Technology Guide

Intel® Ethernet 800 Series Network Adapter - Performance Evolution on DPDK

Authors

Leyi Rong

Bruce Richardson

Georgii Tkachuk

1 Introduction

Intel® Ethernet 800 Series Network Adapters offer exceptional performance to meet requirements for a range of workloads. The 800 series delivers packet-classification and sorting optimizations, hardware-enhanced timing capabilities, and a fully programmable pipeline. Intel's Ethernet product portfolio consistently delivers a reliable experience and proven interoperability.

On DPDK, the ICE Poll Mode Driver (PMD) manages the 800 series adapters, including the 10Gb/25Gb/50Gb/100Gb Intel Ethernet Network Adapter E810. The data plane of the ICE PMD has different, separated code paths, including scalar and vector data paths, which are chosen at runtime for optimum performance. The vector data path is accelerated by using Intel SIMD instructions and other techniques. Based on the specific SIMD instruction sets available, the vector data path selections cover Intel® Streaming SIMD Extensions (Intel® SSE), Intel® Advanced Vector Extension 2 (Intel® AVX2), and Intel® Advanced Vector Extension 512 (Intel® AVX-512) data paths. As the performance of these vector data paths is significantly better than that of the scalar path, we also call a vector path a "fast path" and the scalar path a "slow path."

This technology guide focuses on the data plane performance of the ICE PMD, including the methods used to improve the packet forwarding performance on a CPU core. It also illustrates the performance improvements achieved by using Intel SSE/Intel AVX2/Intel AVX-512 on the driver data path. Such content can help you better understand the details behind the performance improvement.[1]

This document is part of the Network & Edge Platform Experience Kits.

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

Table of Contents

Figures

Tables

Document Revision History

Revision	Date	Description
001	April 2023	Initial release.

1.1 Terminology

Table 1. Terminology

Abbreviation	Description
DPDK	Data Plane Development Kit (dpdk.org)
EAL	Environment Abstraction Layer
IA	Intel® Architecture
ICE	100Gb Intel® Ethernet Network Adapter (C is Roman numeral for 100)
Intel® AVX	Intel® Advanced Vector Extensions (Intel® AVX)
Intel® SSE	Intel® Streaming SIMD Extensions (Intel® SSE)
ISA	Instruction Set Architecture
MMX	MultiMedia eXtension
PMD	Poll Mode Driver
pps	packet per second
SIMD	Single Instruction Multiple Data

1.2 Reference Documentation

Table 2. Reference Documents

Reference	Source
DPDK Official Website	https://www.dpdk.org/
Intel® 64 and IA-32 Architectures Software Developer's Manual	https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.htmlhttps://software.intel.com/content/www/us/en/develop/articles/intelsdm.html
Intel® 64 and IA-32 Architectures Software Optimization Reference Manual	https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf
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® Ethernet Network Adapter E810-CQDA2 Product Guide	https://cdrdv2.intel.com/v1/dl/getContent/709766

2 Overview

Intel Ethernet 800 Series Network Adapters offer efficient workload-optimized performance at Ethernet speeds of 1 to 100Gbps. The 800 series is designed to improve network performance with innovative and versatile capabilities to optimize NFV, storage, HPC-AI, and hybrid cloud workloads.

DPDK is the Data Plane Development Kit that consists of libraries to accelerate packet processing workloads running on a wide variety of CPU architectures. DPDK initially added the ICE poll mode driver (PMD) for the 800 Series Ethernet Network Adapters in the DPDK 19.02 release, providing support for 10/25/50/100Gbps 800 Series Network Adapters, and the driver has been further enhanced in the DPDK releases since then. Although there are many features supported in the ICE PMD, as with other network adapter drivers in DPDK, packet processing performance remains the critical indicator for a DPDK PMD.[2]

Generally speaking, when discussing packet processing performance, we should do so in the context of the different data paths available, both scalar and vector. The vector data paths leverage Intel's SIMD instruction sets, and for the ICE PMD the vector data paths available include those optimized using Intel SSE, Intel AVX2, and Intel AVX-512. Therefore, this paper focuses on the following topics:

- Single core performance comparison with different data paths (scalar/Intel SSE/Intel AVX2/Intel AVX-512)
- Optimization methods used on the accelerated vector data paths, especially in the Intel AVX-512 data path

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

3 Performance Comparison

Intel introduced its first generation of SIMD processors with MMX extensions to the x86 architecture in 1997. Intel then developed Intel® Streaming SIMD Extensions (Intel® SSE) and Intel® Advanced Vector Extensions (Intel® AVX) to further improve the performance of SIMD operations. Intel AVX-512 was proposed by Intel in 2013, consisting of a set of 512-bit extensions to the 256-bit Intel AVX instructions. Intel AVX-512 was first implemented in Intel's Xeon Phi x200 (Knights Landing)[3] and then in the new Intel Xeon Scalable processor family.

