Support for GPUs with GPUDirect RDMA in MVAPICH2

SC’13 NVIDIA Booth

by

D.K. Panda
The Ohio State University
E-mail: panda@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~panda
Outline

• Overview of MVAPICH2-GPU Project
• GPUDirect RDMA with Mellanox IB adaptors
• Other Optimizations for GPU Communication
• Support for MPI + OpenACC
• CUDA and OpenACC extensions in OMB
Drivers of Modern HPC Cluster Architectures

- Multi-core processors are ubiquitous and InfiniBand is widely accepted
- MVAPICH2 has constantly evolved to provide superior performance
- Accelerators/Coprocessors are becoming common in high-end systems
- How does MVAPICH2 help development on these emerging architectures?
InfiniBand + GPU systems (Past)

• Many systems today have both GPUs and high-speed networks such as InfiniBand

• Problem: Lack of a common memory registration mechanism
  – Each device has to pin the host memory it will use
  – Many operating systems do not allow multiple devices to register the same memory pages

• Previous solution:
  – Use different buffer for each device and copy data
GPU-Direct

- Collaboration between Mellanox and NVIDIA to converge on one memory registration technique
- Both devices register a common host buffer
  - GPU copies data to this buffer, and the network adapter can directly read from this buffer (or vice-versa)

- *Note that GPU-Direct does not allow you to bypass host memory*
MPI + CUDA

• Data movement in applications with standard MPI and CUDA interfaces

At Sender:
cudaMemcpy(s_hostbuf, s_devbuf, ...);
MPI_Send(s_hostbuf, size, ...);

At Receiver:
MPI_Recv(r_hostbuf, size, ...);
cudaMemcpy(r_devbuf, r_hostbuf, ...);

High Productivity and Low Performance

• Users can do the Pipelining at the application level using non-blocking MPI and CUDA interfaces

Low Productivity and High Performance
GPU-Aware MPI Library: MVAPICH2-GPU

- Standard MPI interfaces used for unified data movement
- Takes advantage of Unified Virtual Addressing (>= CUDA 4.0)
- Optimizes data movement from GPU memory

At Sender:

MPI_Send(s_devbuf, size, ...);

At Receiver:

MPI_Recv(r_devbuf, size, ...);

High Performance and High Productivity
Pipelined Data Movement in MVAPICH2

- Pipelines data movement from the GPU, overlaps
  - device-to-host CUDA copies
  - inter-process data movement (network transfers or shared memory copies)
  - host-to-device CUDA copies

- 45% improvement compared with a naïve (Memcpy+Send)
- 24% improvement compared with an advanced user-level implementation (MemcpyAsync+Isend)
Outline

- Overview of MVAPICH2-GPU Project
- GPUDirect RDMA with Mellanox IB adaptors
- Other Optimizations for GPU Communication
- Support for MPI + OpenACC
- CUDA and OpenACC extensions in OMB
GPU-Direct RDMA (GDR) with CUDA

- Network adapter can directly read/write data from/to GPU device memory
- Avoids copies through the host
- Fastest possible communication between GPU and IB HCA
- Allows for better asynchronous communication
- OFED with GDR support is under development by Mellanox and NVIDIA
**GPU-Direct RDMA (GDR) with CUDA**

- OFED with support for GPUDirect RDMA is under work by NVIDIA and Mellanox
- OSU has an initial design of MVAPICH2 using GPUDirect RDMA
  - Hybrid design using GPU-Direct RDMA
    - GPUDirect RDMA and Host-based pipelining
    - Alleviates P2P bandwidth bottlenecks on SandyBridge and IvyBridge
  - Support for communication using multi-rail
  - Support for Mellanox Connect-IB and ConnectX VPI adapters
  - Support for RoCE with Mellanox ConnectX VPI adapters

\[\text{SNB E5-2670} / \]
\[\text{IVB E5-2680V2} \]
\[\text{P2P write: } 5.2 \text{ GB/s} \]
\[\text{P2P read: } < 1.0 \text{ GB/s} \]
\[\text{SNB E5-2670} / \]
\[\text{IVB E5-2680V2} \]
\[\text{P2P write: } 6.4 \text{ GB/s} \]
\[\text{P2P read: } 3.5 \text{ GB/s} \]
Performance of MVAPICH2 with GPU-Direct-RDMA: Latency

