Technology Guide

Intel® Data Streaming Accelerator (DSA) – Offloading Memory Copying in VPP by Shared Memory Packet Interface (memif) Plugin with Intel® DSA on 4th Gen Intel® Xeon® Scalable Processors

Authors

Chenxi Wang

Ai Bee Lim

Timothy Miskell

Abhijit Sinha

Key Contributors

Michael Luo

Jing Zhao

Alex Wang

Ping Yu

Sunny Wang

Xiaobing Qian

1 Introduction

Intel® Data Streaming Accelerator (DSA) is a high-performance data copy and transformation accelerator integrated in 4th Gen Intel® Xeon® Scalable Processors and future Intel® processors, targeted for optimizing streaming data movement and transformation operations common with applications for high-performance storage, networking, persistent memory, and various data processing applications.

Shared memory packet interface (memif) provides high performance packet transmit and receive between user applications. Vector Packet Processing (VPP) can utilize DSA work queues created from the Linux IDXD kernel driver to accelerate memory copy on the primary side of memif.

This document outlines the deployment of VPP memif with Direct Memory Access (DMA) option to accelerate memory copying. Through comparing the benchmarking results for various packet sizes with MTU 1500 and MTU 9000 between Software Mode (without Intel® DSA) and Hardware Mode (with Intel® DSA), there is up to 1.9x throughput improvement with 0.001% packet loss rate on 4th Gen Intel ® Xeon® Scalable Processor with Intel® E810 Network Controllers.

This document is intended for VPP community to start the evaluation of Intel® DSA.

This document is part of the Network Transformation Experience Kits.

Table of Contents

1 Introduction........1
1.1 Terminology........3
1.2 Reference Documentation........3
2 Overview........3
2.1 Intel® Data Streaming Accelerator (DSA)........4
2.2 DMA API Support in VPP memif........4
3 Deployment and Benchmarking........5
3.1 Memory Copying Acceleration of Memif........5
3.2 System Configuration........5
3.3 BIOS Settings........6
3.4 Testing Setup........7
3.5 Testing Deployment........7
4 Testing Results........10
5 Summary........12

Figures

Figure 1. Memory Copying on Primary Side of Memif using Intel® DSA........5
Figure 2. Setup of DUT and Packet Generator........7
Figure 3. Throughput Comparison between Software Mode and Hardware Mode (MTU 1500)........12
Figure 4. Throughput Comparison between Software Mode and Hardware Mode (MTU 9000)........122

Tables

Table 1. Terminology........3
Table 2. Reference Documents........3
Table 3. System Configurations........5
Table 4. BIOS Settings........6
Table 5. Throughput Comparison between Hardware Mode and Software Mode (MTU 1500)........10
Table 6. Throughput Comparison between Hardware Mode and Software Mode (MTU 9000)........101

Document Revision History

Revision	Date	Description
001	<October 2023>	Initial release.

1.1 Terminology

Table 1. Terminology

Abbreviation	Description
BIOS	Basic input and output service
DPDK	Data Plane Development Kit
DSA	Intel® Data Streaming Accelerator (Intel® DSA)
DUT	Device Under Test
ENQCMD	An Intel® 64 CPU instruction to enqueue a command to a shared work queue using Deferrable Memory Write (DMWr)
IOMMU	I/O Memory Management Unit. A DMA Remapping Hardware Unit as defined by Intel® Virtualization Technology for Directed I/O
IDXD	Intel® Data Accelerator Driver. A typical kernel driver that identifies devices instances in a system
NIC	Network Interface Card/Controller
NUMA	Non-Uniform Memory Access
PCI	Peripheral Component Interconnect
TRex	An open source, low cost, stateful and stateless traffic generator fueled by DPDK.
memif	Shared memory packet interface. It provides high performance packet transmit and receive between user application and Vector Packet Processing or multiple user applications
MTU	Maximum transmission unit
WQ	A queue in the device used to store descriptors submitted by software until they can be dispatched.
VPP	FD.io's Vector Packet Processor (VPP) is a fast, scalable layer 2-4 multi-platform network stack. It runs in Linux Userspace on multiple architectures including x86, ARM, and Power architectures.

