Intel® AVX-512 - Instruction Set for Packet Processing

Authors

Ray Kinsella

Chris MacNamara

Georgii Tkachuk

Reviewer

Konstantin Ananyev

1 Introduction

Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instruction set is a powerful addition to the packet processing toolkit. As Intel's latest generation of SIMD instruction set, Intel® AVX-512 (also known as AVX-512) is a game changer, doubling register width, doubling the number of available registers, and generally offering a more flexible instruction set compared to its predecessors. Intel® AVX-512 has been available since the 1st Generation Intel® Xeon® Scalable processors and is now optimized in the latest 3rd Generation Intel® Xeon® Scalable processor and the Intel® Xeon® D processor, with compelling performance benefits. The performance, configurations, and feature set may vary for the Intel® Xeon® D processor.

This paper is the first in a series of white papers focusing on how to write packet processing software using the Intel® AVX-512 instruction set. It provides a brief overview of the Intel® AVX-512 instruction set and describes the microarchitecture optimizations for the instruction set in the latest 3rd Generation Intel® Xeon® Scalable processors.

The next paper in the series describes examples of where Intel® AVX-512 is being used in packet processing, including performance benefits using Intel® AVX-512 optimizations that demonstrate performance gains in excess of 300% in microbenchmarks[1]. An executive summary of these papers is also available.

This technology guide is intended for organizations developing or deploying packet processing software on the latest 3rd Generation Intel® Xeon® Scalable processors.

It is a part of the Network Transformation Experience Kit, which is available at https://networkbuilders.intel.com/network-technologies/network-transformation-exp-kits.

[1] See section 4.4.1 for details. See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

Table of Contents

Figures

Tables

Document Revision History

REVISION	DATE	DESCRIPTION
001	February 2021	Initial release.
002	April 2021	Benchmark data updated. Revised the document for public release to Intel® Network Builders.
003	February 2022	Added information regarding Intel® Xeon® D processor in the Introduction section.

1.1 Terminology

ABBREVIATION	DESCRIPTION
AVX	Advanced Vector Extensions
DPDK	Data Plane Development Kit (dpdk.org)
FD.io	Fata Data I/O, an umbrella project for Open Source network projects
FMA	Fused multiply add
HPC	High Performance Computing
IDS/IPS	Intrusion Detection Systems/Intrusion Prevention Systems
ISA	Instruction Set Architecture
SIMD	Single Instruction Multiple Data: a term used to describe vector instructions sets such as Intel® SSE and Intel® AVX
SSE	Streaming SIMD Extensions (predecessor to AVX)
TDP	Thermal Design Power
VPP	FD.io Vector Packet Processing, an Open Source networking stack (part of FD.io)

1.2 Reference Documentation

REFERENCE	SOURCE
Intel® 64 and IA-32 Architectures Optimization Reference Manual	https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf
Intel® 64 and IA-32 Architectures Software Developer Manuals	https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html
Intel® Xeon® Scalable Processors Family Specification Update	https://www.intel.in/content/www/in/en/processors/xeon/xeon-technical-resources.html
Second Generation Intel® Xeon® Scalable Processors Specification Update	https://www.intel.com/content/www/us/en/products/docs/processors/xeon/2nd-gen-xeon-scalable-spec-update.html
New 3rd Generation Intel® Xeon® Scalable Processor	https://hotchips.org/assets/program/conference/day1/HotChips2020_Server_Processors_Intel_Irma_ICX-CPU-final3.pdf
Intel® AVX-512 - Packet Processing with Intel® AVX-512 Instruction Set Solution Brief	https://networkbuilders.intel.com/solutionslibrary/intel-avx-512-packet-processing-with-intel-avx-512-instruction-set-solution-brief
Intel® AVX-512 Writing Packet Processing Software with Intel® AVX-512 Instruction Set Technology Guide	https://networkbuilders.intel.com/solutionslibrary/intel-avx-512-writing-packet-processing-software-with-intel-avx-512-instruction-set-technology-guide
Intel® Intrinsics Guide	https://software.intel.com/sites/landingpage/IntrinsicsGuide/

2 Overview

While SIMD instructions are perhaps best known for their use in domains such as HPC, image processing, and artificial intelligence, what may be less appreciated is SIMD's applicability to packet processing. In fact, SIMD instructions are already commonly used in packet processing applications to speed up compute-bound algorithms. Intel® AVX-512, Intel's very latest SIMD instruction set, is a richer and more flexible instruction set compared to its predecessors, introducing new concepts such as masked operations, broadcasts, instruction set extensions, register modes, and bitwise ternary instructions.

This technology guide has two sections that are structured as follows:

- The first section is a high-level instruction set overview, describing some of the new concepts implemented in the Intel® AVX-512 instruction set.
- The second section has details on how the latest Intel® Xeon® Scalable processors have been optimized for Intel® AVX-512 instruction set.

3 Intel® AVX-512 Instruction Set

This section provides some basic grounding in Intel® AVX-512 instructions, as well as some of the new concepts and improvements that are realized in the instruction set. The subsections explain the increase in register sizes, the concept of mask registers and instruction set extensions, describe vector length instruction set extension, and finally describe SIMD ternary instructions--a new concept available with AVX-512 instructions.

3.1 Register Width and Count

As the name suggests, Intel® AVX-512 expands the width of existing Intel® Advanced Vector Extensions 2 (Intel® AVX2) SIMD registers from 256 bits to 512 bits, and by this expansion it doubles the amount of work a developer can accomplish in a single operation. To understand this concept in detail, consider the example of 64-bit vector addition

Table 1. 64-bit Vector Addition

INSTRUCTION SET	C INTRINSIC FORM OF INSTRUCTION	PACKED 64-BIT INTEGERS	ASSEMBLER FORM
Intel® SSE	__m128i _mm_add_epi64 (__m128i a, __m128i b)	2 (128/64)	paddq
Intel® AVX2	__m256i _mm256_add_epi64 (__m256i a, __m256i b)	4 (256/64)	vpaddq
Intel® AVX-512	__m512i _mm512_add_epi64 (__m512i a, __m512i b)	8 (512/64)	vpaddq

The Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instruction _mm_add_epi64 (**paddq**) adds two 128-bit registers **a** and **b** and returns the result in a single 128-bit integer register. Each register contains two packed 64-bit integers, thereby completing two distinct 64-bit additions in a single instruction.

Intel® AVX2 added the instruction _mm256_add_epi64 (**vpaddq**), that expands integer vector addition to 256-bit registers. This expansion doubles the work accomplished in a single instruction when compared to SSE2 instructions, completing four distinct 64-bit additions in a single instruction.

Now Intel® AVX-512 again doubles the number of packed-integer operations performed in a single instruction. Intel® AVX-512 adds a 512-bit variant in the form of _mm512_add_epi64 (**vpaddq**) instruction, expanding the packed-integer addition to 512-bit registers, allowing completion of eight distinct 64-bit additions in a single instruction.

Figure 1 is a graphical representation of the vector addition instructions described earlier. The figure shows both the register size and the amount of work performed doubling with each instruction set generation, represented by the packed integers **a0** to **a7.**

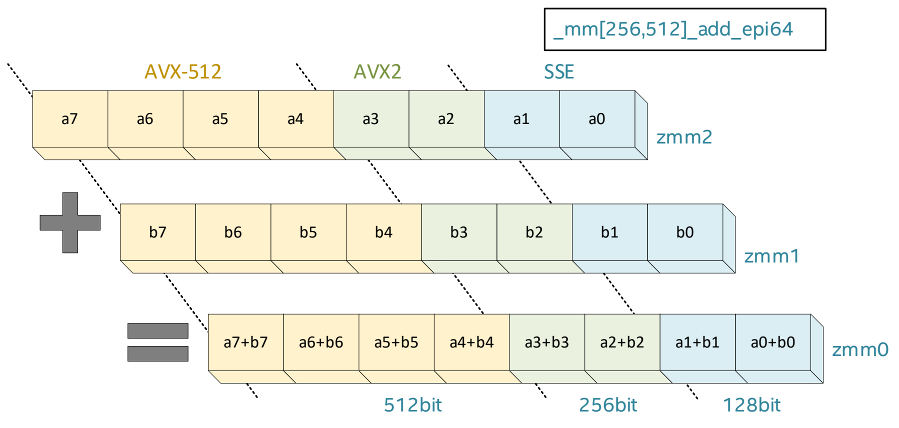


Figure 1. Vector Addition with Intel® SSE, Intel® AVX2, and Intel® AVX-512 Instruction Sets

It is worth mentioning here that the Intel® AVX2 instruction set (AVX2) also introduced three operand non-destructive instructions, featuring a destination operand in addition to the two source operands. This was an improvement in AVX2 over Intel® SSE instructions in that it was no longer necessary for the operation to overwrite one of the source operands. Intel® AVX-512 instructions are also non-destructive in this way.

