

Intel® oneAPI Collective Communications Library Developer Guide and Reference

Contents

Chapter 1: Intel® oneAPI Collective Communications Library	
Release Notes	4
Installation Guide	4
Sample Application	5
Use oneCCL package from CMake	7
Programming Model	8
Host Communication	8
Device Communication.....	9
Limitations	11
General Configuration	11
Execution of Communication Operations	11
Transport Selection.....	12
Advanced Configuration.....	12
Selection of Collective Algorithms	12
Low-precision Datatypes	12
Caching of Communication Operations	13
Prioritization of Communication Operations.....	13
Fusion of Communication Operations	13
Enabling OFI/verbs/dmabuf Support	14
oneCCL API	14
Initialization	15
oneCCL Concepts	15
Communicator.....	15
Context	17
Device.....	17
Event	17
Key-value Store	18
Stream	18
Communication Operations.....	18
Datatypes	19
Collective Operations.....	19
Point-To-Point Operations	33
Environment Variables	36
oneCCL Benchmark User Guide	50
Notices and Disclaimers.....	54

Intel® oneAPI Collective Communications Library



Intel® oneAPI Collective Communications Library (oneCCL) provides an efficient implementation of communication patterns used in deep learning.

oneCCL features include:

- Built on top of lower-level communication middleware – [Intel® MPI Library](#) and [libfabric](#).
- Optimized to drive scalability of communication patterns by allowing to easily trade off compute for communication performance.
- Works across various interconnects: InfiniBand*, Cornelis Networks*, and Ethernet.
- Provides common API sufficient to support communication workflows within Deep Learning / distributed frameworks (such as [PyTorch*](#), [Horovod*](#)).

oneCCL package comprises the oneCCL Software Development Kit (SDK) and the Intel® MPI Library Runtime components.

Get Started

- [Release Notes](#)
- [Installation Guide](#)
 - [System Requirements](#)
 - [Installation using Command Line Interface](#)
 - [Find More](#)
- [Sample Application](#)
 - [Build details](#)
 - [Run the sample](#)
- [Use oneCCL package from CMake](#)
 - [oneCCLConfig files generation](#)

Developer Guide

- [Programming Model](#)
 - [Host Communication](#)
 - [Device Communication](#)
 - [Limitations](#)
- [General Configuration](#)
 - [Execution of Communication Operations](#)
 - [Transport Selection](#)
- [Advanced Configuration](#)
 - [Selection of Collective Algorithms](#)
 - [Low-precision Datatypes](#)
 - [Caching of Communication Operations](#)
 - [Prioritization of Communication Operations](#)
 - [Fusion of Communication Operations](#)
 - [Enabling OFI/verbs/dmabuf Support](#)

Developer Reference

- [oneCCL API](#)
 - [Initialization](#)
 - [oneCCL Concepts](#)
 - [Communication Operations](#)
 - [Generic workflow](#)
 - [Error Handling](#)

- [Environment Variables](#)
 - [Collective Algorithms Selection](#)
 - [Workers](#)
 - [ATL](#)
 - [Multi-NIC](#)
 - [Low-precision datatypes](#)
 - [CCL_LOG_LEVEL](#)
 - [CCL_ITT_LEVEL](#)
 - [Fusion](#)
 - [CCL_PRIORITY](#)
 - [CCL_MAX_SHORT_SIZE](#)
 - [CCL_SYCL_OUTPUT_EVENT](#)
 - [CCL_ZE_LIBRARY_PATH](#)
 - [CCL_RECV](#)
 - [CCL_SEND](#)

Benchmark User Guide

- [oneCCL Benchmark User Guide](#)
 - [Build oneCCL Benchmark](#)
 - [Run oneCCL Benchmark](#)
 - [Example](#)

Release Notes

Refer to [Intel® oneAPI Collective Communications Library Release Notes](#).

Installation Guide

This page explains how to install and configure the Intel® oneAPI Collective Communications Library (oneCCL). oneCCL supports different installation scenarios using command line interface.

System Requirements

Visit [Intel® oneAPI Collective Communications Library System Requirements](#) to learn about hardware and software requirements for oneCCL.

Installation using Command Line Interface

To install oneCCL using command line interface (CLI), follow these steps:

1. Go to the `ccl` folder:

```
cd ccl
```

2. Create a new folder:

```
mkdir build
```

3. Go to the folder created:

```
cd build
```

4. Launch CMake:

```
cmake ..
```

5. Install the product:

```
make -j install
```

In order to have a clear build, create a new `build` directory and invoke `cmake` within the directory.

Custom Installation

You can customize CLI-based installation (for example, specify directory, compiler, and build type):

- To specify **installation directory**, modify the `cmake` command:

```
cmake .. -DCMAKE_INSTALL_PREFIX=</path/to/installation/directory>
```

If no `-DCMAKE_INSTALL_PREFIX` is specified, oneCCL is installed into the `_install` subdirectory of the current build directory. For example, `ccl/build/_install`.

- To specify **compiler**, modify the `cmake` command:

```
cmake .. -DCMAKE_C_COMPILER=<c_compiler> -DCMAKE_CXX_COMPILER=<cxx_compiler>
```

- To enable SYCL devices communication support, specify SYCL compiler (only Intel® oneAPI DPC++/C++ Compiler is supported):

```
cmake .. -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx -DCOMPUTE_BACKEND=dpcpp
```

- To specify the **build type**, modify the `cmake` command:

```
cmake .. -DCMAKE_BUILD_TYPE=[Debug|Release]
```

- To enable `make` verbose output to see all parameters used by `make` during compilation and linkage, modify the `make` command as follows:

```
make -j VERBOSE=1 install
```

Find More

- [oneCCL Get Started Guide](#)
- [oneCCL GitHub Source Code Repository](#)
- [oneCCL Documentation](#)

Sample Application

The sample code below shows how to use oneCCL API to perform allreduce communication for SYCL USM memory.

```
#include <iostream>
#include <mpi.h>
#include "oneapi/ccl.hpp"

void mpi_finalize() {
    int is_finalized = 0;
    MPI_Finalized(&is_finalized);

    if (!is_finalized) {
        MPI_Finalize();
    }
}

int main(int argc, char* argv[]) {
    constexpr size_t count = 10 * 1024 * 1024;

    int size = 0;
    int rank = 0;

    ccl::init();

    MPI_Init(nullptr, nullptr);
```

```

MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

atexit(mpi_finalize);

sycl::default_selector device_selector;
sycl::queue q(device_selector);
std::cout << "Running on " << q.get_device().get_info<sycl::info::device::name>() << "\n";

/* create kvs */
ccl::shared_ptr_class<ccl::kvs> kvs;
ccl::kvs::address_type main_addr;
if (rank == 0) {
    kvs = ccl::create_main_kvs();
    main_addr = kvs->get_address();
    MPI_Bcast((void*)main_addr.data(), main_addr.size(), MPI_BYTE, 0, MPI_COMM_WORLD);
}
else {
    MPI_Bcast((void*)main_addr.data(), main_addr.size(), MPI_BYTE, 0, MPI_COMM_WORLD);
    kvs = ccl::create_kvs(main_addr);
}

/* create communicator */
auto dev = ccl::create_device(q.get_device());
auto ctx = ccl::create_context(q.get_context());
auto comm = ccl::create_communicator(size, rank, dev, ctx, kvs);

/* create stream */
auto stream = ccl::create_stream(q);

/* create buffers */
auto send_buf = sycl::malloc_device<int>(count, q);
auto recv_buf = sycl::malloc_device<int>(count, q);

/* open buffers and modify them on the device side */
auto e = q.submit([&](auto& h) {
    h.parallel_for(count, [=](auto id) {
        send_buf[id] = rank + id + 1;
        recv_buf[id] = -1;
    });
});

int check_sum = 0;
for (int i = 1; i <= size; ++i) {
    check_sum += i;
}

/* do not wait completion of kernel and provide it as dependency for operation */
std::vector<ccl::event> deps;
deps.push_back(ccl::create_event(e));

/* invoke allreduce */
auto attr = ccl::create_operation_attr<ccl::allreduce_attr>();
ccl::allreduce(send_buf, recv_buf, count, ccl::reduction::sum, comm, stream, attr,
deps).wait();

/* open recv_buf and check its correctness on the device side */
sycl::buffer<int> check_buf(count);

```

```
q.submit([&](auto& h) {
    sycl::accessor check_buf_acc(check_buf, h, sycl::write_only);
    h.parallel_for(count, [=](auto id) {
        if (recv_buf[id] != static_cast<int>(check_sum + size * id)) {
            check_buf_acc[id] = -1;
        }
    });
});

q.wait_and_throw();

/* print out the result of the test on the host side */
{
    sycl::host_accessor check_buf_acc(check_buf, sycl::read_only);
    size_t i;
    for (i = 0; i < count; i++) {
        if (check_buf_acc[i] == -1) {
            std::cout << "FAILED\n";
            break;
        }
    }
    if (i == count) {
        std::cout << "PASSED\n";
    }
}

sycl::free(send_buf, q);
sycl::free(recv_buf, q);
}
```

Build details

1. Build oneCCL with SYCL support (only Intel® oneAPI DPC++/C++ Compiler is supported).
2. Set up the library environment.
3. Use the C++ driver with the `-fsycl` option to build the sample:

```
icpx -o sample sample.cpp -lccl -lmpi
```

Run the sample

Intel® MPI Library is required for running the sample. Make sure that MPI environment is set up.

To run the sample, use the following command:

```
mpiexec <parameters> ./sample
```

where `<parameters>` represents optional `mpiexec` parameters such as node count, processes per node, hosts, and so on.

NOTE Explore the complete list of oneAPI code samples in the [oneAPI Samples Catalog](#). These samples were designed to help you develop, offload, and optimize multiarchitecture applications targeting CPUs, GPUs, and FPGAs.

Use oneCCL package from CMake

`oneCCLConfig.cmake` and `oneCCLConfigVersion.cmake` are included into oneCCL distribution.

With these files, you can integrate oneCCL into a user project with the `find_package` command. Successful invocation of `find_package(oneCCL <options>)` creates imported target `oneCCL` that can be passed to the `target_link_libraries` command.

For example:

```
project(Foo)
add_executable(foo foo.cpp)

# Search for oneCCL
find_package(oneCCL REQUIRED)

# Connect oneCCL to foo
target_link_libraries(foo oneCCL)
```

oneCCLConfig files generation

To generate oneCCLConfig files for oneCCL package, use the provided `cmake/scripts/config_generation.cmake` file:

```
cmake [-DOUTPUT_DIR=<output_dir>] -P cmake/script/config_generation.cmake
```

Programming Model

- [Host Communication](#)
- [Device Communication](#)
- [Limitations](#)

NOTE Check out [oneCCL specification](#) that oneCCL is based on.

Host Communication

The communication operations between processes are provided by [Communicator](#).

The example below demonstrates the main concepts of communication on host memory buffers.

Example

Consider a simple oneCCL `allreduce` example for CPU.

