Technology Guide

FD.io VPP – Accelerate the Host Stack with 4th Gen Intel® Xeon® Scalable Processor

Authors

Ping Yu

Marvin Liu

Junfeng Wang

Pengze Qiu

John DiGiglio

Florin Coras
Cisco Systems, Inc.

1 Introduction

With the ongoing rapid increase in internet traffic, as a basic underlying component of the network, network host stacks are a focus of efforts to accelerate network performance. Maintaining and improving performance of the increased bandwidth has motivated industry to explore a multitude of solutions for network stack acceleration, from kernel optimization to user space stack deployment.

As a transport layer, a host stack is the intermediate layer between application space and network space. For security and software layer design considerations, application space and network space are generally isolated, therefore, for both kernel space and user space network space, memory copy between the two spaces is necessary and inevitable. To address the industry challenge, the 4th Gen Intel® Xeon® Scalable processor has embedded a DMA engine to accelerate memory copy in the host stack.

This guide takes the FD.io VPP host stack, an open-source project, as the example to demonstrate the memory copy acceleration benefit based on the latest 4th Gen Intel® Xeon® Scalable processor with Intel® Data Streaming Accelerator (Intel® DSA). We also describe the software API interface to use hardware to help achieve this performance.

This document is intended for communication service providers, or anyone looking to improve the performance of the host stack in their network. Even though the goal of this document is to showcase improvements using the VPP host stack, the technologies enabled here can be used as a reference point for improving performance in other user space or even kernel space host stack deployments.

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

Table of Contents

Figures

Tables

Document Revision History

Revision	Date	Description
001	October 2022	Initial release.
002	January 2023	Revised the document for public release to Intel® Network Builders.

1.1 Terminology

Table 1. Terminology

Abbreviation	Description
DCA	Direct Cache Access
DMA	Direct Memory Access
DPDK	Data Plane Development Kit
DUT	Device Under Test
Intel DSA	Intel Data Streaming Accelerator
Intel® I/OAT	Intel® I/O Acceleration Technology (Intel® I/OAT)
FD.io	Fast Data Input/Output
MSI-X	Extended Message Signaled Interrupts
NIC	Network Interface Card
RSC	Receive Side Coalescing
SVM	Shared Virtual Memory
SW	Software
TCP	Transmission Control Protocol
VPP	Vector Packet Processing

1.2 Reference Documentation

Table 2. Reference Documents

Reference	Source
Intel® I/O Acceleration Technology	https://www.intel.com/content/www/us/en/wireless-network/accel-technology.html
Introducing the Intel® Data Streaming Accelerator (Intel® DSA)	https://01.org/blogs/2019/introducing-intel-data-streaming-accelerator
VPP Wiki	https://wiki.fd.io/view/VPP
VPP/Host Stack Wiki	https://wiki.fd.io/view/VPP/HostStack

2 Overview

This document is intended as a guide to show how host stack performance may be significantly improved based on the latest 4th Gen Intel Xeon Scalable processor. We analyze performance and identify memory copy as a common bottleneck in both kernel space host stack and user space stack.

This document describes Intel® technologies for memory copy acceleration from Intel® I/O Acceleration Technology (Intel® I/OAT) technology to Intel Data Streaming Accelerator (Intel DSA). To help software better use hardware capabilities, `dmadev` and `wq` API interfaces are exposed in user space and kernel space, respectively. FD.io VPP host stack is used to showcase memory copy acceleration technologies that can achieve significant performance gains for HTTP traffic. For our experiments we employed two Device Under Tests (DUTs), each with one socket Intel Xeon Scalable processors. The test setup includes:

- A system with the latest Intel Xeon Scalable processors with Intel Data Streaming Accelerator (Intel DSA)
- Two Intel® Ethernet Network Adapter E810-CQDA2 PCIe 4.0 Network Interface Cards (NICs) with 100 Gbps throughput
- Fast Data Input/Output (FD.io) Vector Packet Processing (VPP) high performance, packet-processing stack
- NGINX open source for HTTP protocol processing