3.1 Benchmarking Platform

As Intel AVX-512 instructions consume wider data per instruction and use more power, in some Intel processors, the CPU frequency may be lower when Intel AVX-512 instructions are executed. As the potential frequency reduction when using Intel AVX-512 is reduced in more recent Intel Xeon Scalable processor families, we use the 4th Gen Intel Xeon Scalable processor as the performance benchmarking platform. Table 3 describes the performance benchmarking platform details[4].

Table 3. Performance Benchmark Platform Configuration

Reference	Source
Architecture	x86_64
CPU Model	Intel® Xeon® Platinum 8468H CPU @ 2.10 GHz
# Socket(s)	2
# NUMA node(s)	2
Core(s) per socket	48
BIOS version	EGSDREL1.SYS.0096.D30.2301031434
Microcode	0x2b0003b1
SpeedStep, TurboBoost	Enabled
Core Frequency	3.8 GHz
Network Adapter	Intel® Ethernet Network Adapter E810-CQDA2
OS	Ubuntu 22.04 (Jammy Jellyfish)
Kernel	5.15.35 with CONFIG_NO_HZ_FULL=y
DPDK Version	V22.11
Compiler	GCC 9.4.0-5ubuntu1
Grub Cmdline	hugepagesz=2M hugepages=8192 default_hugepagesz=2M isolcpus=1-47,49-95,97-143,145-191 rcu_nocbs=1-47,49-95,97-143,145-191 nohz_full=1-47,49-95,97-143,145-191 rcu_nocb_poll skew_tick=1 nomodeset clocksource=tsc kthread_cpus=0 irqaffinity=0 powernow-k8.tscsync=1 panic=30 init=/sbin/init net.ifnames=0 nmi_watchdog=0 audit=0 nosoftlockup hpet=disable mce=off tsc=reliable numa_balancing=disable memory_corruption_check=0 workqueue.power_efficient=false module_blacklist=ast,iavf modprobe.blacklist=ice init_on_alloc=0 intel_iommu=off
Test Command	#dpdk-testpmd --force-max-simd-bitwidth=<64,128,256,512> -l 49,50 -n 8 --socket-mem 512,512 -a 98:00.0 -a a8:00.0 -- --nb-cores 1 --rxq 1 --txq 1 -a --rxd 1024 --txd 1024 --forward-mode=io --burst=32

The performance benchmark platform topology uses two single ports of two dual-port 100Gb Ethernet Network Adapter E810 (1 port per network adapter) connected to Ixia® Traffic Generator, as the Network Adapter E810-CQDA2 only supports up to 100Gb for each adapter. That means a total of 297.6 Mpps aggregated traffic should be handled by the core if the packet size is 64 bytes. The DPDK "Testpmd" reference application is responsible for forwarding back the packets from Ixia, as shown in Figure 1. It is important to note that RFC2544 zero packet loss is used in the test case, and the acceptable packet loss rate is 0.

[3] James Reinders (23 July 2013). "AVX-512 Instructions"