AVX-512 instructions also expand the number of available SIMD registers from 16 to 32, doubling the number of values that can be concurrently held in registers.

3.1.1 Packed Data Types

In the Intel® AVX-512 instruction set, each intrinsic's suffix is used to indicate how the operands are treated, adopting the same instruction naming conventions as its predecessors. The *pi* suffix indicates packed integer operands, *pu* suffix indicates packed unsigned integer operands, the *pd* suffix indicates packed double-precision floating-point operands, and the *ps* suffix indicates packed single-precision floating-point operands.

The C data type of the intrinsic operands is declared as either __m512i, indicating packed integers, or __m512d, indicating packed double precision floating points, or __m512, indicating packed single precision floating points. It follows that AVX-512 intrinsics with a *ps* suffix would therefore have operands with a data type of __m512.

To understand this concept in detail, consider the example of vector addition shown in Table 2.

Table 2. Packed Data Type Vector Addition

INTRINSIC SUFFIX	C INTRINSIC FORM OF INSTRUCTION	PACKED DATA TYPE
_epi64	__m512i _mm512_add_**epi64** (__m512i a, __m512i b)	8 x 64-bit Integer
_epi32	__m512i _mm512_add_**epi32** (__m512i a, __m512i b)	16 x 32-bit Integer
_epi16	__m512i _mm512_add_**epi16** (__m512i a, __m512i b)	32 x 16-bit Integer
_epi8	__m512i _mm512_add_**epi8** (__m512i a, __m512i b)	64 x 8-bit Integer
_pd	__m512d _mm512_add_**pd** (__m512d a, __m512d b)	8 x 64-bit double precision floating point
_ps	__m512 _mm512_add_**ps** (__m512 a, __m512 b)	16 x 32-bit single precision floating point

In Table 2, all the intrinsics are variants of the 512-bit vector addition, with each intrinsic suffix indicating how the operands are treated during the addition. For example, the *epi64* suffix indicates that the intrinsic _mm512_add_epi64 will treat operands **a** and **b** as packed 64-bit integers, with eight 64-bit integers packing into a 512-bit register. Similarly, the *epi32* suffix indicates that the intrinsic _mm512_add_epi32 will treat operands **a** and **b** as packed 32-bit integers, with sixteen 32-bit integers packing into a single 512-bit register, and so on.

3.2 Mask Registers and Operations

Intel® AVX-512 introduces masked operations to give developers much more fine-grained control over the behavior of SIMD instructions. Masked instructions and mask registers enable developers to indicate which packed values should be acted upon or ignored for a given operation, where each bit position in a mask register corresponds to a packed value.

The result of a masked packed 64-bit integer arithmetic operation is controlled by an 8-bit mask register. Each bit position in the 8-bit mask corresponds with one of the eight packed 64-bit values in a single 512-bit register (refer to Figure 2). Similarly, 32-, 16-, and 8-bit integer arithmetic are controlled by 16-, 32-, and 64-bit mask values respectively.

When bits are set in the mask register, the arithmetic operation on the corresponding packed values completes when the instruction retires. However, when bits are cleared, another value is returned in place of the result of the arithmetic operation. In such a case, the returned value may be zero or a read from a source register, depending on whether the *maskz* or *mask* prefix variants of the instruction are used.

To understand this concept in detail, consider the example of masked 64-bit vector addition.

Table 3. 64-bit Masked Vector Addition

INSTRUCTION SET	C INTRINSIC FORM OF INSTRUCTION
Intel® SSE4.2	__m128i _mm_add_epi64 (__m128i a, __m128i b)
Intel® AVX2	__m256i _mm256_add_epi64 (__m256i a, __m256i b)
Intel® AVX-512	__m512i _mm512_add_epi64 (__m512i a, __m512i b)
Intel® AVX-512	__m512i _mm512_**mask**_add_epi64 (__m512i src, __mmask8 k, __m512i a, __m512i b)
Intel® AVX-512	__m512i _mm512_**maskz**_add_epi64 (__mmask8 k, __m512i a, __m512i b)

In Table 3, notice that AVX-512 instructions described in the previous section are included along with two new variants that have no equivalents in the previous instruction set generations:

- _mm512_mask_add_epi64: The mask register **k** controls the generation of the packed return value. For each packed 64-bit integer, it indicates if the value must be read from the **src** register or must be the result of the packed 64-bit addition of **a** and **b.**

- _mm512_maskz_add_epi64: Similarly, here too the mask register **k** indicates that the return value must be zero or the result of the packed addition of **a** and **b**.

Figure 2 is a graphical representation of the intrinsic _mm512_mask_add_epi64. It shows a mask register indicating, via single-bit flags, whether each packed 64-bit integer of the return value is read from corresponding 64-bit values in a *source* register or is the result of the addition of 64-bit values in **a** and **b.**

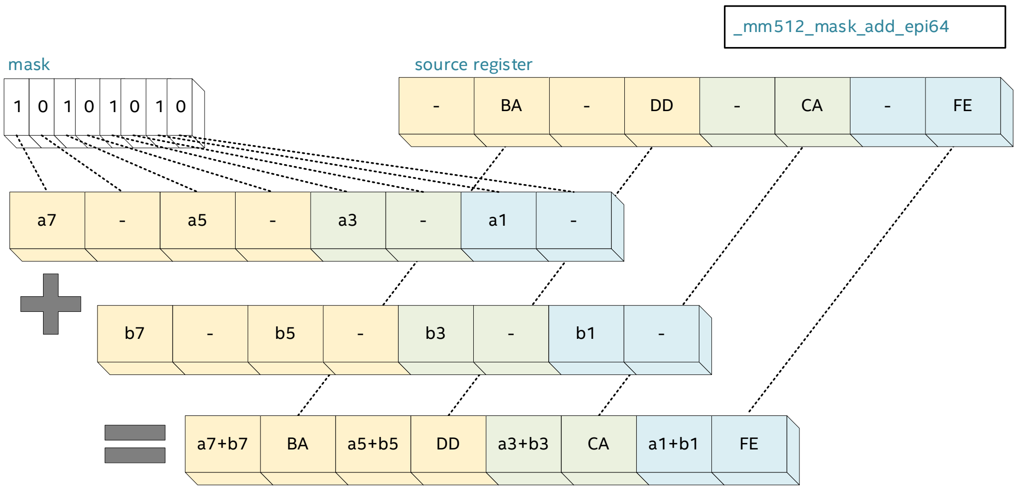


Figure 2. Masked Vector Addition, Mask, and Source Register

Figure 3 is a graphical representation of the intrinsic _mm512_maskz_add_epi64. It shows a mask register indicating, via single-bit flags, whether each packed 64-bit integer of the return value is zero or is the result of the addition of corresponding 64-bit values in **a** and **b.**

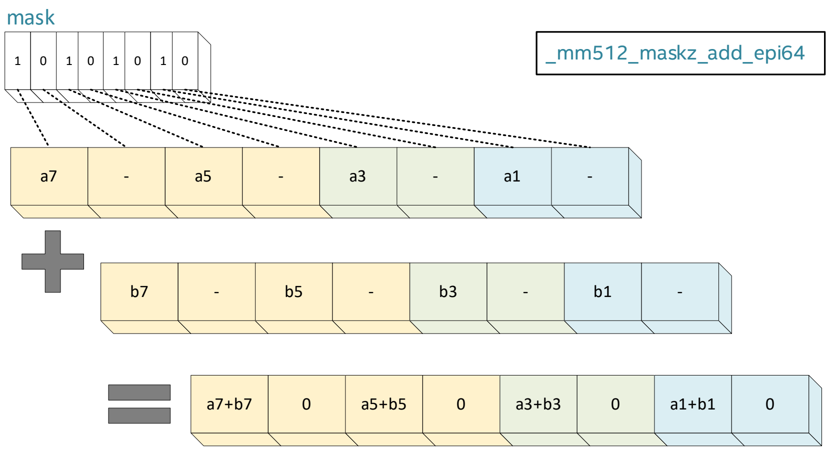


Figure 3. Masked Vector Addition and Mask Register

3.2.1 Generating Masks

Intel® AVX-512 introduces a range of new instructions to generate masks from packed values. The intrinsics for these instructions all share the suffix _mask and return an 8-bit to 64-bit value depending on the variant of the instruction. Some examples are shown in Table 4.