1.2 Reference Documentation

Table 2. Reference Documents

Reference	Source
Intel® DSA Architecture Specification	https://software.intel.com/en-us/download/intel-data-streamingaccelerator-preliminary-architecture-specification
Intel® Data Streaming Accelerator User Guide	https://www.intel.com/content/www/us/en/content-details/759709/intel-data-streaming-accelerator-user-guide.html?wapkw=DSA
Intel® Data Streaming Accelerator (DSA) - Packet Copy Offload in OVS with Intel® DSA	https://networkbuilders.intel.com/solutionslibrary/intel-data-streaming-accelerator-packet-copy-offload-ovs-technology-guide
Intel® Data Streaming Accelerator (DSA) - Accelerating DPDK Vhost	https://networkbuilders.intel.com/solutionslibrary/intel-data-streaming-accelerating-dpdk-vhost-technology-guide
Intel® Data Streaming Accelerator (DSA) - Packet Copy Offload in DPDK with Intel® DSA	https://networkbuilders.intel.com/solutionslibrary/intel-data-streaming-accelerator-packet-copy-offload-dpdk-technology-guide
Intel® Data Streaming Accelerator (DSA) - DPDK-DMA Packet Copying with Intel® DSA on 4th Gen Intel® Xeon® Scalable Processors	https://networkbuilders.intel.com/solutionslibrary/intel-data-streaming-accelerator-dsa-dpdk-dma-packet-copying-with-intel-dsa-on-4th-gen-intel-xeon-scalable-processors
Accel-config	https://github.com/intel/idxd-config
Memif library (libmemif)	https://s3-docs.fd.io/vpp/22.02/interfacing/libmemif/index.html
Trex Manual	https://trex-tgn.cisco.com/trex/doc/trex_manual.html
Release Notes for VPP 23.06	https://s3-docs.fd.io/vpp/23.06/aboutvpp/releasenotes/v23.06.html
Scalar vs Vector Packet Processing	https://s3-docs.fd.io/vpp/23.02/aboutvpp/scalar-vs-vector-packet-processing.html

2 Overview

This document outlines the deployment of VPP memif with Intel® DSA to accelerate memory copying.

Instead of the traditional VPP memory copying operations, we accelerated memory copying to Intel® DSA via the updated memif plugin of VPP 23.06 in the latest release. Comparing the benchmarking results between Software Mode (without Intel® DSA) and Hardware Mode (with Intel® DSA) of packet size ranging from 64 Bytes to 9000 Bytes, it is observed that up to 1.9x throughput improvement of 0.001% packet loss rate on the 4th Gen Intel® Xeon® Scalable Processors with Intel® E810 Network Controllers.

The use case describes a scenario where VPP utilizes DSA DWQs created from the Linux IDXD kernel driver to accelerate memory copying on the Primary side of shared memory packet interface (memif). To utilize DSA with VPP, we built the docker image with VPP installed for environment setup, initializing two containers with the image, and then mapping DSA WQs into containers. In addition, using memif plugin of VPP to support DMA option by utilizing new DMA API.

2.1 Intel® Data Streaming Accelerator (DSA)

The primary objective of Intel® DSA is to provide enhance overall system performance for data mover and transformation operations, while simultaneously freeing up CPU cycles for higher level functions. Intel® DSA hardware supports high-performance data movement capability to and from various memory types including volatile memory, persistent memory, memory-mapped I/O, and remote volatile memory on other nodes within a cluster through a Non-Transparent Bridge (NTB) in the SoC. It provides a PCI Express compatible programming interface to the Operating System and can be controlled through a device driver.

In addition to performing basic data movement operations, Intel® DSA is designed to perform higher-level transformation operations on memory. For example, it can generate and test CRC checksum or Data Integrity Field (DIF) on the memory region to support usages typical with storage and networking applications. It also supports a memory compare operation for equality, generates a delta record, and applies a delta record to a buffer. These are compared and the delta generate/merge functions are utilized by applications such as VM migration, VM fast check-pointing, and software managed memory deduplication usages.

Intel® DSA also facilitates data movement between different address spaces by using the Inter-Domain capabilities of the device. This may have applications in networking, for example with a virtual switch implementation to efficiently copy data between virtual machines or to speed up inter-process communication (IPC) primitives in the OS or VMM. Additionally, it may also be used in message and data passing between processes in application domains like High Performance Computing (HPC) and Machine Learning.

2.2 DMA API Support in VPP memif

FD.io's Vector Packet Processor (VPP) stands as a fast, scalable layer 2-4 multi-platform network stack. It operates in Linux userspace supporting multiple architectures including x86, ARM, and Power architectures. VPP's high performance network stack is becoming the preferred choice for applications around the world. The VPP framework demonstrates remarkable efficiency by leveraging the scalability of contemporary Intel® processors while efficiently managing packet processing through batch operations, commonly referred to as vectors, with a capacity of handling up to 256 packets at a time. There is continuous enhancement in VPP through the use of plugins.

The shared memory packet interface (memif) provides seamless packet transmission and reception between user applications and Vector Packet Processing (VPP). By using *libmemif*, user applications can create shared memory interface in primary or secondary mode and then connect to. Once the connection is established, user application can receive or transmit packets based on *libmemif* API.

The release of VPP 23.06 introduced a new feature to allow memif support DMA option. Since secondary side already support zero copy mode, DMA option is only added in primary side. Based on this new feature of VPP, we utilize the created WQs of Intel® DSA to accelerate memory copy on primary side of memif via DMA functionality.