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

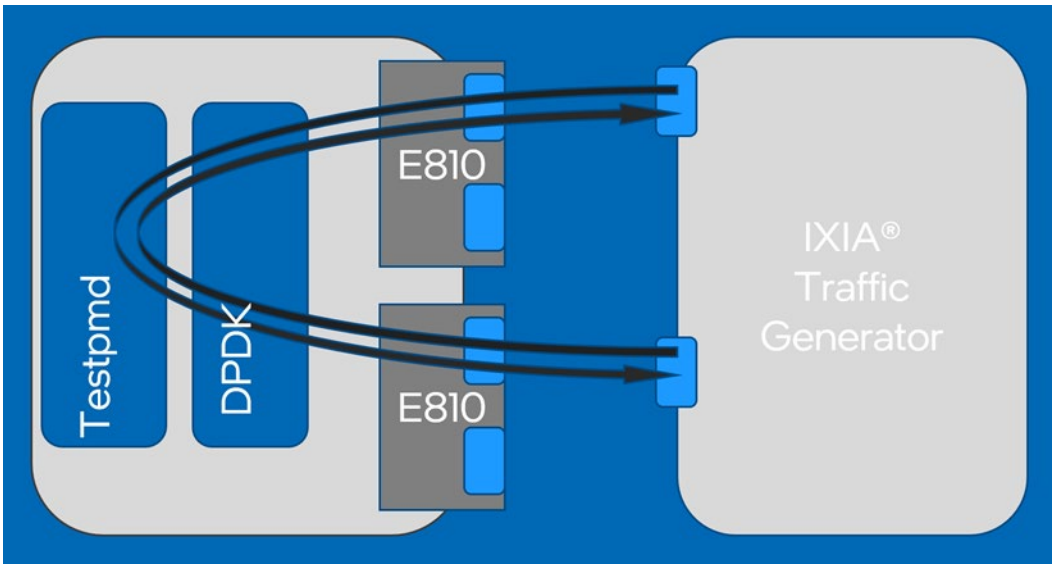

Figure 1. Performance Benchmark Test Topology

3.2 Data Path Selection

As previously mentioned, there are several SIMD accelerated implementations of the data plane in the DPDK ICE PMD. This is not unique to this particular PMD, as multiple accelerated implementations also exist for other Intel high-speed Ethernet adapters in DPDK, for example, in the "ixgbe" and the "i40e" PMDs. Although these fast data paths do not support all network adapter offload features, especially on the transmit (Tx) side, in scenarios in which there is no need for these advanced offload features, it is worth choosing a dedicated fast data path, as such a path can deliver remarkably better performance by using the advanced SIMD extension instruction sets and related optimizations on state-of-the-art modern processors.[5] Taking the ICE PMD for example, three SIMD extension implementations are added, as well as the original scalar data path, so one of four different data paths can be selected depending on the platform and application requirements. These four paths, and the locations of their source code in the driver, are:

1. Scalar data path – `ice_rxtx.c`
2. Intel SSE data path – `ice_rxtx_vec_sse.c`
3. Intel AVX2 data path – `ice_rxtx_vec_avx2.c`
4. Intel AVX-512 data path – `ice_rxtx_vec_avx512.c`

The DPDK testpmd application is used to test the ICE PMD packet forwarding performance. Normally, the driver itself selects what it feels to be the best code path at runtime – normally the Intel AVX2 path, assuming hardware supports it – but we can manually adjust the data path selection by passing the DPDK EAL parameter "`--force-max-simd-bitwidth=<val>`" to the application at runtime. For example, to select the Intel AVX-512 data path (on supporting hardware), append `--force-max-simd-bitwidth=512` to the testpmd command. Similarly, change `<val>` to 64, 128, and 256 to change the data path selection to scalar, Intel SSE, and Intel AVX2 data paths respectively.

3.3 Performance Benchmarking within Different Data Path Selection

To demonstrate the packet forwarding performance attainable using the ICE PMD, Figure 2 shows the packet throughput measured as we move from the scalar data path, through various vectorized data paths, all the way to the Intel AVX-512 path. The test configuration is listed above in Table 3. The 32-byte Flexible Receive (Rx) Descriptor format was used by the driver and the Rx/Tx descriptor ring size was 1024.

