MVAPICH2-GPU: Optimized GPU to GPU Communication for InfiniBand Clusters

D. K. Panda

Network-Based Computing Laboratory
The Ohio State University
Outline

• Introduction
• Problem Statement
• Our Solution: MVAPICH2-GPU
• Design Considerations
• Performance Evaluation
• Conclusion & Future Work
InfiniBand Clusters in Top500

- Percentage share of InfiniBand is steadily increasing
- 41% of systems in TOP 500 using InfiniBand (June ’11)
- 61% of systems in TOP 100 using InfiniBand (June ‘11)

ISC 2011 Hamburg
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
  - Dr. Sumit Gupta briefed industry users at NVIDIA meeting yesterday on programmability advances on GPUs
- 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]

- GPU and InfiniBand require separate memory registration
Collaboration between Mellanox and NVIDIA to converge on one memory registration technique

Both devices can register same host memory:
  - GPU and network adapters can access the buffer
Outline

• Introduction

• Problem Statement

• Our Solution: MVAPICH2-GPU

• Design Considerations

• Performance Evaluation

• Conclusion & Future Work
Problem Statement

- Data movement from/to GPGPUs
  - Performance bottleneck
  - Reduced programmer productivity

- Hard to optimize at the application level
  - CUDA and MPI expertise required for efficient implementation
  - Hardware dependent latency characteristics
  - Hard to support and optimize collectives
  - Hard to support advanced features like one-sided communication
Outline

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

• Support GPU to GPU communication through standard MPI interfaces
  – e.g. enable MPI_Send, MPI_Recv from/to GPU memory

• Provide high performance without exposing low level details to the programmer
  – Pipelined data transfer which automatically provides optimizations inside MPI library without user tuning

• Available to work with
  – GPU Direct
  – Without GPU Direct
Sample Code - without MPI integration

- Naïve implementation with MPI and CUDA

At Sender:
- `cudaMemcpy(s_buf, s_device, size, cudaMemcpyDeviceToHost);`
- `MPI_Send(s_buf, size, MPI_CHAR, 1, 1, MPI_COMM_WORLD);`

At Receiver:
- `MPI_Recv(r_buf, size, MPI_CHAR, 0, 1, MPI_COMM_WORLD, &req);`
- `cudaMemcpy(r_device, r_buf, size, cudaMemcpyHostToDevice);`

- High productivity but poor performance
Sample Code – User optimized code

- Pipelining at user level with non-blocking MPI and CUDA interfaces
- Code repeated at receiver side
- *Good performance but poor productivity*

At Sender:
```c
for (j = 0; j < pipeline_len; j++)
    cudaMemcpyAsync(s_buf + j * block_sz, s_device + j * block_sz, ...);
for (j = 0; j < pipeline_len; j++) {
    while (result != cudaSucess) {
        result = cudaStreamQuery(...);
        if(j > 0) MPI_Test(...);
    }
    MPI_Isend(s_buf + j * block_sz, block_sz, MPI_CHAR, 1, 1, ....);
}
MPI_Waitall();
```
Sample Code – MVAPICH2-GPU

• MVAPICH2-GPU: provides standard MPI interfaces for GPU

At Sender:
   MPI_Send(s_device, size, ...); // s_device is data buffer in GPU

At Receiver:
   MPI_Recv(r_device, size, ...); // r_device is data buffer in GPU

• High productivity and high performance!
Outline

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

• Memory detection
  – CUDA 4.0 introduces *Unified Virtual Addressing (UVA)*
  – MPI library can differentiate between device memory and host memory without any hints from the user

• Overlap CUDA copy and RDMA transfer
  – Pipeline DMA of data from GPU and InfiniBand RDMA
  – Allow for progressing DMAs individual data chunks
Pipelined Design

- Data is divided into chunks
- Pipeline CUDA copies with RDMA transfers
- If system does not have GPU-Direct, an extra copy is required

ISC 2011 Hamburg
Pipeline Design (Cont.)

• 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
• 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, GPUDirect

• OSU Micro Benchmarks
  – The source and destination addresses are in GPU device memory
Ping Pong Latency

- **With GPU-Direct**
  - 45% improvement compared to Memcpy+Send (4MB messages)
  - 24% improvement compared to MemcpyAsync+Isend (4MB messages)

- **Without GPU-Direct**
  - 38% improvement compared to Memcpy+send (4MB messages)
  - 33% improvement compared to MemcpyAsync+Isend (4MB messages)
One-sided Communication

- 45% improvement compared to Memcpy+Put (with GPU Direct)
- 39% improvement compared with Memcpy+Put (without GPU Direct)
- Similar improvement for Get operation
- Major improvement in programming
  - One sided communication not feasible without MPI+CUDA support
Collective Communication

- With GPU Direct
  - 37% improvement for MPI_Gather (1MB) and 32% improvement for MPI_Scatter (4MB)

- Without GPU Direct
  - 33% improvement for MPI_Gather (1MB) and 23% improvement MPI_Scatter (4MB)
Collective Communication: Alltoall

- With GPU Direct
  - 30% improvement for 4MB messages
- Without GPU Direct
  - 26% improvement for 4MB messages
Comparison with Other MPI Stack: Latency

- Open MPI implementation by Rolf vandeVaart (NVIDIA)
  - [https://bitbucket.org/rolfv/ompi-trunk-cuda-3](https://bitbucket.org/rolfv/ompi-trunk-cuda-3)
- Small message latency around **10%** better (1B – 512B)
- Large message latency around **40%** better (4M)

ISC 2011 Hamburg
Comparison with Other MPI Stack: Bandwidth

- Bandwidth for large messages 512K improved by almost *factor of three*
Conclusion & Future Work

- GPUs are key for data-centric applications
- However, data movement is the key bottleneck in current generation clusters with GPUs
- Asynchronous CUDA Copy and RDMA provide opportunity for latency hiding
- Optimizations at application level reduce productivity and achieve sub-optimal performance
- MPI Library can handle this for the users efficiently
- MVAPICH2-GPU shows substantial improvements for MPI operations
  - Will be available in future MVAPICH2 release
- Future work includes collective specific optimizations and applications-level studies with MVAPICH2-GPU
Thank You!

{wangh, potluri, luom, singhas, surs, panda}@cse.ohio-state.edu

Network-Based Computing Laboratory

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

MVAPICL Web Page

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