1. Create a communicator object with user-supplied size, rank, and key-value store:

```
auto ccl_context = ccl::create_context();
auto ccl_device = ccl::create_device();

auto comms = ccl::create_communicators(
    size,
    vector_class<pair_class<size_t, device>>{ { rank, ccl_device } },
    ccl_context,
    kvs);
```

Or for convenience use non-vector form without device and context parameters.

```
auto comm = ccl::create_communicator(size, rank, kvs);
```

2. Initialize `send_buf` (in real scenario it is supplied by the user):

```
const size_t elem_count = <N>;

/* initialize send_buf */
```



```
for (idx = 0; idx < elem_count; idx++) {
    send_buf[idx] = rank + 1;
}
```

3. `allreduce` invocation performs the reduction of values from all the processes and then distributes the result to all the processes. In this case, the result is an array with `elem_count` elements, where all elements are equal to the sum of arithmetical progression:

$$p \cdot (p + 1) / 2$$

```
ccl::allreduce(send_buf,
              recv_buf,
              elem_count,
              reduction::sum,
              comm).wait();
```

4. Check the correctness of `allreduce` operation:

```
auto comm_size = comm.size();
auto expected = comm_size * (comm_size + 1) / 2;

for (idx = 0; idx < elem_count; idx++) {
    if (recv_buf[idx] != expected) {
        std::cout << "unexpected value at index " << idx << std::endl;
        break;
    }
}
```

Device Communication

The communication operations between devices are provided by [Communicator](#).

The example below demonstrates the main concepts of communication on device memory buffers.

Example

Consider a simple oneCCL `allreduce` example for GPU:

1. Create oneCCL communicator objects with user-supplied size, rank <-> SYCL device mapping, SYCL context and key-value store:

```
auto ccl_context = ccl::create_context(sycl_context);
auto ccl_device = ccl::create_device(sycl_device);

auto comms = ccl::create_communicators(
    size,
    vector_class<pair_class<size_t, device>>{ { rank, ccl_device } },
    ccl_context,
    kvs);
```

2. Create oneCCL stream object from user-supplied `sycl::queue` object:

```
auto stream = ccl::create_stream(sycl_queue);
```

3. Initialize `send_buf` (in real scenario it is supplied by the user):

```
const size_t elem_count = <N>;

/* using SYCL buffer and accessor */
```

```
auto send_buf_host_acc = send_buf.get_host_access(h, sycl::write_only);
for (idx = 0; idx < elem_count; idx++) {
    send_buf_host_acc[idx] = rank;
}
```

```
/* or using SYCL USM */
for (idx = 0; idx < elem_count; idx++) {
    send_buf[idx] = rank;
}
```

4. For demonstration purposes, modify the `send_buf` on the GPU side:

```
/* using SYCL buffer and accessor */
sycl_queue.submit([&](cl::sycl::handler& h) {
    auto send_buf_dev_acc = send_buf.get_access<mode::write>(h);
    h.parallel_for(range<1>{elem_count}, [=](item<1> idx) {
        send_buf_dev_acc[idx] += 1;
    });
});
```

```
/* or using SYCL USM */
for (idx = 0; idx < elem_count; idx++) {
    send_buf[idx] += 1;
}
```

5. `allreduce` invocation performs reduction of values from all processes and then distributes the result to all processes. In this case, the result is an array with `elem_count` elements, where all elements are equal to the sum of arithmetical progression:

$$p \cdot (p + 1) / 2$$

```
std::vector<event> events;
for (auto& comm : comms) {
    events.push_back(ccl::allreduce(send_buf,
                                   rcv_buf,
                                   elem_count,
                                   reduction::sum,
                                   comm,
                                   streams[comm.rank()]));
}

for (auto& e : events) {
    e.wait();
}
```

6. Check the correctness of `allreduce` operation on the GPU:

```
/* using SYCL buffer and accessor */

auto comm_size = comm.size();
auto expected = comm_size * (comm_size + 1) / 2;

sycl_queue.submit([&](handler& h) {
    auto rcv_buf_dev_acc = rcv_buf.get_access<mode::write>(h);
    h.parallel_for(range<1>{elem_count}, [=](item<1> idx) {
        if (rcv_buf_dev_acc[idx] != expected) {
            rcv_buf_dev_acc[idx] = -1;
        }
    });
});
```

```
    }
  });
});
...

auto recv_buf_host_acc = recv_buf.get_host_access(sycl::read_only);
for (idx = 0; idx < elem_count; idx++) {
  if (recv_buf_host_acc[idx] == -1) {
    std::cout << "unexpected value at index " << idx << std::endl;
    break;
  }
}

/* or using SYCL USM */

auto comm_size = comm.size();
auto expected = comm_size * (comm_size + 1) / 2;

for (idx = 0; idx < elem_count; idx++) {
  if (recv_buf[idx] != expected) {
    std::cout << "unexpected value at index " << idx << std::endl;
    break;
  }
}
```

Limitations

The list of scenarios not yet supported by oneCCL:

- Creation of multiple ranks within single process

General Configuration

- [Execution of Communication Operations](#)
- [Transport Selection](#)

Execution of Communication Operations

Communication operations are executed by CCL worker threads (workers). The number of workers is controlled by the [CCL_WORKER_COUNT](#) environment variable.

Workers affinity is controlled by [CCL_WORKER_AFFINITY](#).

By setting workers affinity you can specify which CPU cores are used by CCL workers. The general rule of thumb is to use different CPU cores for compute (e.g. by specifying [KMP_AFFINITY](#)) and for CCL communication.

There are two ways to set workers affinity: automatic and explicit.

Automatic setup

To set affinity automatically, set [CCL_WORKER_AFFINITY](#) to `auto`.

Example

In the example below, oneCCL creates four workers per process and pins them to the last four cores available for the process (available if `mpirun` launcher from oneCCL package is used, the exact IDs of CPU cores depend on the parameters passed to `mpirun`) or to the last four cores on the node.

```
export CCL_WORKER_COUNT=4
export CCL_WORKER_AFFINITY=auto
```

Explicit setup

To set affinity explicitly for all local workers, pass ID of the cores to the `CCL_WORKER_AFFINITY` environment variable.

Example

In the example below, oneCCL creates 4 workers per process and pins them to cores with numbers 3, 4, 5, and 6, respectively:

```
export CCL_WORKER_COUNT=4
export CCL_WORKER_AFFINITY=3,4,5,6
```

Transport Selection

oneCCL supports two transports for inter-process communication: [Intel® MPI Library](#) and [libfabric*](#).

The transport selection is controlled by `CCL_ATL_TRANSPORT`.

In case of MPI over libfabric implementation (for example, Intel® MPI Library 2021) or in case of direct libfabric transport, the selection of specific libfabric provider is controlled by the `FI_PROVIDER` environment variable.

Advanced Configuration

- [Selection of Collective Algorithms](#)
- [Low-precision Datatypes](#)
- [Caching of Communication Operations](#)
- [Prioritization of Communication Operations](#)
- [Fusion of Communication Operations](#)
- [Enabling OFI/verbs/dmabuf Support](#)

Selection of Collective Algorithms

oneCCL supports manual selection of collective algorithms for different message size ranges.

Refer to [Collective Algorithms Selection](#) section for details.

Low-precision Datatypes

oneCCL provides support for collective operations on low-precision (LP) datatypes ([bfloat16](#) and [float16](#)).

Reduction of LP buffers (for example as phase in `ccl::allreduce`) includes conversion from LP to FP32 format, reduction of FP32 values and conversion from FP32 to LP format.

oneCCL utilizes CPU vector instructions for FP32 <-> LP conversion.

For BF16 <-> FP32 conversion oneCCL provides `AVX512F` and `AVX512_BF16`-based implementations. `AVX512F`-based implementation requires GCC 4.9 or higher. `AVX512_BF16`-based implementation requires GCC 10.0 or higher and GNU binutils 2.33 or higher. `AVX512_BF16`-based implementation may provide less accuracy loss after multiple up-down conversions.

For FP16 <-> FP32 conversion oneCCL provides `F16C` and `AVX512F`-based implementations. Both implementations require GCC 4.9 or higher.

Refer to [Low-precision datatypes](#) for details about relevant environment variables.

Caching of Communication Operations

Communication operations may have expensive initialization phase (for example, allocation of internal structures and buffers, registration of memory buffers, handshake with peers, and so on). oneCCL amortizes these overheads by caching operation internal representations and reusing them on the subsequent calls.

To control this, use operation attribute and set `true` value for `to_cache` field and unique string (for example, tensor name) for `match_id` field.

Note that:

- `match_id` should be the same for a specific communication operation across all ranks.
- If the same tensor is a part of different communication operations, `match_id` should have different values for each of these operations.

Prioritization of Communication Operations

oneCCL supports prioritization of communication operations that controls the order in which individual communication operations are executed. This allows to postpone execution of non-urgent operations to complete urgent operations earlier, which may be beneficial for many use cases.

The communication prioritization is controlled by priority value. Note that the priority must be a non-negative number with a higher number standing for a higher priority.

There are the following prioritization modes:

- None - default mode when all communication operations have the same priority.
- Direct - you explicitly specify priority using `priority` field in operation attribute.
- LIFO (Last In, First Out) - priority is implicitly increased on each operation call. In this case, you do not have to specify priority.

The prioritization mode is controlled by `CCL_PRIORITY`.

Fusion of Communication Operations

In some cases, it may be beneficial to postpone execution of communication operations and execute them all together as a single operation in a batch mode. This can reduce operation setup overhead and improve interconnect saturation.

oneCCL provides several knobs to enable and control such optimization:

- The fusion is enabled by `CCL_FUSION`.
- The advanced configuration is controlled by:
 - `CCL_FUSION_BYTES_THRESHOLD`
 - `CCL_FUSION_COUNT_THRESHOLD`
 - `CCL_FUSION_CYCLE_MS`

NOTE For now, this functionality is supported for `allreduce` operations only.

Enabling OFI/verbs/dmabuf Support

oneCCL provides experimental support for data transfers between Intel GPU memory and NIC using Linux dmabuf, which is exposed through OFI API for verbs provider.

Requirements

- Linux kernel version ≥ 5.12
- RDMA core version ≥ 34.0
- level-zero-devel package

Usage

oneCCL, OFI and OFI/verbs from Intel® oneAPI Base Toolkit support device memory transfers. Refer to [Run instructions](#) for usage.

If you want to build software components from sources, refer to [Build instructions](#).

Build instructions

OFI

```
git clone --single-branch --branch v1.13.2 https://github.com/ofiwg/libfabric.git
cd libfabric
./autogen.sh
./configure --prefix=<ofi_install_dir> --enable-verbs=<rdma_core_install_dir> --with-ze=<level_zero_install_dir> --enable-ze-dlopen=yes
make -j install
```

NOTE You may also get OFI release package directly from [here](#). No need to run autogen.sh if using the release package.

oneCCL

```
cmake -DCMAKE_INSTALL_PREFIX=<ccl_install_dir> -DLIBFABRIC_DIR=<ofi_install_dir> -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx -DCOMPUTE_BACKEND=dpcpp -DENABLE_OFI_HMEM=1 ..
make -j install
```

Run instructions

1. Set the environment. See [Get Started Guide](#).
2. Run allreduce test with ring algorithm and SYCL USM device buffers:

```
export CCL_ATL_TRANSPORT=ofi
export CCL_ATL_HMEM=1
export CCL_ALLREDUCE=ring
export FI_PROVIDER=verbs
mpirun -n 2 <ccl_install_dir>/examples/sycl/sycl_allreduce_usm_test gpu device
```

oneCCL API

- [Initialization](#)
- [oneCCL Concepts](#)

- Communicator
- Context
- Device
- Event
- Key-value Store
- Stream
- Communication Operations
 - Datatypes
 - Collective Operations
 - Point-To-Point Operations

Generic workflow

Refer to oneCCL specification for more details about [generic workflow](#) with oneCCL API.