3 Deployment and Benchmarking

3.1 Memory Copying Acceleration of Memif

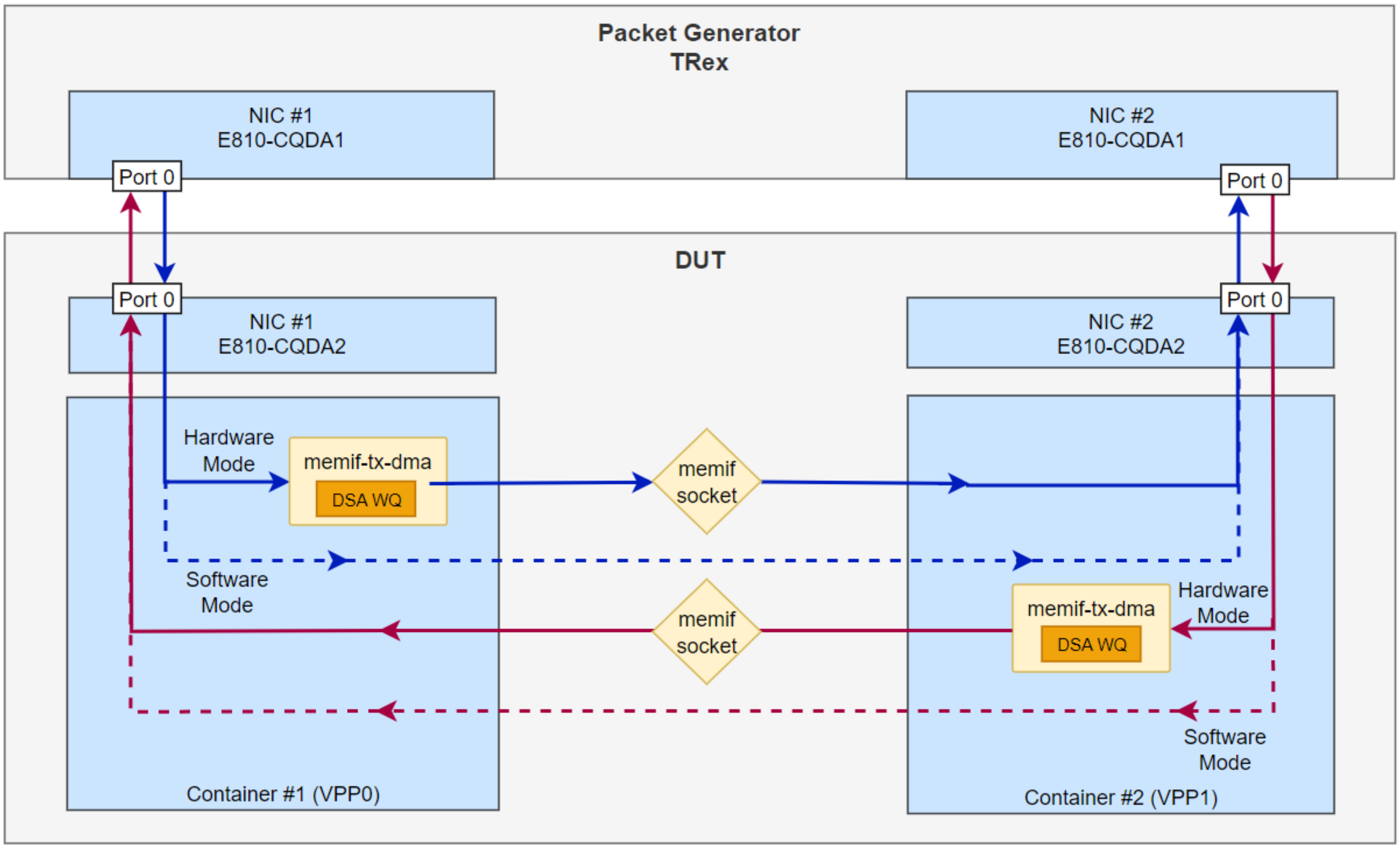


Figure 1. Memory Copying on Primary Side of Memif using Intel® DSA

Note: Blue arrows and Red arrows represent the traffic flows of VPP0 and VPP1, respectively. DSA acceleration only available on the primary side. The benchmarking results are collected by sum of ports of NIC#1 and NIC#2 on Packet Generator.

3.2 System Configuration

To perform the VPP memif memory copying benchmarking on the 4th Gen Intel® Xeon® Scalable Processors, refer the system configuration of DUT as detailed in Note: Please ensure that the kernel version is upgraded and compatible with the accel-config version. More details should consult OS provider for official support.

Table 3.

For Intel® IOMMU driver, it needs to be enabled in the kernel configuration. The Intel® IOMMU driver can also be compiled into the kernel, but not be enabled by default. In both cases, add the following boot parameter to the grub file:

```
intel_iommu=on,sm_on
```

Note: Please ensure that the kernel version is upgraded and compatible with the accel-config version. More details should consult OS provider for official support.