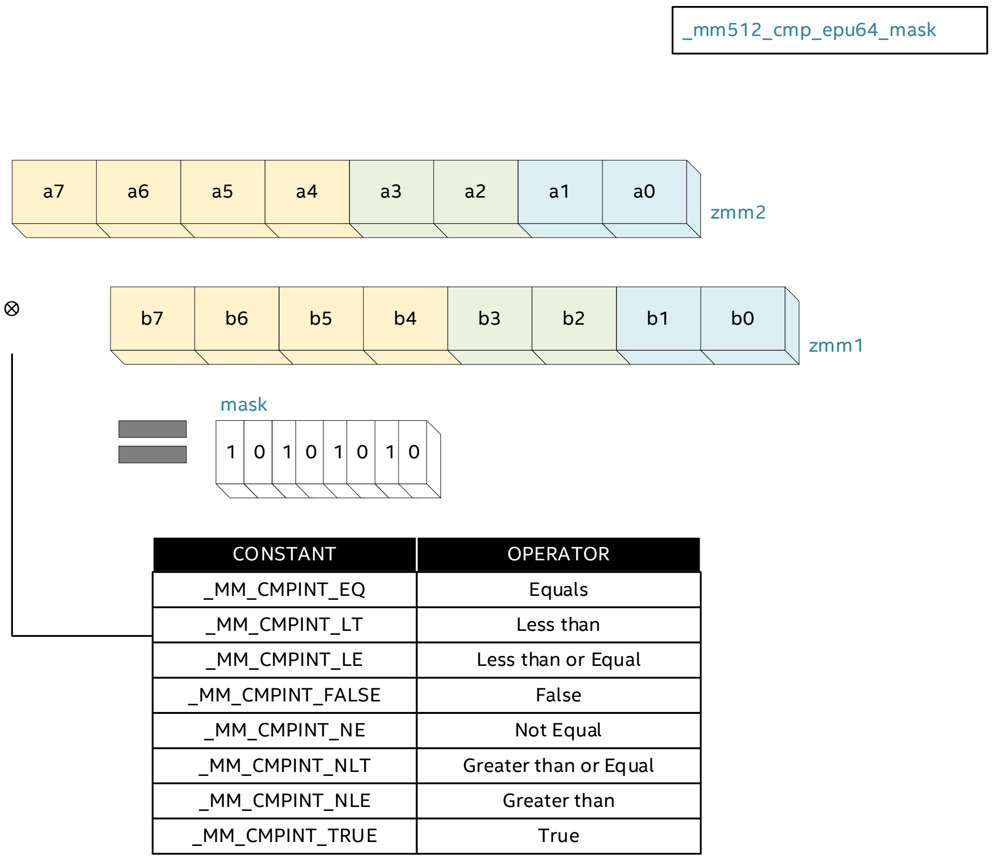
Table 4. Generation of 8-bit Mask from 64-bit Packed Values

INSTRUCTION SET	C INTRINSIC FORM OF INSTRUCTION
Intel® AVX-512	__mmask8 _mm512_movepi64_**mask** (__m512i a)
Intel® AVX-512	__mmask8 _mm512_cmp_epu64_**mask** (__m512i a, __m512i b, _MM_CMPINT_ENUM imm8)

Details of the two AVX-512 intrinsics listed in Table 4 are given below:

- _mm512_movepi64_mask: This instruction sets each bit of the returned 8-bit mask, based on the most significant bit of the corresponding 8-packed 64-bit integer in a 512-bit register **a.**
- _mm512_cmp_epu64_mask: This instruction compares the packed unsigned 64-bit integers in **a** and **b** based on the comparison operand specified in **imm8** and stores the results in an 8-bit mask register.

Figure 4 is a graphical representation of the _mm512_cmp_epu64_mask intrinsic. It generates a mask based on a comparison of two packed 64-bit values **a** and **b.** The type of comparison is controlled by the immediate operator **imm8** with the range of possible values as shown in the table within the figure.



CONSTANT	OPERATOR
_MM_CMPINT_EQ	Equals
_MM_CMPINT_LT	Less than
_MM_CMPINT_LE	Less than or Equal
_MM_CMPINT_FALSE	False
_MM_CMPINT_NE	Not Equal
_MM_CMPINT_NLT	Greater than or Equal
_MM_CMPINT_NLE	Greater than
_MM_CMPINT_TRUE	True

Figure 4. Generating 8-bit Masks from 64-bit Packed Values

The Intel® AVX-512 instruction set also has a number of instructions for manipulating masks, all of which share the prefix _*k*. Examples of 64-bit mask manipulation intrinsics are: _knot_mask64 (not), _kand_mask64 (and), _kor_mask64 (or), _kxor_mask64 (xor) _kshiftri_mask64 (shift right), and_kshiftli_mask64 (shift left).

3.3 Extensions

Intel® AVX-512 is a family of instruction set extensions whose first member is AVX-512 Foundation (AVX512F). AVX512F extended AVX/AVX2 instructions to support 512-bit operands. It also added masked variants of these instructions, as well as instructions for creating and manipulating masks. A series of extensions follows thereafter on top of AVX512F, each of which added a distinct set of new functionalities.

For example, all four of five AVX-512 instructions discussed up to this point—_mm512_add_epi64, _mm512_mask_add_epi64, _mm512_maskz_add_epi64, and _mm512_cmp_epu64_mask — were added as part of AVX512F. The other instruction described—_mm512_movepi64_mask—was added as part of the AVX512DQ extension. It is important to note that AVX-512 is therefore not a single instruction set but is, instead, an improved and extended set with each new generation of Intel Xeon Scalable processors.

- The 1st Generation Intel® Xeon® Scalable processor for servers introduced the following extensions:
  – AVX-512 Foundation (AVX512F)
  – Conflict Detection (AVX512CD)
  – Vector Length (AVX512VL)
  – Doubleword and Quadword (AVX512DQ)

  - Byte and Word (AVX512BW)
- The 2nd Generation Intel® Xeon® Scalable processors added the following extension:
  - Vector Neural Network (AVX-512_VNNI)
- The 3rd Generation Intel® Xeon® Scalable processors adds the following extensions:
  - Vector Byte Manipulation (AVX512_VBMI and AVX512_VBMI2),
  - Vector Popcount Doubleword and Quadword (VPOPCNTDQ),
  - Bit Algorithms (BITALG), and
  - Vector AES (VAES) and Galois field (GFNI)

On most Linux* systems, to determine the extensions supported by a given Intel® microprocessor, parse the `proc` filesystem's `cpuinfo` node, `/proc/cpuinfo`, as shown below. The example below is of instruction set extensions supported by the Intel® Xeon® Gold 6230N CPU microprocessor.

```
$ grep "model name" /proc/cpuinfo | uniq
model name      : Intel(R) Xeon(R) Gold 6230N CPU @ 2.30GHz
$ egrep "^flags" /proc/cpuinfo | uniq | egrep -o "avx512\S*"
avx512f
avx512dq
avx512cd
avx512bw
avx512vl
avx512_vnni
```

3.3.1 Vector Length Extension

The Vector Length (AVX-512VL) extension adds a significant number of instructions to the Intel® AVX-512 instruction set and provides a good example of an extension. AVX512VL adds new instructions for length vectors smaller than 512-bit, such that all instructions added in the AVX512F instruction set now also support 256-bit and 128-bit operands, in effect adding masked variants of AVX2 and SSE instructions. Developers thus get an opportunity to enhance existing SIMD code with new Intel® AVX-512 capabilities.

To understand this concept in detail, consider the example of a 64-bit vector addition.

