Flash Memory Summit

Important new NVMe features for optimizing the data pipeline

Dr. Stephen Bates, CTO

Eideticom

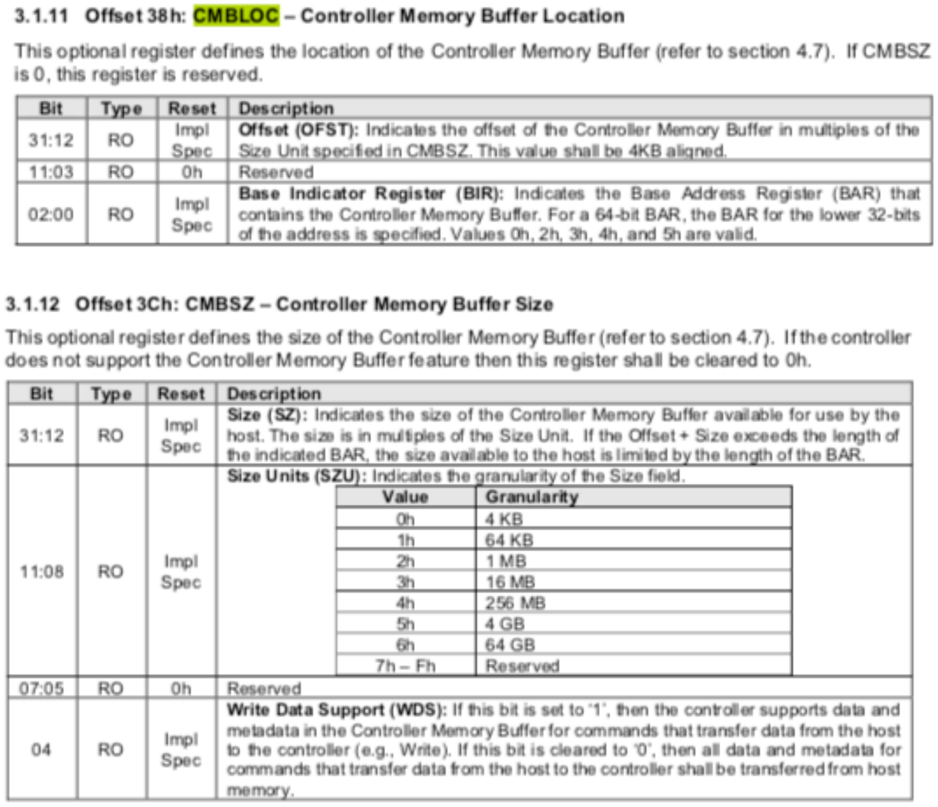
EIDETICOM

Outline

- Intro to NVMe Controller Memory Buffers (CMBs)
- Use cases for CMBs
    - Submission Queue Support (SQS) only
    - RDS (Read Data Support) and WDS (Write Data Support) for NVMe p2p copies
    - SQS, RDS and WDS for optimized NVMe over Fabrics
- Software for NVMe CMBs
    - SPDK (Storage Performance Developer Kit) work for NVMe copies.
    - Linux kernel work for p2pdma and for offload.
- Roadmap for the future

Intro to Controller Memory Buffers

- CMBs were introduced to the NVMe standard in 2014 in version 1.2.
- A NVMe CMB is a PCIe BAR (or part thereof) that can be used for certain NVMe specific data types.
- The main purpose of the CMB is to provide an alternative to:
  - Placing queues in host memory
  - Placing data for DMA in host memory.
- As well as a BAR, two optional NVMe registers are needed:
  - CMBLOC - location
  - CMBSZ - size and supported types
- Multiple vendors support CMB today (Intel, Eideticom, Everspin) or soon (Toshiba, Samsung, WDC etc).



3.1.11 Offset 38h: CMBLOC – Controller Memory Buffer Location

This optional register defines the location of the Controller Memory Buffer (refer to section 4.7). If CMBSZ is 0, this register is reserved.

Bit	Type	Reset	Description
31:12	RO	Impl Spec	**Offset (OFST):** Indicates the offset of the Controller Memory Buffer in multiples of the Size Unit specified in CMBSZ. This value shall be 4KB aligned.
11:03	RO	0h	Reserved
02:00	RO	Impl Spec	**Base Indicator Register (BIR):** Indicates the Base Address Register (BAR) that contains the Controller Memory Buffer. For a 64-bit BAR, the BAR for the lower 32-bits of the address is specified. Values 0h, 2h, 3h, 4h, and 5h are valid.

