SDC

STORAGE DEVELOPER CONFERENCE

SNIA SANTA CLARA, 2017

p2pmem: Enabling PCIe Peer-2-Peer in Linux

Stephen Bates, PhD
Raithlin Consulting

Nomenclature: A Reminder

PCIe Peer-2-Peer using p2pmem is NOT blucky!!

The Rationale: A Reminder

The Rationale: A Reminder

- PCIe End-Points are getting faster and faster (e.g. GPGPUs, RDMA NICs, NVMe SSDs)
- Bounce buffering all IO data in system memory is a waste of resources
- Bounce buffering all IO data in system memory reduces QoS for CPU memory (the noisy neighbor problem)

The/A Solution?

- p2pmem is a Linux kernel framework for allowing PCIe EPs to DMA to each other whilst under host CPU control
- CPU/OS still responsible for security, error handling etc.
- 99.99% of DMA traffic now goes direct between EPs

Linux Kernel Background

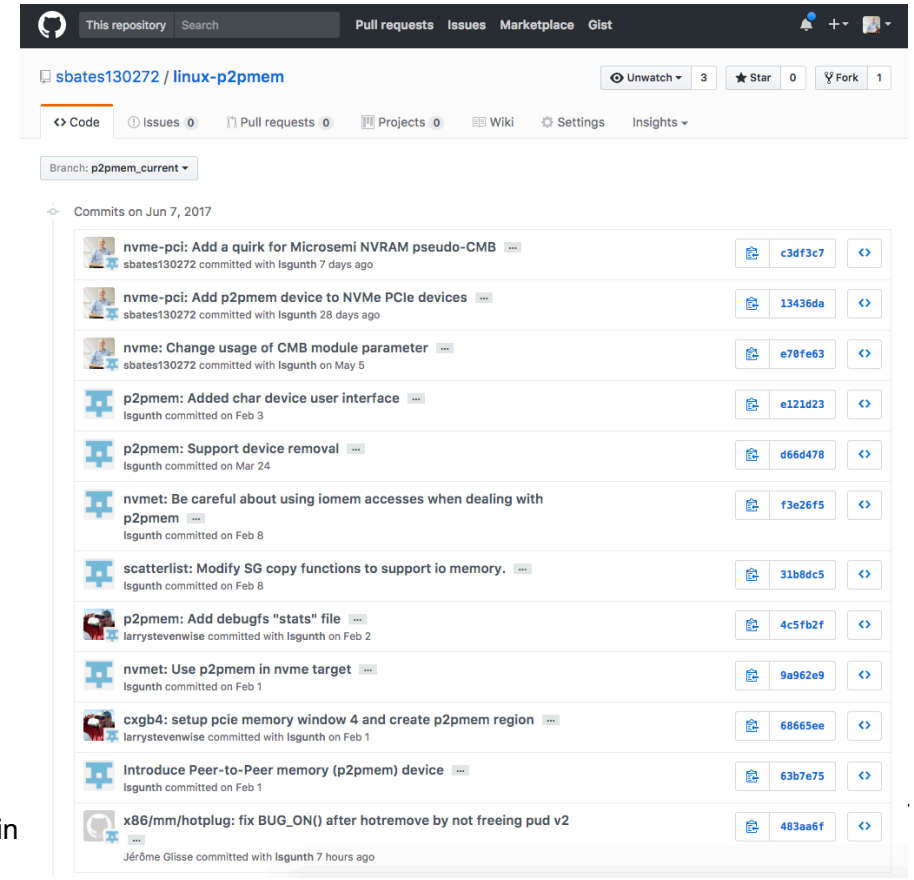
- There have been some recent additions to the kernel in preparation of new memory types on the memory channel:
  - ZONE_DEVICE– The ability to associate a range of memory addresses (PFNs) with a specific driver and not allocate them to system memory.
  - PMEM – A ZONE_DEVICE device driver that takes a memory region and exposes it to the rest of the OS as a DAX enabled block device.
  - DAX – A framework that allows a memory addressable block device to bypass the page-cache. Allows supporting filesystems (like ext4) to be placed on these block devices.
  - STRUCT PAGE SUPPORT – PMEM devices can (optionally) include struct page backing so they can be used for things like DMA.