2.1 Host Stack Memory Copy Overhead Analysis

The most common and available host stack measurement is iperf3. Based on iperf3, we launched a test between servers with Intel® Ethernet Network Adapters. During the test, we used perf3 to identify per-CPU-core performance bottlenecks.

```
PerfTop:    3747 irqs/sec  kernel:99.0%  exact: 100.0% lost: 0/0 drop: 0/0 [4000Hz cycles],  (target_pid: 433210)
-----------------------------------------------------------------------------------------------------------------

  42.00%  [kernel]       [k] copy_user_enhanced_fast_string
  21.91%  [kernel]       [k] __check_object_size
   5.77%  [kernel]       [k] _copy_to_iter
   5.25%  [kernel]       [k] __skb_datagram_iter
   2.57%  [kernel]       [k] __virt_addr_valid
   1.46%  [kernel]       [k] simple_copy_to_iter
   1.15%  [kernel]       [k] tcp_rcv_established
   0.77%  [kernel]       [k] _raw_spin_lock
   0.77%  [kernel]       [k] _raw_spin_lock_irqsave
   0.75%  [kernel]       [k] read_tsc
   0.71%  [kernel]       [k] sock_rfree
   0.65%  [kernel]       [k] tcp_recvmsg_locked
   0.65%  [kernel]       [k] check_stack_object
   0.64%  [kernel]       [k] skb_release_data
   0.60%  [kernel]       [k] copyout
   0.46%  [kernel]       [k] tcp_eat_recv_skb
   0.41%  [kernel]       [k] __domain_mapping
   0.38%  [kernel]       [k] __alloc_skb
```

Figure 1. Memory Copy Overhead in Kernel Space Stack

In Figure 1, you can see that in kernel space, host stack-related memory copy is the biggest bottleneck.

We similarly benchmarked the VPP host stack using LD_PREOAD shim, which intercepts POSIX socket API calls and redirects them into VPP. In this scenario, the VPP network stack is in charge of packet reception, transmission, and processing up to the IP layer, while the host stack performs TCP termination and delivery of control and I/O events to the iperf3 application through shared memory. The following perf report is the result of using this setup.

```
PerfTop:    4057 irqs/sec  kernel: 0.0%  exact: 100.0% lost: 0/0 drop: 0/0 [4000Hz cycles],  (all, CPU: 4)
-----------------------------------------------------------------------------------------------------------------

  36.50%  libsvm.so.22.02.0       [.] svm_fifo_copy_to_chunk_ma_icl
   7.66%  libsvm.so.22.02.0       [.] svm_fifo_enqueue
   5.63%  libvnet.so.22.02.0      [.] tcp4_established_node_fn_icl
   5.33%  libvnet.so.22.02.0      [.] tcp_rcv_ack
   4.96%  libvnet.so.22.02.0      [.] tcp_segment_validate
   3.50%  libvnet.so.22.02.0      [.] ip4_local_inline
   3.33%  libvnet.so.22.02.0      [.] tcp_segment_rcv
   3.32%  libvnet.so.22.02.0      [.] session_lookup_connection_wt4
   3.10%  libvnet.so.22.02.0      [.] ip4_lookup_node_fn_icl
   3.01%  libvnet.so.22.02.0      [.] session_enqueue_stream_connection
   2.92%  dpdk_plugin.so          [.] ice_recv_scattered_burst_vec_avx512_offload
   2.54%  libvnet.so.22.02.0      [.] tcp4_input_node_fn_icl
   2.20%  libvnet.so.22.02.0      [.] ethernet_input_node_fn_icl
   2.11%  libvnet.so.22.02.0      [.] ip4_input_no_checksum_node_fn_icl
   1.87%  dpdk_plugin.so          [.] dpdk_ops_vpp_dequeue_icl
   1.65%  libsvm.so.22.02.0       [.] fsh_try_alloc_chunk
   1.64%  dpdk_plugin.so          [.] dpdk_input_node_fn_icl
   1.21%  libvnet.so.22.02.0      [.] tcp_rcv_sacks
   1.10%  libvnet.so.22.02.0      [.] tcp_session_get_transport
```

Figure 2. Memory Copy Overhead in User Space

From the results shown in Figure 2, you can see that memory copy is a key component for both kernel and user space host stacks. This guide provides a hardware-accelerated DMA engine to reduce the memory copy overhead and help improve performance.