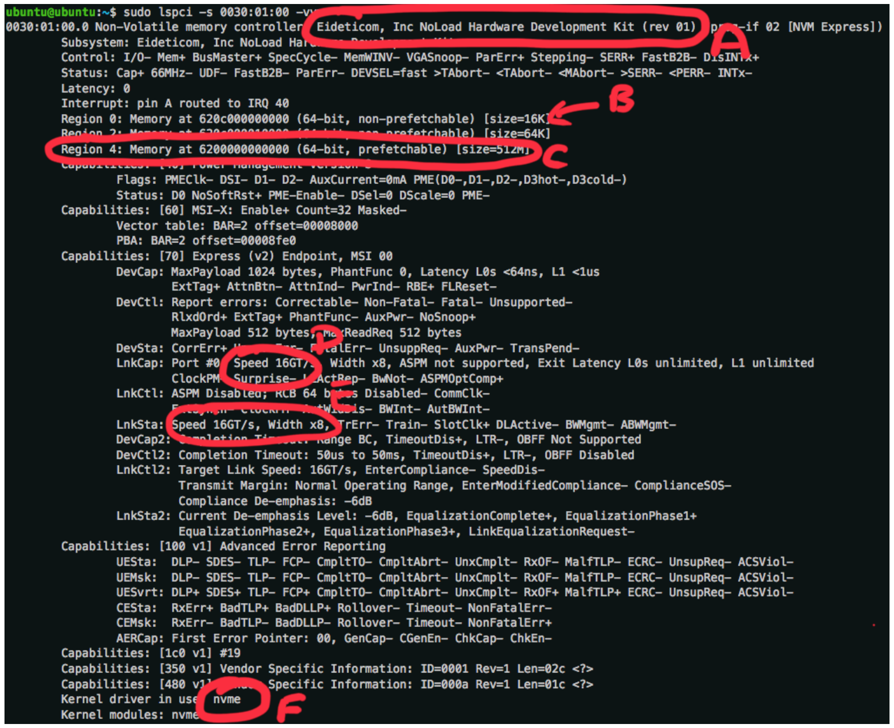
3.1.12 Offset 3Ch: CMBSZ – Controller Memory Buffer Size

This optional register defines the size of the Controller Memory Buffer (refer to section 4.7). If the controller does not support the Controller Memory Buffer feature then this register shall be cleared to 0h.

Bit	Type	Reset	Description
31:12	RO	Impl Spec	**Size (SZ):** Indicates the size of the Controller Memory Buffer available for use by the host. The size is in multiples of the Size Unit. If the Offset + Size exceeds the length of the indicated BAR, the size available to the host is limited by the length of the BAR.
11:08	RO	Impl Spec	**Size Units (SZU):** Indicates the granularity of the Size field.
07:05	RO	0h	Reserved
04	RO	Impl Spec	**Write Data Support (WDS):** If this bit is set to '1', then the controller supports data and metadata in the Controller Memory Buffer for commands that transfer data from the host to the controller (e.g., Write). If this bit is cleared to '0', then all data and metadata for commands that transfer data from the host to the controller shall be transferred from host memory.

Value	Granularity
0h	4 KB
1h	64 KB
2h	1 MB
3h	16 MB
4h	256 MB
5h	4 GB
6h	64 GB
7h – Fh	Reserved