What does linux-p2pmem code do?

https://github.com/sbates130272/linux-p2pmem/tree/p2pmem_nvme

For a simple (work in progress) example see - https://asciinema.org/a/c1qeco5suc8xjzm8ea8k66d27

- In upstream Linux today there are checks placed on any address passed to a PCIe DMA engine (e.g. NVMe SSD, RDMA NIC or GPGPU).
- One check is for IO memory.
- p2pmem code “safely” allows IO memory to be used in a DMA descriptor.

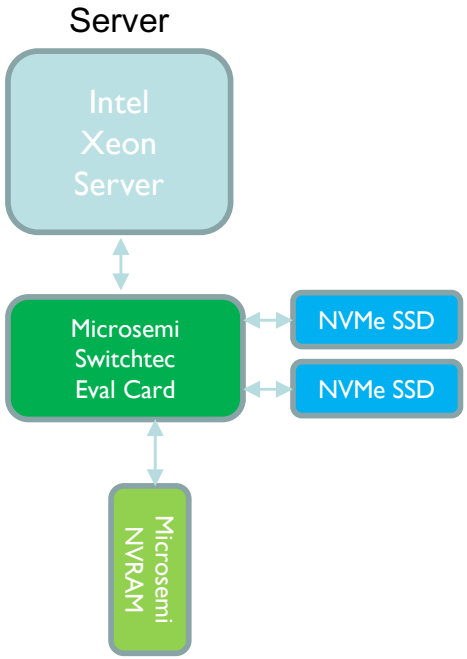


So What?

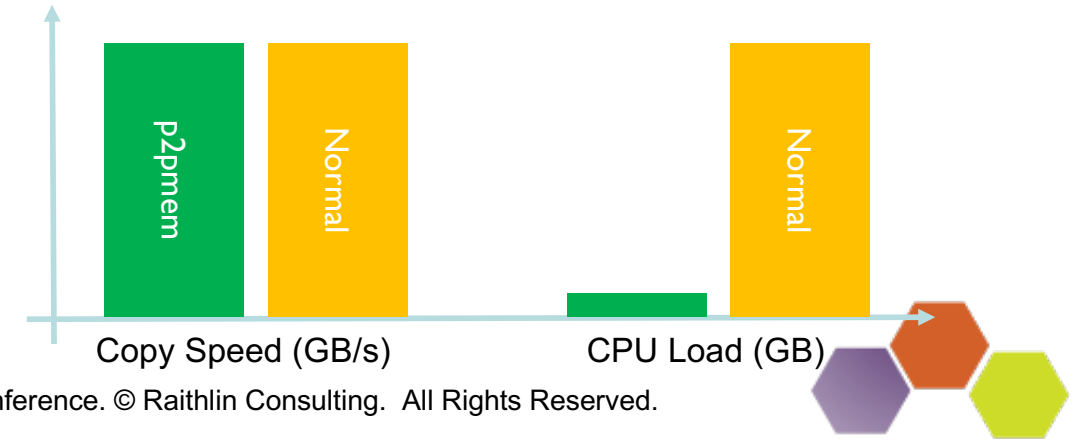
- With p2pmem we can now move data directly from one PCIe EP to another.
- For example we can use a PCIe BAR as the buffer for data being copied between two NVMe SSDs…

Example – NVMe Offloaded Copy 1

Test code is here - https://github.com/sbates130272/p2pmem-test



1. Copy data from one NVMe SSD to another using a Microsemi NVRAM card as a p2pmem buffer.
2. For 8GB copies the speed is about the same (600MB/s) but data on USP drops from ~16GB to 15MB or so!
3. Note we still do two DMAs (one MemWr based one from /dev/nvme0n1 to /dev/p2pmem0 and one MemRd based on from /dev/nvme1n1 to /dev/p2pmem0.



So What (part two)?

