),
           observations(observations_),
           weights(weights_), indices(indices_);


 std::int64_t n_dims;
 std::int64_t n_observations;
 Type* observations;
 Type* weights = nullptr;
 std::int64_t* indices = nullptr;
 static constexpr layout layout = ObservationsLayout;
 };
}
```

Include Files

- `oneapi/mkl/stats.hpp`

Template Parameters

`typename DataType`	Type of dataset. May be `sycl::buffer<float,1>`, `sycl::buffer<double, 1>` (for buffer-based dataset) or `float*`, `double*` (for USM-based dataset).
`oneapi::mkl:: stats::layout ObservationsLayout`	Layout of the observations matrix of the dataset. The specific values are as follows: `oneapi::mkl::stats::layout::row_major` `oneapi::mkl::stats:: layout::col_major`

Struct Members

Name	Type	Description
n_dims	`std::int64_t`	The number of dimensions (variables)
n_observations	`std::int64_t`	The number of observations
layout	`oneapi::mkl::stats::layout`	Characterize the matrix of observations layout (row or column major)
observations	`sycl::buffer<Type, 1> / Type*`	Matrix of observations.
weights	`sycl::buffer<Type, 1> / Type*`	Array of weights of size n_observations. Elements of the arrays are non-negative numbers. If the parameter is not specified, each observation is assigned a weight equal to 1.
indices	`sycl::buffer<std::int64_t, 1> / std::int64_t*`	Array of vector components that will be processed. Size of array is `n_dims`. If the parameter is not specified, all components of the vector are processed.

14.2.4 Service Functions to Create Dataset

Use these functions to create a dataset.

oneapi::mkl::stats::make_dataset

Description

Entry point to create a dataset from the provided parameters.

API

Syntax

Buffer API

```
template<layout ObservationsLayout = layout::row_major,
  typename Type> dataset<ObservationsLayout,
  sycl::buffer<Type, 1>> make_dataset(std::int64_t n_dims,
  std::int64_t n_observations,
  sycl::buffer<Type, 1> observations,
  sycl::buffer<Type, 1> weights = {0},
  sycl::buffer<std::int64_t, 1> indices = {0});
```

USM API

```
template<layout ObservationsLayout = layout::row_major,
  typename Type> dataset<ObservationsLayout,
  Type*> make_dataset(std::int64_t n_dims,
  std::int64_t n_observations,
  Type* observations, Type* weights = nullptr,
  std::int64_t* indices = nullptr);
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

`typename DataType`	Type of dataset. May be `float`, `double`.
`oneapi::mkl::stats::layout ObservationsLayout`	Layout of the observations matrix of the dataset. The specific values are as follows: `oneapi::mkl::stats::layout::row_major` `oneapi::mkl::stats::layout::col_major`

Input Parameters

Name	Type	Description
n_dims	`std::int64_t`	The number of dimensions (variables)
n_observations	`std::int64_t`	The number of observations
observations	`sycl::buffer<Type, 1> / Type*`	Matrix of observations

Optional Input Parameters

Name	Type	Description
weights	`sycl::buffer<Type, 1> / Type*`	Array of weights of size n_observations. Elements of the arrays are non-negative numbers. If the parameter is not specified, each observation is assigned weight equal to 1.
indices	`sycl::buffer<std::int64_t, 1> / std::int64_t*`	Array of vector components that will be processed. Size of array is `n_dims`. If the parameter is not specified, all components of the random vector are processed.

Return Type

Buffer API

`oneapi::mkl::stats::dataset<ObservationsLayout, sycl::buffer<Type, 1>>`	Dataset holding specified parameters

USM API

`oneapi::mkl::stats::dataset<ObservationsLayout, Type*>`	Dataset holding specified parameters

14.2.5 Summary Statistics Routines

The Summary Statistics routines calculate:

- Raw and central sums/moments up to the fourth order
- Variation coefficient
- Skewness and excess kurtosis (further referred to as a kurtosis)
- Minimum and maximum

oneapi::mkl::stats::raw_sum

Description

Entry point to compute arrays of raw sums up to the 4th order.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void raw_sum(sycl::queue& queue,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> sum,
    sycl::buffer<Type, 1> raw_sum_2 = {0},
    sycl::buffer<Type, 1> raw_sum_3 = {0},
    sycl::buffer<Type, 1> raw_sum_4 = {0});
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event max(
    sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* max,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

Name	Description
`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast` `oneapi::mkl::stats::method::one_pass`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::raw_sum()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameters for USM API

Name	Type	Description
dependencies	`const std::vector <sycl::event> &`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
sum	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of sum.
raw_sum_2	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of raw sums of the 2nd order (optional).
raw_sum_3	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of raw sums of the 3rd order (optional).
raw_sum_4	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of raw sums of the 4th order (optional).

USM API

Name	Type	Description
sum	`Type*`	Pointer to the output array of sum.
raw_sum_2	`Type*`	Pointer to the output array of raw sums of the 2nd order (optional).
raw_sum_3	`Type*`	Pointer to the output array of raw sums of the 3rd order (optional).
raw_sum_4	`Type*`	Pointer to the output array of raw sums of the 4th order (optional).
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::central_sum

Description

Entry point to compute arrays of central sums up to the 4th order.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void central_sum(sycl::queue& queue,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> central_sum_2,
    sycl::buffer<Type, 1> central_sum_3 = {0},
    sycl::buffer<Type, 1> central_sum_4 = {0});
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event central_sum(sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* central_sum_2,
    Type* central_sum_3 = nullptr,
    Type* central_sum_4 = nullptr,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast` `oneapi::mkl::stats::method::one_pass`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::central_sum()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameters for USM API

Name	Type	Description
dependencies	`const std::vector <sycl::event> &`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
central_sum_2	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central sums of the 2nd order (optional).
central_sum_3	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central sums of the 3rd order (optional).
central_sum_4	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central sums of the 4th order (optional).

USM API

Name	Type	Description
central_sum_2	`Type*`	Pointer to the output array of central sums of the 2nd order (optional).
central_sum_3	`Type*`	Pointer to the output array of central sums of the 3rd order (optional).
central_sum_4	`Type*`	Pointer to the output array of central sums of the 4th order (optional).
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::central_sum with User-provided Mean

Description

Entry point to compute arrays of central sums up to the 4th order with mean provided by the user.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void central_sum(sycl::queue& queue,
    sycl::buffer<Type, 1> mean,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> central_sum_2,
    sycl::buffer<Type, 1> central_sum_3 = {0},
    sycl::buffer<Type, 1> central_sum_4 = {0});
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event central_sum(sycl::queue& queue,
    Type* mean,
    const dataset<ObservationsLayout, Type*>& data,
    Type* central_sum_2,
    Type* central_sum_3 = nullptr,
    Type* central_sum_4 = nullptr,
    const std::vector<sycl::event> &dependencies = {});;
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::central_sum()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Buffer API

Name	Type	Description
mean	`sycl::buffer<Type,1>`	`sycl::buffer` to the array of mean values provided by the user.

USM API

Name	Type	Description
mean	Type*	Pointer to the output array of mean values provided by the user.

Output Parameters

Buffer API

Name	Type	Description
central_sum_2	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central sums of the 2nd order.
central_sum_3	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central sums of the 3rd order.
central_sum_4	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central sums of the 4th order.

USM API

Name	Type	Description
central_sum_2	Type*	Pointer to the output array of central sums of the 2nd order.
central_sum_3	Type*	Pointer to the output array of central sums of the 3rd order.
central_sum_4	Type*	Pointer to the output array of central sums of the 4th order.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::raw_moment

Description

Entry point to compute arrays of raw moments up to 4th order.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void raw_moment(sycl::queue& queue,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> mean,
    sycl::buffer<Type, 1> raw_moment_2 = {0},
    sycl::buffer<Type, 1> raw_moment_3 = {0},
    sycl::buffer<Type, 1> raw_moment_4 = {0});
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event raw_moment(sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* mean,
    Type* raw_moment_2 = nullptr,
    Type* raw_moment_3 = nullptr,
    Type* raw_moment_4 = nullptr,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

Name	Description
`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast` `oneapi::mkl::stats::method::one_pass`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::raw_moment()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameters for USM API

Name	Type	Description
dependencies	`const std::vector <sycl::event> &`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
mean	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of mean values.
raw_moment_2	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of raw moments of the 2nd order (optional).
raw_moment_3	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of raw moments of the 3rd order (optional).
raw_moment_4	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of raw moments of the 4th order (optional).

USM API

Name	Type	Description
mean	`Type*`	Pointer to the output array of mean values.
raw_moment_2	`Type*`	Pointer to the output array of raw moments of the 2nd order (optional).
raw_moment_3	`Type*`	Pointer to the output array of raw moments of the 3rd order (optional).
raw_moment_4	`Type*`	Pointer to the output array of raw moments of the 4th order (optional).
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::central_moment

Description

Entry point to compute arrays of central moments up to 4th order.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void central_moment(sycl::queue& queue,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> central_moment_2,
    sycl::buffer<Type, 1> central_moment_3 = {0},
    sycl::buffer<Type, 1> central_moment_4 = {0});
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event central_moment(sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* central_moment_2,
    Type* central_moment_3 = nullptr,
    Type* central_moment_4 = nullptr,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast` `oneapi::mkl::stats::method::one_pass`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::central_moment()` routine submits kernels in this queue.
data	`const dataset< ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameters for USM API

Name	Type	Description
dependencies	`const std::vector <sycl::event> &`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
central_moment_2	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central moments of the 2nd order.
central_moment_3	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central moments of the 3rd order (optional).
central_moment_4	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central moments of the 4th order (optional).

USM API

Name	Type	Description
central_moment_2	`Type*`	Pointer to the output array of central moments of the 2nd order.
central_moment_3	`Type*`	Pointer to the output array of central moments of the 3rd order (optional).
central_moment_4	`Type*`	Pointer to the output array of central moments of the 4th order (optional).
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::central_moment with User-provided Mean

Description

Entry point to compute arrays of central moments up to the 4th order with mean provided by the user.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void central_moment(sycl::queue& queue,
    sycl::buffer<Type, 1> mean,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> central_moment_2,
    sycl::buffer<Type, 1> central_moment_3 = {0},
    sycl::buffer<Type, 1> central_moment_4 = {0});
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event central_moment(sycl::queue& queue,
    Type* mean,
    const dataset<ObservationsLayout, Type*>& data,
    Type* central_moment_2,
    Type* central_moment_3 = nullptr,
    Type* central_moment_4 = nullptr,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::central_moment()` routine submits kernels in this queue.
data	`const dataset< ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Buffer API

Name	Type	Description
mean	`sycl::buffer<Type,1>`	`sycl::buffer` to the array of mean values provided by the user.