Table 5. 64-bit Vector Length Extensions and Vector Addition

INSTRUCTION SET	C INTRINSIC FORM OF INSTRUCTION	REGISTER SIZE
Intel® SSE	__m128i _mm_add_epi64 (__m128i a, __m128i b)	128
Intel® AVX2	__m256i _mm256_add_epi64 (__m256i a, __m256i b)	256
AVX512F	__m512i _mm512_add_epi64 (__m512i a, __m512i b)	512
AVX512F	__m512i _mm512_mask_add_epi64 (__m512i src, __mmask8 k, __m512i a, __m512i b)	512
AVX512F	__m512i _mm512_maskz_add_epi64 (__mmask8 k, __m512i a, __m512i b)	512
AVX512VL	__m128i _mm_mask_add_epi64 (__m128i src, __mmask16 k, __m128i a, __m128i b)	128
AVX512VL	__m128i _mm_maskz_add_epi64(__mmask16 k, __m128i a, __m128i b)	128
AVX512VL	__m256i _mm256_mask_add_ep64 (__m256i src, __mmask32 k, __m256i a, __m256i b)	256
AVX512VL	__m256i _mm256_maskz_add_epi64 (__mmask32 k, __m256i a, __m256i b)	256

The AVX512 vector length extensions add four new intrinsics that have no equivalent in previous instruction set generations:

- _mm_mask_add_epi64 and _mm_maskz_add_epi64: These intrinsics are similar to _mm512_mask_add_epi64 and _mm512_maskz_add_epi64 intrinsics described in section 3.2; they are masked variants of the 128-bit SSE _mm_add_epi64 intrinsic.
- _mm256_mask_add_epi64 and _mm256_maskz_add_epi64: These intrinsics are similar to _mm_mask_add_epi64 and _mm_maskz_add_epi64 intrinsics described in section 3.2; they are masked variants of the 256-bit AVX2 _mm256_add_epi64 intrinsic.

Figure 5 is a graphical representation of the _mm_maskz_add_epi64, _mm256_maskz_add_epi64, and _mm512_maskz_add_epi64 intrinsics. It shows masked vector addition extended to support 128-bit, 256-bit, and 512-bit register sizes.

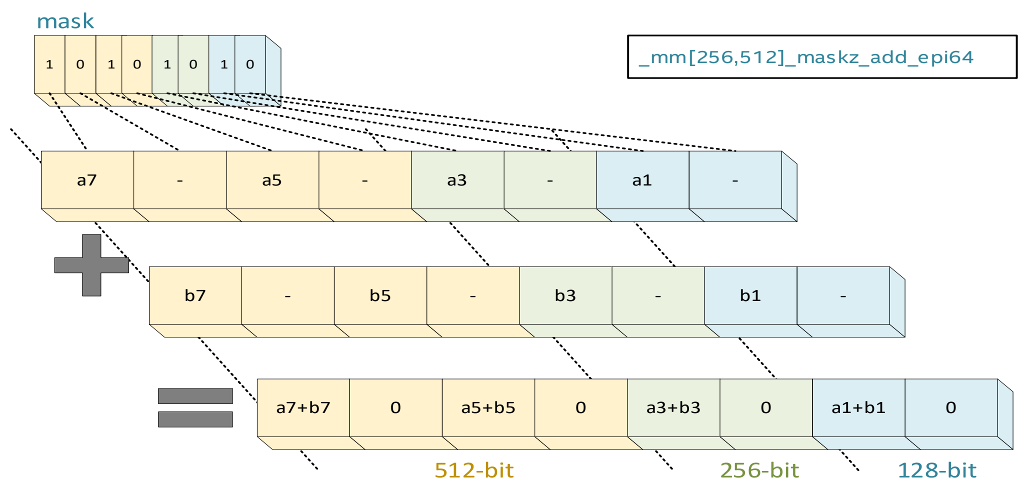


Figure 5. Packed 64-bit Addition with Vector Length Extensions

3.4 Ternary Instructions

Ternary instructions with three operands are a new concept in the Intel® AVX-512 instruction set that has no equivalent in the Intel® SSE or Intel® AVX2 instruction sets. The _mm*_ternarylogic* intrinsics perform two bitwise logical operations on three source operands concurrently and are available in *doubleword* and *quadword* variants.

Performing a bitwise logical ternary operation is a common pattern in software development (refer to Figure 6), where, for example, it is common to find a bitwise operation coupled with a mask operation. The _mm*_ternarylogic* intrinsic is a packed version of these operations, performing the same and similar operations on packed values in a single operation.

```
uint64_t dst, a, b c;
…

dst = (a ^ b) & c;
```

Figure 6. Ternary Bitwise Operations

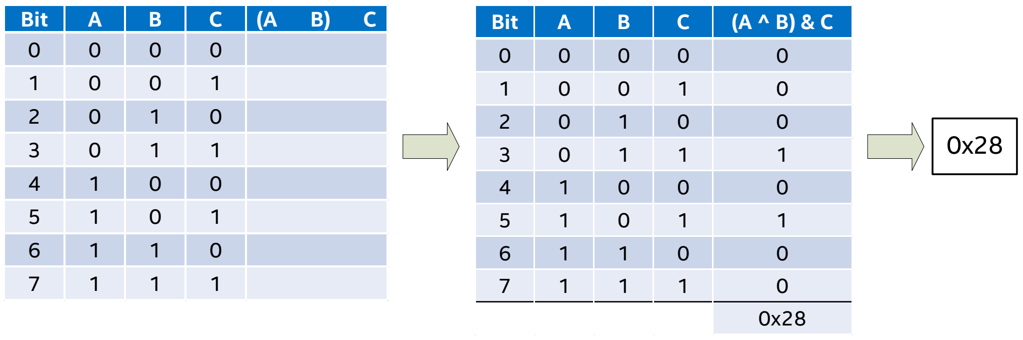
To understand this concept in detail, consider the example of a 64-bit ternary logic.

Table 6. 64-bit Ternary Logic

INSTRUCTION SET	C INTRINSIC FORM OF INSTRUCTION
AVX-512	__m512i _mm512_ternarylogic_epi64 (__m512i a, __m512i b, __m512i c, int imm8)
AVX-512	__m512i _mm512_mask_ternarylogic_epi64 (__m512i src, __mmask8 k, __m512i a, __m512i b, int imm8)
AVX-512	__m512i _mm512_maskz_ternarylogic_epi64 (__mmask8 k, __m512i a, __m512i b, __m512i c, int imm8)

The _mm512_ternarylogic_epi64 intrinsic uses the argument **imm8** to indicate how the input vectors **a, b**, and **c** must be treated, that is, which bitwise operations must be performed. The value for **imm8** is constructed using a truth table, an example of which is shown in Figure 7.

To complete the truth table, substitute the symbol ⨂ with whichever bitwise operation you want to perform on the operands. Record the value that is generated, given the inputs A, B, and C, in the final column. Repeat this for all 8-bits, which is then combined to become the **imm8** value.

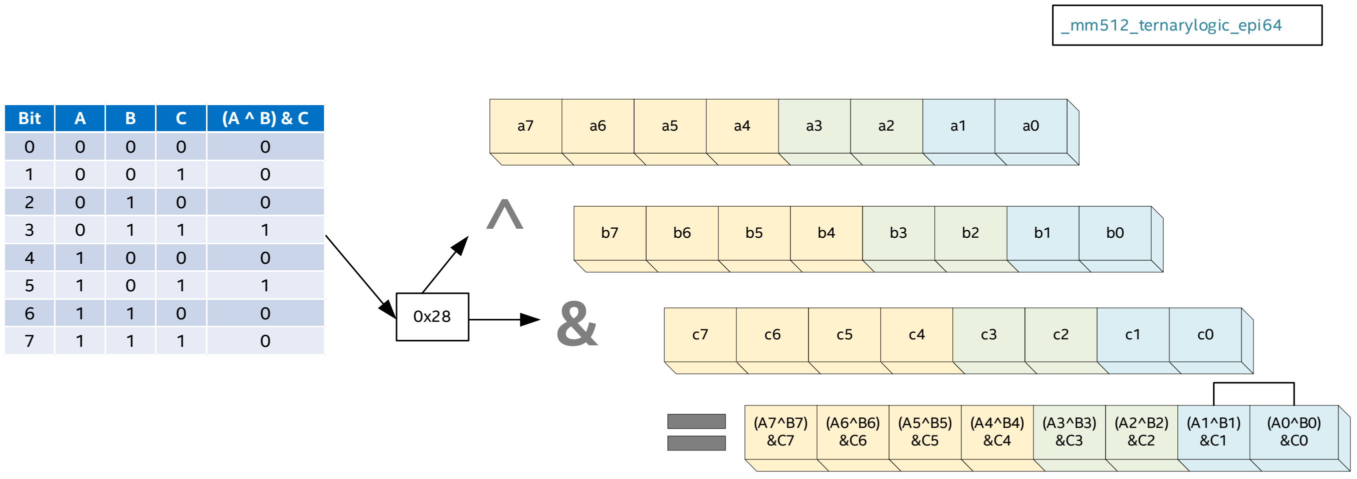


Bit	A	B	C	(A B) C
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	

Bit	A	B	C	(A ^ B) & C
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0
				0x28

Figure 7. Ternary Instruction Truth Table

Figure 8 is a graphical representation of the intrinsic _mm512_ternarylogic_epi64. It shows how to construct the truth table to generate the **imm8** value, and how the result of the intrinsic is generated from the result of the two bitwise operations.



