Enabling Remote Access to Persistent Memory on an IO Subsystem using NVM Express and RDMA

Stephen Bates, PhD

Microsemi
Otherwise Known As
iopmem: pmem for MMIO

Stephen Bates, PhD

Microsemi
PCIe IO devices are getting faster.

Almost always these PCIe devices have either a high performance DMA engine, a number of exposed PCIe BARs or both.

Until this work, (almost) any high-performance transfer of information between two PCIe devices has required the use of a buffer in system memory.

With this patch to Linux the bandwidth between CPU cores and system memory is not compromised when high-throughput transfers occur between PCIe devices.
PCIe NVM Express 101

- NVMe defines a PCIe memory regions and the fields within that region.
- The CPU can mmap/ioremap those regions and then request IO.
- Admin commands used at boot/probe time to configure queues and doorbells.
# How NVM Express Works

The anatomy of a single NVMe Read command.

~10us total time for QD=1 NVMe Read
The Hardware

- NVMe complaint DRAM based SSD.
- Presents to host as a NVMe device (see next slide).
- Also can present some or all of the DRAM as a PCIe BAR (aka NVMe CMB).

Shipping in volume today!
The Hardware

bateste@cgy1-donard:~/qemu-minimal$ sudo lspci -vvv -s 01:00.0

01:00.0 Non-Volatile memory controller: PMC-Sierra Inc. Device f117 (rev 06) (prog-if 02 [NVM Express])

  Subsystem: PMC-Sierra Inc. Device f117
  Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAabort- <TAabort- <MAabort- >SERR- <PERR- INTx-
  Latency: 0, Cache Line Size: 64 bytes

  Interrupt: pin A routed to IRQ 16

  Region 0: Memory at dfa00000 (64-bit, non-prefetchable) [size=16K]

  Region 4: Memory at 381f80000000 (64-bit, prefetchable) [size=1G]

Standard NVMe BAR, standard driver maps this.

1GB CMB we can use as we see fit.
NVMe Controller Memory Buffers

- NVMe specification defines how to advertise a CMB.
- A CMB can be used for a variety of things.
- In this case we use it as a staging area for write/read data.
- NVMe standard is looking at adding persistence to CMBs.
When latency is below 20us it becomes interesting to understand what is contributing to that latency.

What part is due to the SSD and what part is due to the OS and CPU architecture.

We did some measurements to find out.
Latency Analysis

- The IO time contribution from the NVRAM is very consistently about 8-9us (see next slide).
- The IO time measured at the application is ~10us more. This is mostly software overhead.
- The application overhead is also more “spikey”. Why is that?
Latency Analysis

- This is the SSD IO time as a distribution.

- Very tight distribution with a mean of under 9us.

- This SSD is very consistent.
There have been some recent additions to the kernel in preparation of new memory types on the memory channel:

- **ZONEDEVICE**: 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.
How pmem.c works

- User reserves a memory range using a kernel boot option (e.g. memmap=2G!14G).
- The pmem driver binds to this reserved memory region and presents it to user-space as a /dev/pmem0 block device.
- The user can put filesystems on the block device and then run IO to/from it.
### PMEM Performance

<table>
<thead>
<tr>
<th></th>
<th>QD=1, T=1</th>
<th>QD=128, T=1</th>
<th>QD=128, T=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMEM</td>
<td>3.0us (1050MB/s)</td>
<td>520us (1050MB/s)</td>
<td>510us (7500MB/s)</td>
</tr>
<tr>
<td>NVMe (DRAM)</td>
<td>16.0us (228MB/s)</td>
<td>765us (663MB/s)</td>
<td>959us (3980MB/s)</td>
</tr>
<tr>
<td>NVMe (NAND)</td>
<td>88us (42MB/s)</td>
<td>794us (64MB/s)</td>
<td>2554 (1544MB/s)</td>
</tr>
</tbody>
</table>

**Gory Details:** Linux cgy1-donard 4.6.0+3-00004-ga573b70 #118 SMP Fri Jun 3 15:21:30 MDT 2016 x86_64 GNU/Linux. 8 cores on two sockets, Intel(R) Xeon(R) CPU E5-2609 0 @ 2.40GHz, memmap=2G!14G, fio-2.2.10-17-g217b0, 512B random read, direct IO, libaio engine, DDR3 4GB DIMMs @ 8+1 Bytes @ 1066MHz.
Our proposal is an 88 line change to the kernel that enables struct page support for ZONEDEVICE memory that resides in IO memory space.

