Optimizing and Tuning of GPU and Xeon Phi Support in MVAPICH2

MVAPICH User Group (MUG) Meeting

by

Sreeram Potluri

The Ohio State University

E-mail: potluri@cse.ohio-state.edu

http://www.cse.ohio-state.edu/~potluri

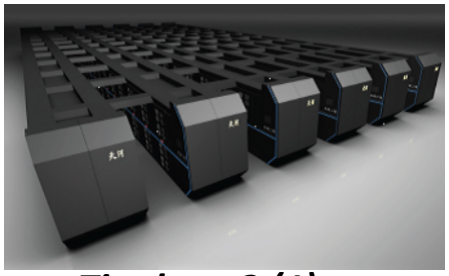
Drivers of Modern HPC Cluster Architectures

Multi-core Processors

High Performance Interconnects - InfiniBand
<1usec latency, >100Gbps Bandwidth

Accelerators / Coprocessors
high compute density, high performance/watt
>1 TFlop DP on a chip

- 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?



Tianhe – 2 (1)

Titan (2)

Stampede (6)

Tianhe – 1A (10)

Outline

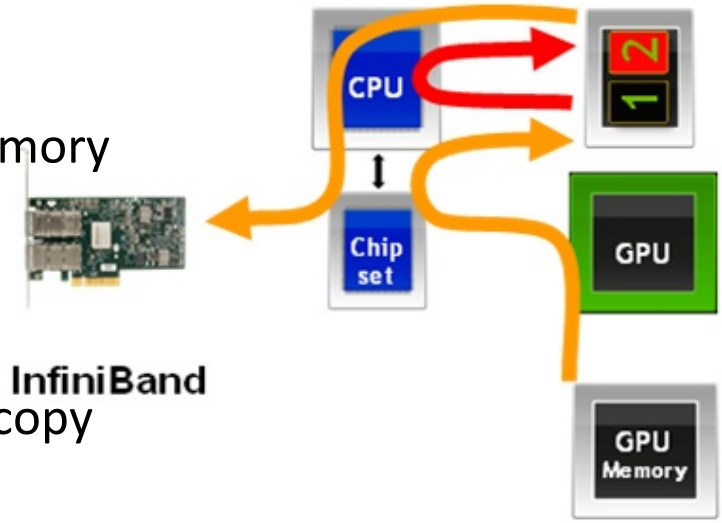
- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
  - Usage: Build, Initialization and Cleanup
  - Support for MPI + OpenACC
  - CUDA and OpenACC extensions in OMB
- MVAPICH2 for Intel Xeon Phi Clusters
- Conclusion

Outline

- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
    - Pipelined data movement
    - GPUDirect RDMA with Mellanox IB
    - CUDA IPC for multi-GPU clusters
    - MPI Collective Communication
    - MPI Datatype Processing
  - Usage: Build, Initialization and Cleanup
  - Support for MPI + OpenACC
  - CUDA and OpenACC extensions in OMB

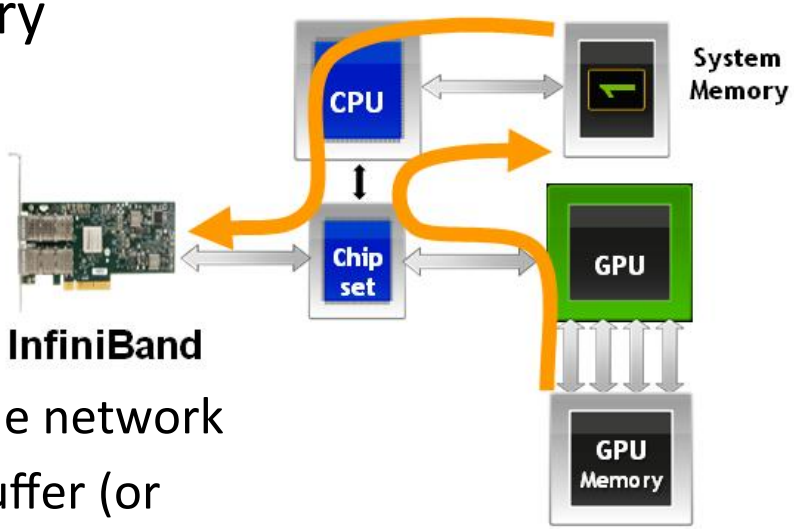
InfiniBand + GPU systems (Past)