USM API

Name	Type	Description
mean	Type*	Pointer to the output array of mean values provided by the user.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
central_moment_2	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central moments of the 2nd order.
central_moment_3	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central moments of the 3rd order.
central_moment_4	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of central moments of the 4th order.

USM API

Name	Type	Description
central_moment_2	Type*	Pointer to the output array of central moments of the 2nd order.
central_moment_3	Type*	Pointer to the output array of central moments of the 3rd order.
central_moment_4	Type*	Pointer to the output array of central moments of the 4th order.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::mean

Description

Entry point to compute the arrays of mean values.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void mean(sycl::queue& queue,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> mean);
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event mean(sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* mean,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

Name	Description
`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast` `oneapi::mkl::stats::method::one_pass`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::mean()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
mean	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of mean values.

USM API

Name	Type	Description
mean	Type*	Pointer to the output array of mean values.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::variation

Description

Entry point to compute the arrays of variation coefficients.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void variation(sycl::queue& queue,
    const dataset< ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> variation;
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event variation(sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* variation,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

Name	Description
`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast` `oneapi::mkl::stats::method::one_pass`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::variation()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
variation	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of variation coefficients.

USM API

Name	Type	Description
variation	`Type*`	Pointer to the output array of variation coefficients.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::variation with User-provided Mean

Description

Entry point to compute the arrays of variation coefficients with the mean provided by the user.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void variation(sycl::queue& queue,
    sycl::buffer<Type, 1> mean,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> variaion);
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event variation(sycl::queue& queue,
    Type* mean,
    const dataset<ObservationsLayout, Type*>& data,
    Type* variation,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

Name	Description
`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::variation()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Buffer API

Name	Type	Description
mean	`sycl::buffer<Type,1>`	sycl::buffer to the array of mean values provided by the user.

USM API

Name	Type	Description
mean	Type*	Pointer to the output array of mean values provided by the user.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
variation	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of variation coefficients.

USM API

Name	Type	Description
variation	Type*	Pointer to the output array of variation coefficients.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::skewness

Description

Entry point to compute the arrays of skewness values.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void skewness(sycl::queue& queue,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> skewness);
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event skewness(sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* skewness,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

Name	Description
`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast` `oneapi::mkl::stats::method::one_pass`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::skewness()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
skewness	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of skewness values.

USM API

Name	Type	Description
skewness	Type*	Pointer to the output array of skewness values.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::skewness with User-provided Mean

Description

Entry point to compute the arrays of skewness values with a mean provided by the user.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void skewness(sycl::queue& queue,
    sycl::buffer<Type, 1> mean,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> skewness);
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event skewness(sycl::queue& queue,
    Type* mean,
    const dataset<ObservationsLayout, Type*>& data,
    Type* skewness,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

Name	Description
`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::skewness()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Buffer API

Name	Type	Description
mean	`sycl::buffer<Type,1>`	`sycl::buffer` to the array of mean values provided by the user.

USM API

Name	Type	Description
mean	Type*	Pointer to the output array of mean values provided by the user.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
skewness	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of skewness values.

USM API

Name	Type	Description
variation	Type*	Pointer to the output array of variation coefficients.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::kurtosis

Description

Entry point to compute the array of kurtosis values.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void kurtosis(sycl::queue& queue,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> kurtosis);
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event kurtosis(sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* kurtosis,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast` `oneapi::mkl::stats::method::one_pass`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::kurtosis()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
kurtosis	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of kurtosis values.

USM API

Name	Type	Description
kurtosis	Type*	Pointer to the output array of kurtosis values.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::kurtosis with User-provided Mean

Description

Entry point to compute the array of kurtosis values with a mean provided by the user.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void kurtosis(sycl::queue& queue,
    sycl::buffer<Type, 1> mean,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> kurtosis);
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event kurtosis(sycl::queue& queue,
    Type* mean,
    const dataset<ObservationsLayout, Type*>& data,
    Type* kurtosis,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::kurtosis()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Buffer API

Name	Type	Description
mean	`sycl::buffer<Type,1>`	`sycl::buffer` to the array of mean values provided by the user.

USM API

Name	Type	Description
mean	Type*	Pointer to the output array of mean values provided by the user

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
kurtosis	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of kurtosis values.

USM API

Name	Type	Description
kurtosis	Type*	Pointer to the output array of kurtosis values.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::min

Description

Entry point to compute the array of minimum values.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void min(
    sycl::queue& queue,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> min);
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event min(
    sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* min,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

Name	Description
`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::min()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
min	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of minimum values.

USM API

Name	Type	Description
min	`Type*`	Pointer to the output array of minimum values.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::max

Description

Entry point to compute the array of maximum values.

API

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void max(
    sycl::queue& queue,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> max);
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event max(
    sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* max,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::max()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
max	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of maximum values.

USM API

Name	Type	Description
max	Type*	Pointer to the output array of maximum values.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

oneapi::mkl::stats::min_max

Description

Entry point to compute the array of minimum and maximum values simultaneously.

Syntax

Buffer API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
void min_max(sycl::queue& queue,
    const dataset<ObservationsLayout, sycl::buffer<Type, 1>>& data,
    sycl::buffer<Type, 1> min, sycl::buffer<Type, 1> max);
```

USM API

```
template<method Method = method::fast, typename Type, layout ObservationsLayout>
sycl::event min_max(sycl::queue& queue,
    const dataset<ObservationsLayout, Type*>& data,
    Type* min, Type* max,
    const std::vector<sycl::event> &dependencies = {});
```

Include Files

- oneapi/mkl/stats.hpp

Template Parameters

Name	Description
`oneapi::mkl::stats::method Method = oneapi::mkl::stats::method::fast`	Computation method. The specific values are as follows: `oneapi::mkl::stats::method::fast`

Input Parameters

Name	Type	Description
queue	`sycl::queue&`	Valid `sycl::queue`, calls of the `oneapi::mkl::stats::min_max()` routine submits kernels in this queue.
data	`const dataset<ObservationsLayout, Type*>&`	Dataset which is used for estimates computation.

Optional Input Parameter for USM API

Name	Type	Description
dependencies	`const std::vector<sycl::event>&`	List of events to wait for before starting computation, if any.

Output Parameters

Buffer API

Name	Type	Description
min	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of minimum values.
max	`sycl::buffer<Type, 1>`	`sycl::buffer` to the output array of maximum values.

USM API

Name	Type	Description
min	`Type*`	Pointer to the output array of minimum values.
max	`Type*`	Pointer to the output array of maximum values.
event	`sycl::event`	Function returns event after submitting task in `sycl::queue`.

15.0 Fourier Transform Functions

The general form of the discrete Fourier transform is:

$$Z_{(k_1,k_2,\ldots,k_d)} = \sigma \times \sum_{j_d=0}^{n_d-1} \cdots \sum_{j_2=0}^{n_2-1} \sum_{j_1=0}^{n_1-1} w_{j_1,j_2\ldots j_d} \exp\left(\delta i 2\pi \sum_{l=1}^{d} j_l k_l / n_l\right)$$

for $k_l = 0, \ldots n_l - 1 (l = 1, \ldots, d)$, where σ is a scale factor, $\delta = -1$ for the forward transform, and $\delta = +1$ for the inverse (backward) transform. In the forward transform, the input (periodic) sequence $\{w_{j1}, w_{j2}, \ldots, w_{jd}\}$ belongs to the set of complex-valued sequences or the set of real-valued sequences, and the output sequence belongs to the set of complex-valued sequences or the set of complex-valued conjugate-even sequences, respectively.

The Intel® oneAPI Math Kernel Library (oneMKL) provides a DPC++ interface for computing a discrete Fourier transform through the fast Fourier transform algorithm. The DPC++ interface emulates the usage model of the oneMKL C and Fortran Discrete Fourier Transform Interface (DFTI).

The DPC++ interface computes an FFT in four steps:

1. Allocate a fresh descriptor for the problem with a call to the descriptor object constructor,

   ```
   descriptor<PRECISION, DOMAIN> desc(dimensions);
   ```

   The descriptor captures the configuration of the transform, such as the dimensionality (or rank), sizes, number of transforms, memory layout of the input/output data (defined by strides), and scaling factors. Many of the configuration settings are assigned default values in this call which you might need to modify in your application.

2. Optionally adjust the descriptor configuration with a call to the `descriptor<PRECISION, DOMAIN>::set_value` function as needed. Typically, you must carefully define the data storage layout for an FFT. The configuration settings of the descriptor, such as the default values, can be obtained with the `descriptor<PRECISION, DOMAIN>::get_value` function.

3. Commit the descriptor with a call to the `descriptor<PRECISION, DOMAIN>::commit` function, that is, make the descriptor ready for the transform computation. Once the descriptor is committed, the parameters of the transform, such as the type and number of transforms, strides and distances, the type and storage layout of the data, and so on, are "frozen" in the descriptor. The commit function takes in a `sycl::queue` object to determine the device associated with the descriptor.

4. Compute the transform with a call to the `compute_forward` or `compute_backward` functions as many times as needed. Because the descriptor is defined and committed separately, the compute functions just take the input and output data and compute the transform as defined. To modify any configuration parameters for another call to a compute function, use

   `descriptor<PRECISION, DOMAIN>::set_value`

   followed by

   `descriptor<PRECISION, DOMAIN>::commit`

   before calling the compute function again.

All the above functions throw a named `std::runtime_exception` in the event of failure.

To use the DPC++ interface, include `oneapi/mkl/dfti.hpp`. The DPC++ interface supports CPU and Intel GPU devices. Some device-specific limitations are noted in the **descriptor<precision, domain>::set_value** section of this document. Refer to the Intel® oneAPI Math Kernel Library Release Notes for known limitations. Working usage examples can be found in the oneMKL installation directory under `examples/dpcpp/dft`.

The FFT functions assume the Cartesian representation of complex data (that is, the real and imaginary parts define a complex number). The Intel® oneAPI Math Kernel Library Vector Mathematical Functions provide efficient tools for conversion to and from polar representation. See the Conversion from Cartesian to polar representation of complex data and Conversion from polar to Cartesian representation of complex data examples.

15.1 descriptor<precision, domain>

Creates a descriptor for the templated precision and forward domain with configuration values. This routine belongs to the `oneapi::mkl::dft` namespace.

15.1.1 Description

This constructor initializes members to default values and throws an `std::runtime_exception` in the case that it fails. Note that the precision and domain are determined via templating. This function allocates memory for the descriptor data structure and instantiates it with all the default configuration settings for the precision, forward domain, and dimensions of the transform.

This function does not perform any significant computational work such as computation of twiddle factors. The function `descriptor::commit` does this work after the function `descriptor::set_value` has set values of all necessary parameters.

The interface supports a single `MKL_LONG` input for 1-D transforms, and an `std::vector` for N-D transforms.

15.1.2 API

Syntax