Intro to Controller Memory Buffers



```
ubuntu@ubuntu:~$ sudo lspci -s 0030:01:00 -vv
0030:01:00.0 Non-Volatile memory controller: Eideticom, Inc NoLoad Hardware Development Kit (rev 01) (prog-if 02 [NVM Express])
        Subsystem: Eideticom, Inc NoLoad Hardware Development Kit
        Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr+ Stepping- SERR+ FastB2B- DisINTx+
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
        Latency: 0
        Interrupt: pin A routed to IRQ 40
        Region 0: Memory at 620c000000000 (64-bit, non-prefetchable) [size=16K]
        Region 2: Memory at [illegible] (64-bit, non-prefetchable) [size=64K]
        Region 4: Memory at 6200000000000 (64-bit, prefetchable) [size=512M]
        Capabilities: [40] Power Management version [illegible]
                Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
                Status: D0 NoSoftRst+ PME-Enable- DSel=0 DScale=0 PME-
        Capabilities: [60] MSI-X: Enable+ Count=32 Masked-
                Vector table: BAR=2 offset=00008000
                PBA: BAR=2 offset=00008fe0
        Capabilities: [70] Express (v2) Endpoint, MSI 00
                DevCap: MaxPayload 1024 bytes, PhantFunc 0, Latency L0s <64ns, L1 <1us
                        ExtTag+ AttnBtn- AttnInd- PwrInd- RBE+ FLReset-
                DevCtl: Report errors: Correctable- Non-Fatal- Fatal- Unsupported-
                        RlxdOrd+ ExtTag+ PhantFunc- AuxPwr- NoSnoop+
                        MaxPayload 512 bytes, MaxReadReq 512 bytes
                DevSta: CorrErr+ UncorrErr- FatalErr- UnsuppReq- AuxPwr- TransPend-
                LnkCap: Port #0, Speed 16GT/s, Width x8, ASPM not supported, Exit Latency L0s unlimited, L1 unlimited
                        ClockPM- Surprise- LLActRep- BwNot- ASPMOptComp+
                LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- CommClk-
                        ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
                LnkSta: Speed 16GT/s, Width x8, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
                DevCap2: Completion Timeout: Range BC, TimeoutDis+, LTR-, OBFF Not Supported
                DevCtl2: Completion Timeout: 50us to 50ms, TimeoutDis+, LTR-, OBFF Disabled
                LnkCtl2: Target Link Speed: 16GT/s, EnterCompliance- SpeedDis-
                         Transmit Margin: Normal Operating Range, EnterModifiedCompliance- ComplianceSOS-
                         Compliance De-emphasis: -6dB
                LnkSta2: Current De-emphasis Level: -6dB, EqualizationComplete+, EqualizationPhase1+
                         EqualizationPhase2+, EqualizationPhase3+, LinkEqualizationRequest-
        Capabilities: [100 v1] Advanced Error Reporting
                UESta:  DLP- SDES- TLP- FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF- MalfTLP- ECRC- UnsupReq- ACSViol-
                UEMsk:  DLP- SDES- TLP- FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF- MalfTLP- ECRC- UnsupReq- ACSViol-
                UESvrt: DLP+ SDES+ TLP- FCP+ CmpltTO- CmpltAbrt- UnxCmplt- RxOF+ MalfTLP+ ECRC- UnsupReq- ACSViol-
                CESta:  RxErr+ BadTLP+ BadDLLP+ Rollover- Timeout- NonFatalErr-
                CEMsk:  RxErr- BadTLP- BadDLLP- Rollover- Timeout- NonFatalErr+
                AERCap: First Error Pointer: 00, GenCap- CGenEn- ChkCap- ChkEn-
        Capabilities: [1c0 v1] #19
        Capabilities: [350 v1] Vendor Specific Information: ID=0001 Rev=1 Len=02c <?>
        Capabilities: [480 v1] Vendor Specific Information: ID=000a Rev=1 Len=01c <?>
        Kernel driver in use: nvme
        Kernel modules: nvme
```

- A - This device's manufacturer has registered its vendor ID and device IDs with the PCIe database. This means you get a human-readable description of it.
- B - This device has three PCIe BARs:
- BAR0 is 16KB and is the standard NVMe BAR that any legitimate NVMe device must have.
- C - The third BAR is the Controller Memory Buffer (CMB) which can be used for both NVMe queues and NVMe data.
- F - Since this device is a NVMe device it is bound to the standard Linux kernel NVMe driver.