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

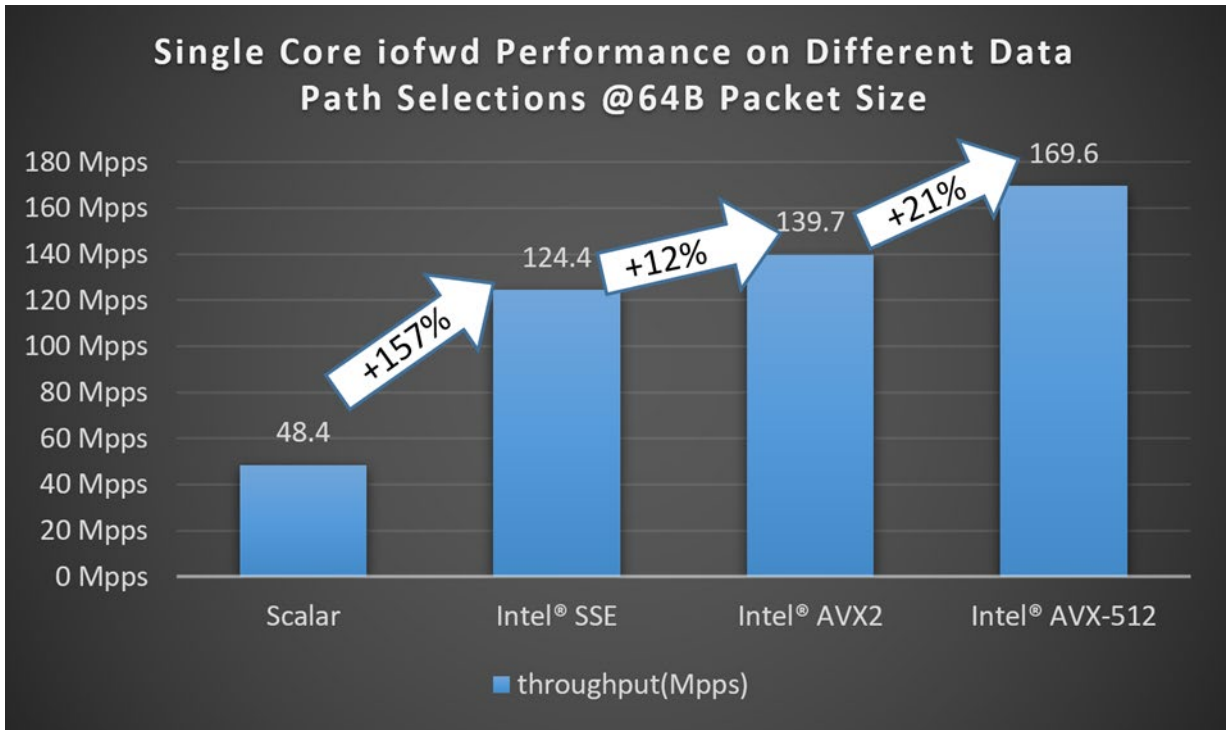

Figure 2. Single Core iofwd Performance on Different Data Path Selections @ 64B Packet Size

The results show that testpmd iofwd (the forwarding mode where we transmit each received packet without modifying it) running on a single core achieves significant performance improvement as one goes from using the scalar data path to the Intel SSE data path, to the Intel AVX2 data path, and finally to the Intel AVX-512 data path. Overall, a ~**3.5x** single core performance increase was measured between using the Intel AVX-512 data path and using the initial scalar data path[6]. Note though, that the features supported on these paths are not entirely equal, as the scalar data path supports more advanced offload features. However, even looking solely across the three different vectorized code paths (which are equivalent in feature set), the data shows that performance improvements can be achieved by using more advanced SIMD instructions.

4 ICE PMD Performance Optimizations

Section 3.3 showed that the driver performance improves hugely when changing the data path from the scalar to the Intel AVX-512 implementation. In this section, we deep dive into how the latest Intel AVX-512 data path optimizes performance on the latest Intel Xeon Scalable processors.

4.1 ICE PMD Packet Pipeline

This subsection gives an overview of how network packets are processed in the PMD.

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

4.1.1 Rx Path Pipeline

Figure 3 shows the steps involved in receiving a network packet. As shown in the diagram, the steps are:

1. CPU/driver writes an Rx descriptor to the receive descriptor ring in memory/cache for the network port. This descriptor includes a reference to an empty buffer where the network adapter can place the newly received packet.
2. Network adapter reads the Rx descriptor to get the buffer address to store the packet.
3. Network adapter uses DMA to write the received packet to the provided buffer.
4. Network adapter writes back a new Rx descriptor to the descriptor ring to inform software that a packet has been received and put in the supplied buffer.
5. CPU reads from the new Rx descriptor the details of the received packet, for example, the packet length or metadata about packet offloads performed.