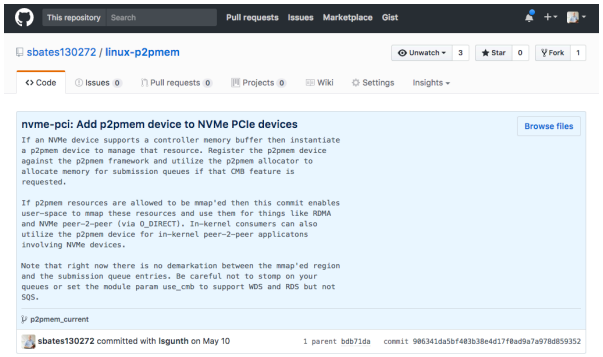
- With p2pmem we can now move data directly from one PCIe EP to another.
- Now consider if the PCIe BAR lives inside one of the NVMe SSDs. This could be a NVMe CMB[1], a NVMe PMR[1] or a separate PCIe function.

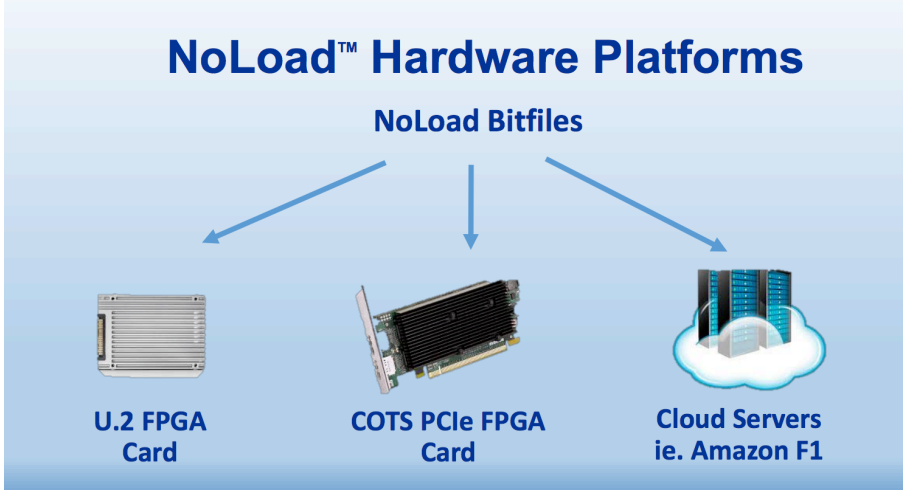
1 A NVMe Controller Memory Buffer is a volatile BAR that can be used for data and commands. A PMR is a (pre-standard) non-volatile BAR that can be used for data.

10

p2pmem and NVMe CMBs

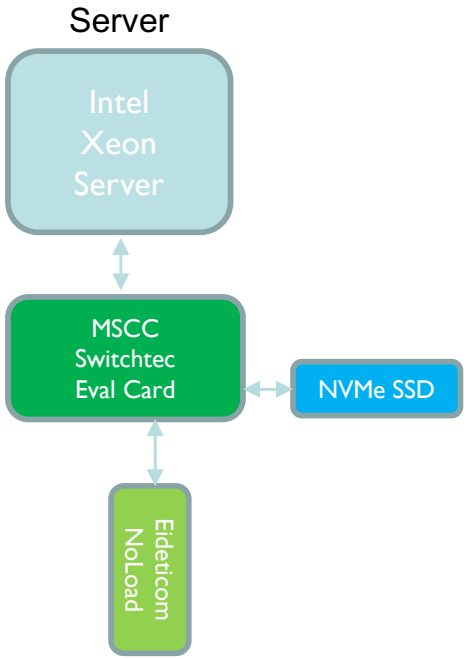
- NVMe CMB defines a PCIe memory region within the NVMe specification.
- p2pmem can leverage a CMB as either a bounce buffer or a true CMB.
- A simple update to the NVMe PCIe driver ties p2pmem and CMB together.
- Hardware tested using Eideticom's NoLoad device
- Can be extended to Persistent CMBs (PMRs) – coming soon!



