Optimized Non-contiguous MPI Datatype Communication for GPU Clusters: Design, Implementation and Evaluation with MVAPICH2


Network-Based Computing Laboratory
The Ohio State University
Outline

• Introduction
• Problem Statement
• Our Solution: MVAPICH2-GPU-NC
• Design Considerations
• Performance Evaluation
• Conclusion & Future Work
InfiniBand Clusters in TOP500

- Percentage share of InfiniBand is steadily increasing
- 41% of systems in TOP500 using InfiniBand (June ’11)
- 61% of systems in TOP100 using InfiniBand (June ‘11)
Growth in GPGPUs

• GPGPUs are gaining significance on clusters for data-centric applications
  – Word Occurrence, Sparse Integer Occurrence
  – K-means clustering, Linear regression

• GPGPUs + InfiniBand are gaining momentum for large clusters
  – #2 (Tianhe-1A), #4 (Nebulae) and #5 (Tsubame) Petascale systems

• GPGPUs programming
  – CUDA or OpenCL + MPI

• Big issues: performance of data movement
  – Latency
  – Bandwidth
  – Overlap
Data Movement in GPU Clusters

- Data movement in InfiniBand clusters with GPUs
  - **CUDA**: Device memory → Main memory  [at source process]
  - **MPI**: Source rank → Destination process
  - **CUDA**: Main memory → Device memory  [at destination process]
MVAPICH/MVAPICH2 Software

- High Performance MPI Library for IB and HSE
  - MVAPICH (MPI-1) and MVAPICH2 (MPI-2.2)
  - Used by more than 1,710 organizations in 63 countries
  - More than 79,000 downloads from OSU site directly
  - Empowering many TOP500 clusters
    - 5th ranked 73,278-core cluster (Tsubame 2.0) at Tokyo Institute of Technology
    - 7th ranked 111,104-core cluster (Pleiades) at NASA
    - 17th ranked 62,976-core cluster (Ranger) at TACC
  - Available with software stacks of many IB, HSE and server vendors including Open Fabrics Enterprise Distribution (OFED) and Linux Distros (RedHat and SuSE)
  - [http://mvapich.cse.ohio-state.edu](http://mvapich.cse.ohio-state.edu)
MVAPICH2-GPU: GPU-GPU using MPI

• Is it possible to optimize GPU-GPU communication with MPI?
  – Support GPU to remote GPU communication using MPI
  – P2P and One-sided were improved
  – Collectives can directly get benefits from p2p improvement

• How to optimize GPU-GPU collectives with different algorithms?
  – Support GPU to GPU Alltoall communication with Dynamic Staging mechanism
  – GPU-GPU Alltoall performance was improved

• How to handle non-contiguous data in GPU device memory?
  – This paper!
  – Support GPU-GPU non-contiguous data communication (P2P) using MPI
  – Vector datatype and SHOC benchmark are optimized
Non-contiguous Data Exchange

Halo data exchange

- Multi-dimensional data
  - Row based organization
  - Contiguous on one dimension
  - Non-contiguous on other dimensions

- Halo data exchange
  - Duplicate the boundary
  - Exchange the boundary in each iteration

Cluster 2011 Austin
Datatype Support in MPI

• Native datatypes support in MPI
  – improve programming productivity
  – Enable MPI library to optimize non-contiguous data transfer

At Sender:

MPI_Type_vector (n_blocks, n_elements, stride, old_type, &new_type);
MPI_Type_commit(&new_type);
...
MPI_Send(s_buf, size, new_type, dest, tag, MPI_COMM_WORLD);

• What will happen if the non-contiguous data is inside GPU device memory?
Outline

• Introduction

• Problem Statement

• Our Solution: MVAPICH2-GPU-NC

• Design Considerations

• Performance Evaluation

• Conclusion & Future Work
Problem Statement

• Non-contiguous data movement from/to GPGPUs
  – Performance bottleneck
  – Reduced programmer productivity

• Hard to optimize GPU-GPU non-contiguous data communication at the user level
  – CUDA and MPI expertise is required for efficient implementation
  – Hardware dependent characteristics, such as latency
  – Different choices of Pack/Unpack non-contiguous data, which is better?
Problem Statement (Cont.)

(a) No pack
(b) Pack by GPU into Host
(c) Pack by GPU inside Device

Which is better?