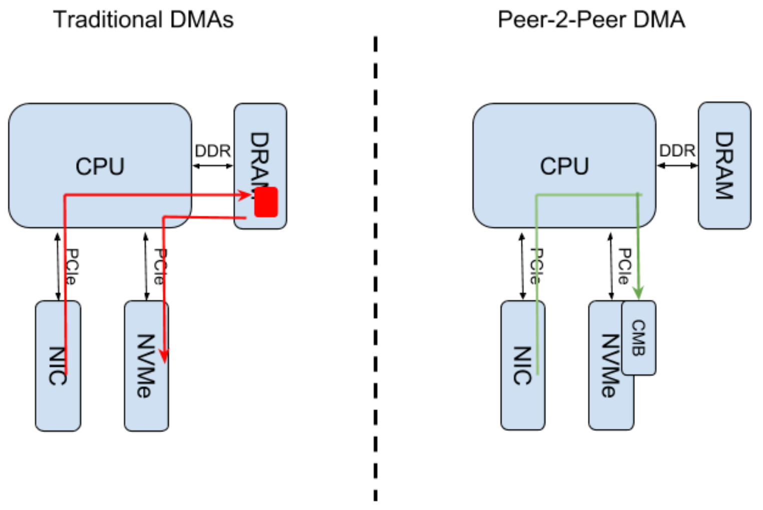
Below is CMBLOC and CMBSZ for a CMB enabled NVMe device

```
cmbloc  : 3
        Offset                 (OFST): 0 (See cmbsz.szu for granularity)
        Base Indicator Register (BIR): 3

cmbsz   : 500003
        Size                      (SZ): 1280
        Size Units               (SZU): 4 KB
        Write Data Support       (WDS): Write Data and metadata transfer in Controller Memory Buffer is Not supported
        Read Data Support        (RDS): Read Data and metadata transfer in Controller Memory Buffer is Not supported
        PRP SGL List Support   (LISTS): PRP/SG Lists in Controller Memory Buffer is Not supported
        Completion Queue Support (CQS): Admin and I/O Completion Queues in Controller Memory Buffer is Supported
        Submission Queue Support (SQS): Admin and I/O Submission Queues in Controller Memory Buffer is Supported
```

Some Fun Use Cases for CMBs

- Placing some (or all) of your NVMe queues in CMB rather than host memory. Reduce latency [Linux Kernel[1] and SPDK[1]].
- Using the CMB as a DMA buffer allows for offloaded NVMe copies. This can improve performance and offloads the host CPU [SPDK[1]].
- Using the CMB as a DMA buffer allows RDMA NICs to directly place NVMe-oF data into the NVMe SSD. Reduce latency and CPU load [Linux Kernel[2]]

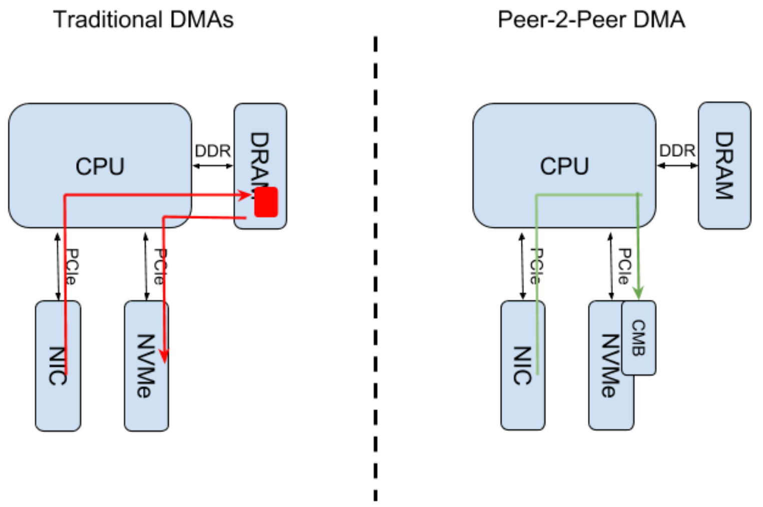


Traditional DMAs (left) load the CPU. P2P DMAs (right) do not load the CPU.

[1] Upstream in the relevant tree.
[2] Proposed patches (see last slide for git repo).

