AMD DATACENTER SOLUTIONS

Accelerating NVM Express Storage with Eideticom's NoLoad®, AMD EPYC CPUs, and p2pdma

Solution Brief
April, 2019

Executive Summary

- Eideticom's NoLoad® NVM Express Computational Storage Processor can offload the CPU by a factor of 70 for industry standard compression. This can save the customer cost, power and physical space when compared to a software-only solution.
- The AMD EPYC™ processor's internal PCIe® switch removes the need for an external PCIe switch which can save the customer cost, power and board space in the server.
- Using Eideticom's NoLoad NVM Express Computational Storage Processor and the Linux® p2pdma framework removes 100% of DMA traffic from the CPU's memory subsystem. This vastly improves the performance of applications running on the CPU.
- The Eideticom NoLoad NVM Express Computational Storage Processor comes in a U.2 form-factor making it easy to deploy in any NVM Express capable server and utilizes adaptable and power-efficient FPGA technology from Xilinx.

70x	4x	20x	Zero
Less CPU	Less Space	Less Power	Memory Bandwidth

Introduction

The recent trend of using high performance PCIe devices to move, manipulate and store data in compute and storage servers is creating new bottlenecks. High performance Network Interface Cards (NICs), NAND based NVM Express (NVMe) SSDs and GPGPUs are all capable of generating multiple-GB/s of PCIe traffic.

This volume of PCIe traffic can be problematic for a variety of reasons:

- Often the connection between the Root Port (RP) on the CPU is narrower that the connection between the PCIe Down-Stream Port (DSP) and the End Point (EP). This can lead to a bottleneck.
- The large volume of PCIe traffic can choke the internal CPU bus causing issues for CPU core to core communication.
- The large volume of PCIe traffic can flood the Last Level Cache (LLC) causing CPU load/stores to cache-miss resulting in additional latency and poorer quality of service for applications.
- The large volume of PCIe traffic can choke the Integrated Memory Controllers (IMCs) on the CPU resulting in additional latency and poorer quality of service for applications.

Eideticom's NVM Express Computational Storage Processor (NoLoad) with Xilinx FPGAs, combined with p2pdma and AMD's EPYC processors can overcome all of the issues noted above.

NoLoad NVM Express Computational Storage Processor

Eideticom's NoLoad U.2 Computational Storage Processor is the storage industries first FPGA accelerator in a 2.5" U.2 NVM Express® (NVMe) drive form factor [A].

Figure 1: *Eideticom's NoLoad in U.2 form factor*

NoLoad's NVMe compatible interface provides seamless integration for all CPU platforms and has been validated on Intel, AMD, ARM® and IBM Power8/9 CPUs. NoLoad supports high performance acceleration of storage and compute workloads on FPGA including Erasure Coding, Deduplication, Compression, Encryption and Analytics. In addition, it is compatible/validated with Broadcom®, Mellanox® and Q-Logic® RDMA NIC's.

Eideticom's p2pdma framework

Linus Torvalds released the first candidate for the 4.20 Linux kernel in November 2018 and it included the upstream version of the Eideticom Peer-to-Peer DMA (p2pdma) framework. This framework is an important part of the evolution of PCIe and NVM Express Computational Storage as it will allow NVMe and other PCIe devices to move data between themselves without having to DMA via CPU memory.

What is a P2P DMA?

In Figure 2 we illustrate the difference between a legacy DMA copy between two PCIe devices and a p2pdma copy.

p2pdma transfers bypass CPU memory and instead use PCIe memory provided by one (or more) of the PCIe devices in the system (e.g., NVMe Controller Memory Buffer (CMB) or PCIe BAR). A P2P capable Root Complex or PCIe switch is needed (note the AMD EPYC processors have very good p2pdma capability). This p2pdma path has not been enabled in Operating Systems until now.

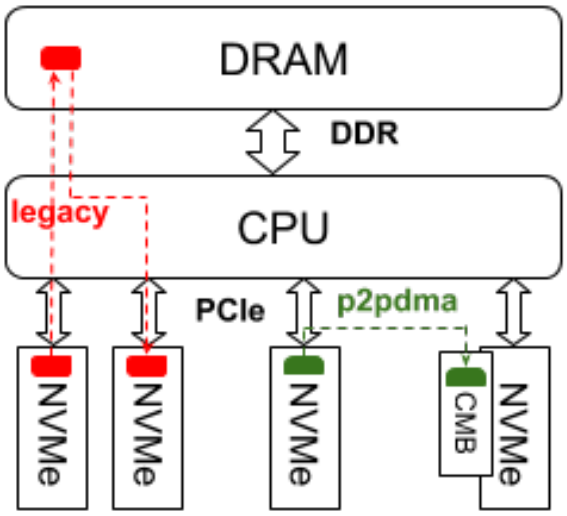

Figure 2*: A legacy DMA vs a p2pdma. The p2pdma avoids the CPU's memory subsystem and can avoid the CPU entirely when PCIe switches are used.*

p2pdma offers the following advantages over the legacy DMA path:

- DMA traffic avoids the CPU's memory subsystem which leaves that memory bandwidth available for applications.
- In systems with many PCIe devices the CPU memory subsystem can become the performance bottleneck. p2pdma avoids this problem.
- With good system design p2pdma performance can be scaled linearly as more devices are added with virtually no risks of bottlenecks occurring.

Why do NVM Express and p2pdma play well together?