Cluster 2011 Austin
Outline

• Introduction
• Problem Statement
• Our Solution: MVAPICH2-GPU-NC
• Design Considerations
• Performance Evaluation
• Conclusion & Future Work
Performance for Vector Pack

- Pack latency (similar for unpack)
  - (a) D2H_nc2nc: D2H, non-contiguous to non-contiguous. Pack by CPU later
  - (b) D2H_nc2c: D2H, non-contiguous to contiguous. Pack by GPU directly to host memory
  - (c) D2D2H_nc2c2c: D2D2H, non-contiguous to contiguous inside GPU

\( (c) \) has up to factor of 8 improvement from \( (a) \)!
MVAPICH2-GPU-NC: Design Goals

• Support GPU-GPU non-contiguous data communication through standard MPI interfaces
  – e.g. MPI_Send / MPI_Recv can operate on GPU memory address for non-contiguous datatype, like MPI_Type_vector

• Provide high performance without exposing low level details to the programmer
  – offload datatype pack and unpack to GPU
    • Pack: pack non-contiguous data into contiguous buffer inside GPU, then move out
    • Unpack: move contiguous data into GPU memory, then unpack to non-contiguous address
  – pipeline data pack/unpack, data movement between device and host, and data transfer on networks
    • Automatically provides optimizations inside MPI library without user tuning
Sample Code - Without MPI Integration

- Simple implementation for vector type with MPI and CUDA
  - Data pack and unpack by CPU

```c
MPI_Type_vector(n_rows, width, n_cols, old_datatype, &new_type);
MPI_Type_commit(&new_type);

At Sender:
cudaMemcpy2D(s_buf, n_cols * datasize, s_device, n_cols * datasize, width * datasize, n_rows, DeviceToHost);
MPI_Send(s_buf, 1, new_type, dest, tag, MPI_COMM_WORLD);

At Receiver:
MPI_Recv(r_buf, 1, new_type, src, tag, MPI_COMM_WORLD, &req);
cudaMemcpy2D(r_device, n_cols * datasize, r_buf, n_cols * datasize, width * datasize, n_rows, HostToDevice);
```

- High productivity but poor performance
Sample Code – User Optimized

- Data pack/upack is done by GPU without MPI data type support
- Pipelining at user level using non-blocking MPI and CUDA interfaces

At Sender:

```c
for (j = 0; j < pipeline_len; j++)
    // pack: from non-contiguous to contiguous buffer in GPU device memory
    cudaMemcpy2DAsync(...);
while (active_pack_stream || active_d2h_stream) {
    if (active_pack_stream > 0 && cudaStreamQuery() == cudaSuccess) {
        // contiguous data move from device to host
        cudaMemcpyAsync(...);
    }
    if (active_d2h_stream > 0 && cudaStreamQuery() == cudaSuccess)
        MPI_Isend(....);
}
MPI_Waitall();
```

*Good performance but poor productivity*
Sample Code – MVAPICH2-GPU-NC

- MVAPICH2-GPU-NC: supports GPU-GPU non-contiguous data communication with standard MPI library
  - Offload data Pack and unpack to GPU
  - Implement pipeline inside MPI library

```c
MPI_Type_vector (n_rows, width, n_cols, old_datatype, &new_type);
MPI_Type_commit(&new_type);
```

**At Sender:**
```c
// s_device is data buffer in GPU
MPI_Send(s_device, 1, new_type, dest, tag, MPI_COMM_WORLD);
```

**At Receiver:**
```c
// r_device is data buffer in GPU
MPI_Recv(r_device, 1, new_type, src, tag, MPI_COMM_WORLD, &req);
```

- **High productivity and high performance!**
Outline

• Introduction
• Problem Statement
• Our Solution: MVAPICH2-GPU-NC

• Design Considerations
• Performance Evaluation
• Conclusion & Future Work
Design Considerations

• Memory detection
  – CUDA 4.0 feature *Unified Virtual Addressing (UVA)*
  – MPI library can differentiate between device memory and host memory without any hints from the user
• Overlap data pack/unpack with CUDA copy and RDMA transfer
  – Data pack and unpack by GPU inside device memory
  – Pipeline data pack/unpack, data movement between device and host, and InfiniBand RDMA
  – Allow for progressing DMAs individual data chunks
Pipeline Design

- Chunk size depends on CUDA copy cost and RDMA latency over the network
- Automatic tuning of chunk size
  - Detects CUDA copy and RDMA latencies during installation
  - Chunk size can be stored in configuration file (mvapich.conf)
- User transparent to deliver the best performance
Outline

• Introduction
• Problem Statement
• Our Solution: MVAPICH2-GPU-NC
• Design Considerations
• Performance Evaluation
• Conclusion & Future Work
Performance Evaluation

• Experimental environment
  – NVIDIA Tesla C2050
  – Mellanox QDR InfiniBand HCA MT26428
  – Intel Westmere processor with 12 GB main memory
  – MVAPICH2 1.6, CUDA Toolkit 4.0

• Modified OSU Micro-Benchmarks
  – The source and destination addresses are in GPU device memory

• SHOC Benchmarks (1.0.1) from ORNL
  – Stencil2D: a two-dimensional nine point stencil calculation, including the halo data exchange

• Run one process per node with one GPU card (8 nodes cluster)
Ping Pong Latency for Vector