Error Handling

Refer to oneCCL specification for more details about [error handling](#).

Initialization

template<class... attr_val_type> init_attr CCL_API create_init_attr (attr_val_type &&... avs)

Creates an attribute object that may be used to control the init operation.

Returns an attribute object

void CCL_API init (const init_attr &attr=default_init_attr)

Initializes the library. Optional for invocation.

Parameters **attr** – optional init attributes

library_version CCL_API get_library_version ()

Retrieves the library version.

oneCCL Concepts

Refer to oneCCL specification for more details about oneCCL **main concepts**.

- Communicator
- Context
- Device
- Event
- Key-value Store
- Stream

Communicator

template<class... attr_val_type> comm_attr CCL_API create_comm_attr (attr_val_type &&... avs)

Creates an attribute object that may be used to control the create_communicator operation.

Returns an attribute object

template<class... attr_val_type> comm_split_attr CCL_API create_comm_split_attr (attr_val_type &&... avs)

Creates an attribute object that may be used to control the split_communicator operation.

Returns an attribute object

template<class DeviceType, class ContextType> vector_class< communicator > CCL_API create_communicators (int size, const vector_class< pair_class< int, DeviceType >> &devices, const ContextType &context, shared_ptr_class< kvs_interface > kvs, const comm_attr &attr=default_comm_attr)

Creates new communicators with user supplied size, ranks, local device-rank mapping and kvs.

Parameters

- **size** – user-supplied total number of ranks
- **rank** – user-supplied rank
- **device** – local device
- **devices** – user-supplied mapping of local ranks on devices
- **context** – context containing the devices
- **kvs** – key-value store for ranks wire-up
- **attr** – optional communicator attributes

Returns vector of communicators / communicator

template<class DeviceType, class ContextType> vector_class< communicator > CCL_API create_communicators (int size, const map_class< int, DeviceType > &devices, const ContextType &context, shared_ptr_class< kvs_interface > kvs, const comm_attr &attr=default_comm_attr)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

template<class DeviceType, class ContextType> communicator CCL_API create_communicator (int size, int rank, DeviceType &device, const ContextType &context, shared_ptr_class< kvs_interface > kvs, const comm_attr &attr=default_comm_attr)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

communicator CCL_API create_communicator (int size, int rank, shared_ptr_class< kvs_interface > kvs, const comm_attr &attr=default_comm_attr)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

template<class DeviceType, class ContextType> vector_class< communicator > CCL_API create_communicators (int size, const vector_class< DeviceType > &devices, const ContextType &context, shared_ptr_class< kvs_interface > kvs, const comm_attr &attr=default_comm_attr)

Creates a new communicators with user supplied size, local devices and kvs. Ranks will be assigned automatically.

Parameters

- **size** – user-supplied total number of ranks
- **devices** – user-supplied device objects for local ranks
- **context** – context containing the devices
- **kvs** – key-value store for ranks wire-up
- **attr** – optional communicator attributes

Returns vector of communicators / communicator

communicator CCL_API create_communicator (int size, shared_ptr_class< kvs_interface > kvs, const comm_attr &attr=default_comm_attr)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

communicator CCL_API create_communicator (const comm_attr &attr=default_comm_attr)

Creates a new communicator with externally provided size, rank and kvs. Implementation is platform specific and non portable.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **attr** – optional communicator attributes
Returns communicator

vector_class< communicator > CCL_API split_communicators (const vector_class< pair_class< communicator, comm_split_attr >> &attrs)

Splits communicators according to attributes.

Parameters **attrs** – split attributes for local communicators
Returns vector of communicators

Context

template<class native_context_type, class = typename std::enable_if<is_context_supported<native_context_type>()>::type> context CCL_API create_context (native_context_type &&native_context)

Creates a new context from @native_context_type.

Parameters **native_context** – the existing handle of context
Returns context object

context CCL_API create_context ()

Device

template<class native_device_type, class = typename std::enable_if<is_device_supported<native_device_type>()>::type> device CCL_API create_device (native_device_type &&native_device)

Creates a new device from @native_device_type.

Parameters **native_device** – the existing handle of device
Returns device object

device CCL_API create_device ()

Event

template<class event_type, class = typename std::enable_if<is_event_supported<event_type>()>::type> event CCL_API create_event (event_type &native_event)

Creates a new event from @native_event_type.

Parameters **native_event** – the existing event
 Returns event object

Key-value Store

template<class... attr_val_type> kvs_attr CCL_API create_kvs_attr (attr_val_type &&... avs)

shared_ptr_class< kvs > CCL_API create_main_kvs (const kvs_attr &attr=default_kvs_attr)

Creates a main key-value store. Its address should be distributed using out of band communication mechanism and be used to create key-value stores on other processes.

Parameters **attr** – optional kvs attributes
 Returns kvs object

shared_ptr_class< kvs > CCL_API create_kvs (const kvs::address_type &addr, const kvs_attr &attr=default_kvs_attr)

Creates a new key-value store from main kvs address.

Parameters • **addr** – address of main kvs
 • **attr** – optional kvs attributes
 Returns kvs object

Stream

template<class native_stream_type, class = typename std::enable_if<is_stream_supported<native_stream_type>()>::type> stream CCL_API create_stream (native_stream_type &native_stream)

Creates a new stream from @native_stream_type.

Parameters **native_stream** – the existing handle of stream
 Returns stream object

stream CCL_API create_stream ()

Communication Operations

Refer to oneCCL specification for more details about **communication operations**.

- [Datatypes](#)
- [Collective Operations](#)
 - [Allgatherv](#)
 - [Allreduce](#)
 - [Alltoall](#)
 - [Alltoallv](#)
 - [Barrier](#)
 - [Broadcast](#)
 - [Reduce](#)
 - [ReduceScatter](#)
 - [Operation Attributes](#)
- [Point-To-Point Operations](#)

- [send](#)
- [recv](#)

Datatypes

template<class... attr_val_type> datatype_attr CCL_API create_datatype_attr (attr_val_type &&... avs)

Creates an attribute object that may be used to register custom datatype.

Returns an attribute object

datatype CCL_API register_datatype (const datatype_attr &attr)

Registers custom datatype to be used in communication operations.

Parameters **attr** – datatype attributes

Returns datatype handle

void CCL_API deregister_datatype (datatype dtype)

Deregisters custom datatype.

Parameters **dtype** – custom datatype handle

size_t CCL_API get_datatype_size (datatype dtype)

Retrieves a datatype size in bytes.

Parameters **dtype** – datatype handle

Returns datatype size

Collective Operations

- [Allgatherv](#)
- [Allreduce](#)
- [Alltoall](#)
- [Alltoallv](#)
- [Barrier](#)
- [Broadcast](#)
- [Reduce](#)
- [ReduceScatter](#)

Operation Attributes

template<class coll_attribute_type, class... attr_val_type> coll_attribute_type CCL_API create_operation_attr (attr_val_type &&... avs)

Creates an attribute object that may be used to customize communication operation.

Returns an attribute object

Allgatherv

event CCL_API allgatherv (const void *send_buf, size_t send_count, void *recv_buf, const vector_class< size_t > &recv_counts, datatype dtype, const communicator &comm, const stream &stream, const allgatherv_attr &attr=default_allgatherv_attr, const vector_class< event > &deps={})

Allgatherv is a collective communication operation that collects data from all the ranks within a communicator into a single buffer. Different ranks may contribute segments of different sizes. The resulting data in the output buffer is the same for each rank.

Parameters

- **send_buf** – the buffer with `send_count` elements of `dtype` that stores local data to be gathered
- **send_count** – the number of elements of type `dtype` in `send_buf`
- **recv_buf** – [out] the buffer to store gathered result of `dtype`, must be large enough to hold values from all ranks, i.e. size should be equal to `dtype` size in bytes * sum of all values in `recv_counts`
- **recv_counts** – array with the number of elements of type `dtype` to be received from each rank
- **dtype** – the datatype of elements in `send_buf` and `recv_buf`
- **comm** – the communicator for which the operation will be performed
- **stream** – abstraction over a device queue constructed via `ccl::create_stream`
- **attr** – optional attributes to customize operation
- **deps** – an optional vector of the events that the operation should depend on

Returns

`ccl::event` an object to track the progress of the operation

event CCL_API allgatherv (const void *send_buf, size_t send_count, void *recv_buf, const vector_class< size_t > &recv_counts, datatype dtype, const communicator &comm, const allgatherv_attr &attr=default_allgatherv_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

event CCL_API allgatherv (const void *send_buf, size_t send_count, const vector_class< void * > &recv_bufs, const vector_class< size_t > &recv_counts, datatype dtype, const communicator &comm, const stream &stream, const allgatherv_attr &attr=default_allgatherv_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overloaded function takes separate receive buffer per rank.

Parameters

recv_bufs – [out] array of buffers to store gathered result, one buffer per rank; each buffer must be large enough to keep the corresponding `recv_counts` elements of `dtype` size

event CCL_API allgatherv (const void *send_buf, size_t send_count, const vector_class< void * > &recv_bufs, const vector_class< size_t > &recv_counts, datatype dtype, const communicator &comm, const allgatherv_attr &attr=default_allgatherv_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overloaded function takes separate receive buffer per rank.

Parameters

recv_bufs – [out] array of buffers to store gathered result, one buffer per rank; each buffer must be large enough to keep the corresponding `recv_counts` elements of `dtype` size

```
template<class BufferType, class = typename
std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API
allgatherv (const BufferType *send_buf, size_t send_count, BufferType *recv_buf, const
vector_class< size_t > &recv_counts, const communicator &comm, const stream &stream, const
allgatherv_attr &attr=default_allgatherv_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_buf** – the buffer with `send_count` elements of `BufferType` that stores local data to be gathered
- **recv_buf** – [out] the buffer to store gathered result of `BufferType`, must be large enough to hold values from all ranks, i.e. size should be equal to `BufferType` size in bytes * sum of all values in `recv_counts`

```
template<class BufferType, class = typename
std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API
allgatherv (const BufferType *send_buf, size_t send_count, BufferType *recv_buf, const
vector_class< size_t > &recv_counts, const communicator &comm, const allgatherv_attr
&attr=default_allgatherv_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_buf** – the buffer with `send_count` elements of `BufferType` that stores local data to be gathered
- **recv_buf** – [out] the buffer to store gathered result of `BufferType`, must be large enough to hold values from all ranks, i.e. size should be equal to `BufferType` size in bytes * sum of all values in `recv_counts`

```
template<class BufferType, class = typename
std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API
allgatherv (const BufferType *send_buf, size_t send_count, vector_class< BufferType * >
&recv_bufs, const vector_class< size_t > &recv_counts, const communicator &comm, const
stream &stream, const allgatherv_attr &attr=default_allgatherv_attr, const vector_class< event
> &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_buf** – the buffer with `send_count` elements of `BufferType` that stores local data to be gathered
- **recv_bufs** – [out] array of buffers to store gathered result, one buffer per rank; each buffer must be large enough to keep the corresponding `recv_counts` elements of `BufferType` size