Software for CMBs - SPDK

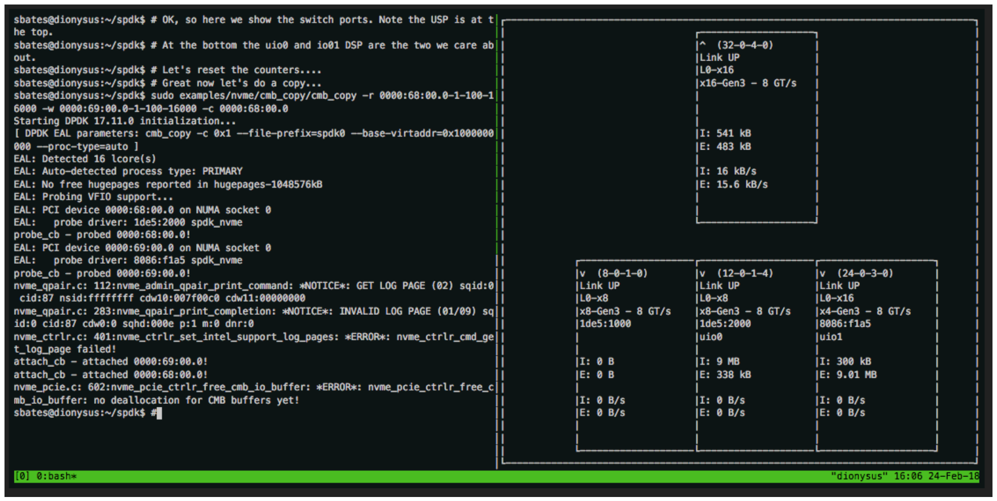
- Storage Performance Development Kit (SPDK) is a Free and Open Source (FOSS) user-space framework for high performance storage.
- Focus on NVMe and NVMe-oF.
- Code added in Feb 2018 to enable P2P NVMe copies when CMBs allow it.
- A simple example of an application using this new API also in SPDK examples (cmb_copy).



cmb_copy is an example application using SPDK's APIs to copy data between NVMe SSDs using P2P DMAs. This bypasses the CPU's memory and PCIe subsystems.

Flash Memory Summit

Software for CMBs - SPDK



```
sbates@dionysus:~/spdk$ # OK, so here we show the switch ports. Note the USP is at t
he top.
sbates@dionysus:~/spdk$ # At the bottom the uio0 and io01 DSP are the two we care ab
out.
sbates@dionysus:~/spdk$ # Let's reset the counters....
sbates@dionysus:~/spdk$ # Great now let's do a copy...
sbates@dionysus:~/spdk$ sudo examples/nvme/cmb_copy/cmb_copy -r 0000:68:00.0-1-100-1
6000 -w 0000:69:00.0-1-100-16000 -c 0000:68:00.0
Starting DPDK 17.11.0 initialization...
[ DPDK EAL parameters: cmb_copy -c 0x1 --file-prefix=spdk0 --base-virtaddr=0x1000000
000 --proc-type=auto ]
EAL: Detected 16 lcore(s)
EAL: Auto-detected process type: PRIMARY
EAL: No free hugepages reported in hugepages-1048576kB
EAL: Probing VFIO support...
EAL: PCI device 0000:68:00.0 on NUMA socket 0
EAL:   probe driver: 1de5:2000 spdk_nvme
probe_cb - probed 0000:68:00.0!
EAL: PCI device 0000:69:00.0 on NUMA socket 0
EAL:   probe driver: 8086:f1a5 spdk_nvme
probe_cb - probed 0000:69:00.0!
nvme_qpair.c: 112:nvme_admin_qpair_print_command: *NOTICE*: GET LOG PAGE (02) sqid:0
 cid:87 nsid:ffffffff cdw10:007f00c0 cdw11:00000000
nvme_qpair.c: 283:nvme_qpair_print_completion: *NOTICE*: INVALID LOG PAGE (01/09) sq
id:0 cid:87 cdw0:0 sqhd:000e p:1 m:0 dnr:0
nvme_ctrlr.c: 401:nvme_ctrlr_set_intel_support_log_pages: *ERROR*: nvme_ctrlr_cmd_ge
t_log_page failed!
attach_cb - attached 0000:69:00.0!
attach_cb - attached 0000:68:00.0!
nvme_pcie.c: 602:nvme_pcie_ctrlr_free_cmb_io_buffer: *ERROR*: nvme_pcie_ctrlr_free_c
mb_io_buffer: no deallocation for CMB buffers yet!
sbates@dionysus:~/spdk$ #
```

```
                    |^  (32-0-4-0)
                    |Link UP
                    |L0-x16
                    |x16-Gen3 - 8 GT/s

                    |I: 541 kB
                    |E: 483 kB

                    |I: 16 kB/s
                    |E: 15.6 kB/s

|v  (8-0-1-0)       |v  (12-0-1-4)      |v  (24-0-3-0)
|Link UP            |Link UP            |Link UP
|L0-x8              |L0-x8              |L0-x16
|x8-Gen3 - 8 GT/s   |x8-Gen3 - 8 GT/s   |x4-Gen3 - 8 GT/s
|1de5:1000          |1de5:2000          |8086:f1a5
|                   |uio0               |uio1

|I: 0 B             |I: 9 MB            |I: 300 kB
|E: 0 B             |E: 338 kB          |E: 9.01 MB

|I: 0 B/s           |I: 0 B/s           |I: 0 B/s
|E: 0 B/s           |E: 0 B/s           |E: 0 B/s
```

```
[0] 0:bash*                                        "dionysus" 16:06 24-Feb-18
```