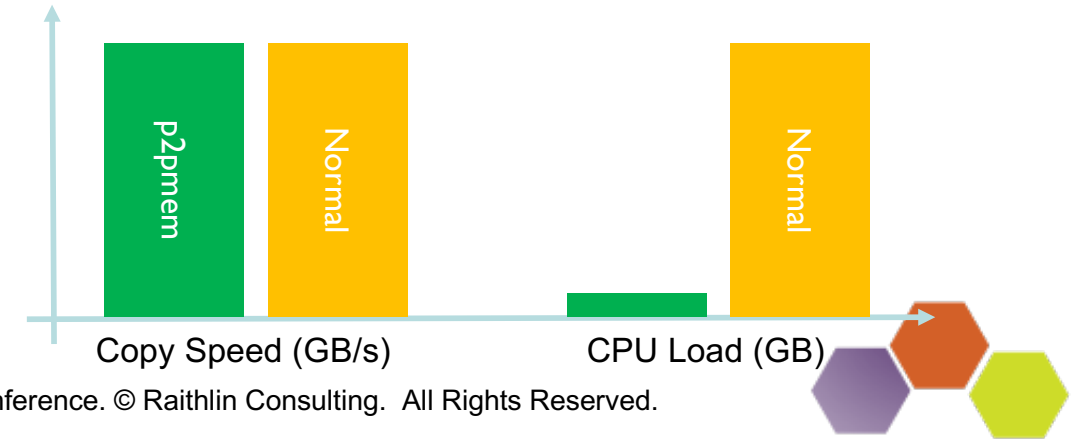


Example – NVMe Offloaded Copy 2

Test code is here - https://github.com/sbates130272/p2pmem-test

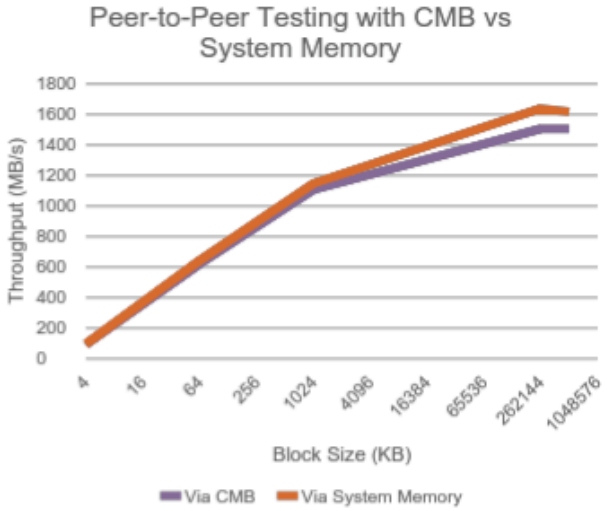


1. Copy data from one NVMe SSD to another using an Eideticom NoLoad and it's CMB as the IO buffer.
2. For 8GB copies the speed is about the same (1600MB/s) but data on USP drops from ~16GB to 15MB or so!
3. **Note only do one DMAs**, the second DMA is now internal to the NoLoad device (from its CMB to the backing store)



Example – NVMe Offloaded Copy 2

Peer-to-Peer CMB Results



- Current setup saturates due to insufficient sources in the test environment

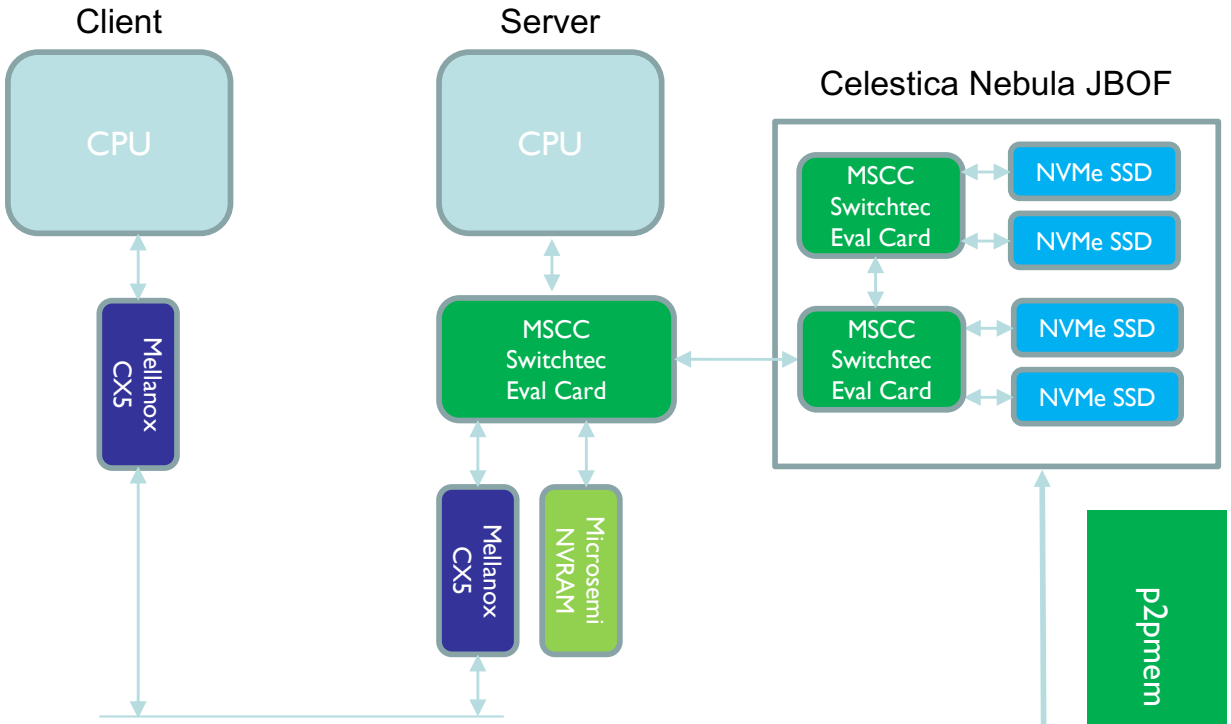
So What (part three)?