```
namespace oneapi::mkl::dft{
  descriptor<PRECISION, DOMAIN> desc (dimension)
}
```

```
namespace oneapi::mkl::dft{
  descriptor<PRECISION, DOMAIN> desc ({dim1, dim2, ...})
}
```

Include Files

- `oneapi/mkl/dfti.hpp`

Template Parameters

Name	Type	Description
PRECISION	`mkl::dft::Precision`	`mkl::dft::Precision::SINGLE` or `mkl::dft::Precision::DOUBLE` are supported precisions. Double precision has limited GPU support and full CPU support.
DOMAIN	`mkl::dft::Domain`	`mkl::dft::Domain::REAL` or `mkl::dft::Domain::COMPLEX` are supported forward domains.

Input Parameters: 1-Dimensional

Name	Type	Description
dimension	MKL_LONG	Dimension of the transform 1-D transform.

Input Parameters: N-Dimensional

Name	Type	Description
dimensions	`std::vector<MKL_LONG>`	Dimensions of the transform.

15.2 descriptor<precision, domain>::set_value

Sets one configuration parameter with the specified configuration value. This routine belongs to the `oneapi::mkl::dft` namespace.

15.2.1 Description

This function sets one configuration parameter with the specified configuration value and throws an `std::runtime_exception` in the case that it fails. Each configuration parameter is a named constant, and the configuration value must have the corresponding type, which can be a named constant or a native type. The DPC++ configuration parameters have type `mkl::dft::config_param`. Each parameter `mkl::dft::config_param::<PARAMETER_NAME>` has the same meaning and possible values as the C interface's `DFTI_<PARAMETER_NAME>`. Some of the options available in the C interface are not fully supported in the DPC++ interface yet. See the note below and the Intel® oneAPI Math Kernel Library Release Notes. For a full description of each parameter's meaning and options, see Config Params.

Note:

- An important difference in the configuration options for DPC++ FFT interface is the use of `FWD_DISTANCE` and `BWD_DISTANCE` instead of `INPUT_DISTANCE` and `OUTPUT_DISTANCE`. The `FWD_DISTANCE` describes the number of elements between different batched FFTs for the forward domain while `BWD_DISTANCE` describes the number of elements between different batched FFTs for the backward domain. This allows a committed descriptor to compute both forward and backward transforms without resetting the distance parameters. It is required for all R2C or C2R transforms to use `FWD_DISTANCE` and `BWD_DISTANCE` instead of `INPUT_DISTANCE` and `OUTPUT_DISTANCE`.
- For `COMPLEX_STORAGE`, only the `DFTI_COMPLEX_COMPLEX` format is currently supported on CPU and GPU devices.
- For batched 3D C2C problems, only `FWD_DISTANCE` or `BWD_DISTANCE` equal to the problem size are supported currently.
- In the case of a real forward domain, the only supported values on GPU devices for `CONJUGATE_EVEN_STORAGE` and `PACKED_FORMAT` are respectively `DFTI_COMPLEX_COMPLEX` and `DFTI_CCE_FORMAT`. These values are set by default and require the setting of the `INPUT_STRIDE` and `OUTPUT_STRIDE` parameters before calling commit. (While these default values differ from the current C interface defaults, the C default values will be brought into alignment with these default values following the full deprecation process of the current C default values. See the deprecation notice at DFTI_COMPLEX_STORAGE, DFTI_REAL_STORAGE, DFTI_CONJUGATE_EVEN_STORAGE.)
- In the case of a real forward domain, arbitrary strides and batch distances are not supported for multi-dimensional transforms on GPU. Considering the last dimension of the data, every element must be separated from its nearest peer(s) (along another dimension and/or in another batch) by a constant distance. For example, to compute a batched, two-dimensional R2C FFT of size `[N2, N1]` with input strides `[0, S2, 1]` (row-major layout with unit elementary stride and no offset), `FWD_DISTANCE` must be equal to `N2*S2` so that every element is separated from its nearest last-dimension counterpart(s) by a distance `S2` (in this example), even across batches.

The set_value function cannot be used to change configuration parameters `mkl::dft::config_param::FORWARD_DOMAIN`, `mkl::dft::config_param::PRECISION` since these are a part of the template. Likewise, `mkl::dft::config_param::LENGTHS` is set with the constructor.

All calls to `set_value()` must be done before calls to `commit()`. This is because the handle may have been moved to an offloaded device after `commit()`.

Function calls needed to configure an FFT descriptor for a particular call to an FFT computation function are summarized in Configuring and Computing an FFT in C/C++.

15.2.2 API

Syntax

```
namespace oneapi::mkl::dft{
  void descriptor<prec,dom>::set_value(config_param param, ...);
}
```

Include Files

- `oneapi/mkl/dfti.hpp`

Input Parameters

Name	Type	Description
param	`mkl::dft::config_param`	Configuration parameter
value	Depends on the configuration parameter	Configuration value

15.3 descriptor<precision, domain>::get_value

Gets the configuration value of one configuration parameter. This routine belongs to the `oneapi::mkl::dft` namespace.

15.3.1 Description

This function gets the configuration value of one particular configuration parameter and will throw an `std::runtime_exception()` in the case that it fails or an invalid configuration is provided.

Each configuration parameter is a named constant, and the configuration value must have the corresponding type, which can be a named constant or a native type. For available configuration parameters and the corresponding configuration values, see DftiGetValue.

15.3.2 API

Syntax

```
namespace oneapi::mkl::dft{
  void descriptor<prec,dom>::get_value (config_param param , ...)
}
```

Include Files

- `oneapi/mkl/dfti.hpp`

Input Parameters

Name	Type	Description
param	enum CONFIG_PARAM	Configuration parameter.
value_ptr	Depends on the configuration parameter	Pointer to configuration value.

15.4 descriptor<precision, domain>::commit

Performs all initialization for the actual FFT computation. This routine belongs to the `oneapi::mkl::dft` namespace.

15.4.1 Description

This function completes initialization of a previously created descriptor, which is required before the descriptor can be used for FFT computations. It throws an `std::runtime_exception()` in the case that it fails. The `sycl::queue` may be associated with a CPU or GPU device. Typically, committing the descriptor performs all initialization that is required for the actual FFT computation on the given device. The initialization done by the function may involve exploring different factorizations of the input length to find the optimal computation method.

Note: Calls to the `descriptor<precision, domain>::set_value` function to change configuration parameters of a descriptor need to happen before `descriptor<precision, domain>::commit`. Typically, a committal function call is immediately followed by a computation function call (see FFT Compute Functions).

15.4.2 API

Syntax

```
namespace oneapi::mkl::dft{
  void descriptor<prec, dom>::commit(sycl::queue &in)
}
```

Include Files

- `oneapi/mkl/dfti.hpp`

Input Parameters

Name	Type	Description
deviceQueue	`sycl::queue`	Sycl queue for a CPU or GPU device.

15.5 compute_forward<typename descriptor_type, typename data_type>

Computes the forward FFT. This routine belongs to the `oneapi::mkl::dft` namespace.

15.5.1 Description

The compute_forward<typename descriptor_type, typename data_type> function accepts a descriptor object followed by one or more data parameters. With a successfully configured and committed descriptor, this function computes the forward FFT, that is, the transform with the minus sign in the exponent, $\delta = -1$. This function throws an `std::runtime_exception()` in the case that it fails.

The FFT descriptor must be properly configured prior to the function call. Function calls needed to configure an FFT descriptor for a particular call to an FFT computation function are summarized in Configuring and Computing an FFT in C/C++.

The number of data parameters to be provided depends exclusively on the descriptor's PLACEMENT configuration parameter. In-place (default) operations require only one data parameter while out-of-place operations take two distinct data parameters. The function's behavior is undefined otherwise.

15.5.2 API

Syntax

Using SYCL buffers

```
namespace oneapi::mkl::dft{
  void compute_forward<descriptor_type,
  data_type>(descriptor_type &desc, sycl::buffer<data_type,
  1> &inout);


  void compute_forward<descriptor_type, input_type,
  output_type>(descriptor_type &desc,
  sycl::buffer<input_type, 1> &in,
  sycl::buffer<output_type, 1> &out);
}
```

Using USM pointers

```
namespace oneapi::mkl::dft{
  sycl::event compute_forward<descriptor_type,
  data_type>(descriptor_type &desc, data_type* inout, const
  std::vector<sycl::event> &dependencies = {});


  sycl::event compute_forward<descriptor_type, input_type,
  output_type>(descriptor_type &desc, input_type* in,
  output_type* out, const
  std::vector<sycl::event> &dependencies = {});
}
```

Include Files

- `oneapi/mkl/dfti.hpp`

Input Parameters

Name	Supported types	Description
inout	i. `sycl::buffer<data_type, 1>` ii. `data_type*`	Input/output data
in	i. `sycl::buffer<input_type, 1>` ii. `input_type*`	Input data

- The parameter name inout (resp. in) corresponds to using a descriptor configured to operate INPLACE (resp. NOT_INPLACE). See descriptor's PLACEMENT.
- Whether using i. SYCL buffers or ii. device-accessible USM pointers, the length of the containing array must be no less than specified at construction of the descriptor.

Output Parameters

Name	Supported types	Description
inout	i. `sycl::buffer<data_type, 1>` ii. `data_type*`	Input/output data
out	i. `sycl::buffer<output_type, 1>` ii. `output_type*`	Output data

- The parameter name inout (resp. out) corresponds to using a descriptor configured to operate INPLACE (resp. NOT_INPLACE). See descriptor's PLACEMENT.
- Whether using i. SYCL buffers or ii. device-accessible USM pointers, the length of the containing array must be no less than specified at construction of the descriptor.
- In case of out-of-place operations, the same type is to be used for the parameters in and out.

15.6 compute_backward<typename descriptor_type, typename data_type>

Computes the backward FFT. This routine belongs to the `oneapi::mkl::dft` namespace.

15.6.1 Description

The `compute_backward<typename descriptor_type, typename data_type>` function accepts a descriptor object followed by one or more data parameters. Given a successfully configured and committed descriptor, this function computes the backward FFT, that is, the transform with the plus sign in the exponent, $\delta = +1$. This function throws an `std::runtime_exception` in the case that it fails.

The FFT descriptor must be properly configured prior to the function call. Function calls needed to configure an FFT descriptor for a particular call to an FFT computation function are summarized in Configuring and Computing an FFT in C/C++.

The number of data parameters to be provided depends exclusively on the descriptor's PLACEMENT configuration parameter. In-place (default) operations require only one data parameter while out-of-place operations take two distinct data parameters. The function's behavior is undefined otherwise.

15.6.2 API

Syntax

Using SYCL buffers

```
namespace oneapi::mkl::dft{
  void compute_backward<descriptor_type,
  data_type>(descriptor_type &desc, sycl::buffer<data_type,
  1> &inout);


void compute_backward<descriptor_type, input_type,
  output_type>(descriptor_type &desc,
  sycl::buffer<input_type, 1> &in,
  sycl::buffer<output_type, 1> &out);
}
```

Using USM pointers