```
template<class BufferType, class = typename
std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API
allgatherv (const BufferType *send_buf, size_t send_count, vector_class< BufferType * >
&recv_bufs, const vector_class< size_t > &recv_counts, const communicator &comm, const
allgatherv_attr &attr=default_allgatherv_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_buf** – the buffer with `send_count` elements of `BufferType` that stores local data to be gathered
- **recv_bufs** – [out] array of buffers to store gathered result, one buffer per rank; each buffer must be large enough to keep the corresponding `recv_counts` elements of `BufferType` size

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API  
allgatherv (const BufferObjectType &send_buf, size_t send_count, BufferObjectType &recv_buf,  
const vector_class< size_t > &recv_counts, const communicator &comm, const stream &stream,  
const allgatherv_attr &attr=default_allgatherv_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_buf** – the buffer of `BufferObjectType` with `send_count` elements that stores local data to be gathered
- **recv_buf** – [out] the buffer of `BufferObjectType` to store gathered result, must be large enough to hold values from all ranks, i.e. size should be equal to `BufferType` size in bytes * sum of all values in `recv_counts`

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API  
allgatherv (const BufferObjectType &send_buf, size_t send_count, BufferObjectType &recv_buf,  
const vector_class< size_t > &recv_counts, const communicator &comm, const allgatherv_attr  
&attr=default_allgatherv_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_buf** – the buffer of `BufferObjectType` with `send_count` elements that stores local data to be gathered
- **recv_buf** – [out] the buffer of `BufferObjectType` to store gathered result, must be large enough to hold values from all ranks, i.e. size should be equal to `BufferType` size in bytes * sum of all values in `recv_counts`

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API  
allgatherv (const BufferObjectType &send_buf, size_t send_count, vector_class<  
reference_wrapper_class< BufferObjectType >> &recv_bufs, const vector_class< size_t >  
&recv_counts, const communicator &comm, const stream &stream, const allgatherv_attr  
&attr=default_allgatherv_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_buf** – the buffer of `BufferObjectType` with `send_count` elements that stores local data to be gathered
- **recv_bufs** – [out] array of buffers to store gathered result, one buffer per rank; each buffer must be large enough to keep the corresponding `recv_counts` elements of `BufferObjectType` size

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API  
allgatherv (const BufferObjectType &send_buf, size_t send_count, vector_class<
```

reference_wrapper_class< BufferObjectType >> &recv_bufs, const vector_class< size_t > &recv_counts, const communicator &comm, const allgather_attr &attr=default_allgather_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_buf** – the buffer of `BufferObjectType` with `send_count` elements that stores local data to be gathered
- **recv_bufs** – [out] array of buffers to store gathered result, one buffer per rank; each buffer must be large enough to keep the corresponding `recv_counts` elements of `BufferObjectType` size

Allreduce

event CCL_API allreduce (const void *send_buf, void *recv_buf, size_t count, datatype dtype, reduction rtype, const communicator &comm, const stream &stream, const allreduce_attr &attr=default_allreduce_attr, const vector_class< event > &deps={})

Allreduce is a collective communication operation that performs the global reduction operation on values from all ranks of communicator and distributes the result back to all ranks.

Parameters

- **send_buf** – the buffer with `count` elements of `dtype` that stores local data to be reduced
- **recv_buf** – [out] the buffer to store reduced result, must have the same dimension as `send_buf`
- **count** – the number of elements of type `dtype` in `send_buf` and `recv_buf`
- **dtype** – the datatype of elements in `send_buf` and `recv_buf``
- **rtype** – the type of the reduction operation to be applied
- **comm** – the communicator for which the operation will be performed
- **stream** – abstraction over a device queue constructed via `ccl::create_stream`
- **attr** – optional attributes to customize operation
- **deps** – an optional vector of the events that the operation should depend on

Returns

`ccl::event` an object to track the progress of the operation

event CCL_API allreduce (const void *send_buf, void *recv_buf, size_t count, datatype dtype, reduction rtype, const communicator &comm, const allreduce_attr &attr=default_allreduce_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API allreduce (const BufferType *send_buf, BufferType *recv_buf, size_t count, reduction rtype, const communicator &comm, const stream &stream, const allreduce_attr &attr=default_allreduce_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API allreduce (const BufferType *send_buf, BufferType *recv_buf, size_t count, reduction rtype, const communicator &comm, const allreduce_attr &attr=default_allreduce_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferObjectType, class = typename
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API allreduce
(const BufferObjectType &send_buf, BufferObjectType &recv_buf, size_t count, reduction rtype,
const communicator &comm, const stream &stream, const allreduce_attr
&attr=default_allreduce_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferObjectType, class = typename
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API allreduce
(const BufferObjectType &send_buf, BufferObjectType &recv_buf, size_t count, reduction rtype,
const communicator &comm, const allreduce_attr &attr=default_allreduce_attr, const
vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Alltoall

```
event CCL_API alltoall (const void *send_buf, void *recv_buf, size_t count, datatype dtype, const
communicator &comm, const stream &stream, const alltoall_attr &attr=default_alltoall_attr,
const vector_class< event > &deps={})
```

Alltoall is a collective communication operation in which each rank sends distinct equal-sized blocks of data to each rank. The *j*-th block of `send_buf` sent from the *i*-th rank is received by the *j*-th rank and is placed in the *i*-th block of `recvbuf`.

Parameters

- **send_buf** – the buffer with `count` elements of `dtype` that stores local data to be sent
- **recv_buf** – [out] the buffer to store received result, must be large enough to hold values from all ranks, i.e. at least `comm_size * count`
- **count** – the number of elements of type `dtype` to be send to or to received from each rank
- **dtype** – the datatype of elements in `send_buf` and `recv_buf`
- **comm** – the communicator for which the operation will be performed
- **stream** – abstraction over a device queue constructed via `ccl::create_stream`
- **attr** – optional attributes to customize operation
- **deps** – an optional vector of the events that the operation should depend on

Returns

`ccl::event` an object to track the progress of the operation

```
event CCL_API alltoall (const void *send_buf, void *recv_buf, size_t count, datatype dtype, const
communicator &comm, const alltoall_attr &attr=default_alltoall_attr, const vector_class< event
> &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

```
event CCL_API alltoall (const vector_class< void * > &send_buf, const vector_class< void * >
&recv_buf, size_t count, datatype dtype, const communicator &comm, const stream &stream,
const alltoall_attr &attr=default_alltoall_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **send_bufs** – array of buffers with local data to be sent, one buffer per rank
- **recv_bufs** – [out] array of buffers to store received result, one buffer per rank

event CCL_API alltoall (const vector_class< void * > &send_buf, const vector_class< void * > &recv_buf, size_t count, datatype dtype, const communicator &comm, const alltoall_attr &attr=default_alltoall_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **send_bufs** – array of buffers with local data to be sent, one buffer per rank
- **recv_bufs** – [out] array of buffers to store received result, one buffer per rank

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API alltoall (const BufferType *send_buf, BufferType *recv_buf, size_t count, const communicator &comm, const stream &stream, const alltoall_attr &attr=default_alltoall_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API alltoall (const BufferType *send_buf, BufferType *recv_buf, size_t count, const communicator &comm, const alltoall_attr &attr=default_alltoall_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API alltoall (const vector_class< BufferType * > &send_buf, const vector_class< BufferType * > &recv_buf, size_t count, const communicator &comm, const stream &stream, const alltoall_attr &attr=default_alltoall_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_bufs** – array of buffers with local data to be sent, one buffer per rank
- **recv_bufs** – [out] array of buffers to store received result, one buffer per rank

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API alltoall (const vector_class< BufferType * > &send_buf, const vector_class< BufferType * > &recv_buf, size_t count, const communicator &comm, const alltoall_attr &attr=default_alltoall_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_bufs** – array of buffers with local data to be sent, one buffer per rank

- **recv_bufs** – [out] array of buffers to store received result, one buffer per rank

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API alltoall  
(const BufferObjectType &send_buf, BufferObjectType &recv_buf, size_t count, const  
communicator &comm, const stream &stream, const alltoall_attr &attr=default_alltoall_attr,  
const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API alltoall  
(const BufferObjectType &send_buf, BufferObjectType &recv_buf, size_t count, const  
communicator &comm, const alltoall_attr &attr=default_alltoall_attr, const vector_class< event  
> &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_bufs** – array of buffers with local data to be sent, one buffer per rank
- **recv_bufs** – [out] array of buffers to store received result, one buffer per rank

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API alltoall  
(const vector_class< reference_wrapper_class< BufferObjectType >> &send_buf, const  
vector_class< reference_wrapper_class< BufferObjectType >> &recv_buf, size_t count, const  
communicator &comm, const stream &stream, const alltoall_attr &attr=default_alltoall_attr,  
const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_bufs** – array of buffers with local data to be sent, one buffer per rank
- **recv_bufs** – [out] array of buffers to store received result, one buffer per rank

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API alltoall  
(const vector_class< reference_wrapper_class< BufferObjectType >> &send_buf, const  
vector_class< reference_wrapper_class< BufferObjectType >> &recv_buf, size_t count, const  
communicator &comm, const alltoall_attr &attr=default_alltoall_attr, const vector_class< event  
> &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Parameters

- **send_bufs** – array of buffers with local data to be sent, one buffer per rank
- **recv_bufs** – [out] array of buffers to store received result, one buffer per rank

Alltoallv

event CCL_API alltoallv (const void *send_buf, const vector_class< size_t > &send_counts, void *recv_buf, const vector_class< size_t > &recv_counts, datatype dtype, const communicator &comm, const stream &stream, const alltoallv_attr &attr=default_alltoallv_attr, const vector_class< event > &deps={})

Alltoallv is a collective communication operation in which each rank sends distinct blocks of data to each rank. Block sizes may differ. The j-th block of `send_buf` sent from the i-th rank is received by the j-th rank and is placed in the i-th block of `recvbuf`.

Parameters

- **send_buf** – the buffer with elements of `dtype` that stores local blocks to be sent to each rank
- **send_bufs** – array of buffers to store send blocks, one buffer per rank
- **recv_buf** – [out] the buffer to store received result, must be large enough to hold blocks from all ranks
- **recv_bufs** – [out] array of buffers to store receive blocks, one buffer per rank
- **send_counts** – array with the number of elements of type `dtype` in send blocks for each rank
- **recv_counts** – array with the number of elements of type `dtype` in receive blocks from each rank
- **dtype** – the datatype of elements in `send_buf` and `recv_buf`
- **comm** – the communicator for which the operation will be performed
- **stream** – abstraction over a device queue constructed via `ccl::create_stream`
- **attr** – optional attributes to customize operation
- **deps** – an optional vector of the events that the operation should depend on

Returns

`ccl::event` an object to track the progress of the operation

event CCL_API alltoallv (const void *send_buf, const vector_class< size_t > &send_counts, void *recv_buf, const vector_class< size_t > &recv_counts, datatype dtype, const communicator &comm, const alltoallv_attr &attr=default_alltoallv_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

event CCL_API alltoallv (const vector_class< void * > &send_bufs, const vector_class< size_t > &send_counts, const vector_class< void * > &recv_bufs, const vector_class< size_t > &recv_counts, datatype dtype, const communicator &comm, const stream &stream, const alltoallv_attr &attr=default_alltoallv_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

event CCL_API alltoallv (const vector_class< void * > &send_bufs, const vector_class< size_t > &send_counts, const vector_class< void * > &recv_bufs, const vector_class< size_t > &recv_counts, datatype dtype, const communicator &comm, const alltoallv_attr &attr=default_alltoallv_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API alltoallv (const BufferType *send_buf, const vector_class< size_t > &send_counts, BufferType *recv_buf, const vector_class< size_t > &recv_counts, const communicator &comm, const stream &stream, const alltoallv_attr &attr=default_alltoallv_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferType, class = typename  
std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API alltoallv  
(const BufferType *send_buf, const vector_class< size_t > &send_counts, BufferType *recv_buf,  
const vector_class< size_t > &recv_counts, const communicator &comm, const alltoallv_attr  
&attr=default_alltoallv_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferType, class = typename  
std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API alltoallv  
(const vector_class< BufferType * > &send_bufs, const vector_class< size_t > &send_counts,  
const vector_class< BufferType * > &recv_bufs, const vector_class< size_t > &recv_counts,  
const communicator &comm, const stream &stream, const alltoallv_attr  
&attr=default_alltoallv_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferType, class = typename  
std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API alltoallv  
(const vector_class< BufferType * > &send_bufs, const vector_class< size_t > &send_counts,  
const vector_class< BufferType * > &recv_bufs, const vector_class< size_t > &recv_counts,  
const communicator &comm, const alltoallv_attr &attr=default_alltoallv_attr, const  
vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API alltoallv  
(const BufferObjectType &send_buf, const vector_class< size_t > &send_counts,  
BufferObjectType &recv_buf, const vector_class< size_t > &recv_counts, const communicator  
&comm, const stream &stream, const alltoallv_attr &attr=default_alltoallv_attr, const  
vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API alltoallv  
(const BufferObjectType &send_buf, const vector_class< size_t > &send_counts,  
BufferObjectType &recv_buf, const vector_class< size_t > &recv_counts, const communicator  
&comm, const alltoallv_attr &attr=default_alltoallv_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferObjectType, class = typename  
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API alltoallv  
(const vector_class< reference_wrapper_class< BufferObjectType >> &send_bufs, const  
vector_class< size_t > &send_counts, const vector_class< reference_wrapper_class<
```