Bit	A	B	C	(A ^ B) & C
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

Figure 8. Ternary Operation with Truth Table

3.5 Additional Concepts

Some additional Intel® AVX-512 enhancements not covered in this document are the new compression and expansion instructions, improved range of conversion instructions, embedded broadcasts, and the new permutation and shuffle instructions. In addition, other AVX-512 concepts worth exploring are in-lane compared to cross-lane instructions.

Additional information on the Intel® AVX-512 instruction set can be found in these sources:

REFERENCE	CHAPTER
Intel® 64 and IA-32 Architectures Optimization Reference Manual	17
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture	15
Intel® Intrinsics Guide	

4 Intel® AVX-512 and Intel® Xeon® Scalable Processor Microarchitecture

This section explores how the Intel® Xeon® Scalable processor microarchitecture is optimized for the Intel® AVX-512 instruction set. The subsections describe the microarchitectural changes—implemented from the 1st Generation Intel® Xeon® Scalable processor through subsequent generations of the processor—to efficiently run Intel® AVX-512 instructions, discuss power utilization of Intel® AVX-512 instructions, and describe how power utilization affects packet processing applications.

4.1 Instruction Throughput

The 1st Generation Intel® Xeon® Scalable processor for servers doubled SIMD execution ports width to 512 bits to accommodate Intel® AVX-512 instructions. It features up to two 512-bit wide execution ports capable of retiring AVX-512 instructions. These two execution ports have the potential to retire Intel® AVX-512 instructions concurrently yielding a maximum throughput of 1024 bits; that is, two AVX-512 instructions retiring in parallel producing 512 bits of data each.

The green stars in Figure 9, represent new features in the 1st Generation Intel® Xeon® Scalable processor microarchitecture for servers compared to the microarchitecture for the client, including a 1 MB L2 cache and an additional AVX-512 execution port. As shown in Figure 9, when executing AVX-512 instructions, Port 0 and Port 1 fuse into a single execution port. Intel® AVX-512 instructions may also run on Port 5, depending on the type of instruction. Note that shuffles, for example, execute only on Port 5.

The 1st Generation Intel® Xeon® Scalable processor microarchitecture for servers features two 512-bit wide **load** ports and a single 512-bit **store** port, enabling two 512-bit **load** and a single 512-bit **store** operations to complete concurrently.

In comparison, Ports 0, 1, and 5 may be used to run Intel® AVX2 or Intel® SSE instructions. The maximum throughput of AVX2 is therefore 768 bits; that is, three AVX2 instructions retiring concurrently yielding 256 bits of data. AVX2 instructions have the advantage of an additional execution port compared to AVX-512, as Port 0 and Port 1 are unfused. However, AVX-512 still has the potential, under ideal conditions, to retire an additional 256 bits of data concurrently compared to AVX2.

The **load** and **store** ports on the 1st Generation Intel® Xeon® Scalable processor microarchitecture for servers may also be used for 256-bit loads and stores, with no adverse effects beyond that of the reduced load and store throughput with AVX2 when compared to AVX-512.

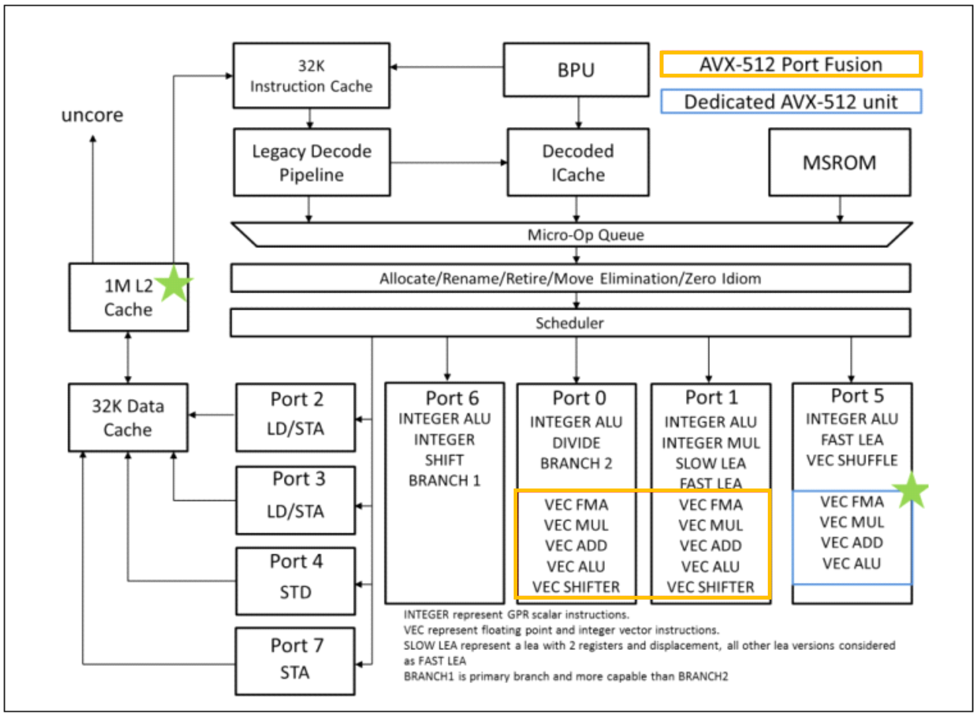

32K Instruction Cache
BPU
AVX-512 Port Fusion
Dedicated AVX-512 unit
uncore
Legacy Decode Pipeline
Decoded ICache
MSROM
Micro-Op Queue
1M L2 Cache
Allocate/Rename/Retire/Move Elimination/Zero Idiom
Scheduler
32K Data Cache
Port 2 LD/STA
Port 3 LD/STA
Port 4 STD
Port 7 STA
Port 6 INTEGER ALU INTEGER SHIFT BRANCH 1
Port 0 INTEGER ALU DIVIDE BRANCH 2 VEC FMA VEC MUL VEC ADD VEC ALU VEC SHIFTER
Port 1 INTEGER ALU INTEGER MUL SLOW LEA FAST LEA VEC FMA VEC MUL VEC ADD VEC ALU VEC SHIFTER
Port 5 INTEGER ALU FAST LEA VEC SHUFFLE VEC FMA VEC MUL VEC ADD VEC ALU
INTEGER represent GPR scalar instructions.
VEC represent floating point and integer vector instructions.
SLOW LEA represent a lea with 2 registers and displacement, all other lea versions considered as FAST LEA
BRANCH1 is primary branch and more capable than BRANCH2

Figure 9. Processor Core Pipeline Functionality of the 1st Generation Intel® Xeon® Scalable Processor Microarchitecture

For more information on the 1st Generation Intel® Xeon® Scalable processor microarchitecture for servers, see section 4.5.

4.2 Power

Power is an important consideration when developing software with Intel® AVX2 and Intel® AVX-512 instructions on Intel® Xeon® Scalable processors. The larger vector SIMD instructions may consume more power per core, with the power increase typically proportional to the performance increase. Hence, using more power typically results in more performance.

The 1st Generation Intel® Xeon® Scalable processor microarchitecture for servers dynamically manages the voltage based on the instruction set in use and this may result in the CPU selecting an appropriate frequency at which a given core runs. The selected frequency depends on several factors, including the instruction mix executing on the core and the number of other cores executing a similar instruction mix. The instruction mix is comprised of the type, width, and number of instructions, including vector instructions, that run over a given period of time.

The maximum Intel® Turbo Boost core frequency (P0n) achievable is influenced by the category of instructions executing on that core and the number of cores sharing that category. The more active cores in the shared category, the lower the frequency of those cores. These categories are called *power levels* and are described in Table 7.

There are three power levels in the Intel® Xeon® Scalable processor that the CPU may apply in non-power limited scenarios. When power is limited, the microprocessor may also choose to adjust the frequency of the cores to stay within its stated TDP (Thermal Design Power) limit.