2.2 Host Stack Use Case

This document primarily covers user space host stack FD.io VPP Intel DSA acceleration.

2.2.1 Fast Data Input/Output (FD.io), Vector Packet Processing (VPP)

FD.io (Fast Data Input/Output) is a Linux Foundation open-source project that provides fast network packets processing capability. FD.io Vector Packet Processing (VPP) is one of many sub-projects within FD.io that provides L2-L4 stack processing.

VPP processes the packet in the burst manner, grouping up to 256 packets into a packet vector. To maximize the utilization of the CPU instruction cache (I-cache), VPP adopts the packet processing graph as its core design. The graph nodes are organized as tree shaped graphs in VPP. The packet vectors flow from NIC RX nodes all the way to TX nodes (or are dropped) based on the processed destinations in each graph node within.

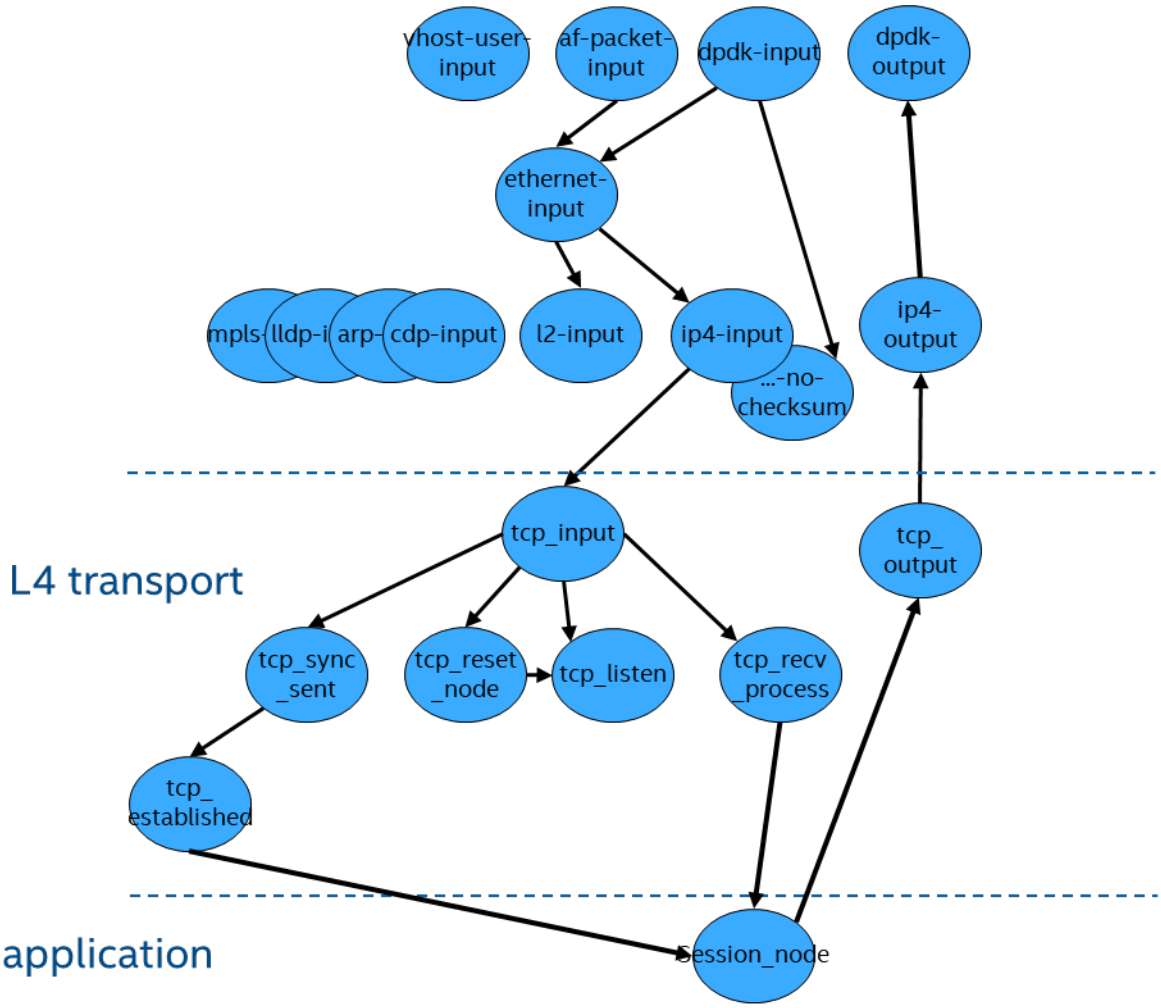

Figure 3. Example of Graph Nodes in VPP

For more information related to VPP, refer to: https://wiki.fd.io/view/VPP.

2.2.2 VPP Host Stack

VPP host stack is a user space network stack implementation for transport layer packet processing. Based on VPP's highly efficient L2 and L3 packet processing capability, the VPP host stack enables fast transport layer for both intermediate built-in and external applications. The host stack consists of two main components. One is the packet processing engine for transport protocol processing, including TCP, UDP, TLS, and QUIC protocols. The other is a shared memory-based interface for application integration. Applications can rely on a POSIX-like API or LD_PRELOAD to integrate the network stack to VPP. Our benchmark test is based on the LD_PRELOAD interface.

A shared memory-based interface provides an intermediate buffer between the application and the VPP host stack. It is a bridge between the network space and application space. If the shared memory performance is accelerated, the overall throughput is also improved.