- 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 - Naive

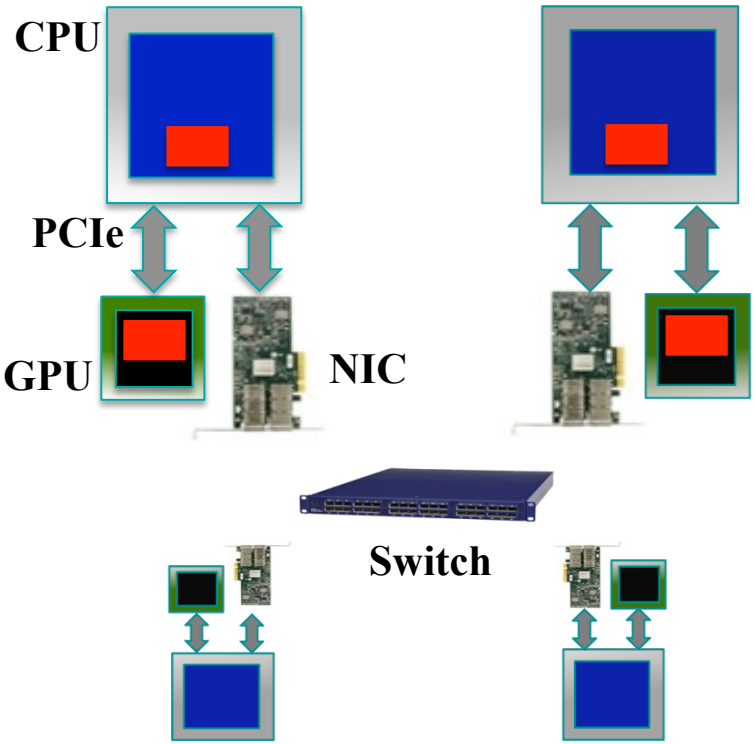
- Data movement in applications with standard MPI and CUDA interfaces

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

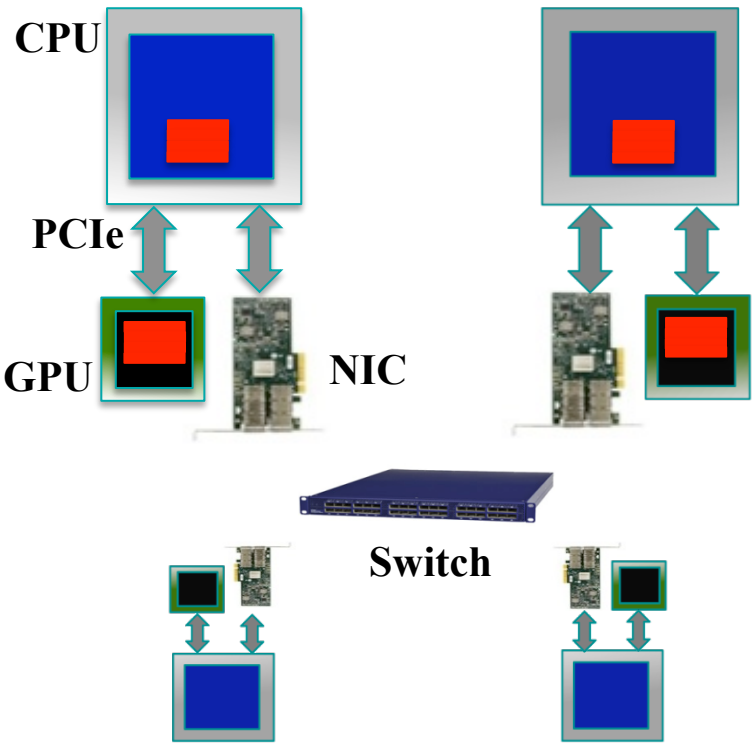
MPI + CUDA - Advanced

- Pipelining at user level with non-blocking MPI and CUDA interfaces

At Sender:

```
for (j = 0; j < pipeline_len; j++)
    cudaMemcpyAsync(s_hostbuf + j * blk, s_devbuf + j * blksz,
      …);
for (j = 0; j < pipeline_len; j++) {
      while (result != cudaSucess) {
            result = cudaStreamQuery(…);
             if(j > 0) MPI_Test(…);
       }
       MPI_Isend(s_hostbuf + j * block_sz, blksz . . .);
 }
MPI_Waitall();
```

<<Similar at receiver>>



Low Productivity and High Performance

Can such optimizations be done within MPI Library?

- 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
  - Pipelining data transfers, using lower level CUDA and GPU features, taking advantage of GPU resources : which *automatically* provides optimizations inside MPI library without user effort
- **New designs were incorporated in MVAPICH2 to support this functionality**