Table 7. Maximum Intel® Turbo Boost Technology Core Frequency Levels

POWER LEVEL	FREQUENCY	INSTRUCTION SET MIX
Power Level 0 Intel® AVX2 Light	Maximum frequency	Scalar, SSE integer and floating-point, and AVX2 integer instructions, except for AVX2 integer multiplication and fused-multiply-add (FMA) instructions
Power Level 1 Intel® AVX2 Heavy/ AVX-512 Light	Generation & SKU dependent	AVX2 integer multiplication, FMA, and floating-point instructions. AVX-512 integer instructions, except for AVX-512 integer multiplication and FMA instructions
Power Level 2 Intel® AVX-512 Heavy	Reduced P0n (Turbo) frequency compared to Power Level 1	AVX-512 integer multiplication, FMA, and floating-point instructions

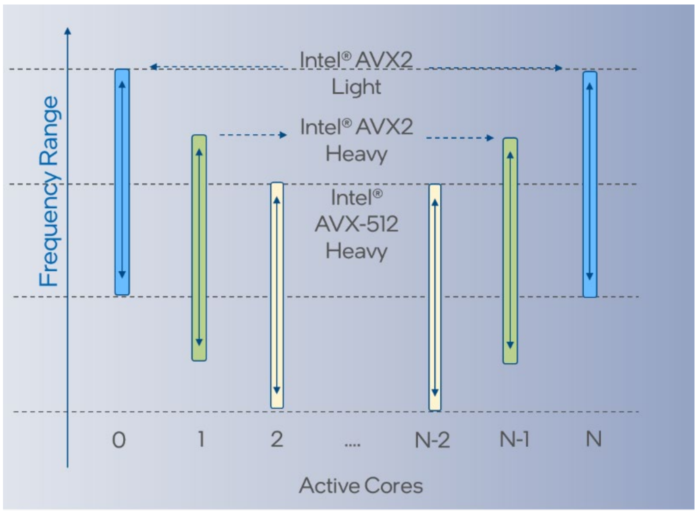


Figure 10. Workload Mix and Core Frequency

Figure 10 is an example of the core frequency range for a given system. Each core frequency is determined independently by the instruction mix of the workload executing on that core and the number of cores with similar instruction mixes.

Note: The 3rd Generation Intel® Xeon® Scalable processors is optimized to reduce the frequency impact for Intel® AVX-512 instructions when executing at Intel® AVX-512 Light (Level 1) and Intel® AVX-512 Heavy (Level 2) power levels, when compared to 1st Generation Intel® Xeon® Scalable processors for servers. See subsection 4.2.1 for more information.

Figure 11 shows some examples of the effect of possible frequency scaling on the Intel® Xeon® Platinum 8180 Processor (8180), Intel® Xeon® Platinum 8280 Processor (8280), and the Intel® Xeon® Gold 6230N Processor (6230N).

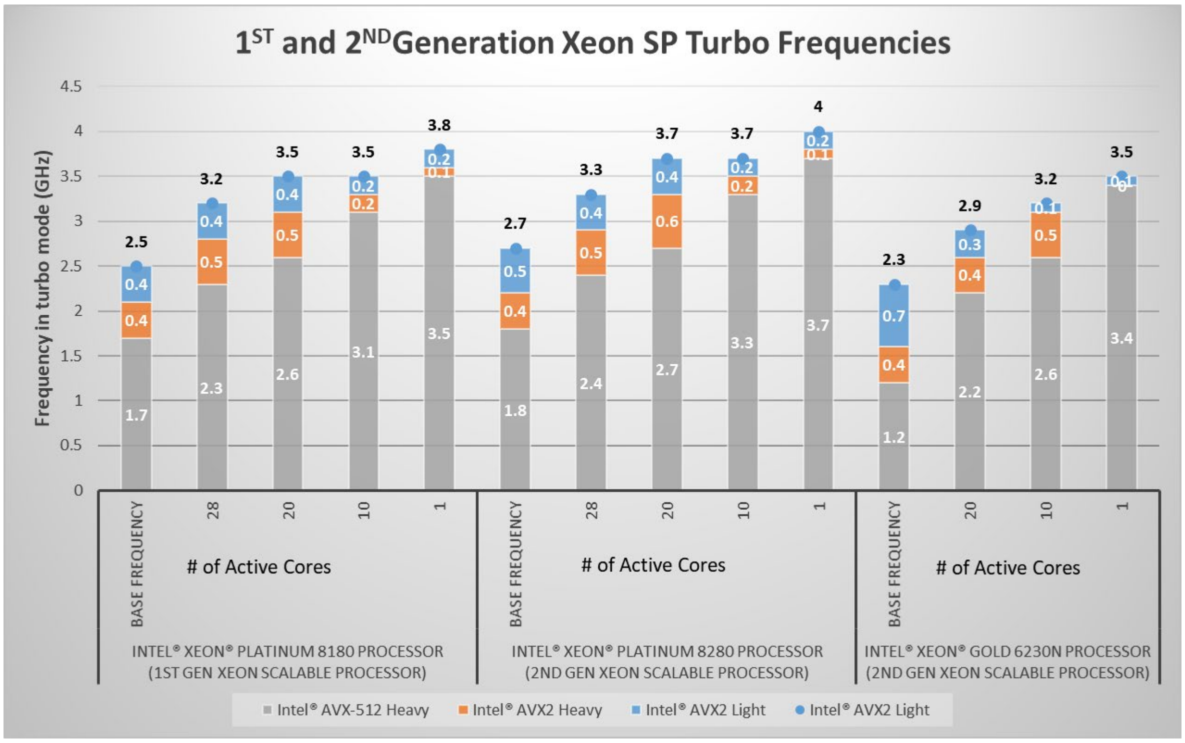


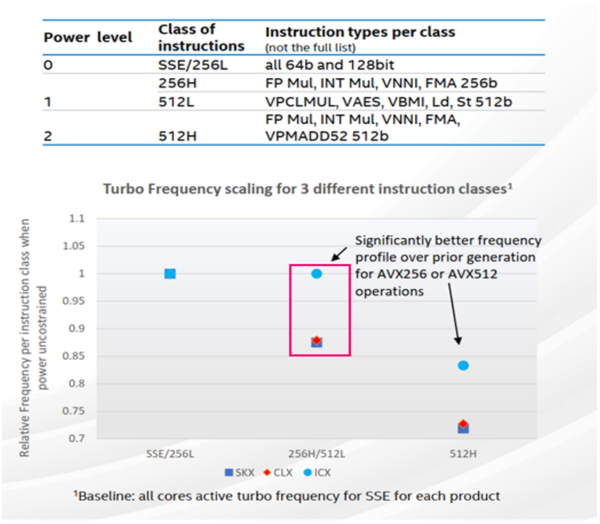
Figure 11. Processor Core Pipeline Functionality of the 1st Generation Intel® Xeon® Scalable Processor Microarchitecture

4.2.1 Power update for 3rd Generation Intel® Xeon® Scalable Processors

In the 3rd Generation Intel® Xeon® Scalable processors, the Power Level 1/Intel® AVX-512 Light power level frequency ranges are significantly improved, as shown in Figure 12. This means that on the 3rd Generation Intel® Xeon® Scalable processors, show as ICX, the IPC gains of using Intel® AVX-512 instructions are typically maintained or enhanced with an improvement in Turbo frequency ranges, when compared to the 1st and 2nd Generation Intel® Xeon® Scalable processors, shown as SKX and CLX respectively.

4.3 Power Level Transitions

Power level transitions are another important consideration when developing software with Intel® AVX-512 instructions; however, as you see in section 4.4, these may be less relevant for packet processing applications.



Power level	Class of instructions	Instruction types per class (not the full list)
0	SSE/256L	all 64b and 128bit
	256H	FP Mul, INT Mul, VNNI, FMA 256b
1	512L	VPCLMUL, VAES, VBMI, Ld, St 512b
2	512H	FP Mul, INT Mul, VNNI, FMA, VPMADD52 512b

Figure 12. Processor Core Pipeline Functionality of the 1st Generation Intel® Xeon® Scalable Processor Microarchitecture

At runtime, the CPU observes and matches the frequency of each core to the required power level. When the CPU detects that a new power level is required, it decides which new core frequency to apply. For example, if the Intel® AVX-512 Light instructions run on a given core with a current power level of *AVX2 Light (Level 0)*, and if that core now requires the *AVX-512 Light (Level 1)* power level, a frequency change may occur. When a frequency change occurs on the 1st Generation Intel® Xeon® Scalable processors for servers, the core frequency transition has a block time of 10-20 µsec to move between frequencies. Block time is the period during which the core is not processing any instruction.

There are three major stages in the CPU setting the power level:

1. The CPU assesses the instructions in flight.
2. As required, a new power level is applied (frequency change).
3. There is then a timeout period to return to a lower power level unless a new power level has been requested.

For example, if no additional instructions execute on a given core requiring the *AVX-512 Light (Level 1)* power level, the core transitions back to the original power level of *AVX2 Light (Level 0)*. However, if additional instructions that do require the *AVX-512 Light (Level 1)* power level run during this time period, the power level is maintained, and no transition occurs.