2.2.3 NGNIX Application

NGINX is an industry-popular, open-source, high-performance HTTP server. NGINX is well known for its high-performance, small memory footprint, rich feature set, stability, and low resource consumption. When NGINX is started, a primary process together with several worker processes are launched. Each worker can be configured to be pinned in one core. By fixing the number of VPP threads and NGINX workers, we can compare the performance improvement based on the same amount CPU core.

2.3 Technology Description

2.3.1 Intel I/OAT Technology

Intel I/OAT is a set of technologies for improving I/O performance, such as Direct Cache Access (DCA), Extended Message Signaled Interrupts (MSI-X), and Receive Side Coalescing (RSC).

As a component of Intel I/OAT, Intel® Quick Data Technology is a platform solution designed to maximize the throughput of server data traffic across a broad range of configurations and server environments to achieve faster, scalable, and more reliable I/O. It provides fast, scalable, and reliable throughput by moving data more efficiently through the server.

2.3.2 Intel DSA Capability

Inherited from Intel Quick Data Technology, Intel DSA is high-performance memory copy and data transformation accelerator for optimizing streaming data movement and transformation operations common with applications for high-performance storage, networking, persistent memory, and various data processing applications.

The goal is to provide higher overall system performance for data mover and transformation operations, while freeing up CPU cycles for higher level functions. Intel DSA enables high performance data mover capability to/from volatile memory, persistent memory, memory mapped I/O, and through a Non-Transparent Bridge (NTB) device to/from remote volatile and persistent memory on another node in a cluster. The following table showcases Intel DSA capabilities.