- With p2pmem we can now move data directly from one PCIe EP to another.
- For example an RDMA NIC can now push data directly to a PCIe BAR.
- Now where have we recently seen NVMe+RMDA? NVMe over Fabrics of course!


14

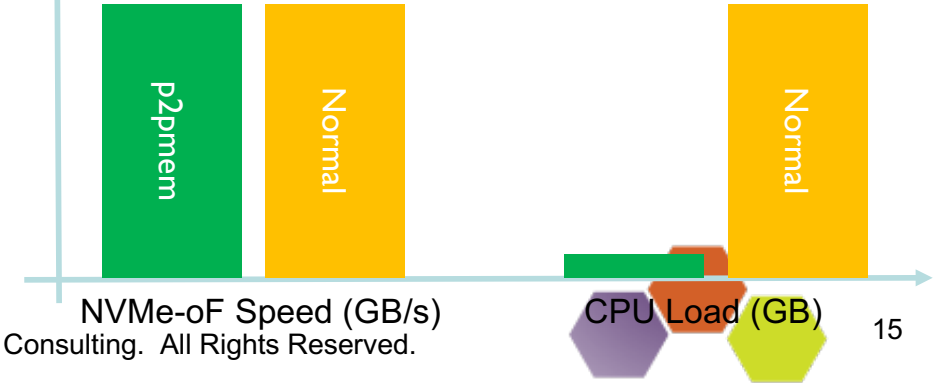
https://github.com/sbates130272/linux-p2pmem/tree/p2pmem_nvme

Example – NVMe over Fabrics



Celestica Mellanox TECHNOLOGIES Microsemi

1. p2pmem enabled for NVMe over Fabrics.
2. Driver uses p2pmem rather than system memory
3. Massive offload of USP and CPU memory sub-system.



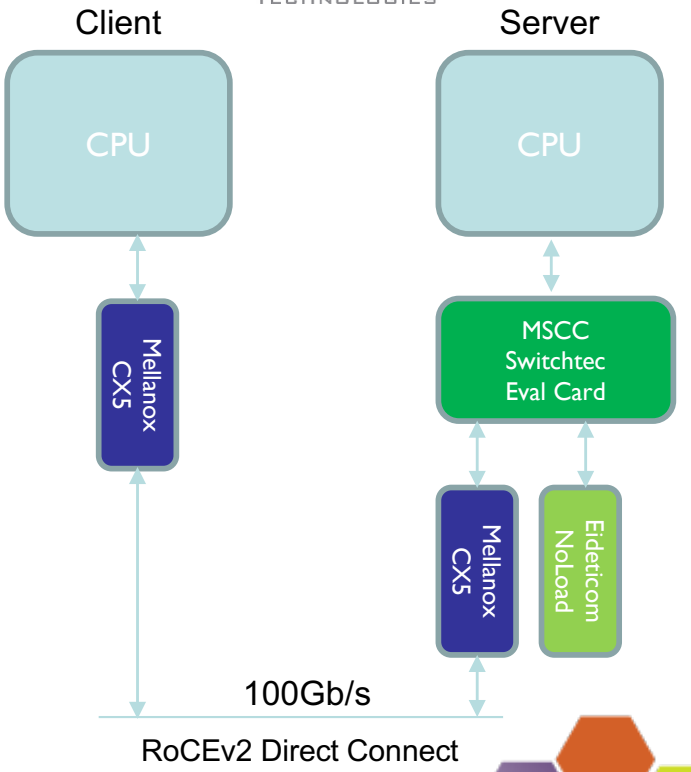
So What (part four)?