```
namespace oneapi::mkl::dft{
  sycl::event compute_backward<descriptor_type,
  data_type>(descriptor_type &desc, data_type* inout, const
  std::vector<sycl::event> &dependencies = {});


  sycl::event compute_backward<descriptor_type, input_type,
  output_type>(descriptor_type &desc, input_type* in,
  output_type* out, const
  std::vector<sycl::event> &dependencies = {});
}
```

Include Files

- `oneapi/mkl/dfti.hpp`

Input Parameters

Name	Supported types	Description
inout	i. `sycl::buffer<data_type, 1>` ii. `data_type*`	Input/output data
in	i. `sycl::buffer<input_type, 1>` ii. `input_type*`	Input data

- The parameter name inout (resp. in) corresponds to using a descriptor configured to operate INPLACE (resp. NOT_INPLACE). See descriptor's PLACEMENT.
- Whether using i. SYCL buffers or ii. device-accessible USM pointers, the length of the containing array must be no less than specified at construction of the descriptor.

Output Parameters

Name	Supported types	Description
inout	i. `sycl::buffer<data_type, 1>` ii. `data_type*`	Input/output data
out	i. `sycl::buffer<output_type, 1>` ii. `output_type*`	Output data

- The parameter name inout (resp. out) corresponds to using a descriptor configured to operate INPLACE (resp. NOT_INPLACE). See descriptor's PLACEMENT.
- Whether using i. SYCL buffers or ii. device-accessible USM pointers, the length of the containing array must be no less than specified at construction of the descriptor.
- In case of out-of-place operations, the same type is to be used for the parameters in and out.

15.7 User-Allocated Workspaces

During the FFT execution, the descriptor object may require additional memory to store intermediate results or pre-computed data. By default, the descriptor allocates this workspace internally. oneMKL DFT provides a combination of existing and new APIs to enable the user to manage this workspace.

Note: User-allocated workspaces are currently supported only for GPU devices.

15.7.1 User's intention regarding workspace

The descriptor allocates any required workspace internally during the `commit` by default. The following needs to be set before `commit` to let oneMKL DFT know that the workspace will be provided by the user.

```
descriptor.set_value(oneapi::mkl::dft::config_param::WORKSPACE, oneapi::mkl::dft::config_value::
↪WORKSPACE_EXTERNAL);
```

The above call will notify the descriptor to not allocate any internal workspace during the `commit`.

Note: If the user did not notify the descriptor before `commit` that an external workspace will be provided, they can directly call `set_workspace`, which will override the internal allocations and utilize the user-workspace.

15.7.2 Get workspace estimate

The below optional call to `get_value` before the `commit` provides a conservative estimate of the workspace size that may be required.

```
size_t workspaceEstimate = 0;
descriptor.get_value(oneapi::mkl::dft::config_param::WORKSPACE_ESTIMATE, &workspaceEstimate);
```

The estimate in bytes will be stored in the workspaceEstimate variable.

Note: The workspace estimate values pertain to GPU devices only.

15.7.3 Get the exact workspace size

The below call to `get_value` returns the exact amount of workspace size that will be required by the descriptor for computing the FFT. This needs to be called after the `commit`.

```
size_t workspaceSize = 0;
descriptor.get_value(oneapi::mkl::dft::config_param::WORKSPACE_BYTES, &workspaceSize);
```

The workspace size in bytes will be stored in the workspaceSize variable.

Note: Trying to get the exact workspace size before `commit` will throw a `std::runtime_exception()`.

15.7.4 Provide the workspace to the descriptor

The below descriptor class member functions will enable the user to set/replace the workspace either in the form of SYCL buffer or device-accessible USM. If the workspace was internally allocated (`config_param::WORKSPACE` was not set to `config_value::WORKSPACE_EXTERNAL` before commit), setting the workspace will deallocate any internal workspace and user-provided workspace will be used.

Syntax

```
namespace oneapi::mkl::dft{
  void descriptor<prec, dom>::set_workspace<data_type>(sycl::buffer<data_type, 1> &
↪workspaceBuf);
  void descriptor<prec, dom>::set_workspace<data_type>(data_type *workspaceUsm);
}
```

Note:

- The descriptor assumes that the workspace will be solely used for FFT, so the user must be careful not to use the same workspace area for other purposes in parallel or before finishing `compute_forward`/`compute_backward` (especially for USM as explicit synchronization is required).
- In case of USM, it is assumed that the workspace is large enough. In case of Buffer, `std::runtime_exception()` will be thrown if the workspace size is smaller than required.
- If the user calls `compute_forward` or `compute_backward` before setting the workspace a `std::runtime_exception()` will be thrown.
- Currently, the workspace type (Buffer or USM) must be the same as the container type(Buffer or USM) used during the compute stage. Otherwise, `std::runtime_exception()` will be thrown.

Include Files

- `oneapi/mkl/dfti.hpp`

16.0 Data Fitting

Intel® oneAPI Math Kernel Library (oneMKL) provides spline-based interpolation capabilities that can be used for spline construction (Linear, Cubic, Quadratic etc.), to perform cell-search operations, and to approximate functions, function derivatives, or integrals.

APIs are experimental. It means that no API or ABI backward compatibility are guaranteed.

APIs are based on SYCL USM (Unfied Shared Memory) input datas.

16.1 Routines

Splines: Splines

Interpolate function: Interpolate Function

16.2 Error Handling

The DPC++ error handling model supports two types of errors:

1. Synchronous errors cause the DPC++ host runtime libraries throw exceptions.
2. Asynchronous errors may only be processed in a user-supplied error handler associated with a SYCL queue.

For routines, handling all errors, synchronous or asynchronous, is a responsibility of the caller. Specifically:

- Exceptions are thrown explicitly by algorithms in the following scenarious:
- input parameters are unexpected;
- provided SYCL device is not supported;
- spline is not fully initialized.
- Exceptions thrown by runtime libraries at the host CPU, including DPC++ synchronous exceptions, are passed through to the caller.
- DPC++ asynchronous errors are not handled.

16.2.1 Common Terms

Glossary

Assume we need to interpolate a function `f(x)` on the `[a, b)` interval using splines.

Let's break `[a, b)` into sub-intervals by n points x_i called partition points or simply **partition**. **Function values** at these points ($y_i = f(x_i), i = 1, \cdots, n$) are also given.

Spline has k degree if it can be expressed by the following polynomial:

$$P(x) = c_1 + c_2(x - x_i) + c_3(x - x_i)^2 + \cdots + c_{k-1}(x - x_i)^k.$$

Splines are constructed on $[x_i, x_{i+1}), i = 1, \cdots, n-1$ sub-intervals. So, for each sub-interval there are following polynomials:

$$P_i(x) = c_{1,i} + c_{2,i}(x - x_i) + c_{3,i}(x - x_i)^2 + \cdots + c_{k-1,i}(x - x_i)^k.$$

$c_{j,i}, j = 1, \cdots, k, i = 1, \cdots, n$ are called spline **coefficients**.

Function is interpolated at points of [a, b). Such points are called interpolation sites or simply **sites**. Sites might or migtn't equals to partition points.

Mathematical Notation in the Data Fitting Component

Concept	Mathematical Notation
Partition	$\{x_i\}_{i=1,\cdots,n}$, where $a = x1 < x2 < \cdots < xn = b$..
Uniform partition	Partition $\{x_i\}_{i=1,\cdots,n}$ which meets the following condition: $x_{i+1} - x_i = x_i - x_{i-1}, i = 2, \cdots, n-1$
Quasi-uniform partition	Partition $\{x_i\}_{i=1,\cdots,n}$ which meets the constraint with a constant C defined as: $1 \leq M/m \leq C$, where $M = max_{i=1,\cdots,n-1}(\Delta_i)$, $m = min_{i=1,\cdots,n-1}(\Delta_i)$, $\Delta_i = x_{i+1} - x_i$
Vector-valued function of dimension p being fit	$f(x) = (f_1(x), \cdots, f_p(x))$.
A k-order derivative of function f(x) at point t	$f^{(k)}(t)$.
Function p agrees with function f at the points $\{x_i\}_{i=1,\cdots,n}$.	For every point ζ in sequence $\{x_i\}_{i=1,\cdots,n}$ that occurs m times, the equality $p^{(i-1)}(\zeta) = f^{(i-1)}(\zeta)$ holds for all $i = 1, \cdots, m$.
The k-th divided difference of function f at points $x_i, ..., x_{i+k}$. This difference is the leading coefficient of the polynomial of order k+1 that agrees with f at $x_i, \cdots, x_{i+k}$.	$[x_i, \cdots, x_{i+k}]\, f$. In particular, $[x_1]\, f = f(x_1)$, $[x_1, x_2]\, f = (f(x_1) - f(x_2))/(x_1 - x_2)$.

Hints in the Data Fitting Component

The Intel® oneAPI Math Kernel Library (oneMKL) Data Fitting component provides ways to specify some "hints" for partitions, function values, coefficients, interpolation sites.

Partition Hints

The following are supported:

- Non-uniform.
- Quasi-uniform.
- Uniform.

Syntax

```
enum class partition_hint {
  non_uniform,
  quasi_uniform,
  uniform
};
```

Function Values Hints

Let $\{x_i\}_{i=1,\cdots,nx}$ is a partition, $\{f_j(x)\}_{j=1,\cdots,ny}$ is a vector-valued function. Function values are stored in the one-dimensional array with `nx * ny` elements. 2 different layouts are possible: row major and column major.

- For row major layout function values are stored as the following:

  Let $\{f_{j,i}\}_{j=1,\cdots,ny,i=1,\cdots,nx}$ is the function value that corresponds to the `i-th` partition point and the `j-th` function.

- For column major:

  Let $\{f_{i,j}\}_{i=1,\cdots,nx,j=1,\cdots,ny}$ is the function value that corresponds to the `i-th` partition point and the `j-th` function.

The following hints are supported:

- Row major.
- Column major.

Syntax

```
enum class function_hint {
  row_major,
  col_major
};
```

Coefficients Hints

Let $\{x_i\}_{i=1,\cdots,nx}$ is a partition, $\{f_j(x)\}_{j=1,\cdots,ny}$ is a vector-valued function. Let cubic spline should be constructed. It means that it requires 4 coefficients per each interpolation interval and function value. Cofficients are stored in the one-dimensional array with `4 * (nx - 1) * ny` elements.

- For row major:

  Let $\{c_{j,i,k}\}_{j=1,\cdots,ny,i=1,\cdots,nx-1,k=1,\cdots,4}$ is the coefficient value that corresponds to the `i-th` partition point, the `j-th` function.

- For column major:

  Let $\{c_{i,j,k}\}_{i=1,\cdots,nx-1,j=1,\cdots,ny,k=1,\cdots,4}$ is the coefficient value that corresponds to the `i-th` partition point, the `j-th` function.

The following is supported:

- row major

Syntax