Table 3. Intel DSA Capability Summary

Category	Features	Description
Accelerator Interfacing	Shared Virtual Memory	DSA can operate in CPU virtual address space, and it supports recoverable page-faults
	Work Queues	Work queues are configurable to operate in dedicated or shared mode
Device Shareability	User mode support	Supports both user-mode and kernel-mode clients
	Virtualization support	Shareability of device from clients
Performance	Bandwidth	~30 GB bandwidth
	QoS	QoS support to provision and manage device operation
	Caching Controls	Can control cache (LLC) allocation on memory write and cache flush operations

Intel DSA supports Shared Virtual Memory (SVM) operation, allowing the device to operate directly in the application's virtual address space without requiring pinned memory. In addition, it can support memory overlap for the source and destination addresses. There is no restriction for memory alignment, which makes software possible to accelerate any type of memory copy.

2.3.3 DPDK DMA Library

The DSA device setup process in user space is through configurable sub-component work queues, groups, and engines. Work queues are on-device storage to contain descriptors that have been submitted to the device.

The DPDK DMA library provides a DMA device framework for hardware DMA poll mode driver integration and generic APIs to DMA operations. For each DMA device, each hardware DMA queue can be represented as a dmadev. A dmadev can create multiple virtual DMA channels, each corresponding to a different transfer context.

Table 4. DPDK DMA Library APIs

API	Interface Usage
`rte_dma_pmd_allocate`	Dynamically allocates a DMA channel
`rte_dma_info_get`	Queries device info and supported features
`rte_dma_copy`	Submits the DMA Copy request to the virtual DMA channel
`rte_dma_fill`	Submits the DMA Fill request to the virtual DMA channel
`rte_dma_submit`	Issues doorbell to hardware
`rte_dma_completed`	Gets the number of successfully completed operations
`rte_dma_completed_status`	Returns the number of completed operations along with the status of each operation
`rte_dma_stats_get`	Gets the statistics from the dmadev device

2.3.4 Kernel Work Queue

An Intel DSA device supports two types of work queues, dedicated and shared. Descriptors are submitted to a dedicated work queue via the `modir64b` instruction. This instruction is posted memory write operation and does not wait for response from the device. Since the submission of a descriptor may fail without any notice, software needs to track the number of in-flight descriptors and ensure that no overflow occurs in the hardware. Descriptors are submitted to shared work queues through the `enqcmds` instruction in the kernel. This instruction is non-posted memory write and returned after completion. Software should check the zero flag for completion status. When the value of the returned zero flag is 1, it indicates that the completion status is failure and the software needs to resubmit the request.

In the kernel, the DSA work queue working model fits into the DMA engine API. The IDXD driver supplies DMA channels that are directly mapped from DSA work queues. Callback functions in the IDXD driver prepare the descriptor and submit it to the physical work queue in async mode.

Table 5. Kernel Work Queue APIs

API	Interface Usage
`dma_request_chan`	Allocates a DMA secondary channel
`dmaengine_submit`	Submits the transaction
`dma_async_issue_pending`	Issues pending DMA requests and waits for callback notification

2.3.5 VPP DMA Infrastructure

The DMA node in VPP bridges the requests of DMA transfer and DMA hardware that are abstracted as back end devices.

In the VPP session layer, the copy operations of shared memory are converted to DMA transfer requests. Several requests are combined into one DMA transfer batch. The DMA node picks a suitable back end source for the batch. Later the DMA back end checks the batch status and does async callbacks after operations finish.

