Flash Memory Summit

Optimizing NVMe-over-Fabrics using NVMe CMBs and Accelerators

Andrew Maier, Software Engineer, Eideticom

Dr. Stephen Bates, CTO, Eideticom

Outline

1. Introduction to NVMe Acceleration and NoLoad™
2. Integration of Accelerators into NVMe over Fabrics (NVMe-oF)
3. Acceleration via NVMe-oF Example
4. NVMe-oF Target/Server CPU Offloading
5. Peer-to-Peer Transfers using NVMe-oF Offload

NoLoad™

- Eideticom's NoLoad™ leverages the NVMe standard to present FPGA Accelerators as NVMe namespaces

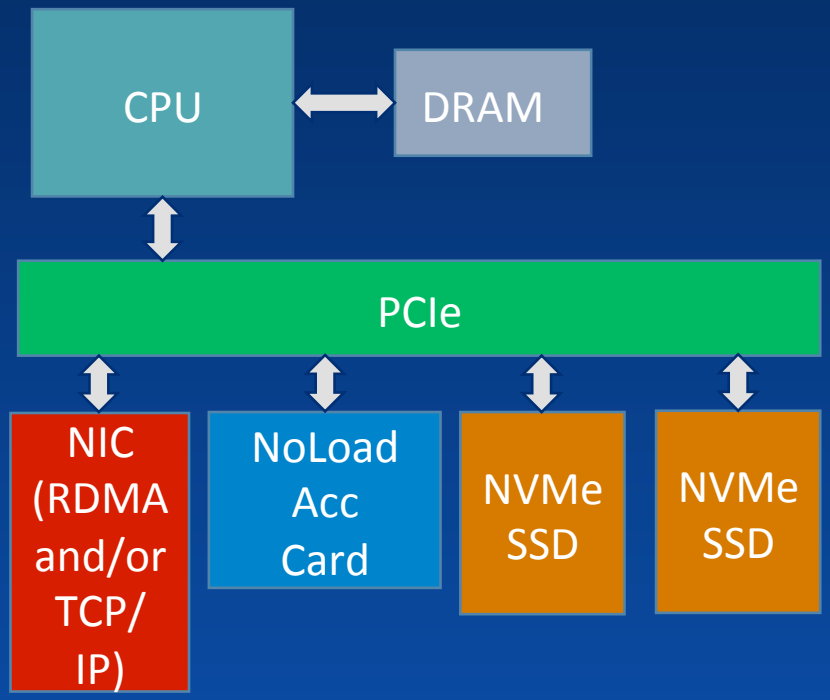
NoLoad Bitfiles

U.2 FPGA Card

COTS PCIe FPGA Card

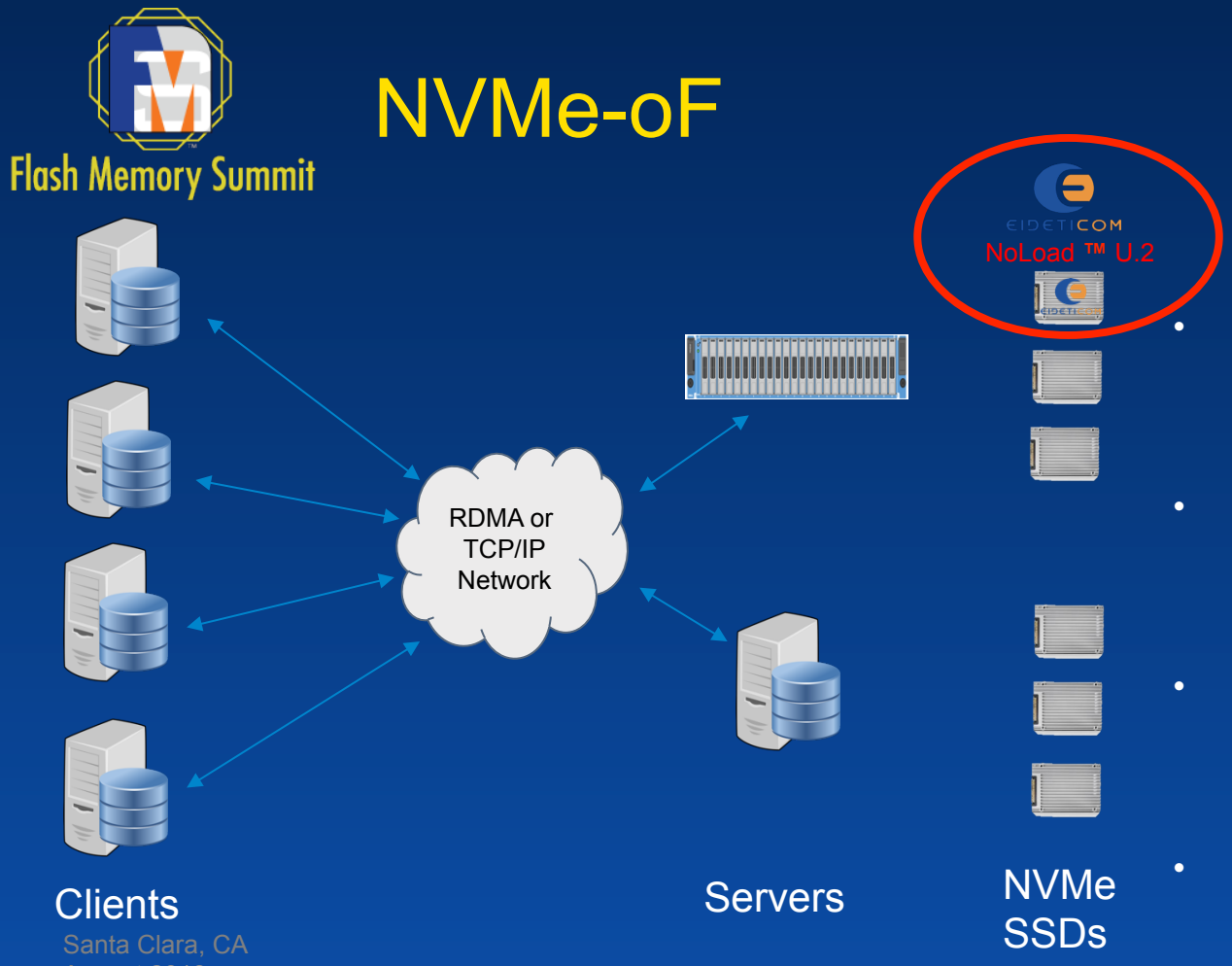
Cloud Servers i.e. Amazon F1

Introduction to NVMe Acceleration



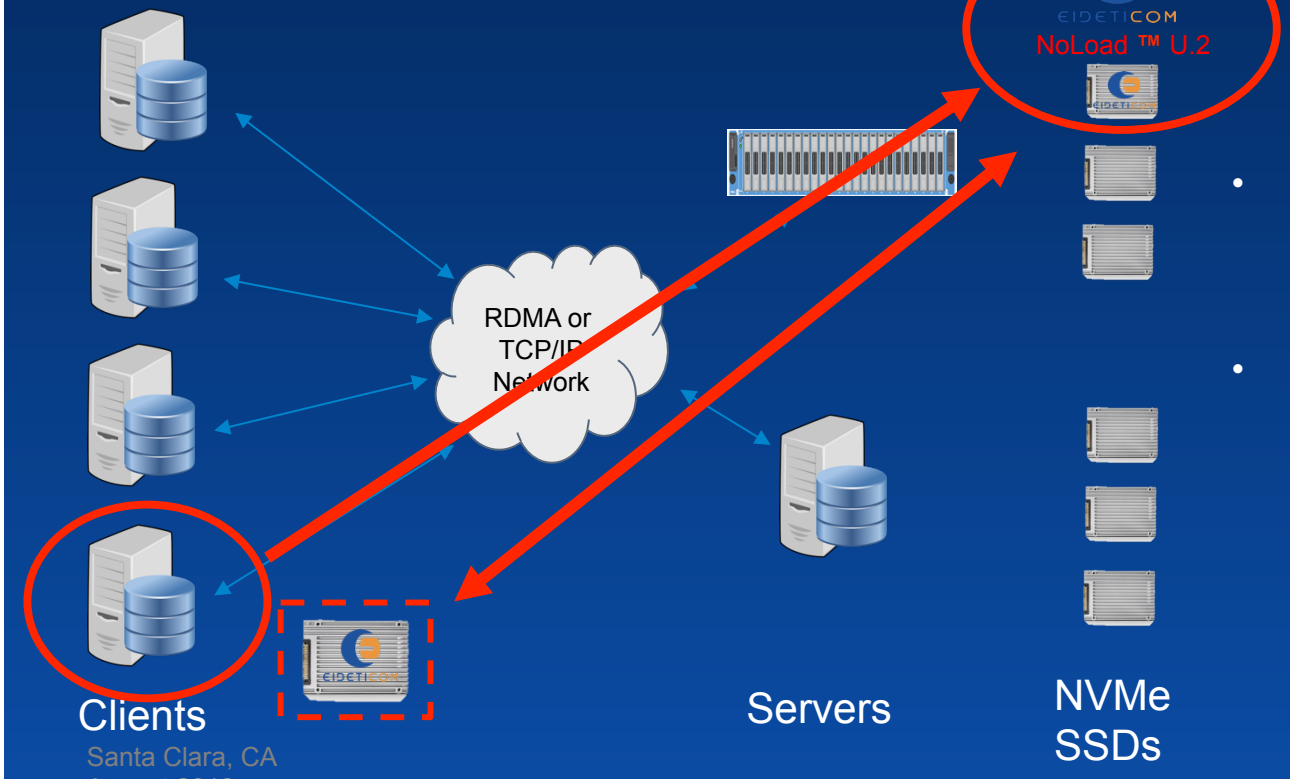
- Why NVMe?
  - NVMe is a low latency, high throughput, low CPU overhead transfer protocol
  - Usage of built-in and industry-standard drivers and tools
    - Why build and maintain a proprietary driver?
  - Ability to use the emerging NVMe over Fabrics ecosystem for storage (and accelerator) disaggregation

