SDC 18
September 24-27, 2018
Santa Clara, CA
www.storagedeveloper.org

Accelerating Storage with NVM Express SSDs and P2PDMA

Stephen Bates, PhD
Chief Technology Officer

EIDETICOM

Outline

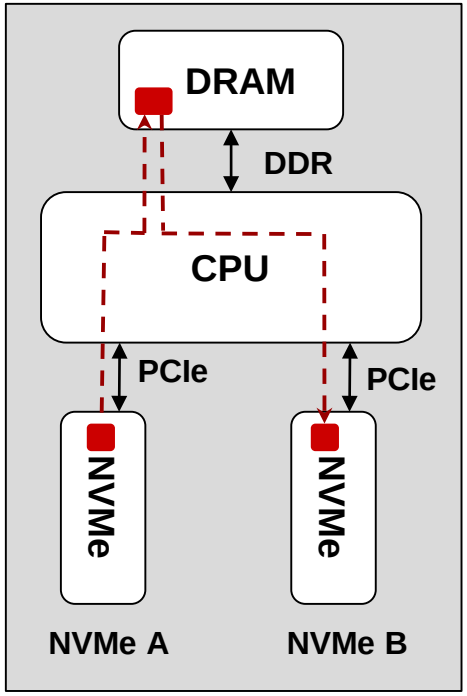
- Motivation for P2PDMA
- What is a P2PDMA?
- An Overview of P2PDMA
  - SPDK
  - Linux Kernel
- NVMe CMBs and P2PDMA
- Applications of P2PDMA
  - NVMe-oF Optimization
  - Offloaded compression

Motivation for P2PDMA

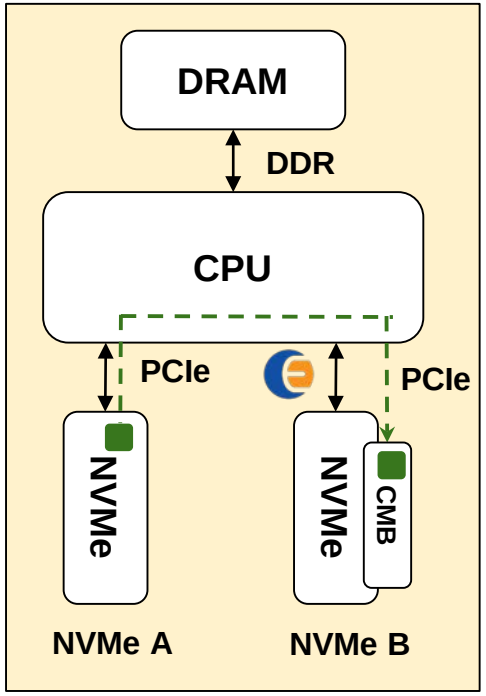
- PCIe devices are getting very fast (e.g. 200GbE NICs, NVMe SSDs, GPGPUs).
- CPUs have lots and lots of PCIe lanes.
- Aggregated IO bandwidth can easily exceed 50GB/s.
- If all DMA traffic has to pass through the CPU memory subsystem a bottleneck occurs.
  - Bandwidth becomes limited
  - Applications on CPUs contend with DMA traffic for memory accesses

What is a P2PDMA?

- P2PDMA transfers bypass CPU memory
- P2PDMAs uses PCIe EP's memory (e.g., NVMe CMB, PCIe BAR)
- A P2P capable Root Complex or PCIe switch is needed
- This path is not enabled in OSes today.