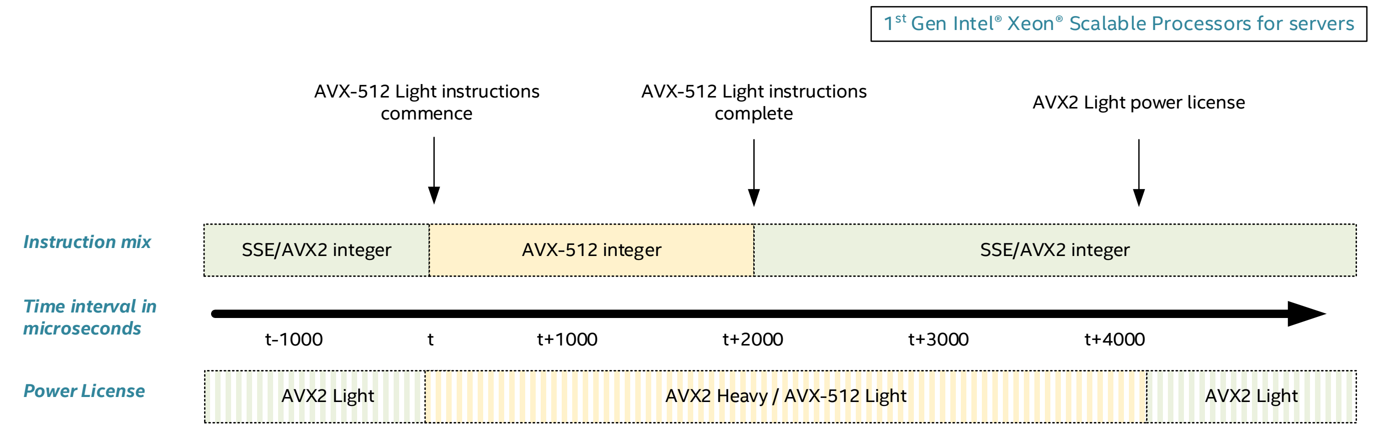

1st Gen Intel® Xeon® Scalable Processors for servers
AVX-512 Light instructions commence
AVX-512 Light instructions complete
AVX2 Light power license
Instruction mix
SSE/AVX2 integer
AVX-512 integer
SSE/AVX2 integer
Time interval in microseconds
t-1000
t
t+1000
t+2000
t+3000
t+4000
Power License
AVX2 Light
AVX2 Heavy / AVX-512 Light
AVX2 Light

Figure 13. Timeline of SIMD Instruction Mix and Power Licensing

An example of Power Level transitions is shown in Figure 13. In the figure, before time interval **t,** a given core is executing Intel® AVX2 integer instructions. At interval **t** an Intel® AVX-512 integer instruction runs and consequently a new *AVX-512 Light* power level is required. The core then blocks for 10-20 µsec until a new frequency is applied. Intel® AVX-512 integer instructions are then run intermittently until **t+2000** µsec, after which the workload returns to executing *AVX2 Light* instructions exclusively. After an additional 2000 µsec, without any additional Intel® AVX-512 instructions executing, the core returns to the *AVX2 Light* power level.

See *Intel® 64 and IA-32 Architectures Optimization Reference Manual, Section 2.4* for more information on 1st Generation Intel® Xeon® Scalable processors for servers.

4.3.1 Power Level Update for 3rd Generation Intel® Xeon® Scalable Processors

Frequency transitions are significantly improved in the 3rd Generation Intel® Xeon® Scalable processors where the block time is reduced to ~0 µsec, allowing for transitions without a latency cost.

4.4 Power Licensing in Packet Processing Applications

Data Plane Development Kit (DPDK) and similar packet processing applications tend to use core isolation and a polling mode of operation to maximize performance. Largely due to their polling behavior and the elimination of context switches, power level transitions in DPDK-like applications are rare as packet processing pipelines generally fall well within the bounds of the 2000 µsec timeout period described in section 4.3. This means that DPDK-like packet processing applications are largely unaffected by power level transitions.

To demonstrate this behavior with DPDK, refer to the *DPDK Getting Started Guide for Linux* that describes how to download, install, and configure DPDK 20.11. In DPDK 20.11, the instruction set mix used is controlled by the Environment Abstraction Layer (EAL) **max-simd-bitwidth** settings. The **max-simd-bitwidth** setting is configurable through the DPDK EAL API and through the DPDK command line `force-max-simd-bitwidth` parameter.

Setting the **max-simd-bitwidth** to **512** causes DPDK to use *AVX-512 Light (Level 1)* instructions and setting **max-simd-bitwidth** to **256** causes DPDK to use *AVX2 Light (Level 0)* instructions. The DPDK EAL API for setting **max-simd-bitwidth** is described in detail in the next paper in this series titled, *Writing Packet Processing applications with the Intel® AVX-512 Instruction Set*.

4.4.1 Test Case

The testing described in this section uses a 3rd Generation Intel® Xeon® Scalable Processor and the DPDK 20.11 release software. The DPDK Packet Classification and Access Control Library (DPDK ACL Library) is used to develop packet processing applications such as Intrusion Detection/Prevention Systems (IDS/IPS).

To demonstrate power licensing in packet processing applications, we use the DPDK **l3fwd-acl** sample application[2]. The sample application demonstrates the use of the DPDK ACL library in the DPDK to implement access control and packet L3 forwarding. Refer to the detailed guide in the DPDK documentation for setting up the **l3fwd-acl** sample application. There is also a guide for building DPDK sample applications such as **l3fwd-acl.**

[2] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

The DPDK **l3fwd-acl** sample application is configured with 4096 forwarding rules, a sample of which is shown below.

```
R10.0.9.121/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.8.165/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.0.203/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.14.39/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.3.6/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.4.139/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.3.31/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.9.8/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.5.56/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.7.34/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.10.171/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.2.137/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
R10.0.4.23/32 0.0.0.0/0 0 : 65535 0 : 65535 0x0/0x0 0
```

Intel® AVX-512 optimization can then be enabled at the sample application's command line by adding the parameters **`--alg=avx512x32`** and **`-force-max-simd-bitwidth=512`**, as follows:

```
./<build_dir>/examples/dpdk-l3fwd-acl -l 1,2 -n 4 -force-max-simd-bitwidth=512 -- -p 0x3 --
config="(0,0,1),(1,0,2)" --rule_ipv4="rule_ipv4.db" --rule_ipv6="rule_ipv6.db" --alg=avx512x32
```

These parameters have the following effect:

- The `--alg=avx512x32` parameter instructs the DPDK ACL Library to use the Intel® AVX-512 optimized lookup algorithm.
- The `-force-max-simd-bitwidth` parameter enables Intel® AVX-512 optimizations throughout DPDK.

Intel® AVX-512 optimizations may be disabled for reference purposes and Intel® AVX2 optimizations may be used instead by adding the parameters `--alg=avx2` and `-force-max-simd-bitwidth=256` as follows.

```
./<build_dir>/examples/dpdk-l3fwd-acl -l 1,2 -n 4 -force-max-simd-bitwidth=256 -- -p 0x3 --
config="(0,0,1),(1,0,2)" --rule_ipv4="rule_ipv4.db" --rule_ipv6="rule_ipv6.db" --alg=avx2
```

Note: The DPDK ACL library is not the only user of Intel® AVX-512 in the packet processing pipeline. Poll-mode-drivers and memory copy routines may also use Intel® AVX-512 optimizations. Therefore, you should be careful of possible misconfigurations such as enabling Intel® AVX-512 optimization in the poll-mode-driver and memory copies (with `-force-max-simd-bitwidth`), while disabling Intel® AVX-512 optimization in the DPDK ACL Library (with `--alg=avx2`), and so on.

After the DPDK **l3fwd-acl** sample application is up and running and forwarding packets, the power level of the core on which it is running can be measured using Linux *perf* and the `core_power.*` events as follows.

```
$ perf stat -e core_power.lvl0_turbo_license -e core_power.lvl1_turbo_license -e core_power.
lvl2_turbo_license  -C <core> -I 1000
    1.000084518      2,294,770,966      core_power.lvl0_turbo_license
    1.000084518                  0      core_power.lvl1_turbo_license
    1.000084518                  0      core_power.lvl2_turbo_license
    2.000232274      2,294,942,336      core_power.lvl0_turbo_license
    2.000232274                  0      core_power.lvl1_turbo_license
    2.000232274                  0      core_power.lvl2_turbo_license
```

Figure 14 shows a trace of the DPDK **l3fwd-acl** sample application using *AVX-512 Light* power level and AVX2 optimized lookup algorithms in two separate runs, over a duration of 16 seconds[3]. Here, it is important to note the following:

a. The DPDK **l3fwd-acl** sample application never transitions between power states. When configured to use Intel® AVX-512 optimizations, it continually stays in the *AVX-512 Light (Level 1)* power level and performance is not adversely impacted due to the power level transitions as described in section 4.3.
b. Similarly, when configured to use Intel® AVX2 optimizations instead of Intel® AVX-512, the sample application stays consistently at the *AVX2 Light (Level 0)* power level.