- **cmb_copy moves data direct from SSD A to SSD B using NVMe CMB(s).**
- **99.99% of UpStream Port (USP) traffic on PCIe switch is eliminated.**
- **OS is still in complete control of the IO and handles any status/error messages.**
- **NVMe SQEs can also be in the CMB or not as desired. SGLs or PRPs can be supported.**

See https://asciinema.org/a/bkd32zDLyKvIq7F8M5BBvdX42

Software for CMBs – The Linux Kernel

- A P2P framework called p2pdma is being proposed for the Linux kernel.
- Much more general than NVMe CMBs. Any PCIe device can utilize it (NICS, GPGPUs etc.).
- PCIe drivers can register device memory (e.g. CMBs or BARs) or request access to P2P memory for DMA.
- Initial patches use p2pdma to optimize the NVMe-oF target code.

Mode of 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
ConnectX-5 Offload + Eideticom NoLoad p2pmem	142/154	0.02	0.02	0.04	1.00	1.00

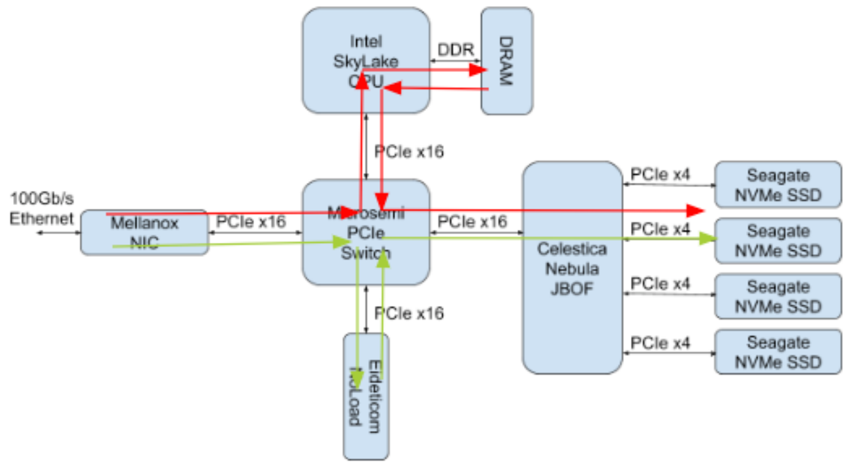
The p2pdma framework can be used to improve NVMe-oF targets. Here we show results from a NVMe-oF target system.

p2pdma can reduce CPU memory load by x50 and CPU PCIe load by x25. NVMe offload can also be employed to reduce CPU core load by x50.