BufferObjectType >> &recv_bufs, const vector_class< size_t > &recv_counts, const communicator &comm, const stream &stream, const alltoallv_attr &attr=default_alltoallv_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferObjectType, class = typename std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API alltoallv (const vector_class< reference_wrapper_class< BufferObjectType >> &send_bufs, const vector_class< size_t > &send_counts, const vector_class< reference_wrapper_class< BufferObjectType >> &recv_bufs, const vector_class< size_t > &recv_counts, const communicator &comm, const alltoallv_attr &attr=default_alltoallv_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Barrier

event CCL_API barrier (const communicator &comm, const stream &stream, const barrier_attr &attr=default_barrier_attr, const vector_class< event > &deps={})

Barrier synchronization is performed across all ranks of the communicator and it is completed only after all the ranks in the communicator have called it.

Parameters

- **comm** – the communicator for which the operation will be performed
- **stream** – abstraction over a device queue constructed via `ccl::create_stream`
- **attr** – optional attributes to customize operation
- **deps** – an optional vector of the events that the operation should depend on

Returns

`ccl::event` an object to track the progress of the operation

event CCL_API barrier (const communicator &comm, const barrier_attr &attr=default_barrier_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Broadcast

event CCL_API broadcast (void *buf, size_t count, datatype dtype, int root, const communicator &comm, const stream &stream, const broadcast_attr &attr=default_broadcast_attr, const vector_class< event > &deps={})

Broadcast is a collective communication operation that broadcasts data from one rank of communicator (denoted as root) to all other ranks.

Parameters

- **buf** – [in,out] the buffer with `count` elements of `dtype` serves as send buffer for root and as receive buffer for other ranks
- **count** – the number of elements of type `dtype` in `buf`
- **dtype** – the datatype of elements in `buf`
- **root** – the rank that broadcasts `buf`
- **comm** – the communicator for which the operation will be performed
- **stream** – abstraction over a device queue constructed via `ccl::create_stream`
- **attr** – optional attributes to customize operation

- **deps** – an optional vector of the events that the operation should depend on

Returns

ccl::event an object to track the progress of the operation

event CCL_API broadcast (void *buf, size_t count, datatype dtype, int root, const communicator &comm, const broadcast_attr &attr=default_broadcast_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API broadcast (BufferType *buf, size_t count, int root, const communicator &comm, const stream &stream, const broadcast_attr &attr=default_broadcast_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API broadcast (BufferType *buf, size_t count, int root, const communicator &comm, const broadcast_attr &attr=default_broadcast_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferObjectType, class = typename std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API broadcast (BufferObjectType &buf, size_t count, int root, const communicator &comm, const stream &stream, const broadcast_attr &attr=default_broadcast_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferObjectType, class = typename std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API broadcast (BufferObjectType &buf, size_t count, int root, const communicator &comm, const broadcast_attr &attr=default_broadcast_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Reduce

event CCL_API reduce (const void *send_buf, void *recv_buf, size_t count, datatype dtype, reduction rtype, int root, const communicator &comm, const stream &stream, const reduce_attr &attr=default_reduce_attr, const vector_class< event > &deps={})

Reduce is a collective communication operation that performs the global reduction operation on values from all ranks of the communicator and returns the result to the root rank.

Parameters	<ul style="list-style-type: none"> • send_buf – the buffer with <code>count</code> elements of <code>dtype</code> that stores local data to be reduced • recv_buf – [out] the buffer to store reduced result, must have the same dimension as <code>send_buf</code>. Used by the <code>root</code> rank only, ignored by other ranks. • count – the number of elements of type <code>dtype</code> in <code>send_buf</code> and <code>recv_buf</code> • dtype – the datatype of elements in <code>send_buf</code> and <code>recv_buf</code> • rtype – the type of the reduction operation to be applied • root – the rank that gets the result of reduction • comm – the communicator for which the operation will be performed • stream – abstraction over a device queue constructed via <code>ccl::create_stream</code> • attr – optional attributes to customize operation • deps – an optional vector of the events that the operation should depend on
Returns	<code>ccl::event</code> an object to track the progress of the operation

event CCL_API reduce (const void *send_buf, void *recv_buf, size_t count, datatype dtype, reduction rtype, int root, const communicator &comm, const reduce_attr &attr=default_reduce_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API reduce (const BufferType *send_buf, BufferType *recv_buf, size_t count, reduction rtype, int root, const communicator &comm, const stream &stream, const reduce_attr &attr=default_reduce_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferType, class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API reduce (const BufferType *send_buf, BufferType *recv_buf, size_t count, reduction rtype, int root, const communicator &comm, const reduce_attr &attr=default_reduce_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferObjectType, class = typename std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API reduce (const BufferObjectType &send_buf, BufferObjectType &recv_buf, size_t count, reduction rtype, int root, const communicator &comm, const stream &stream, const reduce_attr &attr=default_reduce_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

template<class BufferObjectType, class = typename std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API reduce (const BufferObjectType &send_buf, BufferObjectType &recv_buf, size_t count, reduction rtype, int root, const communicator &comm, const reduce_attr &attr=default_reduce_attr, const vector_class< event > &deps={})

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

ReduceScatter

```
event CCL_API reduce_scatter (const void *send_buf, void *recv_buf, size_t recv_count,
datatype dtype, reduction rtype, const communicator &comm, const stream &stream, const
reduce_scatter_attr &attr=default_reduce_scatter_attr, const vector_class< event > &deps={})
```

Reduce-scatter is a collective communication operation that performs the global reduction operation on values from all ranks of the communicator and scatters the result in blocks back to all ranks.

Parameters

- **send_buf** – the buffer with `comm_size * count` elements of `dtype` that stores local data to be reduced
- **recv_buf** – [out] the buffer to store result block containing `recv_count` elements of type `dtype`
- **recv_count** – the number of elements of type `dtype` in receive block
- **dtype** – the datatype of elements in `send_buf` and `recv_buf`
- **rtype** – the type of the reduction operation to be applied
- **comm** – the communicator for which the operation will be performed
- **stream** – abstraction over a device queue constructed via `ccl::create_stream`
- **attr** – optional attributes to customize operation
- **deps** – an optional vector of the events that the operation should depend on

Returns

`ccl::event` an object to track the progress of the operation

```
event CCL_API reduce_scatter (const void *send_buf, void *recv_buf, size_t recv_count,
datatype dtype, reduction rtype, const communicator &comm, const reduce_scatter_attr
&attr=default_reduce_scatter_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

```
template<class BufferType, class = typename
std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API
reduce_scatter (const BufferType *send_buf, BufferType *recv_buf, size_t recv_count, reduction
rtype, const communicator &comm, const stream &stream, const reduce_scatter_attr
&attr=default_reduce_scatter_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferType, class = typename
std::enable_if<is_native_type_supported<BufferType>(), event>::type> event CCL_API
reduce_scatter (const BufferType *send_buf, BufferType *recv_buf, size_t recv_count, reduction
rtype, const communicator &comm, const reduce_scatter_attr
&attr=default_reduce_scatter_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

```
template<class BufferObjectType, class = typename
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API
reduce_scatter (const BufferObjectType &send_buf, BufferObjectType &recv_buf, size_t
recv_count, reduction rtype, const communicator &comm, const stream &stream, const
reduce_scatter_attr &attr=default_reduce_scatter_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.


```
template<class BufferObjectType, class = typename
std::enable_if<is_class_supported<BufferObjectType>(), event>::type> event CCL_API
reduce_scatter (const BufferObjectType &send_buf, BufferObjectType &recv_buf, size_t
recv_count, reduction rtype, const communicator &comm, const reduce_scatter_attr
&attr=default_reduce_scatter_attr, const vector_class< event > &deps={})
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Type-safe version.

Point-To-Point Operations

Point-to-point operations enable direct communication between two specific entities, facilitating data exchange, synchronization, and coordination within a parallel computing environment.

The following point-to-point operations are available in oneCCL:

- send
- recv

send

`send` is a blocking point-to-point communication operation that transfers data from a designated memory buffer (`buf`) to a specific peer rank.

```
event CCL_API send(void *buf,
    size_t count,
    datatype dtype,
    int peer,
    const communicator &comm,
    const stream &stream,
    const pt2pt_attr &attr = default_pt2pt_attr,
    const vector_class<event> &deps = {});
```

Parameters

- `buf` - A buffer with `dtype` count elements that contains the data to be sent.
- `count` - The number of `dtype` elements in a `buf`.
- `dtype` - The datatype of elements in a `buf`.
- `peer` - A destination rank.
- `comm` - A communicator for which the operation is performed.
- `stream` - A stream associated with the operation.
- `attr` - Optional attributes to customize the operation.
- `deps` - An optional vector of the events, on which the operation should depend.