To provide an example usage of our change we provide an example driver (iopmem.c) that can be bound to any PCIe BAR in the system.

Note we are NOT proposing iopmem.c be added to the kernel (though it could be). It would be assumed that PCIe device vendors would update their drivers (e.g. nvme) using our example if they wished to avail of the services provided by the change.

A self-contained PCIe driver. Note that this driver binds to the device so it WILL replace any other driver for that device (e.g. NVMe).

Has a module parameter to identify which of the BARs you want to utilize. For now, the entire BAR is used.

The driver registers a DAX enabled block device and presents this to the OS. The size of the block device is the size of the BAR.

The driver also creates a character device and this can be mmap()ed. This would be for advanced users who do their file layout in user-space.

[i] https://github.com/sbates130272/iopmem
RDMA <-> IOPMEM

- The DMA engines in the RDMA device can target the BAR on the IOPMEM device.
- Can either use the mmap() on the IOPMEM direct OR mmap() files on a DAX mounted filesystem as the RDMA Memory Regions.
- Our system achieved 4GB/s IOPMEM writes and 1.2GB/s IOPMEM reads (limits of our IOPMEM hardware)

Tested using IB, iWARP and RoCE!
RDMA <-> IOPMEM

- Sub 3us read times for small access sizes.
- CX4, RoCE and no RDMA switch.
- Latency rises for larger accesses. Note these are unaligned and byte addressable.
- Results for 4KB are around 6us. This is 6us total, not incremental!

*This is, in essence remote access to a NVMe Persistent CMB. i.e. PMEM over Fabrics!*
The DMA engines on the NVMe device can target the BAR on the IOPMEM device.

Can either use the mmap() on the IOPEM direct OR mmap() files on a DAX mounted filesystem.

Our system achieved 2GB/s IOPMEM writes and 1GB/s IOPMEM reads (limits of the NVMe hardware).
Use Case 1

Background data copying between NVMe devices.

Can use extra processing on the NVMe SSDs to calculate RAID parity, perform dedupe etc.

Host OS still in control (it is issuing all the submission queue entries) but now all DMA traffic is below the PCIe switch.

CPU->DRAM bandwidth is not compromised by DMA traffic. Frees up memory space and bandwidth for other tasks.
Use Case 2

Use FPGA with exposed BAR as iopmem device.

Now it can receive data from standard NVMe SSDs and perform RAID, encryption, dedupe, data search etc.

The FPGA device could even present to the OS as a NVMe device with a CMB! A vendor specific command set could define the acceleration functions.

Again all DMA traffic sits below PCIe switch, offloading the CPU.
Use Case 3

RDMA NIC can push data direct to NVMe CMBs.

Offloads CPU memory subsystem and reduces NVMe latency.

Could be used to improve performance in a NVMe over Fabrics target system.
Use Case 4

RDMA NIC can push data direct to one NVMe w/CMBs. This SSD works as a write-back cache.

Data is then lazily copied out of the NVMe SSD w/CMB onto standard NVMe SSDs.

Avoids the need for all SSDs to be CMB enabled (cost reduction).
Use Case 5

Like use case 4 but add the FPGA card into the mix to offload storage services from the CPU.

You get the idea. Many other permutations possible and we ain’t even added any GPUs yet ;-).
Conclusions

- The Linux kernel has been adapting in preparation of new NVMs and memory attached NVM. We can leverage this work.
- We extend existing work to the PCIe IO memory subsystem and enable IO memory as a DMA target.
- Our example driver exposes IOMEM as both a DAX enabled block device and a mmap( )’able region.
- We show good performance between PCIe devices and note that datapath avoids CPU when a PCIe switch is used.
- There are many interesting use cases when PCIe DMA traffic can be offloaded from the CPU!
Simple Math

\[
\begin{align*}
\text{NVMe} & + \text{RDMA} \\
\text{--------} & \text{--------} \\
\text{AWESOME} & \\
\text{PMEM} & + \text{RDMA} \\
\text{--------} & \text{--------} \\
\text{AWESOME}^2 & \\
\end{align*}
\]