Flash Memory Summit

Software for CMBs – The Linux Kernel

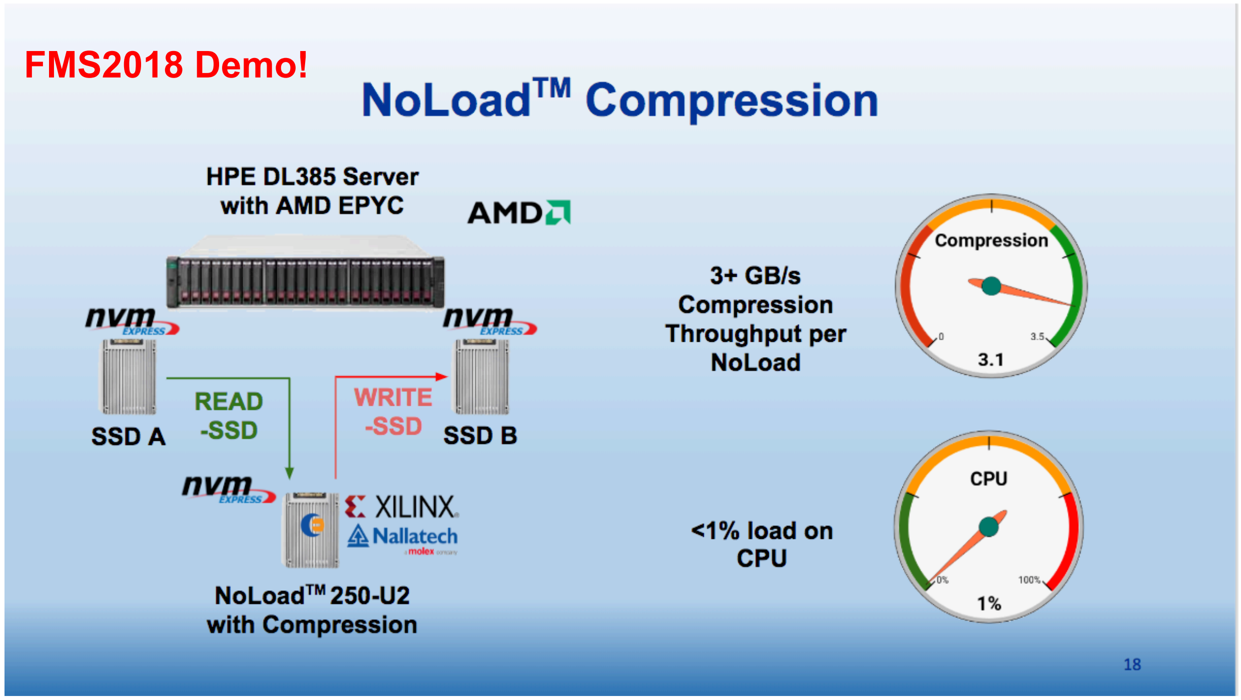
- The hardware setup for the NVMe-oF p2pdma testing is as shown on the right.
- The software setup consisted of a modified Linux kernel and standard NVMe-oF configuration tools (mostly nvme-cli and nvmet).
- The Linux kernel used added support for NVMe offload and Peer-2-Peer DMAs using an NVMe CMB provided by the Eideticom NVMe device.



Green: Legacy Data Path
Red: p2pdma Data Path

This is the NVMe-oF target configuration used. Note RDMA NIC is connected to switch and not CPU Root Port.

Software for CMBs – The Linux Kernel



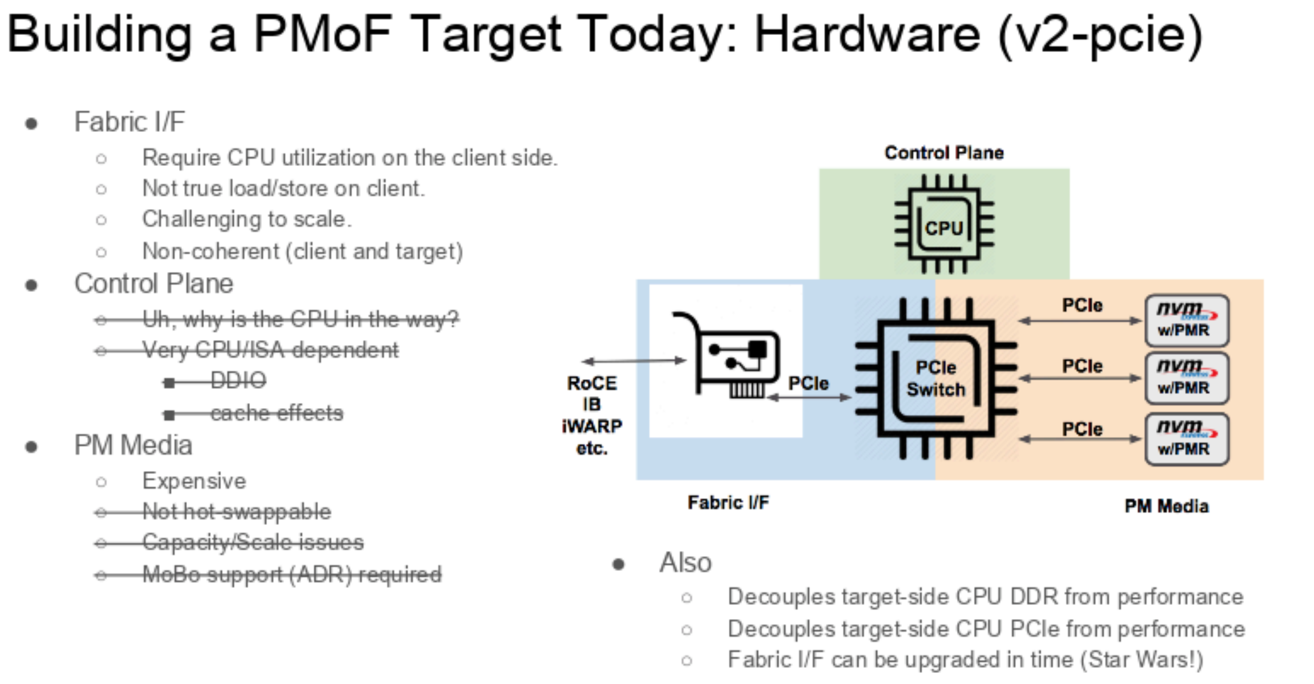
- Eideticom puts accelerators behind NVMe namespaces.
- We can combine this with CMBs to add compute to p2pdma.
- Demo with AMD in Xilinx in Xilinx booth.
- >3GB/s compression (input) via U.2 NVMe accelerator (NoLoad) with p2pdma offloading CPU by >99%.
- Leverages p2pdma to move data from input NVMe SSD to NoLoad and from NoLoad to output NVMe SSD.