```
enum class coefficient_hint {
  row_major
};
```

Sites Hints

The following are supported:

- Non-uniform.
- Uniform.
- Sorted.

Syntax

```
enum class site_hint {
  non_uniform,
  uniform,
  sorted
};
```

Interpolation Results Hints

Let $\{f_j(x)\}_{j=1,\cdots,ny}$ is a vector-valued function, $\{s_i\}_{i=1,\cdots,ns}$ are sites, `d` is a number of derivatives (including interpolation values) that needs to be calculated. So, size of memory to store interpolation results is `nsite * ny * d` elements.

6 different layouts are possible:

- functions-sites-derivatives

  Let $\{r_{j,i,k}\}_{j=1,\cdots,ny,i=1,\cdots,nsite,k=1,\cdots,d}$ is an interpolation result that corresponds to the `i-th` site, the `j-th` function, the `k-th` derivative.

- functions-derivatives-sites

  Let $\{r_{j,k,i}\}_{j=1,\cdots,ny,k=1,\cdots,d,i=1,\cdots,nsite}$ is an interpolation result that corresponds to the `i-th` site, the `j-th` function, the `k-th` derivative.

- sites-functions-derivatives

  Let $\{r_{i,j,k}\}_{i=1,\cdots,nsite,j=1,\cdots,ny,k=1,\cdots,d}$ is an interpolation result that corresponds to the `i-th` site, the `j-th` function, the `k-th` derivative.

- sites-derivatives-functions

  Let $\{r_{i,k,j}\}_{i=1,\cdots,nsite,k=1,\cdots,d,j=1,\cdots,ny}$ is an interpolation result that corresponds to the `i-th` site, the `j-th` function, the `k-th` derivative.

- derivatives-functions-sites

  Let $\{r_{k,j,i}\}_{k=1,\cdots,d,j=1,\cdots,ny,i=1,\cdots,nsite}$ is an interpolation result that corresponds to the `i-th` site, the `j-th` function, the `k-th` derivative.

- derivatives-sites-functions

  Let $\{r_{k,i,j}\}_{k=1,\cdots,d,i=1,\cdots,nsite,j=1,\cdots,ny}$ is an interpolation result that corresponds to the `i-th` site, the `j-th` function, the `k-th` derivative.

The following are supported:

- functions-sites-derivatives
- functions-derivatives-sites
- sites-functions-derivatives
- sites-derivatives-functions

Syntax

```
enum class interpolate_hint {
  funcs_sites_ders,
  funcs_ders_sites,
  sites_funcs_ders,
  sites_ders_funcs
};
```

Derivatives Hints

Following hints are added to choose which derivtive orders need to be computed during the `interpolate` function:

- just compute interpolation values
- compute first derivative of the spline polynomial only
- compute second derivative of the spline polynomial only
- compute third derivative of the spline polynomial only

Syntax

```
enum class derivatives {
  zero,
  first,
  second,
  third
};
```

`operator|` is overloaded to create combinations of derivative orders to be computed by `interpolate`.

Example

Assume that interpolation values, 1-st and 3-rd derivatives need to be computed. To create a bit mask that is passed to `interpolate` it needs following:

```
std::bitset<32> bit_mask = derivatives::zero | derivatives::first | derivatives::third;
```

Boundary Condition Types

Some type of splines requires boundary conditions to be set. The following types are supported:

- Free end ($f^{(2)}(x_1) = f^{(2)}(x_n) = 0$).
- Periodic.
- First derivative.
- Second Derivative.

Syntax

```
enum class bc_type {
  free_end,
  first_left_der,
  first_right_der,
  second_left_der,
  second_right_der,
  periodic
};
```

Note:

1. First derivative and second derivative types must be set on the left and on the right borders.
2. Free end doesn't require any values to be set.

16.2.2 Splines

Header File

```
#include<oneapi/mkl/experimental/data_fitting.hpp>
```

Namespace

```
oneapi::mkl::experimental::data_fitiing
```

Common API for All Spline Types

```
template <
  typename FpType,
  typename SplineType,
  int Dimensions = 1>
class spline {
public:
    using value_type = FpType;
    using spline_type = SplineType;

    spline(
      const sycl::queue& q,
      std::int64_t ny = 1,
      bool were_coeffs_computed = false);

    spline(
      const sycl::device& dev,
      const sycl::context& ctx,
      std::int64_t ny = 1,
      bool were_coeffs_computed = false);

    ~spline();

    spline(const spline<FpType, SplineType, Dimensions>& other) = delete;
    spline(spline<FpType, SplineType, Dimensions>&& other) = delete;
    spline<FpType, SplineType, Dimensions>& operator=(
        const spline<FpType, SplineType, Dimensions>& other) = delete;
    spline<FpType, SplineType, Dimensions>& operator=(
        spline<FpType, SplineType, Dimensions>&& other) = delete;

    spline& set_partitions(
      FpType* input_data,
      std::int64_t nx,
      partition_hint PartitionHint = partition_hint::non_uniform);

    spline& set_function_values(
      FpType* input_data,
      function_hint FunctionHint = storage_hint::row_major);

    spline& set_coefficients(
      FpType* data,
      coefficient_hint CoeffHint = storage_hint::row_major);

    bool is_initialized() const;

    std::int64_t get_required_coeffs_size() const;
```

(continues on next page)

(continued from previous page)

```
    sycl::event construct(const std::vector<sycl::event>& dependencies = {});
};
```

An instance of `spline<T, ST, N>` create the `N`-dimensional spline that operates with the `T` data type. `ST` is a type of spline. `T` can only be `float` or `double`.

Constructor	Description
`spline(const sycl::queue& q, std::int64_t ny = 1, bool were_coeffs_computed = false);`	Create an object with the `q` SYCL queue and `ny` number of functions. If spline coeficients were already computed, provide `true` as a 3-rd parameter. `were_coeffs_computed == false` by default. It means that it needs to call `construct` to compute spline coefficients.
`spline(const sycl::device& dev, const sycl::context& ctx, std::int64_t ny = 1, bool were_coeffs_computed = false);`	Create an object using the `dev` SYCL device, the `ctx` context and `ny` number of functions. If spline coeficients were already computed, provide `true` as a 3-rd parameter. `were_coeffs_computed == false` by default. It means that it needs to call `construct` to compute spline coefficients.

Member function	Description
`spline& set_partitions(FpType* input_data, std::int64_t nx, partition_hint PartitionHint = partition_hint::non_uniform);`	Set partition values that are specified by the `input_data` memory pointer and `nx` partition values. Users can provide `PartitionHint` to specify the layout of data. Default layout is `non_uniform`. If `uniform` is specified, `nx` must equals to 2 and `input_data` must contain only 2 values the left and the right borders of partition. Otherwise, behavior is undefined. If `input_data` layout doesn't satisfy `PartitionHint`, behavior is undefined. Returns a reference to the spline object for which partitions are set. Example, for `uniform`. Let $\{i\}_{i=1,\cdots,n}$ is a partition. So, `input_data` must contain only 2 values: 1, n.
`spline& set_function_values(FpType* input_data, function_hint FunctionHint = storage_hint::row_major);`	Set function values that are specified by the `input_data` memory pointer. Number of function values must equals to `ny * nx` elements. Users can provide `FunctionHint` to specify the layout of data. Default layout is `row_major`. If `input_data` layout doesn't satisfy `FunctionHint`, behavior is undefined. Returns a reference to the spline object for which function values are set.
`spline& set_coefficients(FpType* data, coefficient_hint CoeffHint = storage_hint::row_major);`	Set coefficients that are specified by the `data` memory pointer. Number of coefficients in the memory must equals to the return value of `get_required_coeffs_size()`. Users can provide `CoeffHint` to specify the layout of data. Default layout is `row_major`. If `data` layout doesn't satisfy `CoeffHint`, behavior is undefined. If `were_coeffs_computed == false`, `data` will be rewritten during the `construct` call. Returns a reference to the spline object for which coefficients are set.
`bool is_initialized() const;`	Returns `true` if all required data are set (for example, partitions, function values, coefficients).
`std::int64_t get_required_coeffs_size() const;`	Returns amount of memory that is required for coefficients storage.
`sycl::event construct(const std::vector<sycl::event>& dependencies = {});`	Constructs the spline (calculates spline coefficients if `were_coeffs_computed == false`). The function submits a SYCL kernel and returns the SYCL event to wait on to ensure computation is complete. `dependencies` is a list of SYCL events to wait for before starting computations.

There are some splines that requires internal conditions and boundary conditions to be set. For such spline types, the following member functions must be called.

```
spline& set_internal_conditions(
  FpType* input_data);
```

(continues on next page)

(continued from previous page)

```
spline& set_boundary_conditions(
  bc_type BCType = bc_type::free_end,
  FpType input_value = {});
```

Member function	Description
`spline& set_internal_conditions(FpType* input_data);`	Set internal conditions that are specified by the `input_data` memory pointer. Number of internal condition values must equals to `nx - 2`. Returns a reference to the spline object for which internal conditions are set.
`spline& set_boundary_conditions(bc_type BCType = bc_type::free_end, FpType input_value = {});`	Set the `input_value` boundary condition corresponding to `BCType`. Default value for `input_value` is empty since some boundary conditions doesn't require value to be provided. Returns a reference to the spline object for which boundary condition value is set.

Note: copy/move constructor and copy/move assignment operators are deleted since the `spline` class is just a wrapper over memory that users provide. Memory management responsibility is on user's side.

Supported Spline Types

Currently, the DPC++ Data Fitting APIs support the following spline types (follow the corresponding links to deep-dive into the details):

Linear Spline

Cubic Splines

Linear Spline

- Header File
- Namespace
- Syntax
- Example

Linear spline is a spline whose degree is equal to 1.

It's described by the following polynomial

$$P_i(x) = c_{1,i} + c_{2,i}(x - x_i),$$

where

$$x \in [x_i, x_{i+1}),$$

$$c_{1,i} = f(x_i),$$

$$c_{2,i} = [x_i, x_{i+1}]\, f,$$

$$i = 1, \cdots, n-1.$$

Header File

```
#include<oneapi/mkl/experimental/data_fitting.hpp>
```

Namespace

```
oneapi::mkl::experimental::data_fitiing
```

Syntax

```
namespace linear_spline {
  struct default_type {};
}
```

Example

To create a linear spline object use the following:

```
spline<float, linear_spline::default_type> val(
  /*SYCL queue object*/q,
  /*number of spline functions*/ny
);
```

Follow the Examples section to see more complicated examples.

Cubic Splines

Cubic splines are splines whose degree is equal to 3.

Cubic splines are described by the following polynomial

$$P_i(x) = c_{1,i} + c_{2,i}(x - x_i) + c_{3,i}(x - x_i)^2 + c_{4,i}(x - x_i)^3,$$

where

$$x \in [x_i, x_{i+1}),$$

$$i = 1, \cdots, n-1.$$

There are a lot of different types of cubic splines: Hermite, natural, Akima, Bessel. However, the current version of DPC++ API supports only one type: Hermite.

Header File

```
#include<oneapi/mkl/experimental/data_fitting.hpp>
```

Namespace

```
oneapi::mkl::experimental::data_fitiing
```

Hermite Spline

Coefficients of Hermite spline are calculated using the following formulas:

$$c_{1,i} = f(x_i),$$

$$c_{2,i} = s_i,$$

$$c_{3,i} = ([x_i, x_{i+1}]\, f - s_i) / (\Delta x_i) - c_{4,i}(\Delta x_i),$$

$$c_{4,i} = (s_i + s_{i+1} - 2[x_i, x_{i+1}]\, f) / (\Delta x_i)^2,$$

$$s_i = f^{(1)}(x_i).$$

The following boundary conditions are supported for Hermite spline:

- Free end ($f^{(2)}(x_1) = f^{(2)}(x_n) = 0$).
- Periodic.
- First derivative.
- Second Derivative.

Syntax

```
namespace cubic_spline {
  struct hermite {};
}
```

Example

To create a cubic Hermite spline object use the following:

```
spline<float, cubic_spline::hermite> val(
  /*SYCL queue object*/q,
  /*number of spline functions*/ny
);
```

Follow the Examples section to see more complicated examples.

16.2.3 Interpolate Function

- Header File
- Namespace
- Syntax

Interpolate function performs computations of function and derivatives values at interpolation sites.

If the sites do not belong to interpolation interval `[a, b]`, the library uses:

- interpolant I_0 coefficients computed for interval $[x_0, x_1)$ for the computations at the sites to the left of a.
- interpolant I_{n-2} coefficients computed for interval $[x_{n-2}, x_{n-1})$ for the computations at the sites to the right of b.

Interpolation algorithm depends on interpolant's type (e.g., for cubic spline interpoilation evaluation of third-order polynomial is performed to obtain function values).

Header File

```
#include<oneapi/mkl/experimental/data_fitting.hpp>
```

Namespace

```
oneapi::mkl::experimental::data_fitiing
```

Syntax

```
template <typename Interpolant>
sycl::event interpolate(
    Interpolant& interpolant,
    typename Interpolant::fp_type* sites,
    std::int64_t n_sites,
    typename Interpolant::fp_type* results,
    const std::vector<sycl::event>& dependencies,
    interpolate_hint ResultHint = interpolate_hint::funcs_sites_ders,
    site_hint SiteHint = site_hint::non_uniform); // (1)

template <typename Interpolant>
sycl::event interpolate(
    Interpolant& interpolant,
    typename Interpolant::fp_type* sites,
    std::int64_t n_sites,
    typename Interpolant::fp_type* results,
    std::bitset<32> der_indicator,
    const std::vector<sycl::event>& dependencies = {},
    interpolate_hint ResultHint = interpolate_hint::funcs_sites_ders,
    site_hint SiteHint = site_hint::non_uniform); // (2)

template <typename Interpolant>
sycl::event interpolate(
    sycl::queue& q,
    const Interpolant& interpolant,
    typename Interpolant::fp_type* sites,
    std::int64_t n_sites,
    typename Interpolant::fp_type* results,
    const std::vector<sycl::event>& dependencies,
    interpolate_hint ResultHint = interpolate_hint::funcs_sites_ders,
    site_hint SiteHint = site_hint::non_uniform); // (3)

template <typename Interpolant>
sycl::event interpolate(
    sycl::queue& q,
    const Interpolant& interpolant,
    typename Interpolant::fp_type* sites,
    std::int64_t n_sites,
    typename Interpolant::fp_type* results,
    std::bitset<32> der_indicator,
    const std::vector<sycl::event>& dependencies = {},
    interpolate_hint ResultHint = interpolate_hint::funcs_sites_ders,
    site_hint SiteHint = site_hint::non_uniform); // (4)
```

For all functions users can provide `SiteHint` and `ResultHint` to specify the layout of `sites` and `results` respectively. If `results` layout doesn't satisfy `ResultHint` and/or `sites` layout doesn't satisfy `SiteHint`, behavior is undefined. Returns the SYCL event of the submitted task.

1. Performs computations of function values only using the SYCL queue associated with `interpolant`.
2. Performs computations of certain derivatives (function values is considered as a zero derivative) which are indicated in `der_indicator` (each bit corresponds to certain derivative starting from lower bit) using the SYCL queue associated with `interpolant`.
3. Performs computations of function values only using `q` as an input argument that should be created from the same context and device as the SYCL queue associated with `interpolant`.
4. Performs computations of certain derivatives (function values is considered as a zero derivative) which are indicated in `der_indicator` (each bit corresponds to certain derivative starting from lower bit) using `q` as an input argument that should be created from the same context and device as the SYCL queue associated with `interpolant`.

Follow the **Examples** section to see examples of the interpolation function usage.

16.2.4 Examples

The following example demonstrates how to construct the linear spline and perform the interpolation.

```
#include <cstdint>
#include <iostream>
#include <vector>

#include <CL/sycl.hpp>

#include <oneapi/mkl/experimental/data_fitting.hpp>

constexpr std::int64_t nx = 10'000;
constexpr std::int64_t nsites = 150'000;

int main (int argc, char ** argv) {

    sycl::queue q;
    sycl::usm_allocator<double, sycl::usm::alloc::shared> alloc(q);

    // Allocate memory for spline parameters
    std::vector<double, decltype(alloc)> partitions(nx, alloc);
    std::vector<double, decltype(alloc)> functions(nx, alloc);
    std::vector<double, decltype(alloc)> coeffs(2 * (nx - 1), alloc);
    std::vector<double, decltype(alloc)> sites(nsites, alloc);
    std::vector<double, decltype(alloc)> results(nsites, alloc);

    // Fill parameters with valid data
    for (std::int64_t i = 0; i < nx; ++i) {
        partitions[i] = 0.1 * i;
        functions[i] = i * i;
    }

    for (std::int64_t i = 0; i < nsites; ++i) {
        sites[i] = (0.1 * nx * i) / nsites);
    }

```

(continues on next page)

(continued from previous page)

```
    namespace df = oneapi::mkl::experimental::data_fitting;
    // Set parameters to spline
    df::spline<double, df::linear_spline::default_type> spl(q);
    spl.set_partitions(partitions.data(), nx)
       .set_coefficients(coeffs.data())
       .set_function_values(functions.data());

    // Construct spline
    auto event = spl.construct();
    event = df::interpolate(spl, sites.data(), nsites, results.data(), { event });
    event.wait();