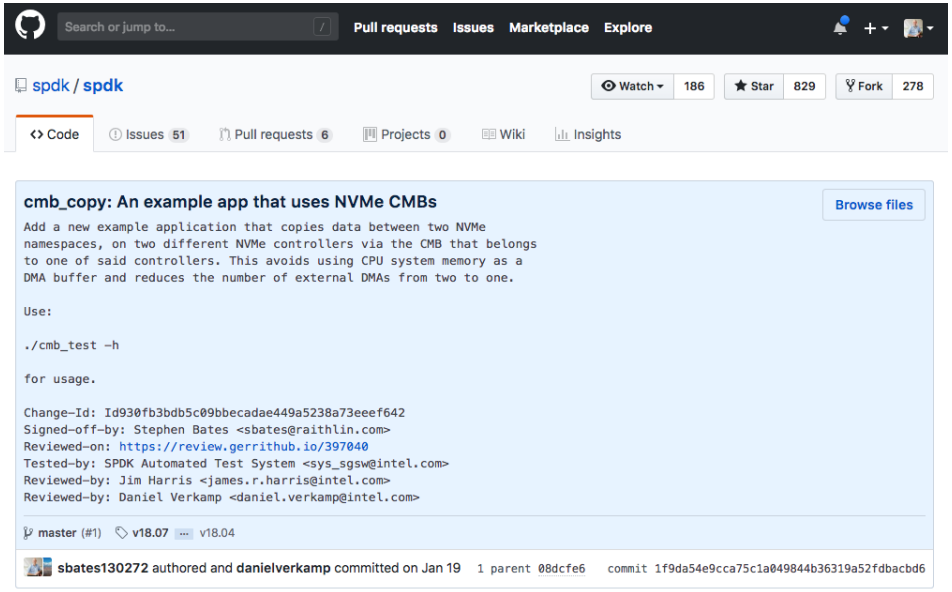
Legacy Datapath



Peer-2-Peer Datapath

An Overview of P2PDMAs: 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).



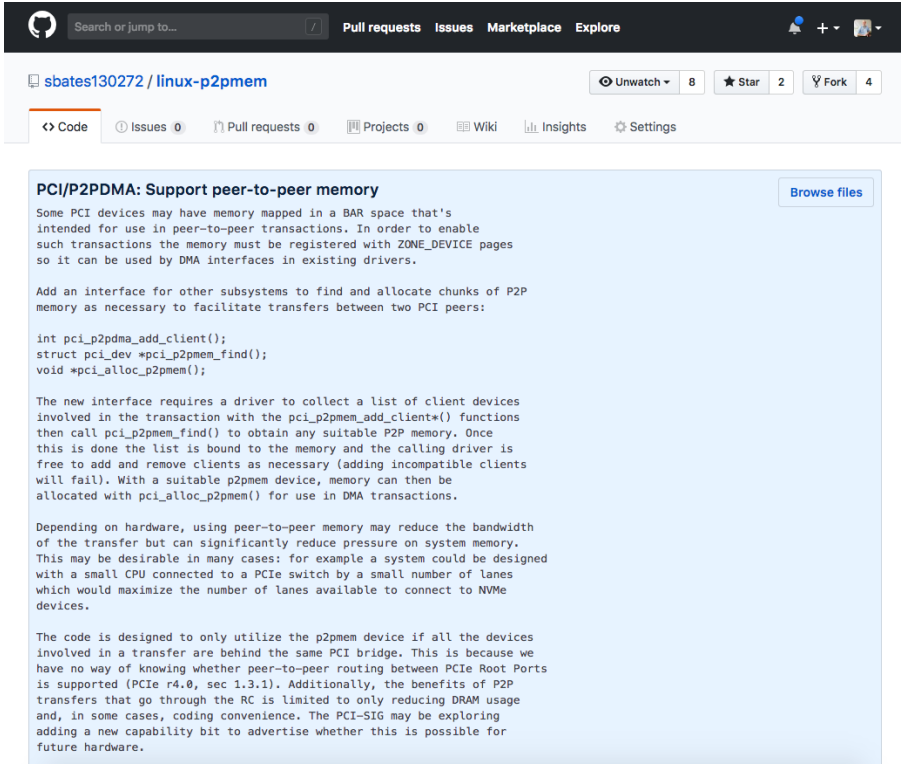
An Overview of P2PDMAs: SPDK

```
buf = spdk_nvme_ctrlr_alloc_cmb_io_buffer(g_config.cmb.ctrlr,
g_config.copy_size);
```

- Allocates a buffer from the CMB of NVMe controller `g_config.cmb.ctrlr` of size `g_config.copy_size` bytes.
- This ensures the DMA engine in the NVMe A SSD sends MemWr or MemRd TLPs to the CMB on the NVMe B SSD.
- Note work still needs to be done on allocator, VFIO and ACS issues.