Persistent Memory Regions

- PMRs add persistent to CMBs.
- PMR features being discussed and standard will be updated. Get involved!
- Small PMRs (10MB-1GB) interesting:
  - Write cache
  - Journal for SQEs
  - Persistent scratchpad for meta-data
- Big PMRs (>>1GB) are ***really*** interesting.

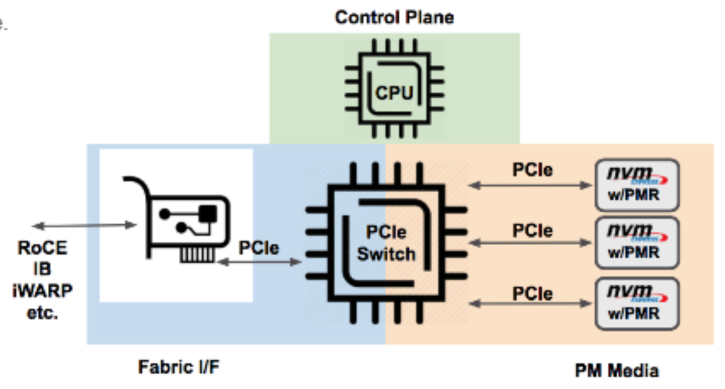
Persistent Memory Regions

See https://www.youtube.com/watch?v=olEem6hHAss&t=1310s



Building a PMoF Target Today: Hardware (v2-pcie)

- Fabric I/F
  - Require CPU utilization on the client side.
  - Not true load/store on client.
  - Challenging to scale.
  - Non-coherent (client and target)
- Control Plane
  - ~~Uh, why is the CPU in the way?~~
  - ~~Very CPU/ISA dependent~~
    - ~~DDIO~~
    - ~~cache effects~~
- PM Media
  - Expensive
  - ~~Not hot-swappable~~
  - ~~Capacity/Scale issues~~
  - ~~MoBo support (ADR) required~~



- Also
  - Decouples target-side CPU DDR from performance
  - Decouples target-side CPU PCIe from performance
  - Fabric I/F can be upgraded in time (Star Wars!)

Large NVMe PMRs would enable PMoF aware filesystems!

Roadmap for CMBs and PMRs and the Software

- NVMe CMBs have been in the standard for a while. However it's only now they are starting to become available and software is starting to utilize them.
- SPDK and the Linux kernel are the two main locations for CMB software enablement today.
- SPDK: NVMe P2P copies. NVMe-oF updates coming.
- Linux kernel. p2pdma framework upstream soon. Will be expanded to support other NVMe/PCIe resources (e.g. doorbells).
- Persistent Memory Regions add non-volatile CMBs and will require (lots of) software enablement too. They will enable a path to Persistent memory storage on the PCIe bus.