    std::cout << "done" << std::endl;
    return 0;
}
```

17.0 Bibliography

For more information about the VS functionality, refer to the following publications:

17.1 VS RNG

[AVX] Intel. **Intel® Advanced Vector Extensions Programming Reference**. (https://software.intel.com/content/www/us/en/develop/home.html)

[Bratley88] Bratley P. and Fox B.L. **Implementing Sobol's Quasirandom Sequence Generator**, ACM Transactions on Mathematical Software, Vol. 14, No. 1, Pages 88-100, March 1988.

[Bratley92] Bratley P., Fox B.L., and Niederreiter H. **Implementation and Tests of Low-Discrepancy Sequences**, ACM Transactions on Modeling and Computer Simulation, Vol. 2, No. 3, Pages 195-213, July 1992.

[Coddington94] Coddington, P. D. **Analysis of Random Number Generators Using Monte Carlo Simulation**. Int. J. Mod. Phys. C-5, 547, 1994.

[IntelSWMan] Intel. **Intel® 64 and IA-32 Architectures Software Developer's Manual**. 3 vols. (https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html)

[L'Ecuyer99] L'Ecuyer, Pierre. **Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure**. Mathematics of Computation, 68, 225, 249-260, 1999.

[L'Ecuyer99a] L'Ecuyer, Pierre. **Good Parameter Sets for Combined Multiple Recursive Random Number Generators**. Operations Research, 47, 1, 159-164, 1999.

[Kirkpatrick81] Kirkpatrick, S., and Stoll, E. **A Very Fast Shift-Register Sequence Random Number Generator**. Journal of Computational Physics, V. 40. 517-526, 1981.

[Matsumoto98] Matsumoto, M., and Nishimura, T. **Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator**, ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 1, Pages 3-30, January 1998.

[Matsumoto00] Matsumoto, M., and Nishimura, T. **Dynamic Creation of Pseudorandom Number Generators**, 56-69, in: Monte Carlo and Quasi-Monte Carlo Methods 1998, Ed. Niederreiter, H. and Spanier, J., Springer 2000, http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/DC/dc.html.

[NAG] NAG Numerical Libraries. https://www.nag.com/content/nag-library

[Saito08] Saito, M., and Matsumoto, M. **SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator**. Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer, Pages 607 – 622, 2008.

http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/ARTICLES/earticles.html

[Salmon11] Salmon, John K., Morales, Mark A., Dror, Ron O., and Shaw, David E., **Parallel Random Numbers: As Easy as 1, 2, 3**. SC '11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, 2011.

[Sobol76] Sobol, I.M., and Levitan, Yu.L. **The production of points uniformly distributed in a multidimensional cube**. Preprint 40, Institute of Applied Mathematics, USSR Academy of Sciences, 1976 (In Russian).

[VS Notes] **|O-MKL| Vector Statistics Notes**, a document presentation on the oneMKL product at https://software.intel.com/content/www/us/en/develop/articles/intel-math-kernel-library-documentation.html

17.2 VM

[IEEE754] IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std 754-2008.

18.0 Appendix A: oneMKL Functionality

This appendix provides an overview of the Intel® oneAPI Math Kernel Library (oneMKL) functionality on the different devices.

18.1 BLAS Functionality

Functionality	CPU	Intel GPU
BLAS Level 1	All	All
BLAS Level 2	All	All
BLAS Level 3	All	All
BLAS-like Extensions	All	All

18.2 LAPACK Functionality

Note: All of the DPC++ LAPACK computational routines have a corresponding *_scratchpad_size function for calculating the required amount of scratchpad space.

Functionality	CPU	Intel GPU
`getrf`	Yes	Yes
`getrs`	Yes	Yes
`getri`	Yes	Yes

Functionality	CPU	Intel GPU
`potrf`	Yes	Yes
`potrs`	Yes	Yes
`potri`	Yes	Yes

Functionality	CPU	Intel GPU
`geqrf`	Yes	Yes
`{or,un}gqr`	Yes	Yes
`{or,un}mqr`	Yes	Yes
`gerqf`	Yes	No
`{or,un}mrq`	Yes	No

Functionality	CPU	Intel GPU
trtrs	Yes	Yes
{sy,he}trf	Yes	No

Functionality	CPU	Intel GPU
{sy,he}ev	Yes	Yes
{sy,he}evd	Yes	Yes
{sy,he}evx	Yes	Yes
{sy,he}trd	Yes	Yes
{or,un}gtr	Yes	No
{or,un}mtr	Yes	No
steqr	Yes	Yes

Functionality	CPU	Intel GPU
{sy,he}gvd	Yes	Yes
{sy,he}gvx	Yes	Yes

Functionality	CPU	Intel GPU
gesvd	Yes	Yes
gebrd	Yes	Yes
{or,un}gbr	Yes	No

Functionality	CPU	Intel GPU
getrf_batch	Yes	Yes
getrs_batch	Yes	Yes
getri_batch	Yes	Yes
potrf_batch	Yes	Yes
potrs_batch	Yes	Yes
geqrf_batch	Yes	Yes
{or,un}gqr_batch	Yes	Yes

18.2.1 Other LAPACK Routines

CPU	Intel GPU
No	No

18.3 DFT Functionality

Functionality	CPU	Intel GPU
1D Complex-to-Complex FFT transformations	Yes	Yes
2D Complex-to-Complex FFT transformations	Yes	Yes
3D Complex-to-Complex FFT transformations	Yes	Yes
4D Complex-to-Complex FFT transformations	No	No
5D Complex-to-Complex FFT transformations	No	No
6D Complex-to-Complex FFT transformations	No	No
7D Complex-to-Complex FFT transformations	No	No
1D Real-to-Complex FFT transformations	Yes	Yes
2D Real-to-Complex FFT transformations	Yes	Yes
3D Real-to-Complex FFT transformations	Yes	Yes
4D Real-to-Complex FFT transformations	No	No
5D Real-to-Complex FFT transformations	No	No
6D Real-to-Complex FFT transformations	No	No
7D Real-to-Complex FFT transformations	No	No

18.4 Sparse BLAS Functionality

In the following table for functionality, sm = sparse matrix, dm = dense matrix, sv = sparse vector, dv = dense vector, sc = scalar.

In the following table for operations, dense vectors = x, y, sparse vectors = w,v, dense matrices = X,Y, sparse matrices = A, B, C, and scalars = alpha, beta, d.

18.4.1 Level 1

Functionality	Operations	CPU	Intel GPU
Sparse Vector - Dense Vector addition (AXPY)	y <- alpha * w + y	No	No
Sparse Vector - Sparse Vector Dot product (SPDOT) (sv.sv -> sc)	d <- dot(w,v)	N/A	N/A
	dot(w,v) = sum(w_i* v_i)	No	No
	dot(w,v) = sum(conj(w_i) * v_i)	No	No
Sparse Vector - Dense Vector Dot product (SPDOT) (sv.dv -> sc)	d <- dot(w,x)	N/A	N/A
	dot(w,v) = sum(w_i* v_i)	No	No
	dot(w,v) = sum(conj(w_i) * v_i)	No	No
Dense Vector - Sparse Vector Conversion (sv <-> dv)	—	N/A	N/A
	x = scatter(w)	No	No
	w = gather(x,windx)	No	No

In the following table for functionality, sm = sparse matrix, dm = dense matrix, sv = sparse vector, dv = dense vector, sc = scalar.

In the following table for operations, dense vectors = x, y, sparse vectors = w,v, dense matrices = X,Y, sparse matrices = A, B, C, and scalars = alpha, beta, d.

18.4.2 Level 2

Functionality	Operations	CPU	Intel GPU
General Matrix-Vector multiplication (GEMV) (sm*dv->dv)	y <- beta*y + alpha * op(A)*x	N/A	N/A
	op(A) = A	Yes	Yes
	op(A) = A^T	Yes	Yes
	op(A) = A^H	No	No
Symmetric Matrix-Vector multiplication (SYMV) (sm*dv->dv)	y <- beta*y + alpha * op(A)*x	N/A	N/A
	op(A) = A	Yes	Yes
	op(A) = A^T	Yes	Yes
	op(A) = A^H	No	No
Triangular Matrix-Vector multiplication (TRMV) (sm*dv->dv)	y <- beta*y + alpha * op(A)*x	N/A	N/A
	op(A) = A	Yes	No
	op(A) = A^T	Yes	No
	op(A) = A^H	No	No
General Matrix-Vector mult with dot product (GEMV-DOT) (sm*dv -> dv, dv.dv->sc)	y <- beta*y + alpha * op(A)*x, d = dot(x,y)	N/A	N/A
	op(A) = A	Yes	Yes
	op(A) = A^T	Yes	Yes
	op(A) = A^H	No	No
Triangular Solve (TRSV) (inv(sm)*dv -> dv)	solve for y, op(A)*y = alpha*x	N/A	N/A
	op(A) = A	Yes	Yes
	op(A) = A^T	Yes	Yes
	op(A) = A^H	No	No

In the following table for functionality, sm = sparse matrix, dm = dense matrix, sv = sparse vector, dv = dense vector, sc = scalar.

In the following table for operations, dense vectors = x, y, sparse vectors = w,v, dense matrices = X,Y, sparse matrices = A, B, C, and scalars = alpha, beta, d.

18.4.3 Level 3

Functionality	Operations	CPU	Intel GPU
General Sparse Matrix - Dense Matrix Multiplication (GEMM) (sm*dm->dm)	Y <- alpha*op(A)*op(X) + beta*Y	N/A	N/A
	op(A) = A, op(X) = X	Yes	Yes
	op(A) = A^T, op(X) = X	Yes	Yes

continues on next page

Table 32 – continued from previous page

Functionality	Operations	CPU	Intel GPU
	op(A) = A^H, op(X) = X	Yes	Yes
	op(A) = A, op(X) = X^T	No	No
	op(A) = A^T, op(X) = X^T	No	No
	op(A) = A, op(X) = X^H	No	No
	op(A) = A^H	No	No
	op(A) = A^T, op(X) = X^H	No	No
	op(A) = A^H, op(X) = X^H	No	No
General Dense Matrix - Sparse Matrix Multiplication (GEMM) (dm*sm->dm)	Y <- alpha*op(X)*op(A) + beta*Y	N/A	N/A
	op(X) = X, op(A)=A	No	No
	op(X) = X^H, op(A)=A	No	No
	op(X) = X^H, op(A)=A	No	No
	op(X) = X, op(A)=A^H	No	No
	op(X) = X^H, op(A)=A^H	No	No
	op(X) = X^H, op(A)=A^H	No	No
	op(X) = X, op(A)=A^H	No	No
	op(X) = X^H, op(A)=A^H	No	No
	op(X) = X^H, op(A)=A^H	No	No
General Sparse Matrix - Sparse Matrix Multiplication (GEMM) (sm*sm->sm)	C <- alpha*op(A)*op(B) + beta*C	N/A	N/A
	op(A)=A, op(B)=B	No	No
	op(A)=A^T, op(B)=B	No	No
	op(A)=A^H, op(B)=B	No	No
	op(A)=A, op(B)=B^T	No	No
	op(A)=A^T, op(B)=B^T	No	No
	op(A)=A^H, op(B)=B^T	No	No
	op(A)=A, op(B)=B^H	No	No
	op(A)=A^T, op(B)=B^H	No	No
	op(A)=A^H, op(B)=B^H	No	No
General Sparse Matrix - Sparse Matrix Multiplication (GEMM) (sm*sm->dm)	Y <- alpha*op(A)*op(B) + beta*Y	N/A	N/A
	op(A)=A, op(B)=B	No	No
	op(A)=A^T, op(B)=B	No	No
	op(A)=A^H, op(B)=B	No	No
	op(A)=A, op(B)=B^T	No	No
	op(A)=A^T, op(B)=B^T	No	No
	op(A)=A^H, op(B)=B^T	No	No
	op(A)=A, op(B)=B^H	No	No
	op(A)=A^T, op(B)=B^H	No	No
	op(A)=A^H, op(B)=B^H	No	No
Symmetric Rank-K update (SYRK) (sm*sm->sm)	C <- op(A)*op(A)^H	N/A	N/A
	op(A)=A	No	No

continues on next page

Table 32 – continued from previous page

Functionality	Operations	CPU	Intel GPU
	op(A)=A^T	No	No
	op(A)=A^H	No	No
Symmetric Rank-K update (SYRK) (sm*sm->dm)	Y <- op(A)*op(A)^H	N/A	N/A
	op(A)=A	No	No
	op(A)=A^T	No	No
	op(A)=A^H	No	No
Symmetric Triple Product (SYPR) (op(sm)*sm*sm -> sm)	C <- op(A)*B*op(A)^H	N/A	N/A
	op(A)=A	No	No
	op(A)=A^T	No	No
	op(A)=A^H	No	No
Triangular Solve (TRSM) (inv(sm)*dm -> dm)	solve for Y, op(A)*Y = alpha*X	N/A	N/A
	op(A)=A	No	No
	op(A)=A^T	No	No
	op(A)=A^H	No	No

In the following table for functionality, sm = sparse matrix, dm = dense matrix, sv = sparse vector, dv = dense vector, sc = scalar.

In the following table for operations, dense vectors = x, y, sparse vectors = w,v, dense matrices = X,Y, sparse matrices = A, B, C, and scalars = alpha, beta, d.

18.4.4 Other

Functionality	Operations	CPU	Intel GPU
Symmetric Gauss-Seidel Preconditioner (SYMGS) (update A*x=b, A=L+D+U)	x0 <- x*alpha; (L+D)*x1=b-U*x0; (U+D)*x=b-L*x1	No	No
Symmetric Gauss-Seidel Preconditioner with Matrix-Vector product (SYMGS_MV) (update A*x=b, A=L+D+U)	x0 <- x*alpha; (L+D)*x1=b-U*x0; (U+D)*x=b-L*x1; y=A*x	No	No
LU Smoother (LU_SMOOTHER) (update A*x=b, A=L+D+U, E~inv(D))	r=b-A*x; (L+D)*E*(U+D)*dx=r; y=x+dr	No	No
Sparse Matrix Add (ADD)	C <- alpha*op(A) + B	No	No
	op(A) = A^T	No	No
	op(A) = A^H	No	No