An Overview of P2PDMAs: Linux Kernel

- The P2PDMA framework proposed for the Linux Kernel is much more general than SPDK.
- P2PDMA supports any PCIe device interested in either:
  - Contributing P2PDMA memory
  - Using P2PDMA memory for DMA
- Central framework for managing P2PDMA memory
- Drivers updated to donate and/or consume P2PDMA memory.
- Currently maintained in linux-p2pmem on GitHub.



An Overview of P2PDMAs: Linux Kernel

- Drivers call the function in red to donate a BAR (or part thereof) to the p2pdma framework.
- This memory is managed by gen_pool, a generic in-kernel allocator.
- It uses devm_memremap_pages() to obtain struct page backing for this memory. This is needed by the DMA API.
- devm_memremap_pages() relies on ZONE_DEVICE which is ARCH specific (x86_64 support only right now, hacks for ARM64 exist).

```
+/**
+ * pci_p2pdma_add_resource - add memory for use as p2p memory
+ * @pdev: the device to add the memory to
+ * @bar: PCI BAR to add
+ * @size: size of the memory to add, may be zero to use the whole BAR
+ * @offset: offset into the PCI BAR
+ *
+ * The memory will be given ZONE_DEVICE struct pages so that it may
+ * be used with any DMA request.
+ */
+int pci_p2pdma_add_resource(struct pci_dev *pdev, int bar, size_t size,
+                            u64 offset)
```

An Overview of P2PDMAs: Linux Kernel

- Drivers call the function in red to find a device which can donate P2PDMA memory in a sane manner.
- We use distance calculations to make a sane decision on which P2PDMA donator memory to use. There may be many devices donating P2PDMA memory in a system (24 NVMe SSDs with CMBs for example).
- The function operations on a list of clients (this list may only have one entry or it may have many).
- Note this can be overridden by configfs. Buyer beware!

```
+/**
+ * pci_p2pmem_find - find a peer-to-peer DMA memory device compatible with
+ *     the specified list of clients and shortest distance (as determined
+ *     by pci_p2pmem_dma())
+ * @clients: list of devices to check (NULL-terminated)
+ *
+ * If multiple devices are behind the same switch, the one "closest" to the
+ * client devices in use will be chosen first. (So if one of the providers are
+ * the same as one of the clients, that provider will be used ahead of any
+ * other providers that are unrelated). If multiple providers are an equal
+ * distance away, one will be chosen at random.
+ *
+ * Returns a pointer to the PCI device with a reference taken (use pci_dev_put
+ * to return the reference) or NULL if no compatible device is found. The
+ * found provider will also be assigned to the client list.
+ */
+struct pci_dev *pci_p2pmem_find(struct list_head *clients)
```

An Overview of P2PDMAs: Linux Kernel

- Drivers call the function in red to obtain memory from the p2pdma device located by the _find() call from the previous slide.
- Since this memory is struct page backed it can be used in DMA operations.
- If NULL is returned the driver can fall back to the traditional memory path.
- Corresponding free() operation returns memory to the pool.
- Note we store the bus addresses in the gen_pool allocator.

```
+/**
+ * pci_alloc_p2p_mem - allocate peer-to-peer DMA memory
+ * @pdev: the device to allocate memory from
+ * @size: number of bytes to allocate
+ *
+ * Returns the allocated memory or NULL on error.
+ */
+void *pci_alloc_p2pmem(struct pci_dev *pdev, size_t size)
```