Table 3. System Configurations

System Configuration	
Baseboard	Jabil 2U8 Midplane
Chassis	Rack Mount Chassis
CPU Model	Intel® Xeon® Gold 6438N
Microarchitecture	SPR

System Configuration	
Sockets	1
Cores per Socket	32
Hyperthreading	Enabled
CPUs	64
Intel Turbo Boost	Disabled
Base Frequency	2.0GHz
All-core Maximum Frequency	2.7GHz
Maximum Frequency	2.0GHz
NUMA Nodes	1
Prefetchers	L2 HW, L2 Adj., DCU HW, DCU IP
PPINs	b964bc27550ac01c
Accelerators	DLB:2, DSA:1, IAX:0, QAT (on CPU):2, QAT (on chipset):0
Installed Memory	128GB (8x16GB DDR5 4800 MT/s [4800 MT/s])
Hugepagesize	1048576 kB
Transparent Huge Pages	madvise
Automatic NUMA Balancing	Disabled
NIC	1x I210 Gigabit Network Connection, 2x Ethernet Controller E810-C for QSFP, 1x Ethernet interface
NIC Firmware	4.20 0x80017784 1.3346.0
Disk	1x 894.3G WUS3BA196C7P3E3, 1x 931.5G INTEL SSDPELKX010T8
BIOS	a2101g
Microcode	0x2b000161
OS	Ubuntu-Server 22.04.1 2022.09.21 (Cubic 2022-09-21 16:19)
Kernel	5.15.0-43-generic
TDP	205 watts
Power & Perf Policy	Performance
Frequency Governor	powersave
Frequency Driver	intel_pstate
Max C-State	9

3.3 BIOS Settings

To enable Intel® DSA and tuning the DUT configuration, refer to the BIOS settings as indicated in Table 4.

Table 4. BIOS Settings

BIOS Menu	Sub Menu	Sub Menu 2	Setting
IIO Configuration	Port configuration		**x8x8 for all ports**
	Intel VT for Directed IO		**Enable**

BIOS Menu	Sub Menu	Sub Menu 2	Setting
	PCIe ENQCMD/ENQCMDS		**Yes**
Advanced Power Management Configuration	CPU P state Control	Speed Step	**Enable**
		AVX License Pre-Grant	**Disable**
		AVX ICCP pre-grant level	**N/A**
	Hardware PM State Control	Hardware P-States	**Native Mode w/o Legacy Support**
	CPU C state Control	CPU C1 auto demotion	**Disable**
		CPU C1 auto undemotion	**Disable**
	Package C State Control	Package C State	**C0/C1 State**
	CPU Advanced PM Tuning	Uncore Freq Scaling	**Disable**
		Uncore Freq RAPL	**Disable**
	Energy perf bias	Power Performance Tuning	**BIOS Controls EPB**
		ENERGY_PERF_BIAS_CFG Mode	**Performance**
		Workload Configuration	**I/O Sensitive**

3.4 Testing Setup

As shown in Figure 2, the network adapter is 100GbE E810-CQDA2 with dual ports, up to 200 Gbps for bandwidth-intensive workloads. More importantly, to saturate the performance of Intel® DSA device, we are adopting one port of each NIC mentioned above, obtaining a total 200 Gbps workload for memory copying application with only one Intel® DSA device.

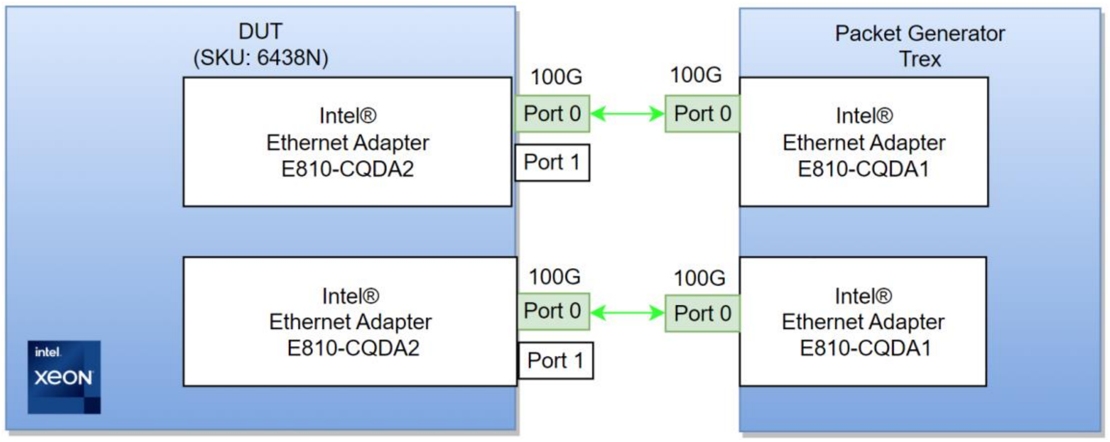


Figure 2. Setup of DUT and Packet Generator

3.5 Testing Deployment

1. Build VPP docker image with VPP installed.

```
# cd /home/user/dockerfile
# DOCKER_BUILDKIT=1 docker build --progress=plain -t memif_dsa:v0.1 -f
Dockerfile_vpp_memif_dsa . --no-cache
```