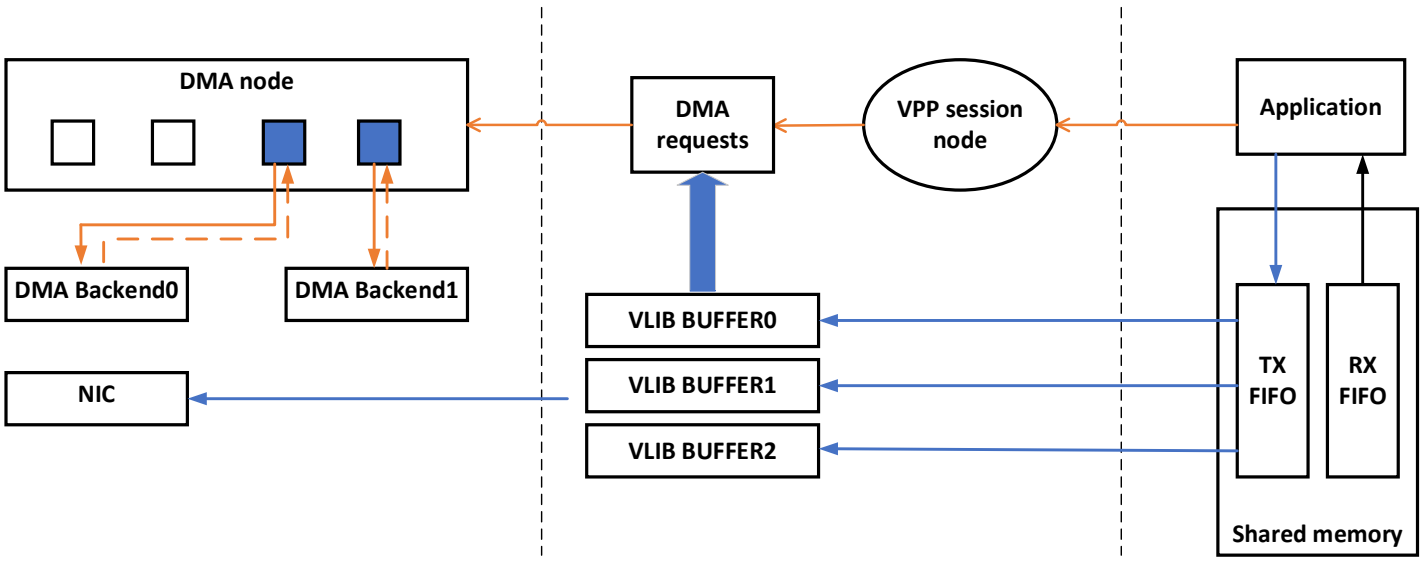

Figure 4. DMA Acceleration Working Model in VPP

3 Deployment and Benefits

We set up a test environment for HTTP throughput test. The client and server are connected with two 100 Gbps network ports. The HTTP server is based on NGNIX, and the TCP/IP stack is configurable. We use VPP default SW stack and VPP SW enhanced by Intel DSA to compare the performance improvement. The client part is multiple wrk instances based on kernel space, and the client configuration is kept the same for the different stacks.

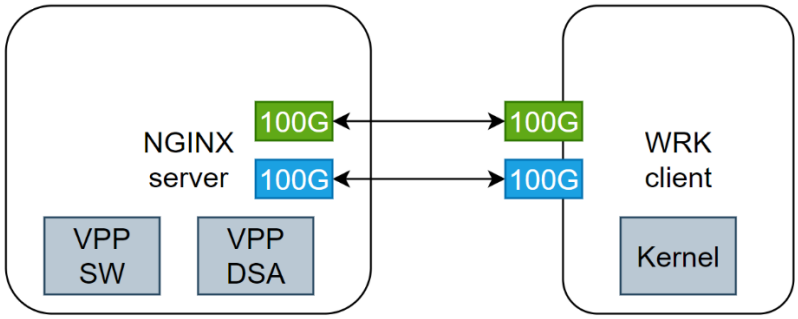

Figure 5. Test Setup Topology

Table 6. Test Configuration

Item	Description
Baseboard	Intel Corporation Archer City
Chassis	Rack Mount Chassis
CPU Model	Intel® Xeon® Platinum 8471N
Microarchitecture	SPR
Cores per Socket	52
Intel Turbo Boost	Disabled
Base Frequency	1.8 GHz
All-core Maximum Frequency	2.8 GHz
Maximum Frequency	3.6 GHz
NUMA Nodes	1
Prefetchers	L2 HW, L2 Adj., DCU HW, DCU IP
Accelerators	DSA:4
Installed Memory	256 GB (8x32 GB 4800 MT/s)
Hugepagesize	1048576 KB
Network Interface Card	2x Ethernet Controller E810-C for QSFP
Disk	1x 223.6 G Intel SSDSC2KB240G8, 1x 240M Disk
BIOS	EGSDCRB1.86B.0079.D34.2205030418
Microcode	0x890000a0
OS	Ubuntu 22.04 LTS
Kernel	5.15.0-27-generic
DDP OS Package	1.3.27.0
VPP Version	v22.02.0

