Asynchronous Peer-to-Peer Device Communication

Feras Daoud, Leon Romanovsky

[ 28 March, 2017 ]
Agenda

- Peer-to-Peer communication
- PeerDirect technology
- PeerDirect and PeerDirect Async
- Performance
- Upstream work
Peer-to-Peer Communication
Peer-to-Peer Communication

“Direct data transfer between PCI-E devices without the need to use main memory as a temporary storage or use of the CPU for moving data.”

- **Main advantages:**
  - Allow direct data transfer between devices
  - Control the peers directly from other peer devices
  - Accelerate transfers between different PCI-E devices
  - Improve latency, system throughput, CPU utilization, energy usage
  - Cut out the middleman
PeerDirect Technology
Timeline

2010
- GPUDirect

2011
- GPUDirect P2P

2013
- GPUDirect RDMA
- PeerDirect

2016
- GPUDirect Async
- PeerDirect Async
GPUs use driver-allocated pinned memory buffers for transfers
- RDMA driver use pinned buffers for zero-copy kernel-bypass communication
- It was impossible for RDMA drivers to pin memory allocated by the GPU
- Userspace needed to copy data between the GPU driver’s system memory region and the RDMA memory region
GPUDirect/GPUDirect P2P

- GPU and RDMA device share the same “pinned” buffers
- GPU copies the data to system memory
- RDMA device sends it from there

Advantages
- Eliminate the need to make a redundant copy in CUDA host memory
- Eliminate CPU bandwidth and latency bottlenecks
CPU synchronizes between GPU tasks and data transfer
HCA directly accesses GPU memory

Advantages
• Direct path for data exchange
• Eliminate the need to make a redundant copy in host memory
while(fin) {
    gpu_kernel <<<… , stream>>> (buf);
    cudaMemcpy(stream);
    ibv_post_send(buf);
    ibv_poll_cq(cqe);
}

100% CPU Utilization
GPUDirect Async/PeerDirect Async

- Control the HCA from the GPU
  - Performance
    - Enable batching of multiple GPU and communication tasks
    - Reduce latency
  - Reduce CPU utilization
    - Light weight CPU
    - Less power
- CPU prepares and queues compute and communication tasks on GPU
- GPU triggers communication on HCA
- HCA directly accesses GPU memory
while(fin) {
    gpu_kernel <<<… , stream>>>(buf);
    gds_stream_queue_send(stream, qp, buf);
    gds_stream_wait_cq(stream, cqe);
}
Peer-to-Peer Evolution

GPUDirect
- Eliminate the need to make a redundant copy in CUDA host memory
- Eliminate CPU bandwidth and latency bottlenecks

PeerDirect
- Eliminate the need to make a redundant copy in host memory
- Direct path for data exchange

PeerDirect Async
- Control RDMA device from the GPU
- Reduce CPU utilization
Allow ibv_reg_mr() to register peer memory

Peer devices implement new kernel module – io_peer_mem

Register with RDMA subsystem - ib_register_peer_memory_client()

io_peer_mem implements the following callbacks:

- **acquire()** – detects whether a virtual memory range belongs to the peer
- **get_pages()** – asks the peer for the physical memory addresses matching the memory region
- **dma_map()** – requests the bus addresses for the memory region
- Matching callbacks for release: **dma_unmap()**, **put_pages()** and **release()**
PeerDirect
Memory Region Registration

User-space Verbs App

RDMA Subsystem

Peer Client

Peer Device

HCA

ibv_reg_mr()

(a) acquire()

mine!

(b) get_pages()

Physical Pages

dma_map()

DMA addresses

Register MR

ibv_reg_mr() Success

Pin Peer Pages

Use MR for PeerDirect

OpenFabrics Alliance Workshop 2017
PeerDirect Async
Allows for peer devices to control the network card

- RDMA NIC provides a bytecode sequence to the peer
- Peer device executes bytecode to trigger sends or detect completions

PeerDirect Async uses dedicated QPs and CQs

PeerDirect Async operations

- `ibv_post_send()` on a PeerDirect Async QP queues a set of operations to be triggered by peer
- `ibv_peer_commit_qp()` – Obtain bytecode for committing pending WQEs for execution
- `ibv_peer_peek_cq()` – Obtain bytecode for detecting a certain number of completions

Device agnostic

- An network card that exports bytecode operations for post_send and poll_cq
- Any peer device that can execute the byte code
  - GPUs, FPGAs, Storage controllers, etc.
Create a QP ->
Mark it for PeerDirect Async ->
Associate it with the peer

1. Post work requests using `ibv_post_send()`
   - Doorbell is not ringed
2. Use `ibv-peer_commit_qp()` to get bytecode for committing all WQEs currently posted to the send work queue
3. Queue the translated bytecode operations on the peer
4. Peer executes the operations after generating outgoing data
Create a CQ ->  
Mark it for PeerDirect Async ->  
Associate it with the peer

1. Use `ibv_peer.peek_cq()` to get bytecode for peeking a CQ for a specific number of completions
2. Queue the translated operations on the peer before the operations that use the received data
3. Synchronize the CPU with the peer to insure that all the operations has ended
4. Use `ibv_poll_cq()` to consume the completion entries
Performance mode

[*] modified ud_pingpong test: recv+GPU kernel+send on each side.
2 nodes: Ivy Bridge Xeon + K40 + Connect-IB + MLNX switch, 10000 iterations, message size: 128B, batch size: 20
Economy Mode

Round-trip latency

- RDMA only: 39.28
- RDMA w/IRQ: 178.62
- +Async: 29.46

CPU utilization

- RDMA only: 45% reduction in CPU utilization

[*] modified ud_pingpong test, HW same as in previous slide
Peer-to-Peer – Upstream Proposals

- **Peer-to-Peer DMA**
  - Mapping DMA addresses of PCI device to IOVA of other device
- **ZONE_DEVICE**
  - Extend ZONE_DEVICE functionality to memory not cached by CPU
- **RDMA extension to DMA-BUF**
  - Allow memory region create from DMA-BUF file handle
- **IOPMEM**
  - A block device for PCI-E memory
- **Heterogeneous Memory Management (HMM)**
  - Common address space will allow migration of memory between devices
THANK YOU

Feras Daoud, Leon Romanovsky