Example of Dockerfile_vpp_memif_dsa

```
# syntax = docker/dockerfile:1
FROM ubuntu:20.04

ARG HTTP_PROXY=XXXX
ARG HTTPS_PROXY=XXXX

ENV http_proxy=$HTTP_PROXY \
    https_proxy=$HTTPS_PROXY \
    TERM=xterm \
    DEBIAN_FRONTEND=noninteractive

RUN mkdir -p /opt
WORKDIR /opt
# install vpp
RUN apt-get update && apt-get install -y \
          apt-utils iproute2 ssh wget net-tools ethtool \
          git gcc automake autoconf libtool make wget python3 \
          bison byacc check chrpath flex sudo meson nasm python3-pip \
          psmisc lsof iperf vim && \
pip3 install --no-cache-dir pyelftools && \
git clone https://github.com/FDio/vpp.git && \
cd vpp && git fetch https://gerrit.fd.io/r/vpp refs/changes/77/38477/1 && git checkout -b
change-38477 FETCH_HEAD && \
sh -c '/bin/echo -e "y\ny" | ./extras/vagrant/build.sh' && \
dpkg -i build-root/*.deb && cd /opt && rm -rf vpp && \
rm -rf /var/lib/apt/lists/* && apt clean && mkdir -p /run/vpp && mkdir -p /var/log/vpp
```

2. Create two WQs from DSA device DSA0, and these WQs will be used by two VPP instances, one WQ for each VPP container. Then bind two NIC PF ports (0000:6f:00.0 and 0000:43:00.0) with vfio-pci by the DPDK script dpdk-devbind.py. The script "dpdk_idxd_cfg.py" is downloaded from https://github.com/DPDK/dpdk, and the script is located at drivers/dma/idxd.

```
# ./dpdk_idxd_cfg.py -q 2 --name-prefix memif 0
# ls /dev/dsa/
wq0.0  wq0.1
# cat /sys/bus/dsa/devices/dsa0/wq0.0/name
memif_wq0.0
# cat /sys/bus/dsa/devices/dsa0/wq0.1/name
memif_wq0.1
# ./dpdk-devbind.py -b vfio-pci 0000:6f:00.0 0000:43:00.0
```

3. VPP Configuration
Update the dpdk section in startup_p0.conf and startup_p1.conf with the BDF of NIC ports which were connected to Trex. Also, update the DSA section in startup_p0.conf and startup_p1.conf with the created wq0.0 and wq0.1, respectively.

```
## Set interface name
Dev 0000:6f:00.0 {
       name eth0
}

dsa {
        ## DSA work queue address. One work queue for each VPP container.
        dev wq0.0
}
```

4. Start VPP in Containers
Start two containers with the built image memif_dsa:v0.1, mapping DSA WQs and VPP config files into containers.

```
# docker run -dt --rm --name=vpp0 --privileged \
                  -v /dev/hugepages:/dev/hugepages \
                  -v /dev/vfio:/dev/vfio \
                  -v /lib/firmware:/lib/firmware \
                  -v /sys/bus/dsa/devices:/sys/bus/dsa/devices \
                  -v /dev/dsa:/dev/dsa \
                  -v /tmp/vpp/socket:/run/vpp/socket \
                  -v /home/npx/vpp_config/config:/opt/vpp/config \
                  --hostname vpp0 \
                  memif_dsa:v0.1 \
                  /bin/bash
# docker run -dt --rm --name=vpp1 --privileged \
                  -v /dev/hugepages:/dev/hugepages \
                  -v /dev/vfio:/dev/vfio \
                  -v /lib/firmware:/lib/firmware \
                  -v /sys/bus/dsa/devices:/sys/bus/dsa/devices \
                  -v /dev/dsa:/dev/dsa \
                  -v /tmp/vpp/socket:/run/vpp/socket \
                  -v /home/npx/vpp_config/config:/opt/vpp/config \
                  --hostname vpp1 \
                  memif_dsa:v0.1 \
                  /bin/bash
```

VPP0

```
# docker exec -it vpp0 /bin/bash
root@vpp0:/opt# vpp -c /opt/vpp/config/startup_p0.conf
perfmon              [warn  ]: skipping source 'intel-uncore' - intel_uncore_init: no uncore
units found

    _______    _        _   _____  ___ 
 __/ __/ _ \  (_)__    | | / / _ \/ _ \
 _/ _// // / / / _ \   | |/ / ___/ ___/
 /_/ /____(_)_/\___/   |___/_/  /_/    
vpp#
```