Returns

`ccl::event` - An object to track the progress of the operation.

```
event CCL_API send(void* buf,
    size_t count,
    datatype dtype,
    int peer,
    const communicator &comm,
    const pt2pt_attr &attr = default_pt2pt_attr,
    const vector_class<event> &deps = {});
```

Below you can find an overloaded member function provided for the convenience. It differs from the above function only in what argument(s) it accepts.

```
template <class BufferType,
         class = typename std::enable_if<is_native_type_supported<BufferType>(), event>::type>
event CCL_API send(BufferType *buf,
                  size_t count,
                  int peer,
                  const communicator &comm,
                  const stream &stream,
                  const pt2pt_attr &attr = default_pt2pt_attr,
                  const vector_class<event>& deps = {});
```

Below you can find an overloaded member function provided for the convenience. It differs from the above function only in what argument(s) it accepts.:

```
event CCL_API send(BufferType *buf,
                  size_t count,
                  int peer,
                  const communicator &comm,
                  const pt2pt_attr &attr = default_pt2pt_attr,
                  const vector_class<event> &deps = {});
```

Below you can find an overloaded member function provided for the convenience. It differs from the above function only in what argument(s) it accepts.

```
event CCL_API send(BufferObjectType &buf,
                  size_t count,
                  int peer,
                  const communicator &comm,
                  const stream &stream,
                  const pt2pt_attr &attr = default_pt2pt_attr,
                  const vector_class<event> &deps = {});
```

Below you can find an overloaded member function provided for the convenience. It differs from the above function only in what argument(s) it accepts.

```
event CCL_API send(BufferObjectType &buf,
                  size_t count,
                  int peer,
                  const communicator &comm,
                  const pt2pt_attr &attr = default_pt2pt_attr,
                  const vector_class<event> &deps = {});
```

recv

`recv` is a blocking point-to-point communication operation that receives data from a peer rank in a memory buffer.

```
event CCL_API recv(void *buf,
                  size_t count,
                  datatype dtype,
                  int peer,
                  const communicator &comm,
                  const stream &stream,
                  const pt2pt_attr &attr = default_pt2pt_attr,
                  const vector_class<event> &deps = {});
```

Parameters

- `buf` - A buffer with `dtype` count elements that contains where the data is received.
- `count` - The number of `dtype` elements in a `buf`.

- `dtype`- The datatype of elements in a `buf`.
- `peer` - A source rank.
- `comm` - A communicator for which the operation is performed.
- `dstream` - A stream associated with the operation.
- `attr` - Optional attributes to customize the operation.
- `deps` - An optional vector of the events, on which the operation should depend.

Returns:

`ccl::event` - An object to track the progress of the operation.

```
event CCL_API recv(void *buf,
    size_t count,
    datatype dtype,
    int peer,
    const communicator &comm,
    const pt2pt_attr &attr = default_pt2pt_attr,
    const vector_class<event>& deps = {});
```

Below you can find an overloaded member function provided for the convenience. It differs from the above function only in what argument(s) it accepts.

```
template <class BufferType,
    class = typename std::enable_if<is_native_type_supported<BufferType>(), event>:::type>
event CCL_API recv(BufferType *buf,
    size_t count,
    int peer,
    const communicator &comm,
    const stream &stream,
    const pt2pt_attr &attr = default_pt2pt_attr,
    const vector_class<event> &deps = {});
```

Below you can find an overloaded member function provided for the convenience. It differs from the above function only in what argument(s) it accepts.

```
event CCL_API recv(BufferType *buf,
    size_t count,
    int peer,
    const communicator &comm,
    const pt2pt_attr &attr = default_pt2pt_attr,
    const vector_class<event> &deps = {});
```

Below you can find an overloaded member function provided for the convenience. It differs from the above function only in what argument(s) it accepts.

```
event CCL_API recv(BufferObjectType &buf,
    size_t count,
    int peer,
    const communicator &comm,
    const stream &stream,
    const pt2pt_attr &attr = default_pt2pt_attr,
    const vector_class<event> &deps = {});
```

Below you can find an overloaded member function provided for the convenience. It differs from the above function only in what argument(s) it accepts.

```
event CCL_API recv(BufferObjectType &buf,
    size_t count,
    int peer,
```

```
const communicator &comm,
const pt2pt_attr &attr = default_pt2pt_attr,
const vector_class<event> &deps = {});
```

Environment Variables

Collective Algorithms Selection

oneCCL supports collective operations for the host (CPU) memory buffers and device (GPU) memory buffers. Below you can see how to select the collective algorithm depending on the type of buffer being utilized.

Device (GPU) Memory Buffers

Collectives that use GPU buffers are implemented using two phases:

- Scaleup phase. Communication between ranks/processes in the same node.
- Scaleout phase. Communication between ranks/processes on different nodes.

SCALEUP

Use the following environment variables to select the scaleup algorithm:

```
CCL_REDUCE_SCATTER_MONOLITHIC_KERNEL
```

Syntax

```
CCL_REDUCE_SCATTER_MONOLITHIC_KERNEL=<value>
```

Arguments

<value>	Description
1	Uses compute kernels to transfer data across GPUs for the ALLREDUCE, REDUCE, and REDUCE_SCATTER collectives.
0	Uses copy engines to transfer data across GPUs for the ALLREDUCE, REDUCE, and REDUCE_SCATTER collectives. The default value.

Description

Set this environment variable to enable compute kernels for the ALLREDUCE, REDUCE, and REDUCE_SCATTER collectives using device (GPU) buffers.

```
CCL_ALLGATHERV_MONOLITHIC_PIPELINE_KERNEL
```

Syntax

```
CCL_ALLGATHERV_MONOLITHIC_PIPELINE_KERNEL=<value>
```

Arguments

<value>	Description
1	Uses compute kernels to transfer data across GPUs for the ALLGATHERV collective. The default value.
0	Uses copy engines to transfer data across GPUs for the ALLGATHERV collective.

Description

Set this environment variable to enable compute kernels that pipeline data transfers across tiles in the same GPU with data transfers across different GPUs, for the ALLGATHERV collective using device (GPU) buffers.

CCL_REDUCE_SCATTER_MONOLITHIC_PIPELINE_KERNEL

Syntax

```
CCL_REDUCE_SCATTER_MONOLITHIC_PIPELINE_KERNEL=<value>
```

Arguments

<value>	Description
1	Uses compute kernels for the ALLREDUCE, REDUCE, and REDUCE_SCATTER collectives. The default value.
0	Uses copy engines to transfer data across GPUs for the ALLREDUCE, REDUCE, and REDUCE_SCATTER collectives.

Description

Set this environment variable to enable compute kernels that pipeline data transfers across tiles in the same GPU with data transfers across different GPUs for the ALLREDUCE, REDUCE, and REDUCE_SCATTER collectives using the device (GPU) buffers.

CCL_ALLTOALLV_MONOLITHIC_KERNEL

Syntax

```
CCL_ALLTOALLV_MONOLITHIC_KERNEL=<value>
```

Arguments

<value>	Description
1	Uses compute kernels to transfer data across GPUs for the ALLTOALL and ALLTOALLV collectives. The default value.
0	Uses copy engines to transfer data across GPUs for the ALLTOALL and ALLTOALLV collectives.

Description

Set this environment variable to enable compute kernels for the ALLTOALL and ALLTOALLV collectives using device (GPU) buffers CCL_<coll_name>_SCALEOUT.

CCL_SKIP_SCHEDULER

Syntax

```
CCL_SKIP_SCHEDULER=<value>
```

Arguments

<value>	Description
1	Enable SYCL kernels
0	Disable SYCL kernels. The default value.

Description

Set this environment variable to 1 to enable the SYCL kernel-based implementation for `ALLGATHERV`, `ALLREDUCE`, and `REDUCE_SCATTER`. This new optimization enhances all message sizes and supports the `int32`, `fp32`, `fp16`, and `bf16` data types, sum operations, and single nodes. oneCCL falls back to other implementations when the support is unavailable with SYCL kernels. Therefore, you can safely set this environment variable.

SCALEOUT

The following environment variables can be used to select the scaleout algorithm used.

Syntax

To set a specific algorithm for scaleout for the device (GPU) buffers for the whole message size range:

```
CCL_<coll_name>_SCALEOUT=<algo_name>
```

To set a specific algorithm for scaleout for the device (GPU) buffers for a specific message size range:

```
CCL_<coll_name>_SCALEOUT="<algo_name_1>[:<size_range_1>][;<algo_name_2>:<size_range_2>][;...]"
```

Where:

- `<coll_name>` is selected from a list of the available collective operations ([Available collectives](#)).
- `<algo_name>` is selected from a list of the available algorithms for the specific collective operation ([Available collectives](#)).
- `<size_range>` is described by the left and the right size borders in the `<left>-<right>` format. The size is specified in bytes. To specify the maximum message size, use reserved word `max`.

oneCCL internally fills the algorithm selection table with sensible defaults. Your input complements the selection table. To see the actual table values, set `CCL_LOG_LEVEL=info`.

Example

```
CCL_ALLREDUCE_SCALEOUT="recursive_doubling:0-8192;rabenseifner:8193-1048576;ring:1048577-max"
```

Available Collectives

Available collective operations (`<coll_name>`):

- `ALLGATHERV`
- `ALLREDUCE`
- `ALLTOALL`
- `ALLTOALLV`
- `BARRIER`
- `BCAST`
- `REDUCE`
- `REDUCE_SCATTER`

Available algorithms

Available algorithms for each collective operation (`<algo_name>`):

ALLGATHERV algorithms

<code>direct</code>	Based on <code>MPI_Iallgatherv</code>
<code>naive</code>	Send to all, receive from all
<code>flat</code>	Alltoall-based algorithm
<code>multi_bcast</code>	Series of broadcast operations with different root ranks

ring	Ring-based algorithm
------	----------------------

ALLREDUCE algorithms

direct	Based on <code>MPI_Iallreduce</code>
rabenseifner	Rabenseifner's algorithm
nreduce	May be beneficial for imbalanced workloads
ring	<code>reduce_scatter</code> + <code>allgather</code> ring. Use <code>CCL_RS_CHUNK_COUNT</code> and <code>CCL_RS_MIN_CHUNK_SIZE</code> to control pipelining on <code>reduce_scatter</code> phase.
double_tree	Double-tree algorithm
recursive_doubling	Recursive doubling algorithm
2d	Two-dimensional algorithm (<code>reduce_scatter</code> + <code>allreduce</code> + <code>allgather</code>). Only available for the host (CPU) buffers.

ALLTOALL algorithms

direct	Based on <code>MPI_Ialltoall</code>
naive	Send to all, receive from all
scatter	Scatter-based algorithm

ALLTOALLV algorithms

direct	Based on <code>MPI_Ialltoallv</code>
naive	Send to all, receive from all
scatter	Scatter-based algorithm

BARRIER algorithms

direct	Based on <code>MPI_Ibarrier</code>
ring	Ring-based algorithm

NOTE The `BARRIER` algorithm does not support the `CCL_BARRIER_SCALEOUT` environment variable. To change the algorithm for `BARRIER`, use the `CCL_BARRIER` environment variable.

BCAST algorithms

direct	Based on <code>MPI_Ibcast</code>
ring	Ring
double_tree	Double-tree algorithm
naive	Send to all from root rank

NOTE The `BCAST` algorithm does not yet support the `CCL_BCAST_SCALEOUT` environment variable. To change the algorithm for `BCAST`, use the `CCL_BCAST` environment variable.

REDUCE algorithms

<code>direct</code>	Based on <code>MPI_Ireduce</code>
<code>rabenseifner</code>	Rabenseifner's algorithm
<code>tree</code>	Tree algorithm
<code>double_tree</code>	Double-tree algorithm

REDUCE_SCATTER algorithms

<code>direct</code>	Based on <code>MPI_Ireduce_scatter_block</code>
<code>ring</code>	Use <code>CCL_RS_CHUNK_COUNT</code> and <code>CCL_RS_MIN_CHUNK_SIZE</code> to control pipelining.

NOTE The `REDUCE_SCATTER` algorithm does not yet support the `CCL_REDUCE_SCATTER_SCALEOUT` environment variable. To change the algorithm for `REDUCE_SCATTER`, use the `CCL_REDUCE_SCATTER` environment variable.