An Overview of P2PDMAs: Linux Kernel

- The P2PDMA patchset adds support for P2PDMA based memory to the block layer.
- This requires a new scatter-gather mapping and (for now) we require all (or no) entries in the scatter-gather list are p2pdma pages.
- request_queues are also updated to indicate if they want to support P2PDMA memory based IO or not.
- This enables NVMe and other block devices to have P2PDMA IO issues against them.

```
diff --git a/include/linux/blkdev.h b/include/linux/blkdev.h
index d6869e0e2b64..7bf80ca802e1 100644
--- a/include/linux/blkdev.h
+++ b/include/linux/blkdev.h
@@ -699,6 +699,7 @@ struct request_queue {
 #define QUEUE_FLAG_SCSI_PASSTHROUGH 27 /* queue supports SCSI commands */
 #define QUEUE_FLAG_QUIESCED    28      /* queue has been quiesced */
 #define QUEUE_FLAG_PREEMPT_ONLY        29      /* only process REQ_PREEMPT
requests */
+#define QUEUE_FLAG_PCI_P2PDMA  30      /* device supports pci p2p requests */

 #define QUEUE_FLAG_DEFAULT     ((1 << QUEUE_FLAG_IO_STAT) |            \
                                 (1 << QUEUE_FLAG_SAME_COMP)    |       \
@@ -731,6 +732,8 @@ bool blk_queue_flag_test_and_clear(unsigned int flag, struct
request_queue *q);
 #define blk_queue_dax(q)       test_bit(QUEUE_FLAG_DAX, &(q)->queue_flags)
 #define blk_queue_scsi_passthrough(q)  \
        test_bit(QUEUE_FLAG_SCSI_PASSTHROUGH, &(q)->queue_flags)
+#define blk_queue_pci_p2pdma(q)        \
+       test_bit(QUEUE_FLAG_PCI_P2PDMA, &(q)->queue_flags)

 #define blk_noretry_request(rq) \
        ((rq)->cmd_flags & (REQ_FAILFAST_DEV|REQ_FAILFAST_TRANSPORT| \
```

An Overview of P2PDMAs: Linux Kernel

Our userspace code which includes the host API and several example applications is location at [1]. There is a pretty good README located there and this code works on any ARCH.

The latest p2pdma kernel patches are here [2]. Note that we use an additional patch [3] which is not going upstream to expose p2pdma to userspace (we can discuss that more at SDC. We have a longer term plan for userspace.). The p2pdma stuff only works on Intel and AMD but with additional hacky patches we can get it up on ARM64 [4] and are working to enable it on RISC-V too. We also have a simple p2p copy application which we use a lot for debug and performance testing [5] and fio also has support for using p2pdma memory via the iomap flag [6]. We don't have a QEMU model for NoLoad but we do use the NVMe model in QEMU for p2pdma testing. We added support for NVMe models with CMBs a while back to the upstream QEMU [7]. Finally I try and maintain a script that always builds the latest p2pdma kernel with sane .config options for x86_64 [8].

[1] https://github.com/Eideticom/NoLoad-Demos

[2] https://lkml.org/lkml/2018/9/13/54

[3] https://github.com/sbates130272/linux-p2pmem/commit/9a5eccff0781f455ac6b2b146007f93c480166ff

[4] https://github.com/sbates130272/linux-p2pmem/commits/pci-p2p-v5-plus-ioremap

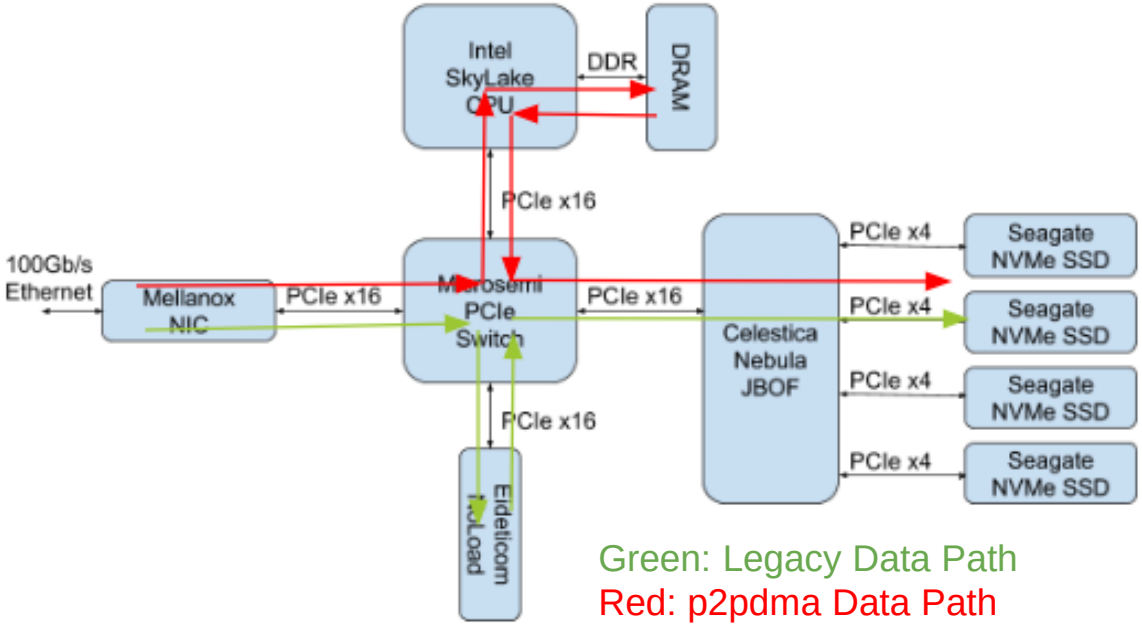
[5] https://github.com/sbates130272/p2pmem-test