[3] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

c. The DPDK **l3fwd-acl** sample application exclusively uses *AVX-512 Light (Level 1)* power level instructions and entirely avoids the more impactful frequency shifts associated with using *AVX-512 Heavy (Level 2)* power level.

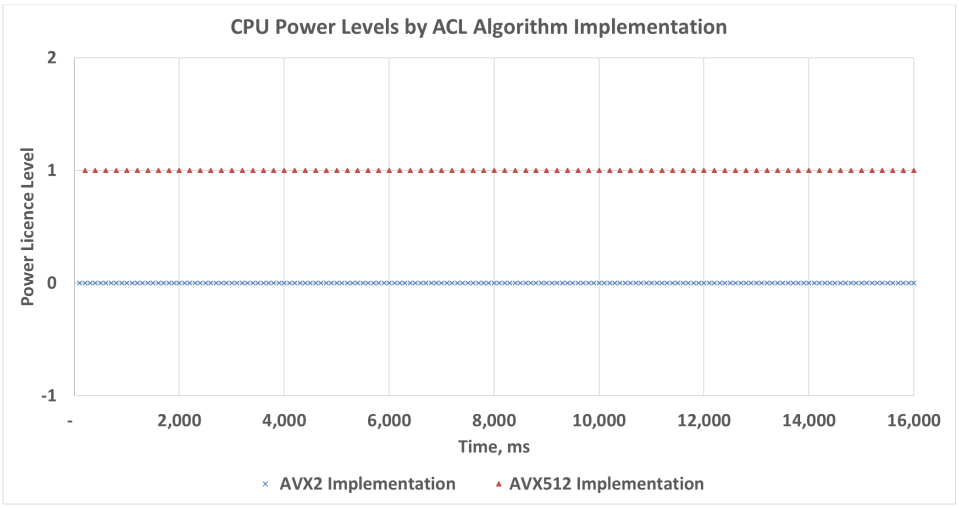


Figure 14. Power License Level Transitions Over Time by SIMD Implementation

As shown in Figure 15, the Intel® AVX-512 instruction set improves the flow search performance of the DPDK ACL library up to 300% compared to scalar lookups when tested with an ACL flow lookup microbenchmark[4]. When the same DPDK ACL library is used within a Layer 3 forwarding sample application **(l3fwd)**, the Intel® AVX-512 instruction set improves packet processing performance by up to 1.35x.

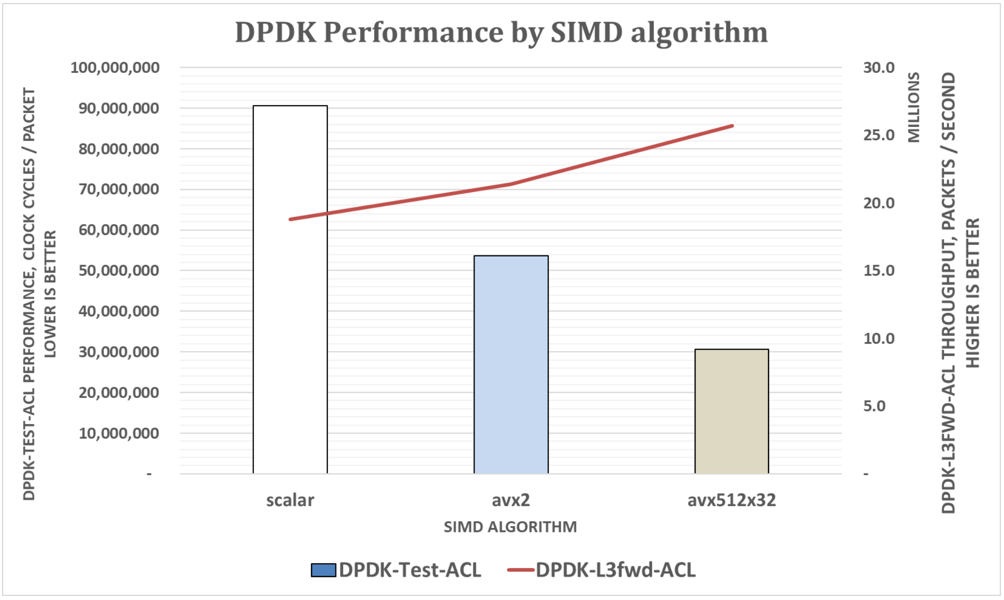


Figure 15. Single Core, Single Thread, 64-byte Packet Performance with DPDK L3FWD-ACL and DPDK-TEST-ACL Example Applications, 4096 Flows, 4096 ACL Rules on 3rd Generation Intel® Xeon® Scalable Processors

The test configuration used to perform the DPDK **l3fwd** ACL power transitions are listed in Table 8.

Table 8. DPDK L3fwd ACL Power Transitions Test Configuration

PARAMETERS	*DESCRIPTION*
Test by	Intel
Test date	28 Mar 2021

[4] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

PARAMETERS	*DESCRIPTION*
Platform	Intel Corporation Reference Platform*
# Nodes	1
# Sockets	2
CPU	Intel® Xeon® Gold 6338N CPU
Cores/socket, Threads/socket	32, 64
CPU Frequency	2.2 GHz
L3 Cache	48MiB
microcode	0x8d055260
BIOS version	WLYDCRB1.SYS.0020.P86.2103050636
System DDR Mem Config: slots/capacity/run-speed	16/16384 GB/3200 MT/s
Turbo, P-states	OFF
Hyper Threads	Enabled
NIC	Intel Corporation Ethernet Controller E810-2CQDA2 for QSFP (rev 02)
OS	Ubuntu 20.04.1 LTS (Focal Fossa)
Kernel	5.4.0-40-generic
DPDK	v21.02
Compiler	GCC 9.3.0-10 ubuntu2
	* Intel® Reference Platform (RP) for 3rd Generation Intel® Xeon® Scalable Processor

4.5 Additional Information

Additional information on the 3rd Generation and 1st Generation Intel® Xeon® Scalable processor's microarchitectures can be found at the following sources.

REFERENCE	CHAPTER
Intel® 64 and IA-32 Architectures Optimization Reference Manual (Ice Lake Client microarchitecture)	2.1
Intel® 64 and IA-32 Architectures Optimization Reference Manual (Skylake Server microarchitecture)	2.2
Intel® 64 and IA-32 Architectures Optimization Reference Manual (Skylake Server Power Management)	2.2.3
Intel® Xeon® processor Scalable family specification update (see Technical Documents)	NA
Second Generation Intel® Xeon® Scalable Processors Specification Update	NA
New 3rd Gen Intel® Xeon® Scalable Processor	Hot Chips 2020

5 Summary

If you are a network software engineer, motivated to solve compute-bound problems in your packet processing software, this document's intention is to clearly establish that AVX-512 development skills are a worth-while addition to your skill set and to provide enough pointers to get started.

To this end, the first section, Intel® AVX-512 Instruction Set, describes improvements in throughput and flexibility in the Intel® AVX-512 instruction set. It covers the basics of improvements with AVX-512, explaining how it doubles the throughput of each instruction compared to its predecessors. It then talks about the improvements in flexibility, introducing some new concepts such as masked and ternary operations and instruction set extensions.

The second section, Intel® AVX-512 and Intel® Xeon® Scalable Processor Microarchitecture, describes how the latest generations of Intel® Xeon® Scalable processors are optimized to run AVX-512 instructions. It describes how the microprocessors' execution ports have been widened to 512 bits, and how AVX-512 execution compares with that of the Intel® AVX2 instruction set. The section also describes power licensing, with examples using familiar packet processing applications.

The next white paper in this series describes Intel® AVX-512 usage in packet processing, concluding the series with some simple examples of how Intel® AVX-512 has been used to create significant performance improvements[5] in some well-known packet processing software. It describes the different development approaches to vector intrinsics and literals for writing AVX-512 code, and how Intel® AVX-512 support is detected and implemented in the software. It then walks through simple examples of optimizations achieved with Intel® AVX-512 instructions and the performance gains they generated.

0222/DN/WIT/PDF 633930-003US

[5] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.