Host (CPU) Memory Buffers

`CCL_<coll_name>`

Syntax

To set a specific algorithm for the host (CPU) buffers for the whole message size range:

```
CCL_<coll_name>=<algo_name>
```

To set a specific algorithm for the host (CPU) buffers for a specific message size range:

```
CCL_<coll_name>="<algo_name_1>[:<size_range_1>][;<algo_name_2>:<size_range_2>][;...]"
```

Where:

- `<coll_name>` is selected from a list of available collective operations ([Available collectives](#)).
- `<algo_name>` is selected from a list of available algorithms for a specific collective operation ([Available algorithms](#)).
- `<size_range>` is described by the left and the right size borders in a format `<left>-<right>`. Size is specified in bytes. Use reserved word `max` to specify the maximum message size.

oneCCL internally fills algorithm selection table with sensible defaults. User input complements the selection table. To see the actual table values set `CCL_LOG_LEVEL=info`.

Example

```
CCL_ALLREDUCE="recursive_doubling:0-8192;rabenseifner:8193-1048576;ring:1048577-max"
```

CCL_RS_CHUNK_COUNT

Syntax

```
CCL_RS_CHUNK_COUNT=<value>
```


Arguments

<value>	Description
COUNT	Maximum number of chunks.

Description

Set this environment variable to specify maximum number of chunks for reduce_scatter phase in ring allreduce.

CCL_RS_MIN_CHUNK_SIZE

Syntax

```
CCL_RS_MIN_CHUNK_SIZE=<value>
```

Arguments

<value>	Description
SIZE	Minimum number of bytes in chunk.

Description

Set this environment variable to specify minimum number of bytes in chunk for reduce_scatter phase in ring allreduce. Affects actual value of CCL_RS_CHUNK_COUNT.

Workers

The group of environment variables to control worker threads.

CCL_WORKER_COUNT

Syntax

```
CCL_WORKER_COUNT=<value>
```

Arguments

<value>	Description
N	The number of worker threads for oneCCL rank (1 if not specified).

Description

Set this environment variable to specify the number of oneCCL worker threads.

CCL_WORKER_AFFINITY

Syntax

```
CCL_WORKER_AFFINITY=<cpulist>
```

Arguments

<cpulist>	Description
auto	Workers are automatically pinned to last cores of pin domain. Pin domain depends from process launcher. If <code>mpirun</code> from oneCCL package is used then pin domain is MPI process pin domain. Otherwise, pin domain is all cores on the node.

<cpulist>	Description
<cpulist>	A comma-separated list of core numbers and/or ranges of core numbers for all local workers, one number per worker. The <i>i</i> -th local worker is pinned to the <i>i</i> -th core in the list. For example <a>, -<c> defines list of cores containing core with number <a> and range of cores with numbers from to <c>. The core number should not exceed the number of cores available on the system. The length of the list should be equal to the number of workers.

Description

Set this environment variable to specify cpu affinity for oneCCL worker threads.

CCL_WORKER_MEM_AFFINITY

Syntax

```
CCL_WORKER_MEM_AFFINITY=<nodelist>
```

Arguments

<nodelist>	Description
auto	Workers are automatically pinned to NUMA nodes that correspond to CPU affinity of workers.
<nodelist>	A comma-separated list of NUMA node numbers for all local workers, one number per worker. The <i>i</i> -th local worker is pinned to the <i>i</i> -th NUMA node in the list. The number should not exceed the number of NUMA nodes available on the system.

Description

Set this environment variable to specify memory affinity for oneCCL worker threads.

ATL

The group of environment variables to control ATL (abstract transport layer).

CCL_ATL_TRANSPORT

Syntax

```
CCL_ATL_TRANSPORT=<value>
```

Arguments

<value>	Description
mpi	MPI transport (default).
ofi	OFI (libfabric*) transport.

Description

Set this environment variable to select the transport for inter-process communications.

CCL_ATL_HMEM

Syntax

```
CCL_ATL_HMEM=<value>
```

Arguments

<value>	Description
1	Enable heterogeneous memory support on the transport layer.
0	Disable heterogeneous memory support on the transport layer (default).

Description

Set this environment variable to enable handling of HMEM/GPU buffers by the transport layer. The actual HMEM support depends on the limitations on the transport level and system configuration.

```
CCL_ATL_SHM
```

Syntax

```
CCL_ATL_SHM=<value>
```

Arguments

<value>	Description
0	Disables the OFI shared memory provider. The default value.
1	Enables the OFI shared memory provider.

Description

Set this environment variable to enable the OFI shared memory provider to communicate between ranks in the same node of the host (CPU) buffers. This capability requires OFI as the transport (CCL_ATL_TRANSPORT=ofi).

The OFI/SHM provider has support to utilize the [Intel\(R\) Data Streaming Accelerator* \(DSA\)](#). To run it with DSA*, you need: * Linux* OS kernel support for the DSA* shared work queues * Libfabric* 1.17 or later

To enable DSA, set the following environment variables:

```
FI_SHM_DISABLE_CMA=1
FI_SHM_USE_DSA_SAR=1
```

Refer to Libfabric* Programmer's Manual for the additional details about DSA* support in the SHM provider: https://ofiwg.github.io/libfabric/main/man/fi_shm.7.html.

```
CCL_PROCESS_LAUNCHER
```

Syntax

```
CCL_PROCESS_LAUNCHER=<value>
```

Arguments

<value>	Description
hydra	Uses the MPI hydra job launcher. The default value.
torch	Uses a torch job launcher.

<value>	Description
pmix	Is used with the PALS job launcher that uses the pmix API. The mpiexec command should be similar to: <pre>CCL_PROCESS_LAUNCHER=pmix CCL_ATL_TRANSPORT=mpi mpiexec -np 2 --ppn 2 --pmi=pmix ...</pre>
none	No job launcher is used. You should specify the values for CCL_LOCAL_SIZE and CCL_LOCAL_RANK.

Description

Set this environment variable to specify the job launcher.

CCL_LOCAL_SIZE

Syntax

```
CCL_LOCAL_SIZE=<value>
```

Arguments

<value>	Description
SIZE	A total number of ranks on the local host.

Description

Set this environment variable to specify a total number of ranks on a local host.

CCL_LOCAL_RANK

Syntax

```
CCL_LOCAL_RANK=<value>
```

Arguments

<value>	Description
RANK	Rank number of the current process on the local host.

Description

Set this environment variable to specify the rank number of the current process in the local host.

Multi-NIC

CCL_MNIC, CCL_MNIC_NAME and CCL_MNIC_COUNT define filters to select multiple NICs. oneCCL workers will be pinned on selected NICs in a round-robin way.

CCL_MNIC

Syntax

```
CCL_MNIC=<value>
```

Arguments

<value>	Description
global	Select all NICs available on the node.

<value>	Description
local	Select all NICs local for the NUMA node that corresponds to process pinning.
none	Disable special NIC selection, use a single default NIC (default).

Description

Set this environment variable to control multi-NIC selection by NIC locality.

CCL_MNIC_NAME

Syntax

```
CCL_MNIC_NAME=<namelist>
```

Arguments

<namelist>	Description
<namelist>	A comma-separated list of NIC full names or prefixes to filter NICs. Use the ^ symbol to exclude NICs starting with the specified prefixes. For example, if you provide a list <code>m1x5_0,m1x5_1,^m1x5_2</code> , NICs with the names <code>m1x5_0</code> and <code>m1x5_1</code> will be selected, while <code>m1x5_2</code> will be excluded from the selection.

Description

Set this environment variable to control multi-NIC selection by NIC names.

CCL_MNIC_COUNT

Syntax

```
CCL_MNIC_COUNT=<value>
```

Arguments

<value>	Description
N	The maximum number of NICs that should be selected for oneCCL workers. If not specified then equal to the number of oneCCL workers.

Description

Set this environment variable to specify the maximum number of NICs to be selected. The actual number of NICs selected may be smaller due to limitations on transport level or system configuration.

Low-precision datatypes

The group of environment variables to control processing of low-precision datatypes.

CCL_BF16

Syntax

```
CCL_BF16=<value>
```

Arguments

<value>	Description
avx512f	Select implementation based on AVX512F instructions.
avx512bf	Select implementation based on AVX512_BF16 instructions.

Description

Set this environment variable to select implementation for BF16 <-> FP32 conversion on reduction phase of collective operation. Default value depends on instruction set support on specific CPU. AVX512_BF16-based implementation has precedence over AVX512F-based one.

CCL_FP16

Syntax

```
CCL_FP16=<value>
```

Arguments

<value>	Description
f16c	Select implementation based on F16C instructions.
avx512f	Select implementation based on AVX512F instructions.

Description

Set this environment variable to select implementation for FP16 <-> FP32 conversion on reduction phase of collective operation. Default value depends on instruction set support on specific CPU. AVX512F-based implementation has precedence over F16C-based one.

CCL_LOG_LEVEL

Syntax

```
CCL_LOG_LEVEL=<value>
```

Arguments

<value>
error
warn (default)
info
debug
trace

Description

Set this environment variable to control logging level.

CCL_ITT_LEVEL

Syntax

```
CCL_ITT_LEVEL=<value>
```

Arguments

<value>	Description
1	Enable support for ITT profiling.
0	Disable support for ITT profiling (default).

Description

Set this environment variable to specify Intel® Instrumentation and Tracing Technology (ITT) profiling level. Once the environment variable is enabled (value > 0), it is possible to collect and display profiling data for oneCCL using tools such as Intel® VTune™ Profiler.

Fusion

The group of environment variables to control fusion of collective operations.

CCL_FUSION

Syntax

```
CCL_FUSION=<value>
```

Arguments

<value>	Description
1	Enable fusion of collective operations
0	Disable fusion of collective operations (default)

Description

Set this environment variable to control fusion of collective operations. The real fusion depends on additional settings described below.

CCL_FUSION_BYTES_THRESHOLD

Syntax

```
CCL_FUSION_BYTES_THRESHOLD=<value>
```

Arguments

<value>	Description
SIZE	Bytes threshold for a collective operation. If the size of a communication buffer in bytes is less than or equal to SIZE, then oneCCL fuses this operation with the other ones.

Description

Set this environment variable to specify the threshold of the number of bytes for a collective operation to be fused.

CCL_FUSION_COUNT_THRESHOLD

Syntax

```
CCL_FUSION_COUNT_THRESHOLD=<value>
```

Arguments

<value>	Description
COUNT	The threshold for the number of collective operations. oneCCL can fuse together no more than COUNT operations at a time.

Description

Set this environment variable to specify count threshold for a collective operation to be fused.

CCL_FUSION_CYCLE_MS

Syntax

```
CCL_FUSION_CYCLE_MS=<value>
```

Arguments

<value>	Description
MS	<p>The frequency of checking for collectives operations to be fused, in milliseconds:</p> <ul style="list-style-type: none"> • Small MS value can improve latency. • Large MS value can help to fuse larger number of operations at a time.

Description

Set this environment variable to specify the frequency of checking for collectives operations to be fused.

CCL_PRIORITY

Syntax

```
CCL_PRIORITY=<value>
```

Arguments

<value>	Description
direct	You have to explicitly specify priority using <code>priority</code> .
lifo	Priority is implicitly increased on each collective call. You do not have to specify priority.
none	Disable prioritization (default).