On the server side, a total of five cores are utilized. VPP and NGINX are independent processes, and they could be pinned to CPU cores. In our testing, two cores are configured to be pinned for VPP and three cores are pinned for NGINX. VPP and NGINX use shared memory for data plane (FIFO) and control plane (message queue) to implement the host stack interface. For the Intel DSA setting, we start one instance and use one engine for the instance, one work queue per one VPP process, and ATS enabled per work queue. The Intel DSA memory copy batch size is set to be 32. To reduce the impact of file I/O operation, we configure NGINX to directly return a memory-based JSON file. Refer to below performance results for reference.

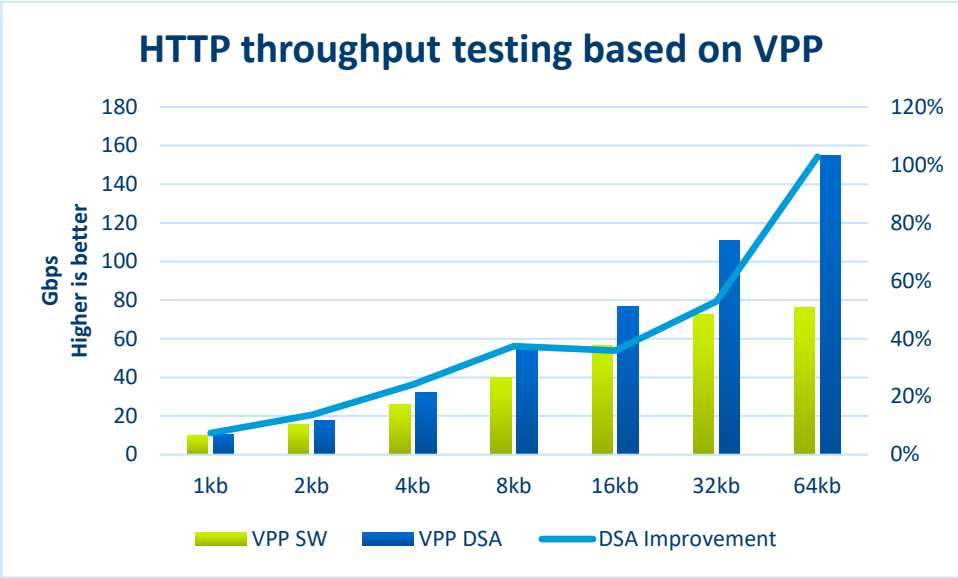

Figure 6. HTTP Throughput Performance Comparison for Intel DSA Improvement

Compared with previous performance CPU usage, we can see that with Intel DSA acceleration, the ratio of memory copy is greatly reduced, which explains the performance benefit from Intel DSA acceleration.

```
   PerfTop:    3856 irqs/sec  kernel: 0.1%  exact: 100.0% lost: 0/0 drop: 0/0 [4000Hz cycles],  (all, CPU: 4)
-------------------------------------------------------------------------------------------------------------

    10.59%  libvnet.so.22.02.0      [.] session_tx_fifo_peek_and_snd
     7.05%  libvnet.so.22.02.0      [.] tcp_update_burst_snd_vars
     4.73%  dpdk_plugin.so          [.] ice_xmit_pkts
     4.33%  libvlib.so.22.02.0      [.] dsa_add_dma_transfer
     3.92%  libsvm.so.22.02.0       [.] svm_fifo_segments
     3.68%  libvnet.so.22.02.0      [.] transport_connection_reschedule
     3.65%  dpdk_plugin.so          [.] dpdk_device_class_tx_fn_icl
     3.01%  libvnet.so.22.02.0      [.] session_add_self_custom_tx_evt
     2.75%  libvnet.so.22.02.0      [.] app_send_io_evt_rx
     2.57%  libvnet.so.22.02.0      [.] session_dma_completion_cb
     2.51%  libsvm.so.22.02.0       [.] svm_fifo_dequeue_drop
     2.32%  libvnet.so.22.02.0      [.] tcp_session_push_header
     2.10%  libsvm.so.22.02.0       [.] fsh_collect_chunks
     2.00%  libsvm.so.22.02.0       [.] svm_msg_q_lock_and_alloc_msg_w_ring
     1.91%  libvnet.so.22.02.0      [.] tcp4_output_node_fn_icl
     1.87%  libvnet.so.22.02.0      [.] session_queue_node_fn
     1.86%  libvnet.so.22.02.0      [.] tcp_make_ack_i
     1.85%  libvnet.so.22.02.0      [.] tcp_rcv_ack