[6] https://github.com/axboe/fio/blob/master/HOWTO [see iomap mapshared option]

[7] https://github.com/qemu/qemu/commit/b2b2b67a0057407e19cfa3fdd9002db21ced8b01

[8] https://github.com/sbates130272/kernel-tools/blob/master/build-latest-p2pdma-kernel [pull the whole kernel-tools repo and run this script, only supports Debian and derivates right now].

Applications of P2PDMA: NVMe-oF



Green: Legacy Data Path
Red: p2pdma Data Path

Applications of P2PDMA: NVMe-oF

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

- p2pdma provides x50 offload from the host CPU's memory subsystem compared to vanilla NVMe-oF driver code.
- Can be combined with other optimizations (like NVMe command offload from Mellanox).

Applications of P2PDMA:Offloaded compression

- Eideticom NoLoad customer
- Required libz compression
- Hyper-converged environment
- Wanted to use P2PDMA to minimize DMA impact on VMs
- U.2 form-factor required.
- Data located on standard NVMe SSDs with ext4.

Eideticom NoLoad™ U.2

- **Standard U.2 SSD form-factor:** Utilizing SFF-8639 connector.
- **PCIe Gen4 ready:** 16GB/s of data ingestion/egestion.
- **Eideticom NoLoad™ IP:**
  - NVM Express end-point
  - Storage and analytics accelerator: RAID, EC, Compression
  - NVMe SGL support.
  - CMB and PMR support.
- **Available Now: sales@eideticom.com**

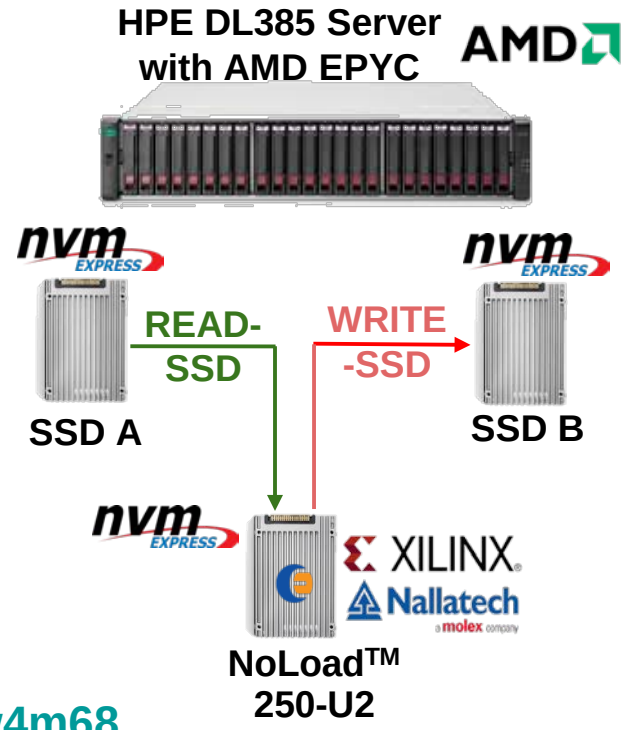
Confidential

12

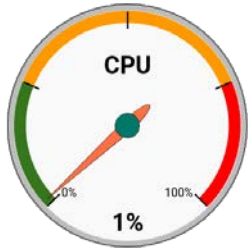
Applications of P2PDMA:Offloaded compression