VPP1

```
# docker exec -it vpp1 /bin/bash
root@vpp1:/opt# vpp -c /opt/vpp/config/startup_p1.conf
perfmon              [warn  ]: skipping source 'intel-uncore' - intel_uncore_init: no uncore
units found

    _______    _        _   _____  ___ 
 __/ __/ _ \  (_)__    | | / / _ \/ _ \
 _/ _// // / / / _ \   | |/ / ___/ ___/
 /_/ /____(_)_/\___/   |___/_/  /_/    

vpp#
```

5. Check the interface and DSA status (Hardware Mode only) in VPP0 and VPP1.

VPP0

```
vpp# show int addr
eth0 (up):
  L3 192.168.10.10/24
local0 (dn):
memif10/0 (up):
  L3 10.10.10.10/24
memif11/0 (up):
  L3 20.20.20.20/24
vpp# show dma
Config 0 thread 0 dma 0/0 request 0                hw 0                cpu 0
Config 0 thread 1 dma 0/0 request 0                hw 0                cpu 0
```

VPP1

```
vpp# show int addr
eth0 (up):
  L3 192.168.20.20/24
local0 (dn):
memif10/0 (up):
  L3 10.10.10.11/24
memif11/0 (up):
  L3 20.20.20.21/24
vpp# show dma
Config 0 thread 0 dma 0/1 request 0                hw 0                cpu 0
Config 0 thread 1 dma 0/1 request 0                hw 0                cpu 0

vpp# ping 10.10.10.10
116 bytes from 10.10.10.10: icmp_seq=1 ttl=64 time=1.0644 ms
116 bytes from 10.10.10.10: icmp_seq=2 ttl=64 time=4.0647 ms
116 bytes from 10.10.10.10: icmp_seq=3 ttl=64 time=8.0628 ms
116 bytes from 10.10.10.10: icmp_seq=4 ttl=64 time=1.0626 ms
116 bytes from 10.10.10.10: icmp_seq=5 ttl=64 time=5.0615 ms

Statistics: 5 sent, 5 received, 0% packet loss
```

6. Generate Traffic from Trex and Measure the Throughput. Start trex with the port connected with DUT. Using TRex to generate traffic with destination IP (e.g. 192.168.0.2) to the DUT port0, and destination IP (e.g. 192.168.1.2) to the DUT port1.

```
## Sample output
....
packets lost from 0 --> 1: 86314597 pkts
packets lost from 1 --> 0: 86318782 pkts
total_received=946966840; total_sent=1119600219; frame_loss=172633379; target_duration=60;
All Ports TX Throughput (Mpps): 18.88
All Ports TX_L1 Throughput (Gbps): 157.72
All Ports TX Throughput (Gbps): 154.69
All Ports RX Throughput (Mpps): 15.92
All Ports RX_L1 Throughput (Gbps):  132.97
All Ports RX Throughput (Gbps): 130.42
```

4 Testing Results

The Software Mode and Hardware Mode follow the same metrics. In VPP configuration file, it is necessary to add 'use-dma' while creating memif interface to enable Hardware Mode with DSA. In addition, we performed extra testing for MTU 9000, compared with MTU 1500 to demonstrate the throughput improvement for jumbo packets transmission.

- VPP core: 1 core for main thread, 1 core for worker thread
- DSA config: 1 device, 1 WQ for each VPP container
- DSA WQs: 1
- RX queues: 1
- TX queues: 1
- Packet loss rate: 0.001%
- Hugepagesize: 1G
- Copy mode: SW/HW
- Turbo: Disabled
- TRex core amount: 4

As shown in Table 5, based on the benchmarking results for MTU 1500, the throughput in Hardware Mode (with Intel® DSA) outperforms the Software Mode when the packet size equal to or greater than 128 Bytes. We selected the packet size ranging from 64 Bytes to 1500 Bytes for typical utilization.

Table 5. Throughput Comparison between Hardware Mode and Software Mode (MTU 1500)

Packet Size (Bytes)	Software Mode (Gbits/s)	Hardware Mode (Gbits/s)
64	9.99	9.95
128	13.97	17.95
256	23.98	33.94
512	37.94	63.88
1024	47.77	131.69
1500	63.85	185.37

Additionally, Hardware Mode (with Intel® DSA) takes advantages of the throughput under MTU 9000 as well for jumbo packets transmission, as shown in Table 6. In particular, we observed that NICs reached the first point of fully utilizing for our DSA scenario with packet size 1800 Bytes.