- **MV2-GPU-NC**
  - 88% improvement compared with Cpy2D+Send at 4MB
  - have the similar latency with Cpy2DAasync+CpyAsync+Isend, but much easier for programming

Cluster 2011 Austin
• Time breakdown for process Rank 1
  – 2x4 process grid; 8K x 8K matrix; 32K bytes halo data for each dimension
  – Contiguous data exchange with Rank 5, non-contiguous data exchange with Rank 0 and Rank 2

• Non-contiguous data dominates data transfer time
## Code Complexity Comparison

- MV2-GPU-NC uses GPU non-contiguous data address as parameters

<table>
<thead>
<tr>
<th></th>
<th>Stencil2D-Default</th>
<th>Stencil2D-MV2-GPU-NC</th>
</tr>
</thead>
</table>
| Function calls         | MPI_Irecv: 4  
MPI_Send: 4  
MPI_Waitall: 2  
cudaMemcpy: 4  
cudaMemcpy2D: 4 | MPI_Irecv: 4  
MPI_Send: 4  
MPI_Waitall: 2  
cudaMemcpy: 0  
cudaMemcpy2D: 0 |
| Lines of code          | 245                                                               | 158                              |

- **36% code reduction in Stencil2D communication kernel!**
Comparing Median Execution Time

- Single precision
  - 1 x 8: only existing non-contiguous data exchange
  - 8 x 1: only existing contiguous data exchange
  - 2 x 4: 60% non-contiguous, 40% contiguous data exchange
  - 4 x 2: 40% non-contiguous, 60% contiguous data exchange

<table>
<thead>
<tr>
<th>Process Grid (Matrix Size / Process)</th>
<th>Stencil2D-Default (sec)</th>
<th>Stencil2D-MV2-GPU-NC (sec)</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 8 (64K x 1K)</td>
<td>0.547788</td>
<td>0.314085</td>
<td>42%</td>
</tr>
<tr>
<td></td>
<td><strong>non-contiguous optimization in this paper</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 x 1 (1K x 64K)</td>
<td>0.33474</td>
<td>0.272082</td>
<td>19%</td>
</tr>
<tr>
<td></td>
<td><strong>contiguous optimization in ISC’11 paper</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x 4 (8K x 8K)</td>
<td>0.36016</td>
<td>0.261888</td>
<td>27%</td>
</tr>
<tr>
<td></td>
<td><strong>both contiguous and non-contiguous optimizations</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 x 2 (8K x 8K)</td>
<td>0.33183</td>
<td>0.258249</td>
<td>22%</td>
</tr>
<tr>
<td></td>
<td><strong>both contiguous and non-contiguous optimizations</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparing Median Execution Time

- Double precision

<table>
<thead>
<tr>
<th>Process Grid (Matrix Size / Process)</th>
<th>Stencil2D-Default (sec)</th>
<th>Stencil2D-MV2-GPU-NC (sec)</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 8 (64K x 1K)</td>
<td>0.780297</td>
<td>0.474613</td>
<td>39%</td>
</tr>
<tr>
<td>8 x 1 (1K x 64K)</td>
<td>0.563038</td>
<td>0.438698</td>
<td>22%</td>
</tr>
<tr>
<td>2 x 4 (8K x 8K)</td>
<td>0.57544</td>
<td>0.424826</td>
<td>26%</td>
</tr>
<tr>
<td>4 x 2 (8K x 8K)</td>
<td>0.546968</td>
<td>0.431908</td>
<td>21%</td>
</tr>
</tbody>
</table>

- **MV2-GPU-NC:**
  - Up to 42% improvement for Stencil2D with single precision data set;
  - Up to 39% improvement for Stencil2D with double precision data set;
Conclusion & Future Work

- GPU-GPU non-contiguous P2P communication is optimized by MVAPICH2-GPU-NC
  - Support GPU-GPU non-contiguous p2p communication using standard MPI functions; improve the programming productivity
  - Offload non-contiguous data pack/unpack to GPU
  - Overlap Pack/Unpack, data movement between device and host, and data transfer on networks
  - get up to 88% latency improvement compared with without MPI level optimization for vector type
  - get up to 42% and 39% improvement compared with default implementation of Stencil2D in SHOC benchmarks
Conclusion & Future Work

• Future work
  – evaluate non-contiguous datatype performance with our design on larger scale GPGPUs cluster
  – improve more benchmarks and applications with MVAPICH2-GPU-NC
  – integrate this design into MVAPICH2 future releases
Thank You!

{wangh, potluri, luom, singhas, ouyangx, sura, panda}

@cse.ohio-state.edu

Network-Based Computing Laboratory

http://nowlab.cse.ohio-state.edu/

MVAPICH Web Page

http://mvapich.cse.ohio-state.edu/

Cluster 2011 Austin