GPUrdma: GPU-side library for high performance networking from GPU kernels

Feras Daoud
Technion – Israel Institute of Technology

Mark Silberstein
Technion

Amir Watad
Technion
What

• A GPU-side library for performing RDMA directly from GPU kernels

Why

• To improve communication performance between distributed GPUs

Results

• 5 µsec GPU-to-GPU communication latency and up to 50 Gbps transfer bandwidth
Evolution of GPU-HCA interaction:

Naive Version

Data Path
Control Path
Evolution of GPU-HCA interaction:

Naive Version

GPUDirect RDMA

Direct Data Path

Data Path

Control Path
Evolution of GPU-HCA interaction:

- **Naive Version**
  - Data Path: GPU → RAM → HCA
  - Control Path: GPU → CPU → RAM → HCA

- **GPUDirect RDMA**
  - Data Path: GPU → RAM → HCA (Direct)
  - Control Path: GPU → CPU → RAM → HCA

- **GPUrdma**
  - Data Path: GPU → RAM → HCA (Direct)
  - Control Path: GPU → CPU → RAM → HCA
Motivations

GPUDirect RDMA Node

CPU_rdma_read()
GPU_kernel<<<>>> {
  GPU_Compute()
}
CPU_rdma_write()
Motivations

GPUDirect RDMA Node

CPU_rdma_read()
GPU_kernel<<<>>> {
    GPU_Compute()
}
CPU_rdma_write()

CPU Overhead
Motivations

GPUDirect RDMA Node

- CPU_rdma_read() (Communication)
- GPU_kernel<<<>>> {
  GPU_Compute()
}
- CPU_rdma_write() (Communication)

Bulk-synchronous design and explicit pipelining
Motivations

GPUDirect RDMA Node

CPU_rdma_read()
GPU_kernel<<<>>>(
    GPU_Compute()
)
CPU_rdma_write()

1. kernel calls overhead
2. Inefficient shared memory usage
Motivations

GPUDirect RDMA Node

CPU_rdma_read()
GPU_kernel<<<>>>(
  Find_Even_Num()
)
CPU_rdma_write()

Sparse data

Offsets

0x0
0x3
0x6
GPUrdma library

**GPUrdma Node**

```c
GPU_kernel<<<>>>(
    GPU_rdma_read()
    GPU_Compute()
    GPU_rdma_write()
)
```

- No CPU intervention
- Overlapping communication and computation
- One kernel call
- Efficient shared memory usage
- Send spare data directly from the kernel
InfiniBand Background

1. Queue pair buffer (QP)
   - Send queue
   - Receive queue

2. Work Queue Element
   - Contains communication instructions

3. Completion queue buffer (CQ)
   - Contains completion elements

4. Completion queue element
   - Contains information about completed jobs
InfiniBand Background

Ring the Door-Bell to execute jobs
- MMIO address
- Informs the HCA about new jobs

1. Write work queue element to QP buffer
2. Ring the Door-Bell
3. Check completion queue element status
Agenda

1. Introduction
2. InfiniBand Background
3. GPURdma
4. GPURdma Evaluation
5. GPI2
GPUrdma Node

- Direct path for data exchange
- Direct HCA control from GPU kernels
- No CPU intervention

Native GPU Node
GPUrdma Implementation

![Diagram of GPU and CPU memory with data exchange](image)
GPUrdma Implementation

Data Path - GPUDirect RDMA
GPUrdma Implementation

1. Move QP, CQ to GPU memory
GPUrdma Implementation

1. Move QP, CQ to GPU memory

Modify InfiniBand Verbs
• `ibv_create_qp()`
• `ibv_create_cq()`
GPUrdma Implementation

2. Map the HCA doorbell address into GPU address space
2. Map the HCA doorbell address into GPU address space

---

Modify NVIDIA driver
Agenda

1. Introduction
2. InfiniBand Background
3. GPUrdma
4. GPUrdma Evaluation
5. GPI2
GPURdma Evaluation

- Single QP
- Multiple QP
- Scalability - Optimal QP/CQ location

NVIDIA Tesla K40c GPU    Mellanox Connect-IB HCA
GPURdma – 1 thread, 1 QP

- Best Performance CPU controller VS GPU controller
GPURdma – 1 thread, 1 QP

- GPU controller – Optimize doorbell rings
GPURdma – 1 thread, 1 QP

- GPU controller – Optimize CQ poll
GPURdma – 32 threads, 1 QP

- GPU controller – Write parallel jobs

![Bandwidth vs. Message Size graph](image)
GPUDirect RDMA

• CPU controller
GPURdma – 30 QPs

- 1 QP per Block vs 30 QPs per Block
Scalability – Optimal QP/CQ location:

1. QP and CQ in GPU memory
2. QP in GPU and CQ in system memory
3. CQ in GPU and QP in system memory
4. QP and CQ in GPU memory
Scalability – Optimal QP/CQ location:

1. QP and CQ in GPU memory
2. QP in GPU and CQ in system memory
3. CQ in GPU and QP in system memory
4. QP and CQ in GPU memory
Scalability – Optimal QP/CQ location:

1. QP and CQ in GPU memory
2. QP in GPU and CQ in system memory
3. CQ in GPU and QP in system memory
4. QP and CQ in GPU memory
Scalability – Optimal QP/CQ location:

1. QP and CQ in GPU memory
2. QP in GPU and CQ in system memory
3. CQ in GPU and QP in system memory
4. QP and CQ in system memory
Optimal QP/CQ location:

- Throughput: No difference

- Latency:

<table>
<thead>
<tr>
<th></th>
<th>QP in CPU</th>
<th>QP in GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ in CPU</td>
<td>8.6</td>
<td>6.2</td>
</tr>
<tr>
<td>CQ in GPU</td>
<td>6.8</td>
<td>4.8</td>
</tr>
</tbody>
</table>

Transfer latency [μsec]
Limitations

GPUDirect RDMA - CUDA v7.5:
Running kernel may observe STALE DATA or data that arrives OUT-OF-ORDER

Scenario:
Intensive RDMA writes to GPU memory

Good news:
NVIDIA announced a CUDA 8 feature that enables consistent update

Suggested fix:
CRC32 integrity check API for error detection
GPI2 for GPUs:

**GPI** - A framework to implement **Partitioned Global Address Space (PGAS)**

**GPI2** - Extends this global address space to GPU memory
GPI2 code example

**CPU Node**
- `gaspi_segment_create (CPU_MEM)`
- Initialize data
- `gaspi_write_notify`
- `gaspi_notify_waitsome`
- `gaspi_proc_term`

**GPU Node**
- `gaspi_segment_create (GPU_MEM)`
- `gaspi_notify_waitsome`
- `GPU_Compute_data<<<>>>`
- `gaspi_write_notify`
- `gaspi_proc_term`
GPI2 using GPURdma

**CPU Node**
- `gaspi_segment_create (CPU_MEM)`
- Initialize data
- `gaspi_write_notify`
- `gaspi_notify_waitsome`
- `gaspi_proc_term`

**GPU Node**
- `gaspi_segment_create (GPU_MEM)`
- `GPU_start_kernel <<<>>>
  - `{`
  - `gpu_gaspi_notify_waitsome`
  - `Compute_data()`
  - `gpu_gaspi_write_notify`  
  - `}`
- `gaspi_proc_term`
GPUrdma Multi-Matrix vector product

<table>
<thead>
<tr>
<th>Batch size [Vectors]</th>
<th>GPI2</th>
<th>GPUrdma</th>
</tr>
</thead>
<tbody>
<tr>
<td>480</td>
<td>2.6</td>
<td>11.7</td>
</tr>
<tr>
<td>960</td>
<td>4.8</td>
<td>18.8</td>
</tr>
<tr>
<td>1920</td>
<td>8.4</td>
<td>25.2</td>
</tr>
<tr>
<td>3840</td>
<td>13.9</td>
<td>29.1</td>
</tr>
<tr>
<td>7680</td>
<td>19.9</td>
<td>30.3</td>
</tr>
<tr>
<td>15360</td>
<td>24.3</td>
<td>31.5</td>
</tr>
</tbody>
</table>

- System throughput in millions of 32x1 vector multiplications per second as a function of the batch size

**CPU Node**
- Start timer
- gaspi_write_notify
- gaspi_notify_waitsome
- Stop timer

**GPU Node**
- gpu_notify_waitsome
- Matrix_compute()
- gpu_write_notify

42
Related works

Lena Oden, Fraunhofer Institute for Industrial Mathematics:
• Infiniband-Verbs on GPU: A case study of controlling an Infiniband network device from the GPU
• Analyzing Put/Get APIs for Thread-collaborative Processors

Mark Silberstein, Technion – Israel Institute of Technology:
• GPUnet: networking abstractions for GPU programs
• GPUfs: Integrating a file system with GPUs

Thanks