GPU-GPU Internode MPI Latency

Small Message Latency

Large Message Latency

Based on MVAPICH2-2.0b
Intel Ivy Bridge (E5-2680 v2) node with 20 cores
NVIDIA Telsa K40c GPU, Mellanox Connect-IB Dual-FDR HCA
CUDA 5.5, Mellanox OFED 2.0 with GPU-Direct-RDMA Patch
Performance of MVAPICH2 with GPU-Direct-RDMA: Bandwidth

GPU-GPU Internode MPI Uni-Directional Bandwidth

Small Message Bandwidth

- 1-Rail
- 2-Rail
- 1-Rail-GDR
- 2-Rail-GDR

Large Message Bandwidth

- 1-Rail
- 2-Rail
- 1-Rail-GDR
- 2-Rail-GDR

Based on MVAPICH2-2.0b
Intel Ivy Bridge (E5-2680 v2) node with 20 cores
NVIDIA Tesla K40c GPU, Mellanox Connect-IB Dual-FDR HCA
CUDA 5.5, Mellanox OFED 2.0 with GPU-Direct-RDMA Patch
Performance of MVAPICH2 with GPU-Direct-RDMA: Bi-Bandwidth

**GPU-GPU Internode MPI Bi-directional Bandwidth**

**Small Message Bi-Bandwidth**
- 1-Rail
- 2-Rail
- 1-Rail-GDR
- 2-Rail-GDR

**Large Message Bi-Bandwidth**
- 1-Rail
- 2-Rail
- 1-Rail-GDR
- 2-Rail-GDR

Based on MVAPICH2-2.0b
Intel Ivy Bridge (E5-2680 v2) node with 20 cores
NVIDIA Telsa K40c GPU, Mellanox Connect-IB Dual-FDR HCA
CUDA 5.5, Mellanox OFED 2.0 with GPU-Direct-RDMA Patch

SC'13 NVIDIA Booth presentation
How can I get started with GDR Experimentation?

- Two modules are needed
  - Alpha version of OFED kernel and libraries with GPUDirect RDMA (GDR) support from Mellanox
  - Alpha version of MVAPICH2-GDR from OSU (currently a separate distribution)
- Send a note to hpc@mellanox.com
- You will get alpha versions of GDR driver and MVAPICH2-GDR (based on MVAPICH2 2.0a release)
- You can get started with this version
- MVAPICH2 team is working on multiple enhancements (collectives, datatypes, one-sided) to exploit the advantages of GDR
- As GDR driver matures, successive versions of MVAPICH2-GDR with enhancements will be made available to the community
Outline

• Overview of MVAPICH2-GPU Project
• GPUDirect RDMA with Mellanox IB adaptors
• Other Optimizations for GPU Communication
• Support for MPI + OpenACC
• CUDA and OpenACC extensions in OMB
Multi-GPU Configurations

- Multi-GPU node architectures are becoming common
- Until CUDA 3.2
  - Communication between processes staged through the host
  - Shared Memory (pipelined)
  - Network Loopback (asynchronous)
- CUDA 4.0 and later
  - Inter-Process Communication (IPC)
  - Host bypass
  - Handled by a DMA Engine
  - Low latency and Asynchronous
  - Requires creation, exchange and mapping of memory handles - overhead
Designs in MVAPICH2 and Performance

- MVAPICH2 takes advantage of CUDA IPC for MPI communication between GPUs
- Hides the complexity and overheads of handle creation, exchange and mapping
- Available in standard releases from MVAPICH2 1.8

---

**SC'13 NVIDIA Booth presentation**
Collectives Optimizations in MVAPICH2: Overview

- Optimizes data movement at the collective level for small messages
- Pipelines data movement in each send/recv operation for large messages
- Several collectives have been optimized
  - Bcast, Gather, Scatter, Allgather, Alltoall, Gatherv, Scatterv, Gatherv, Allgatherv, Alltoallv
- Collective level optimizations are completely transparent to the user
- Pipelining can be tuned using point-to-point parameters
MPI Datatype Support in MVAPICH2

• 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
MPI Datatype Support in MVAPICH2