In the following table for operations, dense vectors = x, y, sparse vectors = w,v, dense matrices = X,Y, sparse matrices = A, B, C, and scalars = alpha, beta, d.

18.4.5 Helper Functions

Functionality	Operations	CPU	Intel GPU
Sort Indices of Matrix (ORDER)	N/A	No	No
Transpose of Sparse Matrix (TRANSPOSE)	A <- op(A) with op=trans or conjtrans	N/A	N/A
	transpose CSR/CSC matrix	No	No
	transpose BSR matrix	No	No
Sparse Matrix Format Converter (CONVERT)	N/A	No	No
Dense to Sparse Matrix Format Converter (CONVERT)	N/A	No	No
Copy Matrix Handle (COPY)	N/A	No	No
Create CSR Matrix Handle	N/A	Yes	Yes
Create CSC Matrix Handle	N/A	No	No
Create COO Matrix Handle	N/A	No	No
Create BSR Matrix Handle	N/A	No	No
Export CSR Matrix	Allows access to internal data in the CSR Matrix handle	No	No
Export CSC Matrix	Allows access to internal data in the CSC Matrix handle	No	No
Export COO Matrix	Allows access to internal data in the COO Matrix handle	No	No
Export BSR Matrix	Allows access to internal data in the BSR Matrix handle	No	No
Set Value in Matrix	N/A	No	N

In the following table for functionality, sm = sparse matrix, dm = dense matrix, sv = sparse vector, dv = dense vector, sc = scalar.

In the following table for operations, dense vectors = x, y, sparse vectors = w,v, dense matrices = X,Y, sparse matrices = A, B, C, and scalars = alpha, beta, d.

18.4.6 Optimize Stages

Functionality	Operations	CPU	Intel GPU
add MEMORY hint and optimize	Chooses to allow larger memory requiring optimizations or not.	No	No
Add GEMV hint and optimize	N/A	Yes	No
Add SYMV hint and optimize	N/A	Yes	No
Add TRMV hint and optimize	N/A	Yes	No
add TRSV hint and optimize	N/A	Yes	No
add GEMM hint and optimize	N/A	Yes	No
add TRSM hint and optimize	N/A	No	No
add DOTMV hint and optimize	N/A	Yes	No
add SYMGS hint and optimize	N/A	No	No
add SYMGS_MV hint and optimize	N/A	No	No
add LU_SMOOTHER hint and optimize	N/A	No	No

18.5 Sparse Solvers Functionality

Functionality	CPU	Intel GPU
Sparse Cholesky Factorization	No	No
Sparse LU Factorization	No	No
Sparse QR factorization	No	No
Hermitian/Symmetric Eigensolver on intervals for Sparse Matrices	No	No
Extremal Eigensolvers for Sparse Matrix	No	No
Poisson Solver	No	No
Trust Region Solver	No	No

18.6 Graphs Functionality

Functionality	CPU	Intel GPU
mxv	No	No
vxm	No	No
mxm	No	No
transpose	No	No

Functionality	CPU	Intel GPU
matrix_create	No	No
matrix_destroy	No	No
vector_create	No	No
vector_destroy	No	No
descriptor_create	No	No
descriptor_destroy	No	No

Functionality	CPU	Intel GPU
matrix_set_csr	No	No
matrix_get_csr	No	No
matrix_set_csc	No	No
matrix_get_csc	No	No
vector_set_dense	No	No
vector_get_dense	No	No
vector_set_sparse	No	No
vector_get_sparse	No	No

Functionality	CPU	Intel GPU
optimize_mxv	No	No
optimize_mxm	No	No
matrix_get_property	No	No
vector_get_property	No	No

Functionality	CPU	Intel GPU
descriptor_set_field	No	No

18.7 Random Number Generators Functionality

18.7.1 Engines

Functionality	CPU	Intel GPU
MRG32K3A	Yes	Yes
MT2203	Yes	Yes
MT19937	Yes	Yes
PHILOX4X32X10	Yes	Yes
SOBOL	Yes	Yes
ARS5	Yes	No
MCG59	Yes	Yes
NIEDERR	Yes	No
MCG31	Yes	Yes
WH	Yes	No
SFMT19937	Yes	No
R250	Yes	No
NONDETERM	Yes	No
DABSTRACT	No	No
SABSTRACT	No	No
SABSTRACT	No	No

18.7.2 Distributions

Functionality	CPU	Intel GPU
Uniform (single/double/integer)	Yes	Yes
UniformBits32 UniformBits64	Yes	Yes
Lognormal (single/double)	Yes	Yes
Gaussian (single/double)	Yes	Yes
Poisson	Yes	Yes
UniformBits	Yes	Yes
Bernoulli	Yes	Yes
Beta (single/double)	Yes	No
Binomial	Yes	No
ChiSquare (single/double)	Yes	No
Exponential (single/double)	Yes	Yes
Gamma (single/double)	Yes	No
Geometric	Yes	Yes
Gumbel (single/double)	Yes	Yes
Hyper Geometric	Yes	No
Laplace (single/double)	Yes	Yes
Multinomial	Yes	No
Negative Binomial	Yes	No
PoissonV	Yes	No
Rayleigh (single/double)	Yes	Yes
Weibull (single/double)	Yes	Yes
Cauchy (single/double)	Yes	Yes
GaussianMV (single/double)	Yes	No

18.8 Vector Math Functionality

Functionality	CPU	Intel GPU
Vector Math Functions, Single Precision	Yes	Yes
Vector Math Functions, Double Precision	Yes	Yes
Vector Math Functions, Single Precision Complex	Yes	No
Vector Math Functions, Double Precision Complex	Yes	No

OpenMP* offload to the GPU is implemented in the Linux* OS, but not in the Windows* OS. The Windows OS implementation will be available in a future release.

18.9 Data Fitting Functionality

18.9.1 Splines, Spline Type

Functionality	CPU	Intel GPU
default linear	Yes (since oneMKL 2022.1)	Yes (since oneMKL 2022.1)
default quadratic	No	No
default cubic	No	No
subbotin	No	No
natural	No	No
hermite	Yes (since oneMKL 2022.1)	Yes (since oneMKL 2022.1)
akima	No	No
bessel	No	No
hyman	No	No
lookup interpolant	No	No
cr stepwise const interpolant	No	No
cl stepwise const interpolant	No	No

18.9.2 Computation Routines

Functionality	CPU	Intel GPU
Construct1D	Yes (since oneMKL 2022.1)	Yes (since oneMKL 2022.1)
Interpolate1D	Yes (since oneMKL 2022.1)	Yes (since oneMKL 2022.1)
Interpolate1DEx	No	No
Integrate1D	No	No
Integrate1DEx	No	No
SearchCells1D	No	No
SearchCells1DEx	No	No
InterpolationCallBack	No	No
IntegrateCallBack	No	No
SearchCellsCallBack	No	No

18.10 Summary Statistics Functionality

Functionality	CPU	Intel GPU
min	Yes	No
max	Yes	No
raw sum 2nd order raw sum 3rd order raw sum 4th order raw sum	Yes	No
2nd order central sum 3rd order central sum 4th order central sum	Yes	No
mean 2nd order raw moment 3rd order raw moment 4th order raw moment	Yes	No
2nd order central moment 3rd order central moment 4th order central moment	Yes	No
kurtosis	Yes	No
skewness	Yes	No
variation coefficient	Yes	No
covariance matrix	No	No
correlation matrix	No	No
cross-product matrix	No	No
pooled covariance matrix	No	No
pooled mean	No	No
group covariance matrix	No	No
group mean	No	No
quantiles	No	No
order statistics	No	No
robust covariance matrix	No	No
ouliers detection	No	No
partial covariance matrix	No	No
partial covariance matrix	No	No
missing values	No	No
parameterized correlation matrix	No	No
stream quantiles	No	No
mean absolute deviation	No	No
median absolute deviation	No	No
sorted observations	No	No

19.0 Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit https://www.intel.com/content/www/us/en/benchmarks/benchmark.html.

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

19.1 Third Party Content

Intel® oneAPI Math Kernel Library includes content from several 3rd party sources that was originally governed by the licenses referenced below:

- Portions© Copyright 2001 Hewlett-Packard Development Company, L.P.

- Sections on the Linear Algebra PACKage (LAPACK) routines include derivative work portions that have been copyrighted:
  - © 1991, 1992, and 1998 by The Numerical Algorithms Group, Ltd.
- Intel® oneAPI Math Kernel Library supports LAPACK 3.5 set of computational, driver, auxiliary and utility routines under the following license:

  Copyright © 1992-2011 The University of Tennessee and The University of Tennessee Research Foundation. All rights reserved.

  Copyright © 2000-2011 The University of California Berkeley. All rights reserved.

  Copyright © 2006-2012 The University of Colorado Denver. All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
  - Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

  The original versions of LAPACK from which that part of Intel® oneAPI Math Kernel Library was derived can be obtained from http://www.netlib.org/lapack/index.html. The authors of LAPACK are E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen.
- The original versions of the Basic Linear Algebra Subprograms (BLAS) from which the respective part of Intel® oneAPI Math Kernel Library was derived can be obtained from http://www.netlib.org/blas/index.html.

- XBLAS is distributed under the following copyright:

  Copyright © 2008-2009 The University of California Berkeley.

  All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
  - Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- The original versions of the Basic Linear Algebra Communication Subprograms (BLACS) from which the respective part of Intel® oneAPI Math Kernel Library was derived can be obtained from http://www.netlib.org/blacs/index.html. The authors of BLACS are Jack Dongarra and R. Clint Whaley.
- The original versions of Scalable LAPACK (ScaLAPACK) from which the respective part of Intel® oneAPI Math Kernel Library was derived can be obtained from http://www.netlib.org/scalapack/index.html. The authors of ScaLAPACK are L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley.
- The original versions of the Parallel Basic Linear Algebra Subprograms (PBLAS) routines from which the respective part of Intel® oneAPI Math Kernel Library was derived can be obtained from http://www.netlib.org/scalapack/html/pblas_qref.html.
- PARDISO (PARallel DIrect SOlver)* in Intel® oneAPI Math Kernel Library was originally developed by the Department of Computer Science at the University of Basel (http://www.unibas.ch). It can be obtained at http://www.pardiso-project.org.
- The Extended Eigensolver functionality is based on the Feast solver package and is distributed under the following license:

  Copyright © 2009, The Regents of the University of Massachusetts, Amherst.

  Developed by E. Polizzi

- Some Fast Fourier Transform (FFT) functions in this release of Intel® oneAPI Math Kernel Library have been generated by the SPIRAL software generation system (http://www.spiral.net/) under license from Carnegie Mellon University. The authors of SPIRAL are Markus Puschel, Jose Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo.
- Open MPI is distributed under the New BSD license, listed below.

  Most files in this release are marked with the copyrights of the organizations who have edited them. The copyrights below are in no particular order and generally reflect members of the Open MPI core team who have contributed code to this release. The copyrights for code used under license from other parties are included in the corresponding files.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- The Safe C Library is distributed under the following copyright:

  Copyright (c)

  Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

- HPL Copyright Notice and Licensing Terms

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution.
  3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed at the University of Tennessee, Knoxville, Innovative Computing Laboratories.
  4. The name of the University, the name of the Laboratory, or the names of its contributors may not be used to endorse or promote products derived from this software without specific written permission.