NVMe-oF



- NVMe over Fabrics (NVMe-oF) allows namespaces to be shared across existing networks
- Using built-in drivers, we expose the NVMe namespaces to client machines
- Since NoLoad is a standard namespace. It can shared in the same way!
- So how does it work?

NVMe-oF



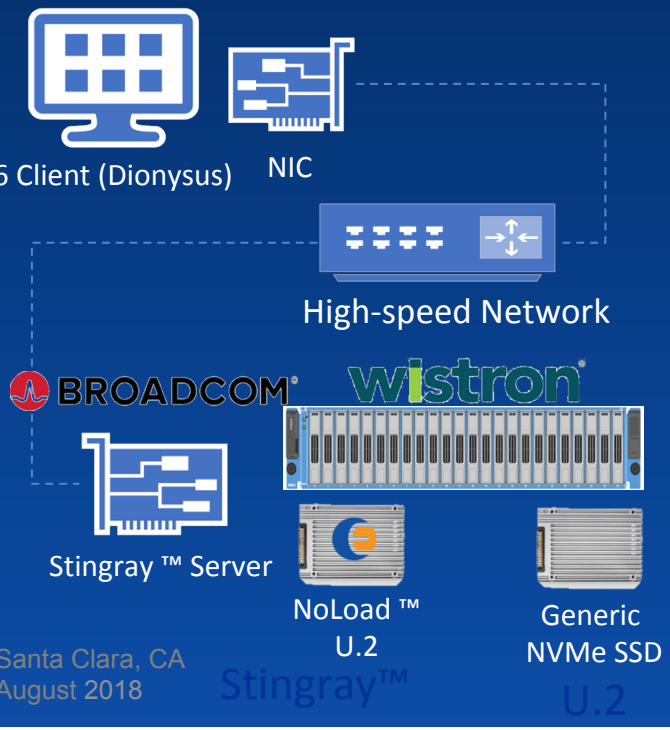
- Clients request to borrow a namespace(s) (or accelerators) from the server.
- Client is given access to the namespace over the connection.

NVMe-oF

```
amaier@dionysus:~$ sudo nvme list
Node             SN                   Model                                    Namespace Usage                      Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- -------------------------- ---------------- --------
/dev/nvme1n1     2018-03-05-0001      Eideticom NoLoad Accelerator Alpha       1           0.00   B /   2.15  GB    512   B +  0 B   1.7.2719
/dev/nvme1n2     2018-03-05-0001      Eideticom NoLoad Accelerator Alpha       2           0.00   B /   2.10  MB    512   B +  0 B   1.7.2719
/dev/nvme1n3     2018-03-05-0001      Eideticom NoLoad Accelerator Alpha       3           0.00   B /   8.59  GB    512   B +  0 B   1.7.2719
/dev/nvme1n4     2018-03-05-0001      Eideticom NoLoad Accelerator Alpha       4           0.00   B /   8.59  GB    512   B +  0 B   1.7.2719
/dev/nvme1n5     2018-03-05-0001      Eideticom NoLoad Accelerator Alpha       5           0.00   B /   8.59  GB    512   B +  0 B   1.7.2719
/dev/nvme1n6     2018-03-05-0001      Eideticom NoLoad Accelerator Alpha       6           0.00   B /   8.59  GB    512   B +  0 B   1.7.2719
/dev/nvme2n1     289d9c51523fb07c     Linux                                    1         750.16  GB / 750.16  GB    512   B +  0 B   4.14.49-
/dev/nvme2n2     289d9c51523fb07c     Linux                                    2          97.00  GB /  97.00  GB    512   B +  0 B   4.14.49-
```