- Datatypes support in MPI
  - Operate on customized datatypes to improve productivity
  - Enable MPI library to optimize non-contiguous data

At Sender:

```c
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);
```

- Inside MVAPICH2
  - Use datatype specific CUDA Kernels to pack data in chunks
  - Optimized `vector` datatypes Kernel based pack/unpack in MVAPICH2 2.0b
  - Efficiently move data between nodes using RDMA
  - Transparent to the user

Application Level Evaluation (LBMGPU-3D)

- LBM-CUDA (Courtesy: Carlos Rosale, TACC)
  - Lattice Boltzmann Method for multiphase flows with large density ratios
  - 3D LBM-CUDA: one process/GPU per node, 512x512x512 data grid, up to 64 nodes

- Oakley cluster at OSC: two hex-core Intel Westmere processors, two NVIDIA Tesla M2070, one Mellanox IB QDR MT26428 adapter and 48 GB of main memory
Outline

• Overview of MVAPICH2-GPU Project
• GPUDirect RDMA with Mellanox IB adaptors
• Other Optimizations for GPU Communication
• Support for MPI + OpenACC
• CUDA and OpenACC extensions in OMB
OpenACC is gaining popularity
Several sessions during GTC
A set of compiler directives (#pragma)
Offload specific loops or parallelizable sections in code onto accelerators

```c
#pragma acc region
{
  for(i = 0; i < size; i++) {
    A[i] = B[i] + C[i];
  }
}
```

- Routines to allocate/free memory on accelerators
  ```c
  buffer = acc_malloc(MYBUFSIZE);
  acc_free(buffer);
  ```
- Supported for C, C++ and Fortran
- Huge list of modifiers – copy, copyout, private, independent, etc..
Using MVAPICH2 with OpenACC 1.0

- `acc_malloc` to allocate device memory
  - No changes to MPI calls
  - MVAPICH2 detects the device pointer and optimizes data movement
  - Delivers the same performance as with CUDA

```c
A = acc_malloc(sizeof(int) * N);

......

#pragma acc parallel loop deviceptr(A) . . .
//compute for loop

MPI_Send (A, N, MPI_INT, 0, 1, MPI_COMM_WORLD);

......
acc_free(A);
```
Using MVAPICH2 with OpenACC 2.0

- `acc_deviceptr` to get device pointer (in OpenACC 2.0)
  - Enables MPI communication from memory allocated by compiler when it is available in OpenACC 2.0 implementations
  - MVAPICH2 will detect the device pointer and optimize communication
  - Delivers the same performance as with CUDA

```c
A = malloc(sizeof(int) * N);

......

#pragma acc data copyin(A) . . .
{

#pragma acc parallel loop . . .
//compute for loop
MPI_Send(acc_deviceptr(A), N, MPI_INT, 0, 1, MPI_COMM_WORLD);

}

......
free(A);
```
Outline

• Overview of MVAPICH2-GPU Project
• GPUDirect RDMA with Mellanox IB adaptors
• Other Optimizations for GPU Communication
• Support for MPI + OpenACC
• CUDA and OpenACC extensions in OMB
CUDA and OpenACC Extensions in OMB

- OSU Micro-benchmarks are widely used to compare performance of different MPI stacks and networks
- Enhancements to measure performance of MPI communication from GPU memory
  - Point-to-point: Latency, Bandwidth and Bi-directional Bandwidth
  - Collectives: support all collectives.
- Support for CUDA and OpenACC
- Flexible selection of data movement between CPU(H) and GPU(D): D->D, D->H and H->D
- Available from [http://mvapich.cse.ohio-state.edu/benchmarks](http://mvapich.cse.ohio-state.edu/benchmarks)
- Available in an integrated manner with MVAPICH2 stack
Summary

• MVAPICH2 evolving to efficiently support MPI communication on heterogeneous clusters with NVIDIA GPU
• Simplifying task of porting MPI applications to these new architectures
• Optimizing data movement while hiding system complexity from the user
• Users have to still be aware of system configurations and the knobs MVAPICH2 have to offer
• User feedback critical as the implementations mature
Web Pointers

NOWLAB Web Page
http://nowlab.cse.ohio-state.edu

MVAPICH Web Page
http://mvapich.cse.ohio-state.edu