- With p2pmem we can now move data directly from one PCIe EP to another.
- For example an RDMA NIC can now push data directly to a PCIe BAR.
- Now rather than using that BAR as a temporary buffer it could be a NVMe PMR giving us **standards based** remote access to byte addressable persistent data!

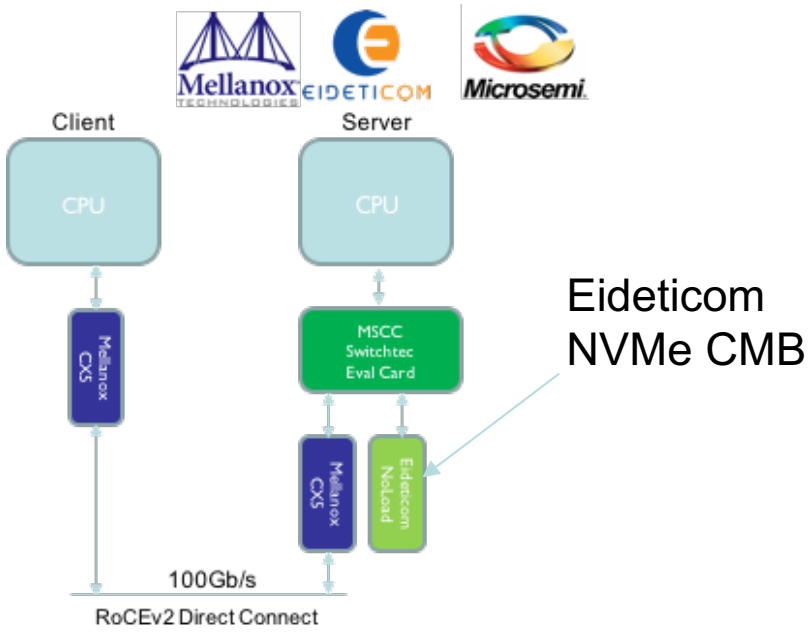

16

Example – Remote access to (Persistent) PCIe/NVMe Memory

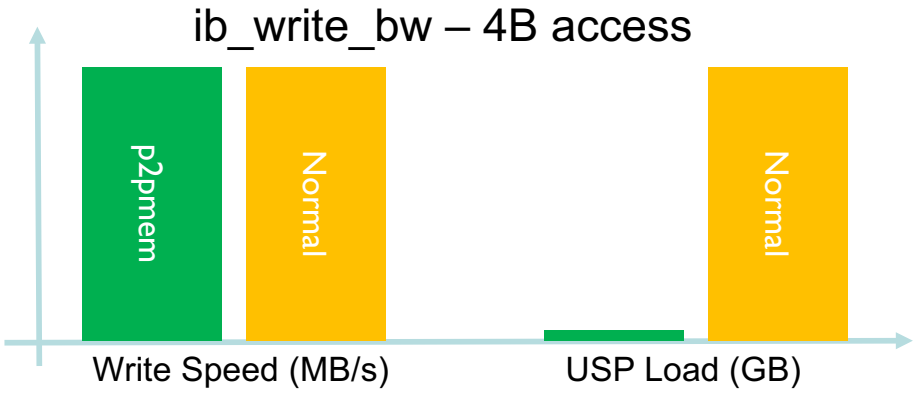
- Use InfiniBand perf tools (e.g. ib_write_bw) with the mmap option to mmap /dev/p2pmem0
- Remote client access that memory using RDMA verbs.
- Measure performance and CPU load in classic and p2pmem modes…