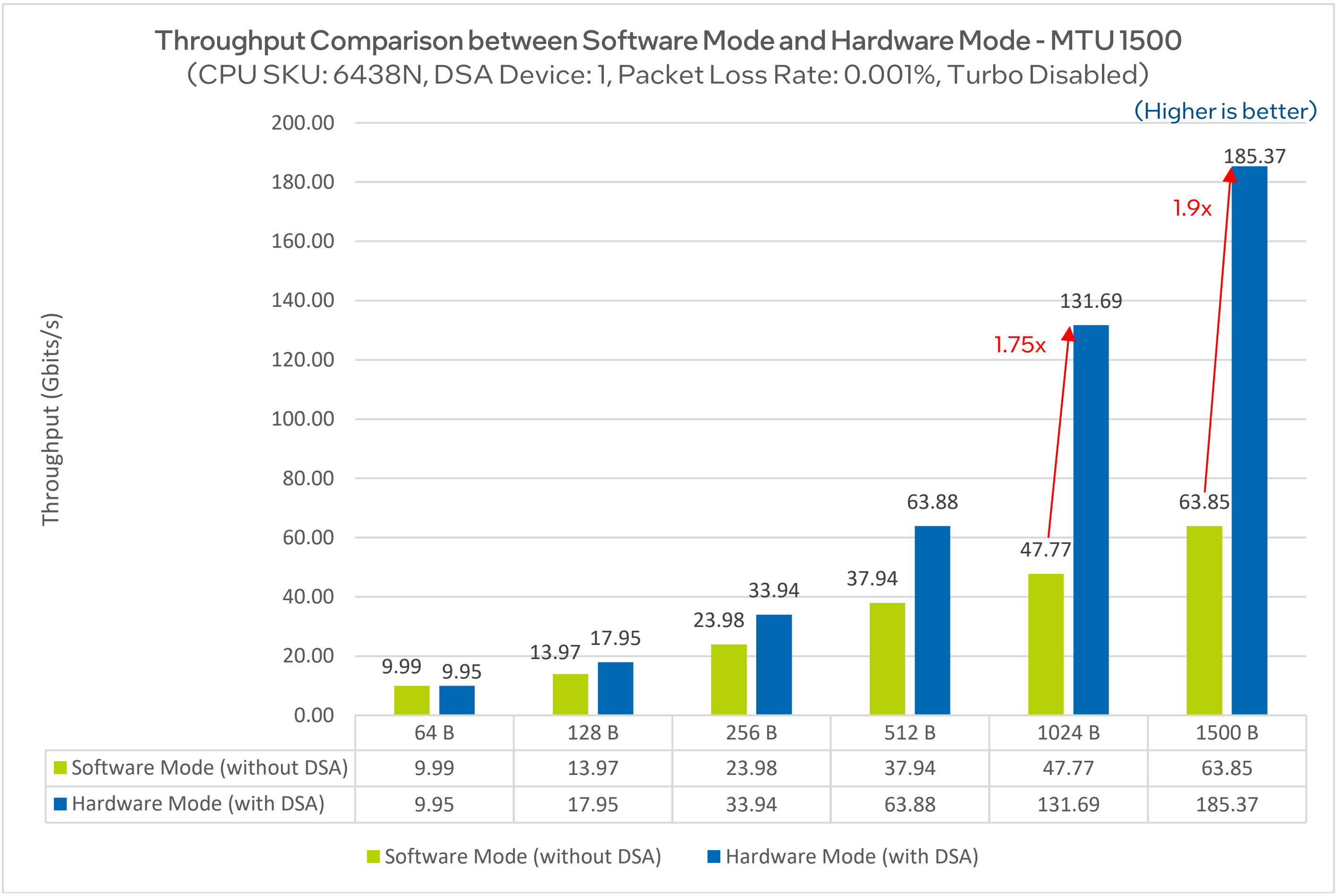
Table 6. Throughput Comparison between Hardware Mode and Software Mode (MTU 9000)

Packet Size (Bytes)	Software Mode (Gbits/s)	Hardware Mode (Gbits/s)
64	9.98	9.95
128	13.98	17.96
256	23.96	34.02
512	37.94	65.64

Packet Size (Bytes)	Software Mode (Gbits/s)	Hardware Mode (Gbits/s)
1024	47.92	131.64
1500	63.90	185.58
1800	69.85	199.53
9000	115.74	199.65

