Acceleration

- Storage I/O Bandwidth rapidly increasing
- Storage workloads taxing on host CPU
- FPGAs provide a compelling solution for storage workloads
- NoLoad = NVMe Offload
NoLoad Accelerator

- NoLoad Bitfiles
  - U.2 FPGA Card
  - COTS PCIe FPGA Card
  - Cloud Servers i.e. Amazon F1
Why NVMe?

- Accelerators require:
  - Low latency
  - High throughput
  - Low CPU overhead
  - Multicore awareness
  - QoS awareness

- Accelerators require:
  - Low latency
  - High throughput
  - Low CPU overhead
  - Multicore awareness
  - QoS awareness

Why develop and maintain a driver when NVMe capabilities align so well with accelerator needs and you can have world-class driver writers working on your driver? Real question is “Why not NVMe?”
Accelerator and Controller Architecture

- Host CPU communicates with accelerators via NVMe controller using standard NVMe commands
- NVMe controller pushes and pulls commands and data via DMA engine
- NVMe controller is in-house developed soft controller on a RISC-V
- Board has external DDR for accelerators that require large data storage
- Controller supports command queue and data CMB (using portion of DDR)
- Developed an accelerator wrapper to handle details of NVMe
NVMe for Accelerators

- Presents as NVMe 1.3 device with multiple namespaces
  - One namespace per accelerator
- Accelerators map to namespaces and are discovered using identify namespace command
  - Vendor specific fields provide accelerator specific information
- Configuration using in-situ data path configuration or vendor specific command
- Input data and in-situ configuration are transferred using NVMe Writes to the namespace associated with the accelerator
- Output data and in-situ status are transferred using NVMe Reads to the namespace associated with the accelerator
NVMe for Accelerators

- In-house NVMe controller supports advanced features including queue and data CMB, SGL and NVMe-oF
- Also support peer-to-peer (P2P) operation
- No customized drivers required – all inbox drivers!
- Leverage industry-standard NVMe test tools
  - FIO and nvme-cli
  - Assist with deployment and benchmarking
- Take advantage of rich NVMe ecosystem
  - Can leverage servers and storage systems developed for NVMe
libnoload

- Developed user API to assist with common tasks associated with acceleration over NVMe
- Provides C and C++ libraries
- Handles discovering NoLoad adapters and enumerating accelerators on the adapters
- Provides support to lock/unlock accelerators
- Provides thin wrappers over system calls for writing data to and reading results back from the accelerators
- Handles seamless integration with our accelerator interface IP
- API is BSD licensed and available via our public github
Controller Performance

- Results for single RISC-V core controller implementation
- Saturating bus for ≥32 kB block transfer for Gen3x8 (COTS FPGA card)
- Saturating bus for ≥16 kB block transfer for Gen3x4 (U.2 form factor)
- Focus to date has been on accelerators with ≥16 kB block sizes (i.e. EC, compression)
- Working on multicore RISC-V system that drastically improves small block performance
NVMe-over-Fabrics (NVMe-oF) allows namespaces to be shared across networks.

- Expose NVMe namespaces to client machines using inbox drivers.
- NoLoad is a standard namespace:
  - Can share it in the same way as any other NVMe device.
Clients request to borrow namespace(s) from server
- Recall that accelerators map to namespaces
- Client given access to the namespace (aka accelerator) over the connection

Clients

RDMA or TCP/IP Network

Servers

NVMe SSDs

NoLoad™ U.2

NVMe-over-Fabrics
Clients see newly acquired namespaces as local NVMe block devices

Normal NVMe operations can be executed as if resources local attached to client machine
Case Study: Compression-over-Fabrics

- Demonstrate GZIP compression-over-fabrics
  - Both RoCE and TCP/IP networking
- Both NoLoad and generic NVMe SSD located on remote server
- U.2 accelerator form factor (Gen 3x4)
- Local client running the application is unaware that it is using an over-Fabrics acceleration device
  - User space code is exactly the same direct attach, over-Fabrics, or peer-to-peer

Local Client running application

- x86 Client
- NIC
- High-speed Network

- Broadcom
- Stingray™ Server
- NoLoad™ U.2
- Generic NVMe SSD

- SDC 2018 Storage Developer Conference. © Eideticom. All Rights Reserved.
Case Study: Compression-oF Results

- Compression over RoCE or TCP/IP attains same throughput as direct attach
- Fabrics latency is hidden by having multiple compression operations in flight
- More about impact on target machine to follow
Case Study: EC-over-Fabrics

- ISA-L compatible Reed-Solomon based EC over RoCE and TCP/IP
- Supports up to 32+4 disk groups with block sizes ranging from 16kB to 128kB
- Gen3x16 accelerator form factor
- Both NoLoad and generic NVMe SSD located on remote server
- User space code is exactly the same direct attach, over-Fabrics, or peer-to-peer
Case Study: EC-oF Results

- Client software is not aware that it is performing EC over fabrics connection
- Results have a small latency penalty vs direct attach results
Mitigating Target Impact (CPU)

- What is the impact on resources in target machine?
  - All transfers flow into and out of DDR on CPU
  - Target CPU has to process all commands
  - How to mitigate?
- NVMe-oF Offload allows the NIC to directly connect to NVMe devices
  - Using Mellanox ConnectX-5 can offload the NVMe work from the target CPU

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency (read/write) us</th>
<th>CPU Utilization</th>
<th>CPU Memory Bandwidth</th>
<th>CPU PCIe Bandwidth</th>
<th>NVMe Bandwidth</th>
<th>Ethernet Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla NVMe-oF</td>
<td>188/227</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>ConnectX-5 Offload</td>
<td>128/138</td>
<td><strong>0.02</strong></td>
<td><strong>2.40</strong></td>
<td>1.03</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Mitigating Target Impact (Memory)

- NVMe CMB (Controller Memory Buffer) is a PCIe BAR that can be used for Submission and Completion Queues, PRPs, SGLs, and data
- PCIe drivers can register memory or request access to memory for DMA
- P2P framework called p2pmem is being proposed for Linux kernel
- P2P DMA allows us to bypass CPU DRAM
## Processor Offload

- NVMe Offload + p2pmem = Big Savings

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency (read/write) us</th>
<th>CPU Utilization</th>
<th>CPU Memory Bandwidth</th>
<th>CPU PCIe Bandwidth</th>
<th>NVMe Bandwidth</th>
<th>Ethernet Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla NVMe-oF</td>
<td>188/227</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>ConnectX-5 Offload</td>
<td>128/138</td>
<td>0.02</td>
<td>2.40</td>
<td>1.03</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Eideticom NoLoad p2pmem</td>
<td>167/212</td>
<td>0.55</td>
<td>0.09</td>
<td>0.01</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>CX5 Offload + Eideticom NoLoad p2pmem</td>
<td>142/154</td>
<td>0.02</td>
<td>0.02</td>
<td>0.04</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>