NVM Express SSDs are PCIe End Points (EPs) with a very well-defined interface to the host CPU. NVMe EPs have the capability to support a new optional feature called Controller Memory Buffer (CMB) and this can be used as the PCIe memory for p2pdma. In the upstream kernel any NVMe SSD with a CMB that has Read Data Support (RDS) and Write Data Support (WDS) capability will register that memory against the p2pdma framework and can be used by other PCIe EPs for DMAs. The Eideticom NVMe NoLoad platform has a large and high performance CMB which can therefore be leveraged in p2pdma to move data into and out of our acceleration namespaces

Digging deeper into the p2pdma API

The API for p2pdma is described in an article by Marta Rybczyńska on lwn.net [B]. p2pdma currently works on select Intel and AMD CPU platforms, with additional support for ARM®64 [C] and RISC-V [D] in development. A simple p2p copy application which can be used for debug and performance testing is available [E]. fio also has support for using p2pdma memory via the iomap flag [F]. Finally, support for NVMe models with CMBs was added to upstream QEMU [G].

AMD EPYC Processors for p2pdma

The AMD EPYC family of CPUs is an ideal choice for p2pdma. The EPYC has been designed with p2pdma in mind by enabling high-performance p2pdma between PCIe devices connected to the CPU. This is not true for other CPUs from other vendors where p2pdma is either not possible or is not performant.

The p2pdma optimized architecture of the EPYC removes the need for an external PCIe switch. This reduces the power, cost, and complexity of the solution compared to alternatives.

Case Study

We used an HPE DL385 AMD EPYC powered server [H] to develop a p2pdma application using NoLoad. This server came with 256GB of DDR4 memory, two AMD EPYC™ 7501 series processors with total of 64 cores running at 2.0GHz, and support for up to 24 NVMe drives in U.2 form factor. This server was specifically used because it has 128 PCIe lanes per CPU and enables good p2pdma performance between PCIe devices.

We attached eighteen NVMe SSDs and six U.2 NoLoad devices (on BittWare 250-U2 FPGA cards [I]) to this server in our tests. Each of the NoLoad devices were programmed to perform ZLIB data compression via an NVMe namespace. Also, twelve of the NVMe SSDs were loaded with data files from the Protein corpus [J] to be used as source data for measuring the compression performance. We used the remaining six SSDs as the destination for the resultant compressed data.

Each NoLoad was assigned to three U.2 NVMe SSDs. All devices were directly connected to one of the two EPYC CPUs and no external PCIe switches were needed. Each group of one NoLoad and three NVMe SSDs were connected to the same NUMA node to optimize the performance and to ensure p2pdma traffic was not traversing the socket to socket bus between the two AMD EPYC CPUs.

We measured the total data compression throughput in two scenarios. In each test, data was read from 12 source SSDs and written to six destination SSD after being compressed. In our software compression test data was first loaded into the host memory from 12 SSDs. The data was then compressed using all available 64 physical CPU cores using the standard software ZLIB library [K] at compression level 1 (fastest). This is the typical software approach for data compression, and we measured the total compression throughput across all 64 physical CPU cores to be about 44 Gb/s. Also, the entire compression I/O occupied about[1] 88 Gb/s of memory bandwidth in this scenario.

***Table 1**: Summary of compression performance in different data transfer modes.*

Test	NoLoads	SSDs	CPU Usage	Memory Bandwidth	Throughput (Maximum)
Software ZLIB	NA	18	100 %	88Gb/s	44Gb/s
p2pdma NoLoad	1	3	0.8%	0Gb/s	27Gb/s
p2pdma NoLoad	2	6	1.6%	0Gb/s	53Gb/s
p2pdma NoLoad	3	9	2.5%	0Gb/s	81Gb/s
p2pdma NoLoad	4	12	3.2%	0Gb/s	108Gb/s
p2pdma NoLoad	5	15	4.2%	0Gb/s	135Gb/s
p2pdma NoLoad	6	18	5.0%	0Gb/s	159Gb/s

In our second test we compressed data with six NoLoads instead of using CPU cores. In this test, we used NVMe's p2pdma feature to transfer the data from two source SSDs to one NoLoad compression accelerator and from there to a destination SSD. This pattern was then scaled out to the total of 18 SSDs and six NoLoad engines. Using p2pdma and NoLoads resulted in offload of the host CPU and memory as the data.

In our p2pdma test we achieved a maximum data throughput of 159 Gb/s. We reached this performance using only 5% of the host's CPU cores and none of the host's memory or cache bandwidth. Note that the host's CPU cores were only used to orchestrate the p2pdma data transfers between the SSDs and the NoLoad accelerators. The results of these tests are given in Table 1. These results show how effective an NVMe Computational Storage Processor with p2pdma capability can be as a computational resource in offloading the host machine.

[1] This number is an estimate since a percentage of the data could be transferred through the CPU cache.

References

[A] https://www.eideticom.com/
[B] https://lwn.net/Articles/767281/
[C] https://github.com/sbates130272/linux-p2pm em/commits/pci-p2p-v5-plus-ioremap
[D] https://github.com/sbates130272/linux-p2pm em/tree/riscv-p2p-sifive
[E] https://github.com/sbates130272/p2pmem-test
[F] https://github.com/axboe/fio/blob/master/HOWTO [see iomap mapshared option]
[G] https://github.com/qemu/qemu/commit/b2b2 b67a0057407e19cfa3fdd9002db21ced8b01
[H] https://www.hpe.com/ca/en/product-catalog/ servers/proliant-servers/pip.overview.hpe-proliant-dl385-gen10-server.1011015902.html
[I] https://www.nallatech.com/store/fpga-accele rated-comp uting/ pcie-accelerator-cards/250- u2/
[J] http://data-compression.info/Corpora/Protein Corpus/
[K] http://github.com/madler/zlib