GPU-Aware MPI Library: MVAPICH2-GPU

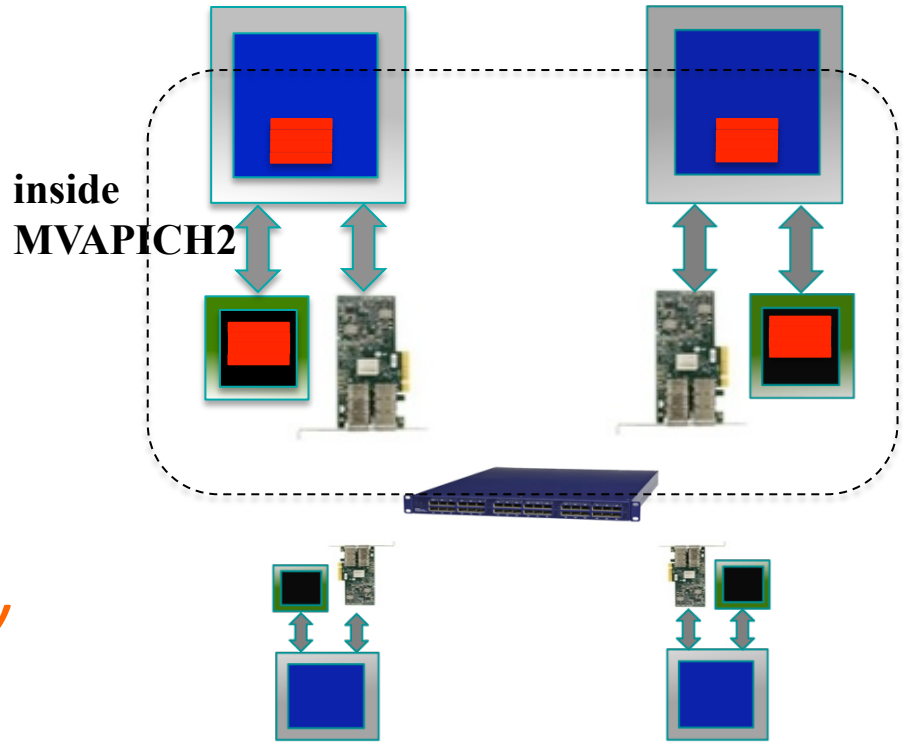
- Standard MPI interfaces used for unified data movement
- Takes advantage of Unified Virtual Addressing (>= CUDA 4.0)
- Optimizes data movement from GPU memory

At Sender:

```
MPI_Send(s_devbuf, size, …);
```

At Receiver:

```
MPI_Recv(r_devbuf, size, …);
```



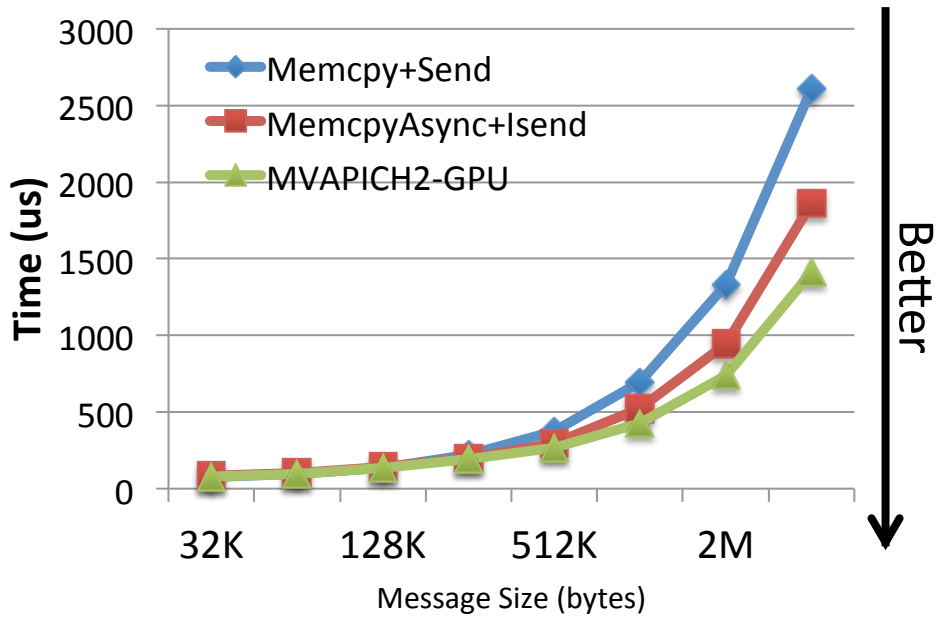
High Performance and High Productivity

Outline

- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
    - Pipelined data movement
    - GPUDirect RDMA with Mellanox IB Adapters
    - CUDA IPC for multi-GPU clusters
    - MPI Collective Communication
    - MPI Datatype Processing
  - Usage: Build, Initialization and Cleanup
  - Support for MPI + OpenACC
  - CUDA and OpenACC extensions in OMB

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



Internode osu_latency large

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

Pipelined Data Movement in MVAPICH2: Tuning

Parameter	Significance	Default	Notes
MV2_USE_CUDA	• Enable / Disable GPU designs	0 (Disabled)	• Disabled to avoid pointer checking overheads for host communication • Always enable to support MPI communication from GPU Memory
MV2_CUDA_BLOCK_SIZE	• Controls the pipeline blocksize	256 KByte	• Tune for your system and application • Varies based on - CPU Platform, IB HCA and GPU - CUDA driver version - Communication pattern (latency/bandwidth)