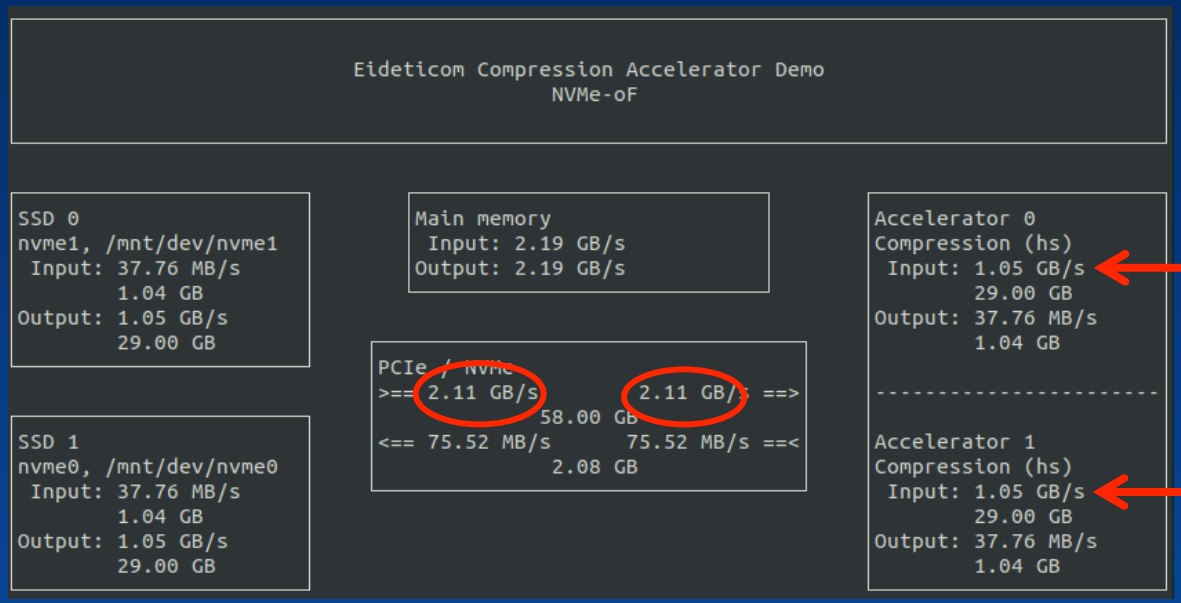
- Clients then see the newly acquired namespaces as local NVMe block devices
- Normal NVMe operations can then be executed as if it were locally in the client machine
- With the latest (soon to be upstreamed) NVMe over Fabrics passthru patches from Chaitanya Kulkarni, the client has access to all vendor specific functionality as well.