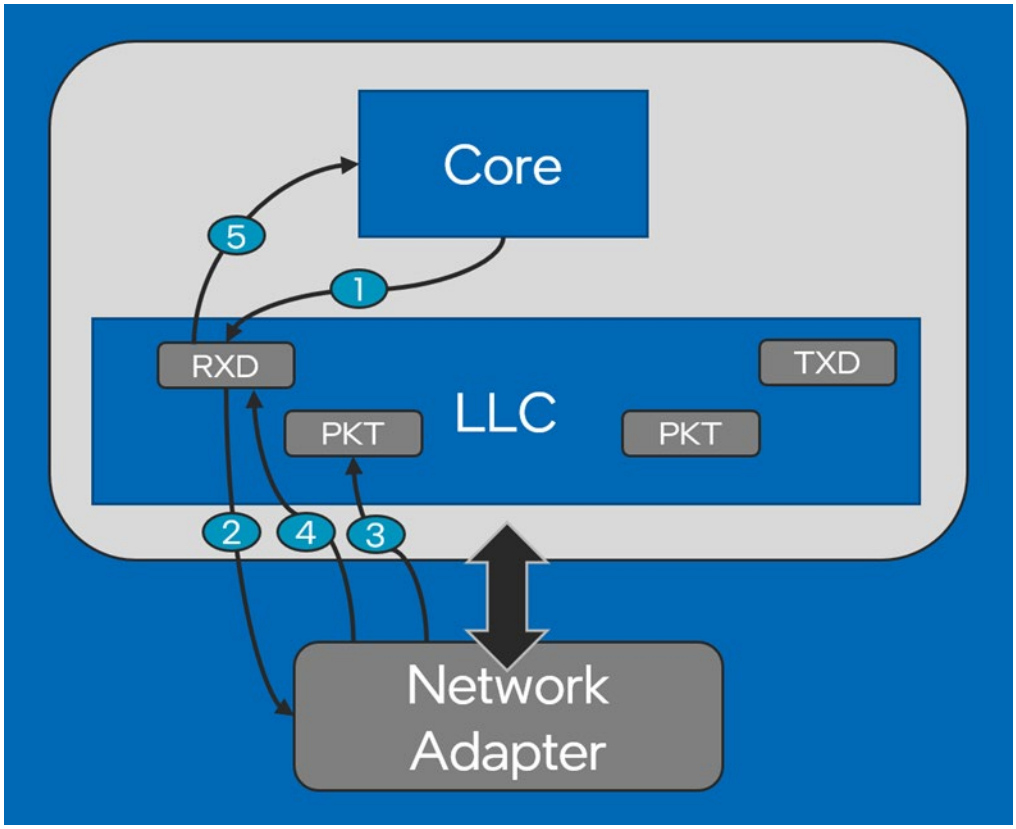

Figure 3. Rx Path Pipeline

4.1.2 Tx Path Pipeline

Figure 4 similarly shows the egress packet processing pipeline. The steps in transmitting a packet are:

1. CPU writes any new data to the packet buffer. In the case of a newly created packet, this would be the entire packet data, including the protocol headers. Where a previously-received packet is being retransmitted back out of the system, the CPU may only write a few bytes of data, for example to update packet headers, before transmit.
2. CPU reads the next Tx descriptor to check if the network adapter has completed the previous transaction, if any, using this slot of the Tx descriptor ring. If so, the descriptor slot can be reused and the transmission process can continue. If the previous transaction has not completed, there are no descriptor ring slots available and the packet transmission fails.
3. CPU writes a Tx descriptor to the descriptor ring slot. That descriptor includes a pointer to the buffer where the packet to be transmitted resides, as well as other necessary metadata, such as the length of the packet. For the scalar code path, this descriptor may include details of some transformations to be performed on the packet by the network adapter (that is, any Tx offloads) before transmitting the packet.
4. Network adapter reads Tx descriptor to get packet buffer address and packet length.
5. Network adapter reads Tx packet from the buffer and transmits it.
6. Network adapter writes back a new Tx descriptor to update the transition status. For the scalar driver code, the network adapter writes back a completion after each packet is transmitted. However, for the vector data paths, a writeback is only performed after N packets (32 by default) in order to save PCI bandwidth and to give improved performance. This

is possible as the vector path uses only one descriptor per packet, as it does not support all Tx offload features or packets requiring multiple buffers. For the scalar path, such an optimization is not possible as the network adapter only performs writeback on the final descriptor of a packet, and it is not possible to guarantee that each Nth descriptor is a final descriptor of a packet.