Description

Set this environment variable to control priority mode of collective operations.

CCL_MAX_SHORT_SIZE

Syntax

```
CCL_MAX_SHORT_SIZE=<value>
```

Arguments

<value>	Description
SIZE	Bytes threshold for a collective operation (0 if not specified). If the size of a communication buffer in bytes is less than or equal to SIZE, then oneCCL does not split operation between workers. Applicable for <code>allreduce</code> , <code>reduce</code> and <code>broadcast</code> .

Description

Set this environment variable to specify the threshold of the number of bytes for a collective operation to be split.

CCL_SYCL_OUTPUT_EVENT

Syntax

```
CCL_SYCL_OUTPUT_EVENT=<value>
```

Arguments

<value>	Description
1	Enable support for SYCL output event (default).
0	Disable support for SYCL output event.

Description

Set this environment variable to control support for SYCL output event. Once the support is enabled, you can retrieve SYCL output event from oneCCL event using `get_native()` method. oneCCL event must be associated with oneCCL communication operation.

CCL_ZE_LIBRARY_PATH

Syntax

```
CCL_ZE_LIBRARY_PATH=<value>
```

Arguments

<value>	Description
PATH/NAME	Specify the name and full path to the <code>Level-Zero</code> library for dynamic loading by oneCCL.

Description

Set this environment variable to specify the name and full path to `Level-Zero` library. The path should be absolute and validated. Set this variable if `Level-Zero` is not located in the default path. By default oneCCL uses `libze_loader.so` name for dynamic loading.

Point-To-Point Operations

CCL_RECV

Syntax

```
CCL_RECV=<value>
```

Arguments

<value>	Description
direct	Based on the MPI*/OFI* transport layer.
topo	Uses XeLinks across GPUs in a multi-GPU node. Default for GPU buffers.
offload	Based on the MPI*/OFI* transport layer and GPU RDMA when supported by the hardware.

CCL_SEND

Syntax

```
CCL_SEND=<value>
```

Arguments

<value>	Description
direct	Based on the MPI*/OFI* transport layer.
topo	Uses XeLinks across GPUs in a multi-GPU node. Default for GPU buffers.
offload	Based on the MPI*/OFI* transport layer and GPU RDMA when supported by the hardware.

oneCCL Benchmark User Guide

The oneCCL benchmark provides performance measurements for the collective operations in oneCCL, such as:

- allreduce
- reduce
- allgather
- alltoall
- alltoallv
- reduce-scatter
- broadcast

The benchmark is distributed with the oneCCL package. You can find it in the examples directory within the oneCCL installation path.

Build oneCCL Benchmark

CPU-Only

To build the benchmark, complete the following steps:

1. Configure your environment. Source the installed oneCCL library for the CPU-only support:

```
source <ccl installation dir>/ccl/latest/env/vars.sh --ccl-configuration=cpu
```

2. Navigate to <oneCCL install location>/share/doc/ccl/examples

3. Build the benchmark with the following command:

```
cmake -S . -B build -DCMAKE_INSTALL_PREFIX=$(pwd)/build/_install && cmake --build build -j $  
(nproc) -t install
```

CPU-GPU

1. Configure your environment.

- Source the Intel(R) oneAPI DPC++/C++ Compiler. See the [documentation](#) for the instructions.
- Source the installed oneCCL library for the CPU-GPU support:

```
source <ccl installation dir>/ccl/latest/env/vars.sh --ccl-configuration=cpu_gpu_dpcpp
```

2. Navigate to <oneCCL install location>/share/doc/ccl/examples.

3. Build the SYCL benchmark with the following command:

```
cmake -S . -B build -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx -DCOMPUTE_BACKEND=dpcpp -DCMAKE_INSTALL_PREFIX=$(pwd)/build/_install && cmake --build build -j $(nproc) -t install
```

Run oneCCL Benchmark

To run the benchmark, use the following command:

```
mpirun -np <N> -ppn <P> benchmark [arguments]
```

Where:

- **N** is the overall number of processes
- **P** is the number of processes within a node

The benchmark reports:

- `#bytes` - the message size in the number of bytes
- `#repetitions` - the number of iterations
- `t_min` - the average time across iterations of the fastest process in each iteration
- `t_max` - the average time across iterations of the slowest process in each iteration
- `t_avg` - the average time across processes and iterations
- `stddev` - standard deviation
- `wait_t_avg` - the average wait time after the collective call returns and until it completes To enable, use the `-x` option.

Notice that `t_min`, `t_max`, and `t_avg` measure the total collective execution time. It means the timer starts before calling oneCCL API and ends once the collective completes. While `wait_t_avg` only measures the wait time. It means the timer starts after the collective API call returns control to the host/calling thread and ends once the collective completes. Thus, `wait_t_avg` does not include the time spent on the oneCCL API call, while `t_min`, `t_max`, and `t_avg` include that time. Time is reported in **µsec**.

Benchmark Arguments

To see the benchmark arguments, use the `--help` argument.

The benchmark accepts the following arguments:

Option	Description	Default Value
<code>-b, --backend</code>	Specify the backend. The possible values are <code>host</code> and <code>sycl</code> . For a CPU-only build, the backend is automatically set to <code>host</code> , and only the <code>host</code> option is available. For a CPU-GPU build, <code>host</code> and <code>sycl</code> options are available, and <code>sycl</code> is the default value. The <code>host</code> value allocates buffers in the host (CPU) memory, while the <code>sycl</code> value allocates buffers in the device (GPU) memory.	<code>sycl</code>

Option	Description	Default Value
-i, --iters	Specify the number of iterations executed by the benchmark.	16
-w, --warmup_iters	Specify the number of the warmup iterations. It means the number of iterations the benchmark runs before starting the timing of the iterations specified with the -i argument.	16
-j, --iter_policy	Specify the iteration policy. Possible values are <code>off</code> and <code>auto</code> . When the iteration policy is <code>off</code> , the number of iterations is the same across the message sizes. When the iteration policy is <code>auto</code> , the number of iterations reduces based on the message size of the collective operation.	auto
-n, --buf_count	Specify the number of collective operations the benchmark calls in each iteration. Each collective uses different <code>send</code> and <code>receive</code> buffers. The explicit wait calls are placed for each collective after all of them are called.	1
-f, --min_elem_count	Specify the minimum number of elements used for the collective.	1
-t, --max_elem_count	Specify the maximum number of elements used for the collective.	128
-y, --elem_counts	Specify a list with the number of elements used for the collective, , such as [-y 4, 8, 32, 131072].	[1, 2, 4, 8, 16, 32, 64, 128]
-c, --check	Check for correctness. The possible values are <code>off</code> (disable checking), <code>last</code> (check the last iteration), and <code>all</code> (check all the iterations).	last
-p, --cache	Specify whether to use persistent collectives (p=1) or not (p=0).	1
	<hr/> <p>NOTE A collective is persistent when the same collective is called with the same parameters multiple times. OneCCL generates a schedule for each collective it runs and can apply optimizations when persistent collectives are used. It means the schedule is generated once and reused across the subsequent invocations, saving the time to generate the schedule.</p> <hr/>	
-q, --inplace	Specify for oneCCL to use in-place (1) or out-of-place (0) buffers. With the in-place buffers, the send and receive buffers used by the collective are the same. With the out-of-place, the buffers are different.	0

Option	Description	Default Value
-a, --sycl_dev_type	Specify the type of the SYCL device. The possible values are <code>host</code> , <code>cpu</code> , and <code>gpu</code> .	<code>gpu</code>
-g, --sycl_root_dev	Specify to use the root devices (0) and sub-devices (1).	0
-m, --sycl_mem_type	Specify the type of SYCL memory. The possible values are <code>usm</code> (unified shared memory) and <code>buf</code> (buffers).	<code>usm</code>
-u, --sycl_usm_type	Specify the type of SYCL device. The possible values are <code>device</code> or <code>shared</code> .	<code>device</code>
-e, --sycl_queue_type	Specify the type of the SYCL queue. The possible values are <code>in_order</code> and <code>out_order</code> .	<code>out_order</code>
-l, --coll	Specify the collective to run. Accept a comma-separated list, without whitespace characters, of collectives to run. The available collectives are <code>allreduce</code> , <code>reduce</code> , <code>alltoallv</code> , <code>alltoall</code> , <code>allgatherv</code> , <code>reduce-scatter</code> , and <code>broadcast</code> .	<code>allreduce</code>
-d, --dtype	Specify the datatype. Accept a comma-separated list, without whitespace characters, of datatypes to benchmark. The available types are <code>int8</code> , <code>int32</code> , <code>int64</code> , <code>uint64</code> , <code>float16</code> , <code>float32</code> , and <code>bfloat16</code> .	<code>float32</code>
-r, --reduction	Specify the type of the reduction. Accept a comma-separated list, without whitespace characters, of the reduction operations to run. The available operations are <code>sum</code> , <code>prod</code> , <code>min</code> , and <code>max</code> .	<code>sum</code>
-o, --csv_filepath	Specify to store the output in the specified CSV file. User specifies the <code>csv_filepath/file_to_store</code> CSV-formatted data into	
-x, --ext	Specify to show the additional information. The possible values are <code>off</code> , <code>auto</code> , and <code>on</code> . With <code>on</code> , it also displays the average wait time.	<code>auto</code>
-h, --help	Show all of the supported options.	

NOTE The `-t` and `-f` options specify the count in number of elements, so the total number of bytes is obtained by multiplying the number of elements by the number of bytes of the data type the collective uses. For instance, with `-f 128` and `fp32` datatype, the total amount of bytes is 512 (128 element count * 4 bytes FP32). The benchmark runs and reports time for message sizes that correspond to the `-t` and `-f` arguments and all message sizes that are powers of two in between these two numbers.

Example

GPU

The following example shows how to run the benchmark with the GPU buffers:

```
mpirun -n <N> -ppn <P> benchmark -a gpu -m usm -u device -l allreduce -i 20 -j off -f 1024 -t 67108864 -d float32 -p 0 -e in_order
```

The above command runs:

- The `allreduce` collective operation
- With a total of `N` processes
- With `P` processes per node allocating the memory in the GPU
- Using SYCL Unified Shared Memory (USM) of the device type
- 20 iterations
- With the element count from 1024 to 67108864 (the benchmark runs with all the powers of two in that range) of float32 datatype, assuming the collective is not persistent and using a SYCL in-order queue

Similar for `allreduce` and `reduce_scatter`:

```
mpirun -n <N> -ppn <P> benchmark -a gpu -m usm -u device -l allreduce,reduce_scatter -i 20 -j off -f 1024 -t 67108864 -d float32 -p 0 -e in_order
```

NOTE In this case, the time reported is the accumulated time corresponding to the execution time of `allreduce` and `reduce_scatter`.

CPU

```
mpirun -b host -n <N> -ppn <P> benchmark -l allreduce -i 20 -j off -f 1024 -t 67108864 -d float32 -p 0
```

The above command specifies to run

- The `allreduce` collective operation
- With a total of `N` processes
- With `P` processes per node
- 20 iterations
- With the element count from 1024 to 67108864 (the benchmark runs with all the powers of two in that range) of float32 datatype, assuming the collective is not persistent

Similar for `allreduce` and `reduce_scatter`:

```
mpirun -b host -n <N> -ppn <P> benchmark -l allreduce,reduce_scatter -i 20 -j off -f 1024 -t 67108864 -d float32 -p 0
```

Notices and Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

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.