Case Study: Compression over Fabrics



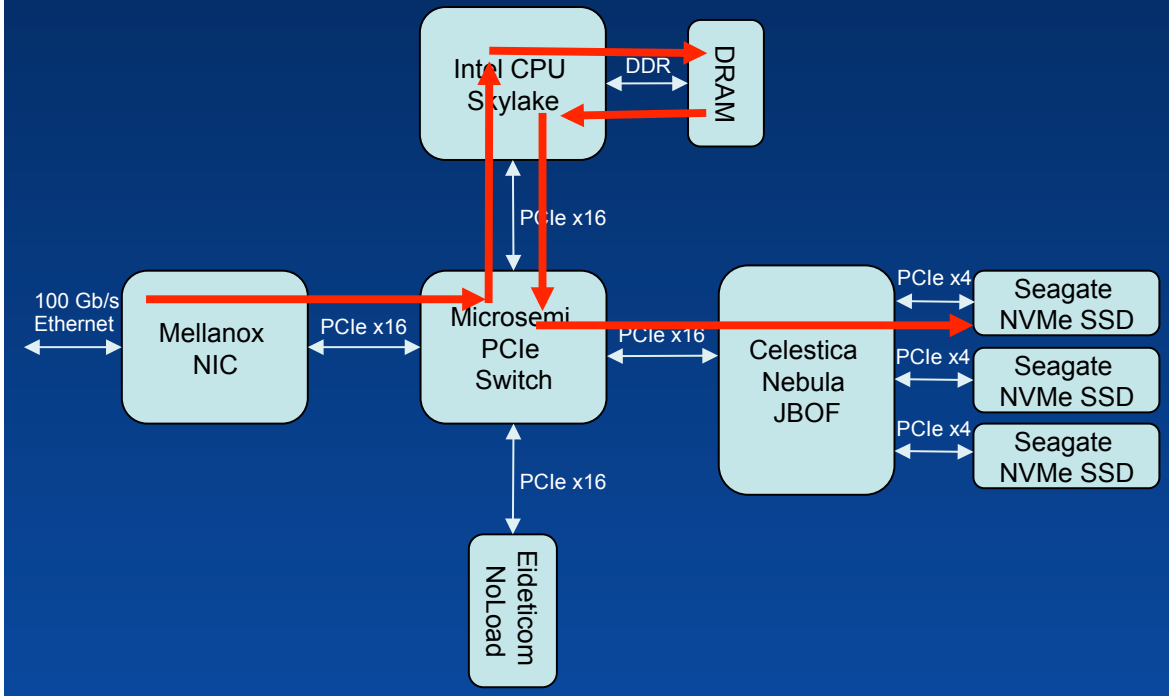
- Implemented compression core FPGA accelerator
  - Each compression core capable of >1GB/s throughput
  - Multiple accelerator cores can be integrated into a NoLoad FPGA
  - Each accelerator core is its own NVMe namespace
- Both NoLoad and a generic NVMe SSD located on remote U.2 JBOF
  - Both are shared via NVMe-oF
- Client process generates data, sends it to the compression accelerator, and then outputs it to the SSD.

Case Study: Compression over Fabrics



- The 2x compression core test over fabrics achieves about 1 GB/s per core
  - This means we are still able to get the same throughput over fabrics! (Given sufficient fabrics bandwidth of course)
- But how much impact is there on resources in the target machine?

Target CPU Efficiency



NVMe-oF target configuration

- Let's look at a different example but with the Mellanox ConnectX-5's
- In vanilla NVMe-oF target CPU is responsible for handling communication with NVMe drive
- This data flow heavily uses the target CPU and DRAM
- How can we reduce the load on the target machine?
  - NVMe-oF offload!

NVMe-oF Offload

- NVMe-oF Offload allows the NIC to directly control NVMe devices
- Using Mellanox ConnectX-5's we can offload the NVMe work from the target CPU

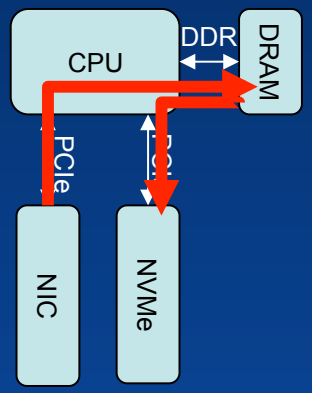
Operation	Latency (read/write) us	CPU Utilization	CPU Memory Bandwidth	CPU PCIe Bandwidth	NVMe Bandwidth	Ethernet Bandwidth
Vanilla NVMe-oF	188/227	1.00	1.00	1.00	1.00	1.00
ConnectX-5 Offload	128/138	0.02	2.40	1.03	1.00	1.00

- ConnectX-5 Offload reduces the target CPU load by **x50** but doesn't decrease the memory bandwidth
- How can we reduce the memory utilization?
  - With peer-to-peer transfers!

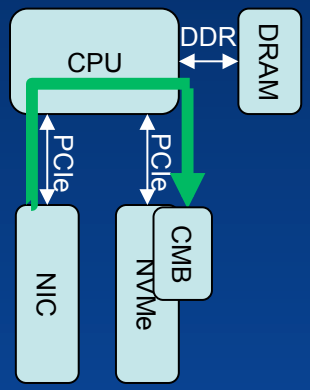
NVMe CMBs and P2P Transfers

- For p2p transfers, we need to make use of NVMe CMBs (Controller Memory Buffers)
- A NVMe CMB is a PCIe BAR (or part thereof) that can be used for certain NVMe specific data types.
- A P2P framework called p2pmem is being proposed for the Linux kernel
- PCIe drivers can register memory (e.g. CMBs) or request access to memory for DMA
- With P2P transfers, we can skip the DRAM copy reducing latency and DRAM usage.