     1.84%  libsvm.so.22.02.0       [.] svm_msg_q_add_raw
     1.70%  libvnet.so.22.02.0      [.] ip4_lookup_node_fn_icl
     1.56%  libvnet.so.22.02.0      [.] tcp4_established_node_fn_icl
     1.46%  libvnet.so.22.02.0      [.] tcp_session_custom_tx
     1.32%  libvnet.so.22.02.0      [.] tcp_session_send_params
     1.30%  libvnet.so.22.02.0      [.] session_lookup_connection_wt4
     1.25%  libvnet.so.22.02.0      [.] tcp_segment_validate
     1.24%  libvnet.so.22.02.0      [.] tcp4_input_node_fn_icl
     1.21%  libsvm.so.22.02.0       [.] svm_fifo_enqueue
     1.11%  libvppinfra.so.22.02.0  [.] mspace_usable_size_with_delta
     1.05%  libvlib.so.22.02.0      [.] vlib_worker_loop
     0.99%  libvnet.so.22.02.0      [.] ip4_local_inline
     0.95%  libsvm.so.22.02.0       [.] svm_msg_q_sub_raw
     0.86%  dpdk_plugin.so          [.] ice_recv_scattered_burst_vec_avx512_offload
     0.85%  libvnet.so.22.02.0      [.] session_wrk_handle_mq
     0.82%  libvnet.so.22.02.0      [.] ip4_rewrite_node_fn_icl
     0.78%  libvnet.so.22.02.0      [.] ethernet_input_node_fn_icl
     0.75%  libvnet.so.22.02.0      [.] session_evt_alloc_new
     0.72%  libvnet.so.22.02.0      [.] cubic_rcv_ack
     0.71%  libsvm.so.22.02.0       [.] svm_msg_q_free_msg
     0.69%  libvnet.so.22.02.0      [.] ip4_input_no_checksum_node_fn_icl
     0.67%  dpdk_plugin.so          [.] dpdk_input_node_fn_icl
     0.64%  libvnet.so.22.02.0      [.] vnet_interface_output_node_fn_icl
     0.60%  libvnet.so.22.02.0      [.] tcp_session_get_transport

[vpp]  1:vpp  2:dpdk  3:zsh  4:zsh
```

Figure 7. Screenshot Showing 32 KB Test Results

4 Summary

This guide demonstrates how Intel DSA can increase host stack performance by accelerating memory copy, giving the CPU more cycles to process the TCP/IP stack.

This guide analyzes host stack performance based on NGINX and wrk benchmarks and identifies memory copy as a key bottleneck for performance improvement. It then introduces Intel DSA hardware functionality and software API definition in DPDK and kernel. This guide also describes how VPP leverages Intel DSA memory copy capability based on back end and config design. Finally, this guide provides the benchmark and performance gain based on Intel DSA acceleration.

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

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.

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.

0123/DN/WIPRO/PDF 739269-002US