- NoLoad™ with three compression cores
- Process steps:
  1. **SSD-A → NoLoad::CMB**
  2. **NoLoad Compression**
  3. **NoLoad::CMB → SSD-B**
- Eideticom's P2P Compression demo with Xilinx, AMD, HP:

www.youtube.com/watch?v=4Sg8cgw4m68



3+ GB/s Compression T'put per NoLoad



<1% load on CPU

Conclusions

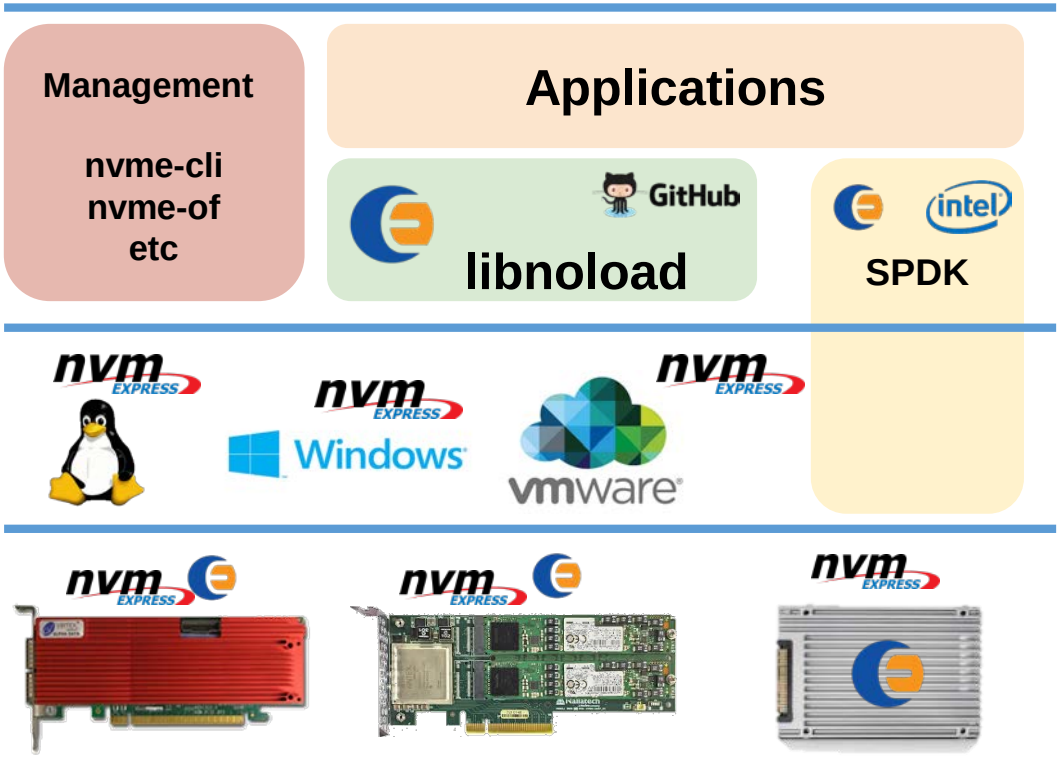
- P2PDMAs make a lot of sense as PCIe devices speed up.
- SPDK has upstream support.
- Linux kernel will (hopefully) be fully supported by 4.20 or 4.21
- Initial applications are NVMe centric but others will follow
- Go forth and test!

Thanks!

Eidetic Communications Inc.
3553 31st NW, Calgary, AB
Canada T2L 2K7

www.eideticom.com, sales@eideticom.com

NoLoad™ Software



- **Userspace:** both kernel & userspace frameworks supported
- **OS:** use inbox NVMe driver (no changes)
- **Hardware:** NoLoad™ Hardware Eval Kits