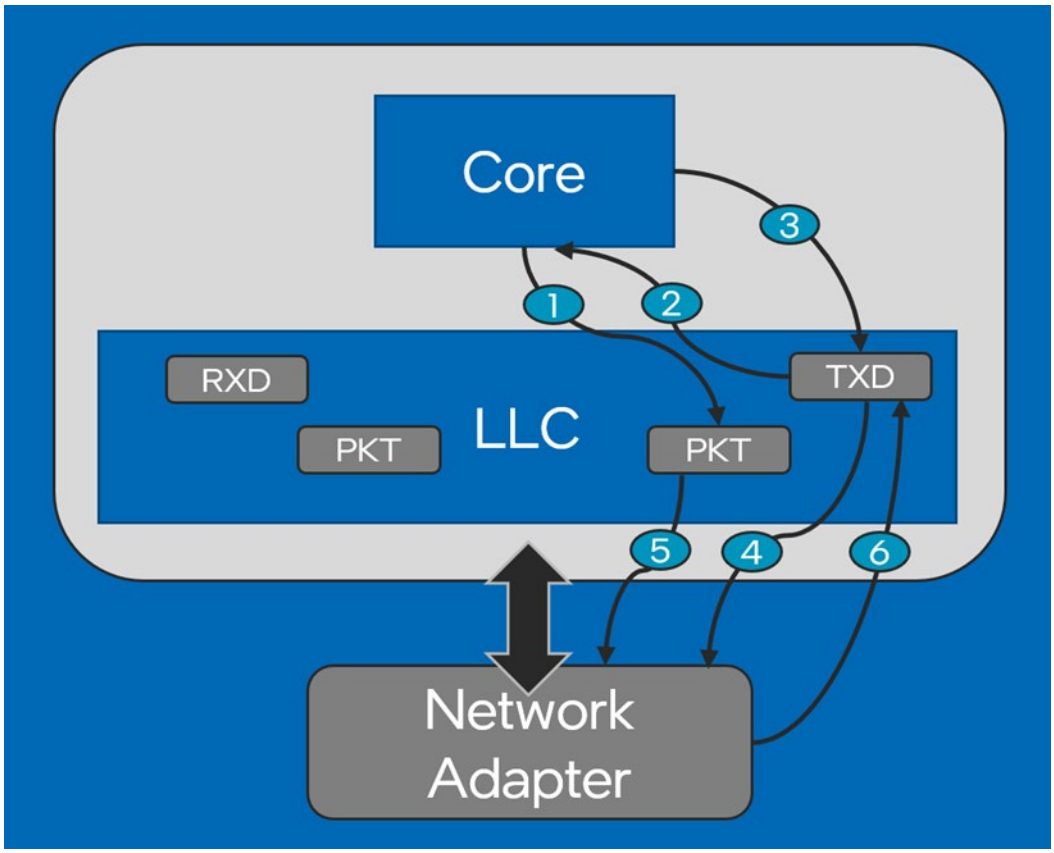

Figure 4. Tx Path Pipeline

4.2 Optimizations on the Intel AVX-512 Data Path

Intel AVX-512 is the latest Intel SIMD extension technology on the x86 platform. It doubles the number of available SIMD registers and broadens the width of each register from 256 to 512 bits compared with its predecessor Intel AVX2. Besides more and wider SIMD registers, which allows us to operate on more data per instruction, Intel AVX-512 also extends the instruction set to provide more advanced functionality, including masked operations, embedded broadcast, instruction prefix-embedded rounding control, and compressed address displacements.

As shown in Figure 2, the Intel AVX-512 data path selection is undoubtedly the best-performing data path inside the ICE PMD.[7] The significant performance improvement over earlier driver paths is not only brought by using Intel AVX-512 instructions, but also by some other code optimizations. The following sections look at a couple of these in more detail.

4.2.1 Vectorized Processing by Using SIMD Instructions

Section 4.1 describes the processing that occurs to receive a packet on a core. In terms of the CPU cycle cost of doing so, a significant portion of the core cycles is taken up in processing the Rx descriptor from the network adapter and transferring the information from the descriptor to the DPDK internal buffer metadata, "mbuf", structure (step 5 of Section 4.1.1). We can use SIMD instructions to accelerate this by processing the information from multiple descriptors simultaneously.

The default 32-byte receive flex descriptor format, as written by the Intel Ethernet 800 Series Network Adapter, is shown in Figure 5[8].