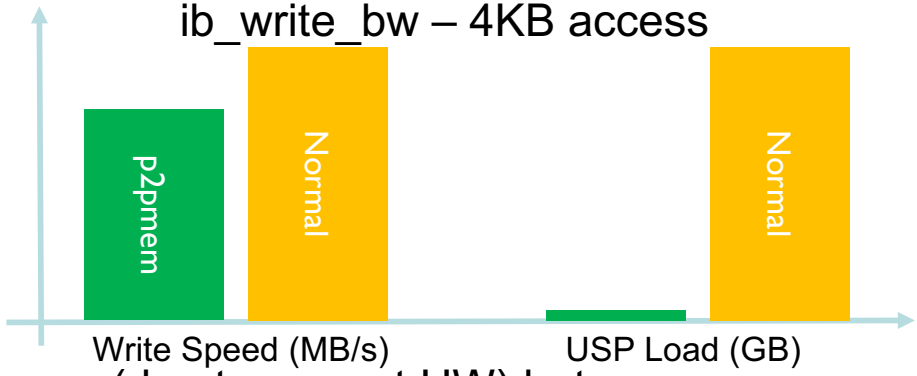


Example – Remote access to (Persistent) PCIe/NVMe Memory



Eideticom NVMe CMB





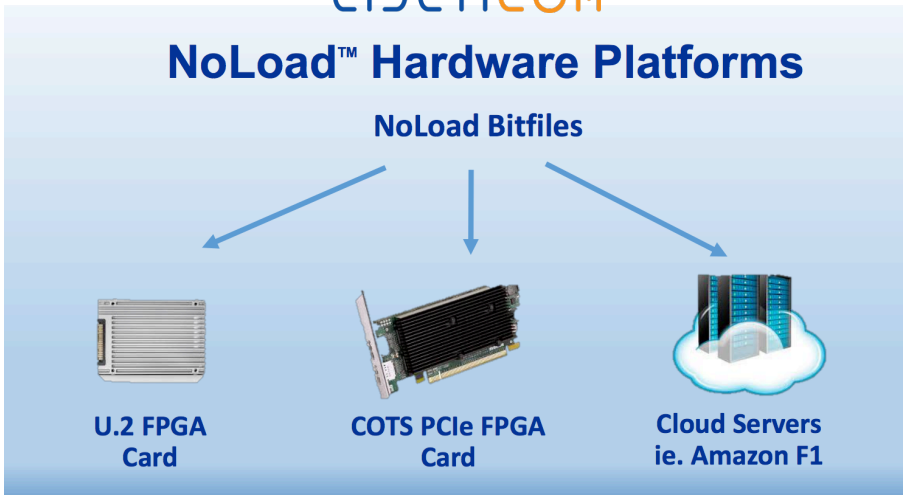
p2pmem mode saturates sooner (due to current HW) but offloads USP by > 6 orders of magnitude!!

The p2pmem Eco-System

EVERSPIN TECHNOLOGIES

Microsemi

EIDETICOM



The p2pmem Upstream Effort

- p2pmem leverages ZONE_DEVICE
  - In x86_64 and AMD
  - Added to ppc64el in 4.13
  - Coming soon for ARM64
- Still some issues around p2pmem patches:
  - IOMEM is not system memory
  - PCIe Routability
  - IOMMU issues

p2pmem Status Summary

ARCH	ZONE_DEVICE	p2pmem	Upstream
x86_64	Yes	Yes	No
ppc64el	Yes	No	No
arm64	No	No	No

Currently working to test p2pmem on ppc64el and add ZONE_DEVICE support to arm64. Note ZONE_DEVICE depends on MEMORY_HOTPLUG and patches for this exist for arm64[1].

[1] https://lwn.net/Articles/707958/

Roadmap

- Upstreaming of p2pmem in Linux kernel ;-)!
- Integration of p2pmem into SPDK/DPDK
- Tie into other frameworks like GPUDirect (Nvidia) and ROCm (AMD/ATI).
- Evolve into emerging memory-centric buses like OpenGenCCIX ;-)