Accelerator and accelerated systems

046278/236278

Mark Silberstein
EE
Technion
GPURdma: Controlling NIC from GPU
GPUDirect RDMA idea
GPUDirectRDMA: how to use it

CPU_rdma_read()
GPU_kernel<<<>>> {
    GPU_Compute()
}
CPU_rdma_write()
GPUDirectRDMA Problem: CPU invocation

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

Network latency is affected!
GPUDirectRDMA Problem:
kernel must stop to communicate

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

Bulk-synchronous design and explicit pipelining
Real problem

- GPU cannot send/recv network packets without the CPU
If only GPU could issue RDMA calls

**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
GPUrdma: in-GPU networking library

- Eliminate the CPU from the control path!
InfiniBand 101 (again)

Queue pair buffer (QP):
Send queue
Receive queue

Work Queue Element:
Contains communication instructions

Completion queue buffer (CQ):
Contains completion elements

Completion queue element:
Contains information about completed jobs
Doorbell register

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

1. Write work queue element to QP buffer
2. Ring the Door-Bell
3. Check completion queue element status
RDMA control from GPU
Start from the CPU RDMA
Use GPUdirectRDMA to map data buffers into GPU
Use GPUdirectRDMA to map QP/CQ to GPU
Map doorbell register to GPU

Required modifying NVIDIA Driver prior to CUDA 8.0.
But will it be efficient?

- GPUs are very slow sequential machines
- Should we allow each GPU thread use IB VERBs?
- Will IB be overwhelmed with 10000 doorbell writes?
GPUrdma Evaluation

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

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

- 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
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>Location</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

Symptom:
GPURdma 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
Interim results summary

- 5 µsec GPU to GPU communication latency
- GPUrdma outperforms the CPU RDMA for small packets by factor of 4.5X
- 50 Gbps transfer bandwidth for messages from 16KB and larger
Agenda

1. Introduction
2. InfiniBand Background
3. GPUrdma
4. GPUrdma Evaluation
5. GPI2
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 <<<>>>`
  ```cpp
  {
  gpu_gaspi_notify_waitsome
  Compute_data()
  gpu_gaspi_write_notify 
  }
  ```
- `gaspi_proc_term`
GPUrdma Ping-Pong

- 10 µsec roundtrip latency
- 40.5 Gb/sec throughput
- 52 Gb/sec throughput with overlapping
- 38 Gb/sec for GPI2

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

GPU Node
- gpu_gaspi_notify_waitsome
- gpu_gaspi_write_notify
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