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

[8] Intel® Ethernet Controller E810 Datasheet

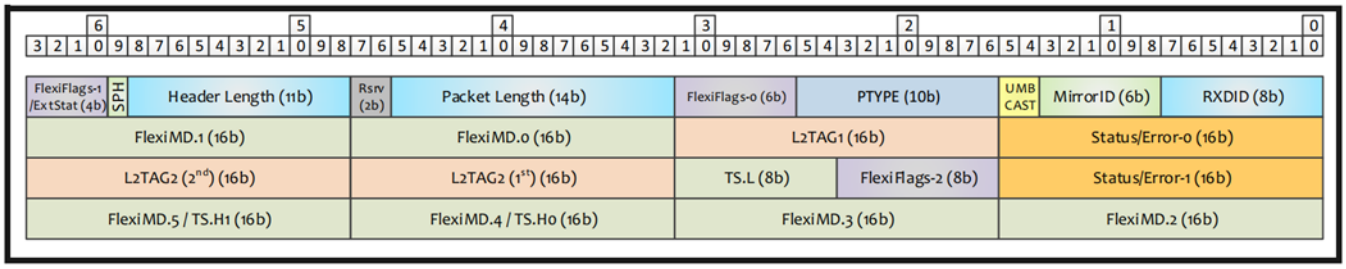

Figure 5. Receive Flex Descriptor

As most of the required packet information, such as packet length, packet type, descriptor done indicator, and end of packet indicator, is provided in the first 16 bytes of the 32-byte Rx descriptor, the data from up to four Rx descriptors can be merged into a single (64-byte) Intel AVX-512 ZMM register. These four descriptors can then be further processed simultaneously by leveraging Intel AVX-512 instructions. The four combined Rx descriptors are formatted in the register as shown in Figure 6.

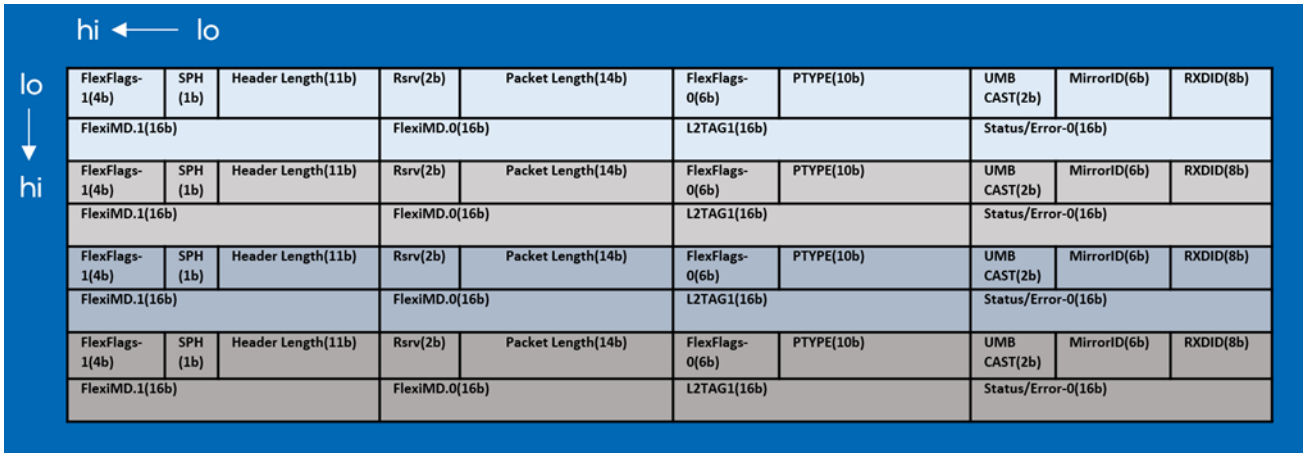

Figure 6. Format of Combined Four Rx Descriptors in One Intel AVX-512 Register

After gathering four Rx descriptors into a single Intel AVX-512 register, the PMD starts to manipulate and interpret the packet information from the Rx descriptor to the mbuf.[9] Using a mix of 512-bit and 256-bit instructions [such as `_mm512_shuffle_epi8()`] to shuffle the packed elements by specified control mask [`_mm512_extracti64x4_epi64()`] to extract the selected 256-bit half of any 512-bit width data [or `_mm512_mask_blend_epi32()`] to blend packed integers using the specified control mask, the driver parses and reorganizes the descriptor's fields to match the memory layout of fields in the DPDK mbuf structure . After these vectorized operations, the mbuf structure for each packet can be appropriately filled up with a single 32-byte store instruction, as shown in Figure 7. For more information, see Intel® 64 and IA-32 Architectures Software Developer's Manual.