Traditional DMAs

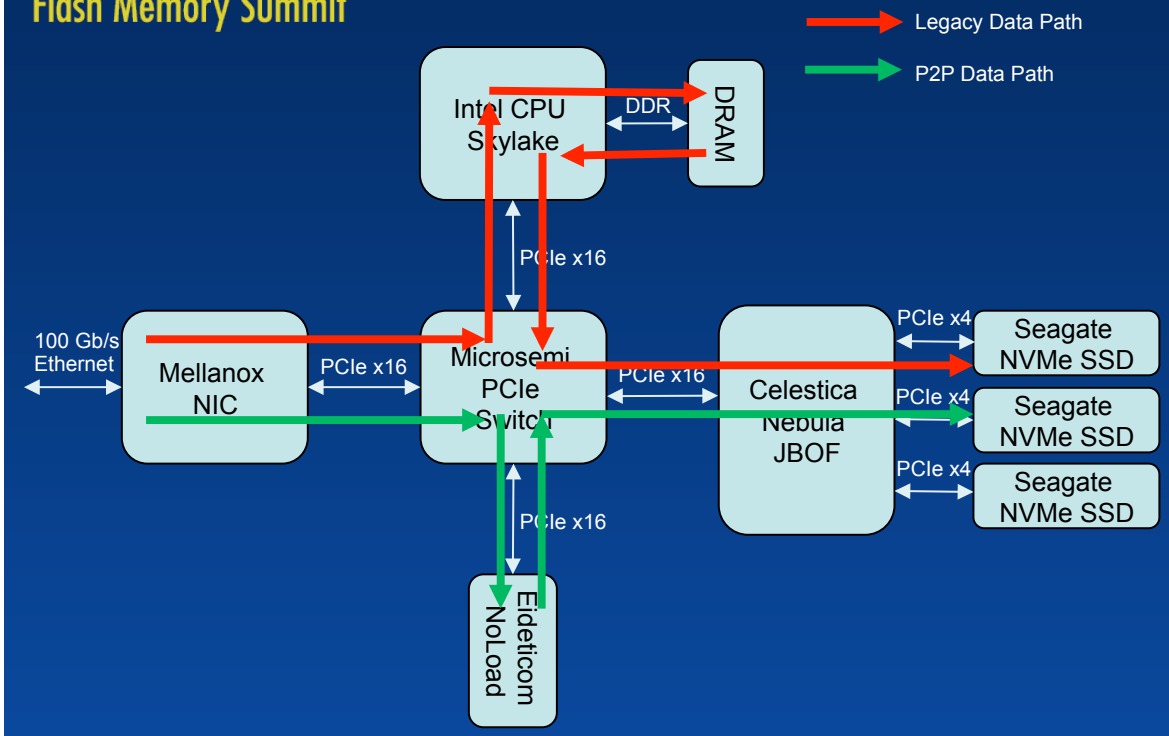


P2P DMAs



- Traditional DMAs (left) load the CPU. P2P DMAs (right) do not load the CPU

NVMe-oF Offload with P2P and CMB



NVMe-oF target configuration

- Now let's retry the previous example with P2P/CMB with the ConnectX-5 Offload
- The P2P path offloads both the CPU and the DRAM

NVMe-oF Offload with P2P and CMB

Operation	Latency (read/write) us	CPU Utilization	CPU Memory Bandwidth	CPU PCIe Bandwidth	NVMe Bandwidth	Ethernet Bandwidth
Vanilla NVMe-oF	188/227	1.00	1.00	1.00	1.00	1.00
ConnectX-5 Offload	128/138	0.02	2.40	1.03	1.00	1.00
Eideticom NoLoad p2pmem	167/212	0.55	0.09	0.01	1.00	1.00
CX5 Offload + Eideticom NoLoad p2pmem	142/154	0.02	0.02	0.04	1.00	1.00

- Combining p2pmem and CX5 Offload provides significant reduction of CPU utilization (x50), CPU memory bandwidth (x50), and CPU PCIe bandwidth (x25)

FMS 2018 Eideticom Demos

- The discussed compression example NVMe-oF with Broadcom at booth #729
- Compression/Decompression acceleration via P2P transfers with Xilinx at booth #313
- Come check them out ;)