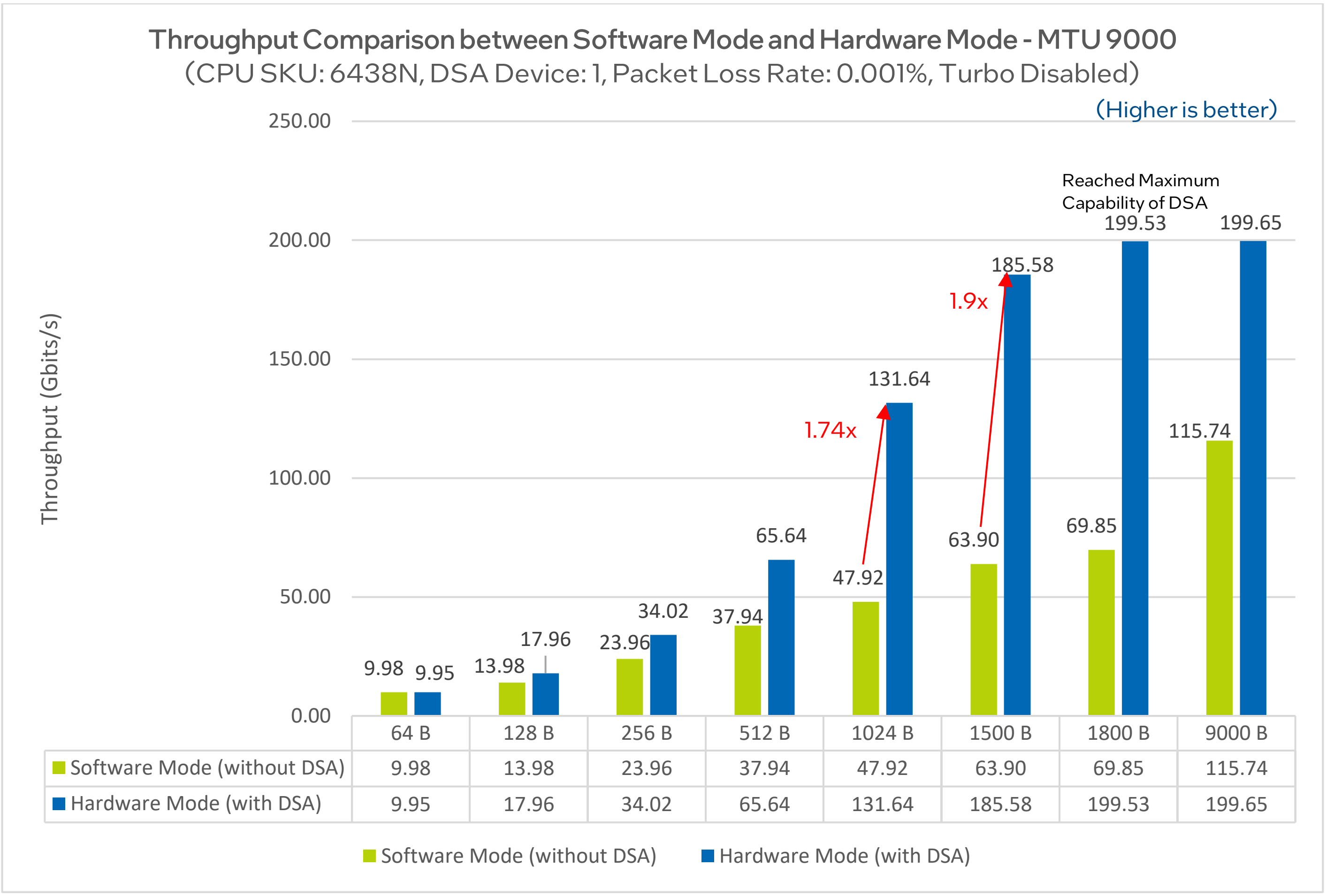
As per the comparisons in

Figure and Error! Reference source not found., it is observed that when the packet size gradually increases, the throughput of the Hardware Mode increases sharply compared with that of the Software Mode, and gradually widens the huge gap. When the packet size increased to 1024 Bytes, up to 1.75x throughput improvement by Intel® DSA acceleration on 4th Gen Intel ® Xeon® Scalable Processor with Intel® E810 Network Controllers.



	64 B	128 B	256 B	512 B	1024 B	1500 B
🟩 Software Mode (without DSA)	9.99	13.97	23.98	37.94	47.77	63.85
🟦 Hardware Mode (with DSA)	9.95	17.95	33.94	63.88	131.69	185.37

Figure 3. Throughput Comparison between Software Mode and Hardware Mode (MTU 1500)



	64 B	128 B	256 B	512 B	1024 B	1500 B	1800 B	9000 B
🟩 Software Mode (without DSA)	9.98	13.98	23.96	37.94	47.92	63.90	69.85	115.74
🟦 Hardware Mode (with DSA)	9.95	17.96	34.02	65.64	131.64	185.58	199.53	199.65

Figure 4. Throughput Comparison between Software Mode and Hardware Mode (MTU 9000)

5 Summary

In conclusion, our deployment and benchmarking of VPP memif plugin for memory copying have demonstrated the value proposition of Intel® DSA on 4th Gen Intel® Xeon® Scalable Processors. The memif device driver of VPP 23.06 supports DMA option by utilizing new DMA API. Based on the benchmarking results, we compared the performance of Hardware Mode (with Intel® DSA) or Software Mode (without Intel® DSA) for various packet sizes with MTU 1500 and MTU 9000.

The benchmarking results showed that Hardware Mode (with Intel® DSA) consistently outperformed Software Mode (without Intel® DSA) in terms of throughput when the packet size is equal to or greater than 128 Bytes. As the packet size increases, the throughput of Intel® DSA accelerated copying also increases sharply, the throughput improvement reaching up to 1.9x over software memory copying.

1023/ABL/Wipro/PDF 784198-001US