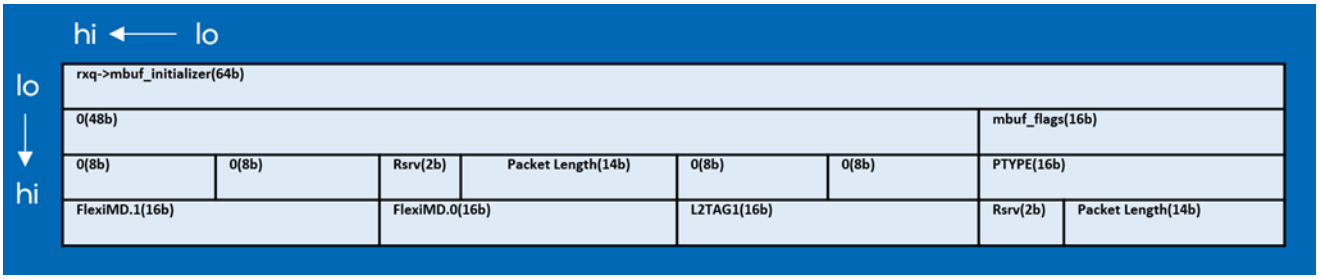

Figure 7. Reorganized Descriptor's Content to Fill Up DPDK "mbuf" Structure

Similar to the above process, vectorized instructions are also used elsewhere in this data path to parallelize processing to help achieve improved performance. Although this guide does not include all the details of this vectorized processing, one can still learn from the example of the snippet described above.

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

4.2.2 Directly Operating on Memory Pools

In the Tx path described in Section 4.2.1, the number of available Tx descriptors is checked by the PMD before it starts to transmit packets. After the free Tx descriptor count drops below the threshold defined by `tx_free_thresh`, the PMD must attempt to free the buffers of those packets with completed transmissions and make the descriptor ring slots they occupied available for reuse.

Inside a PMD, a sizeable proportion of processing time can be spent on buffer allocation and freeing. In particular, the freeing of buffers post-transmit can be costly as each packet may need to be individually inspected to determine which buffer "pool" it belongs to and then freed back to that pool. Performance of this freeing of buffers can be improved in a number of ways, all of which are implemented in the Intel AVX-512 transmit path in the ICE PMD.

Firstly, at network adapter port configuration time, DPDK allows an application to specify, by way of the `FAST_FREE` flag, that buffers being transmitted on the port all belong to the same memory pool and that no packets larger than a single buffer will be used. When this flag is provided, the buffer freeing function in the PMD need only look at the buffer pool for the first packet to be freed and can then free all buffers to the same memory pool in a single `put` operation.

Secondly, the PMD takes advantage of the fact that it is returning a fixed number of elements to the mempool in all cases, allowing the PMD to avoid generic code for the mempool implementation and instead use optimized Intel AVX-512 operations to store the pointers inside the mempool. While the mempool code attempts to copy the buffer pointers one at a time, the driver code instead can use AVX-512 to load and store eight pointers at a time. This vastly reduces the number of load/store operations required, improving performance.

4.2.3 Miscellaneous Optimization Tactics

The primary optimization approaches are covered in previous sections of this guide. This section covers minor but useful tricks that can provide performance improvement.

- Fine-tune Rx/Tx threshold constants to achieve a tradeoff between throughput and latency.
- Use Write Combining Store instead of regular MMIO writes to update the queue tail registers. For smaller bursts on transmit, this can provide a sizeable benefit on supporting platforms. By making smaller bursts less expensive, this change also helps applications achieve lower latency.
- Use of inlining – for example through explicit inlining via function attribute – helps ensure that the code does not pay a penalty for being split up into separate functions, by ensuring the compiler inlines the subfunctions into the main code blocks.
- Minimize load/store operations by merging multiple loads/stores into one vectorized operation. For example, doing a single 32-byte or 64-byte store instead of multiple 4- or 8-byte store operations.

5 Summary

This technology guide demonstrates the benefits to be achieved by using the latest Intel AVX-512 instruction set. It does this by showing the performance achieved by the DPDK poll mode driver for the Intel® Ethernet 800 Series Network Adapter, when using different instruction sets – both scalar and vector[10]. The Intel AVX-512 code path is shown as noticeably faster than the previous vectorized implementations using either Intel SSE or Intel AVX2, as well as being significantly faster than the scalar code – albeit while offering a reduced offload set than the latter.

Beyond this, this paper examines some of the main optimization approaches taken by the AVX-512 data path, explaining how these optimizations contribute to the performance benefit shown.

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

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

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

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.

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

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

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.

0423/DN/WIT/PDF 767813-001US