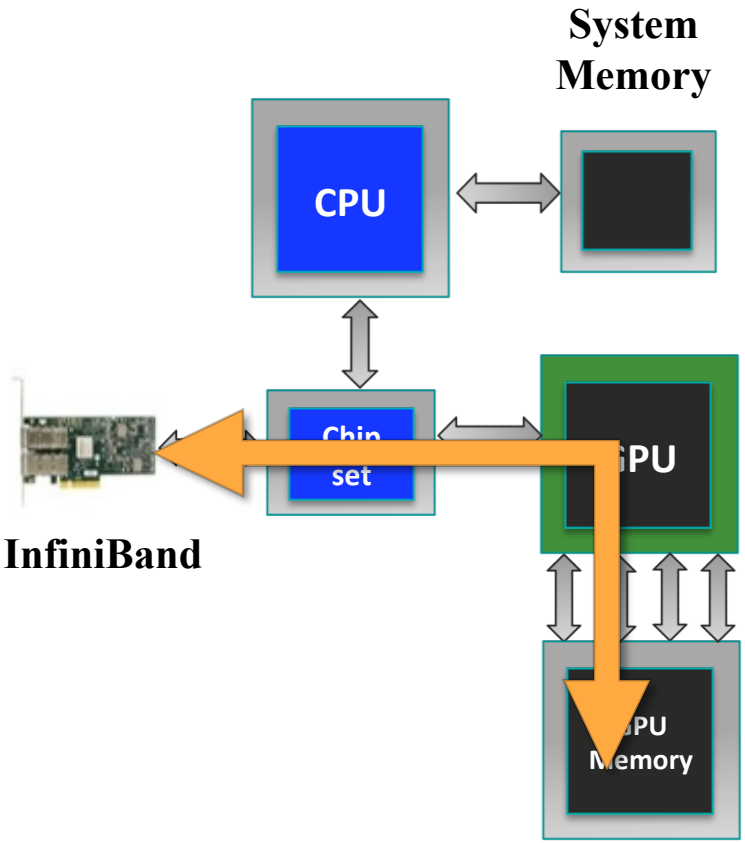
Refer to *"Running on Clusters with NVIDIA GPU Accelerators"* section of the user guide: http://mvapich.cse.ohio-state.edu/support/user_guide_mvapich2-2.0a.html#x1-840006.20

Outline

- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
    - Pipelined data movement
    - GPUDirect RDMA with Mellanox IB Adapters
    - CUDA IPC for multi-GPU clusters
    - MPI Collective Communication
    - MPI Datatype Processing
  - Usage: Build, Initialization, Cleanup
  - Support for MPI + OpenACC
  - CUDA and OpenACC extensions in OMB

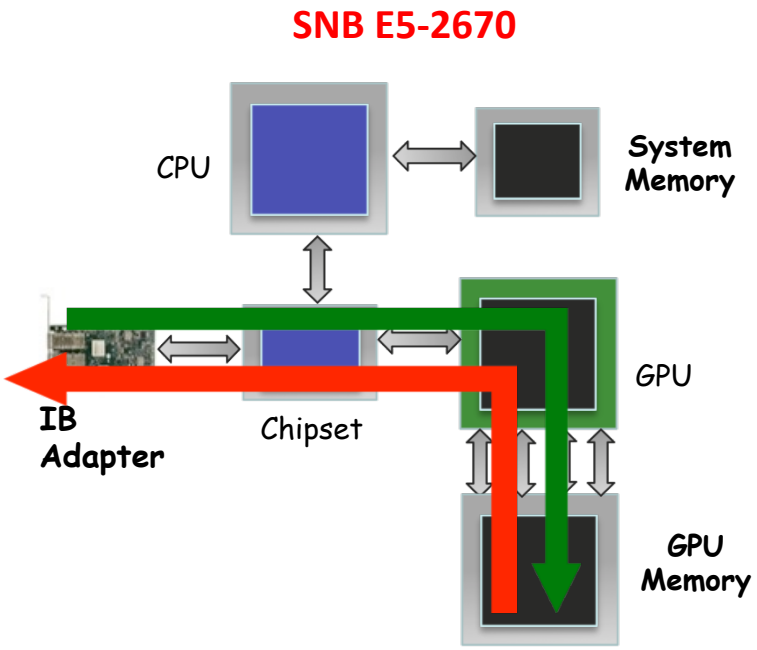
GPU-Direct RDMA (GDR) with CUDA 5.0

- 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



GPUDirect RDMA(GDR) Designs in MVAPICH2: Overview

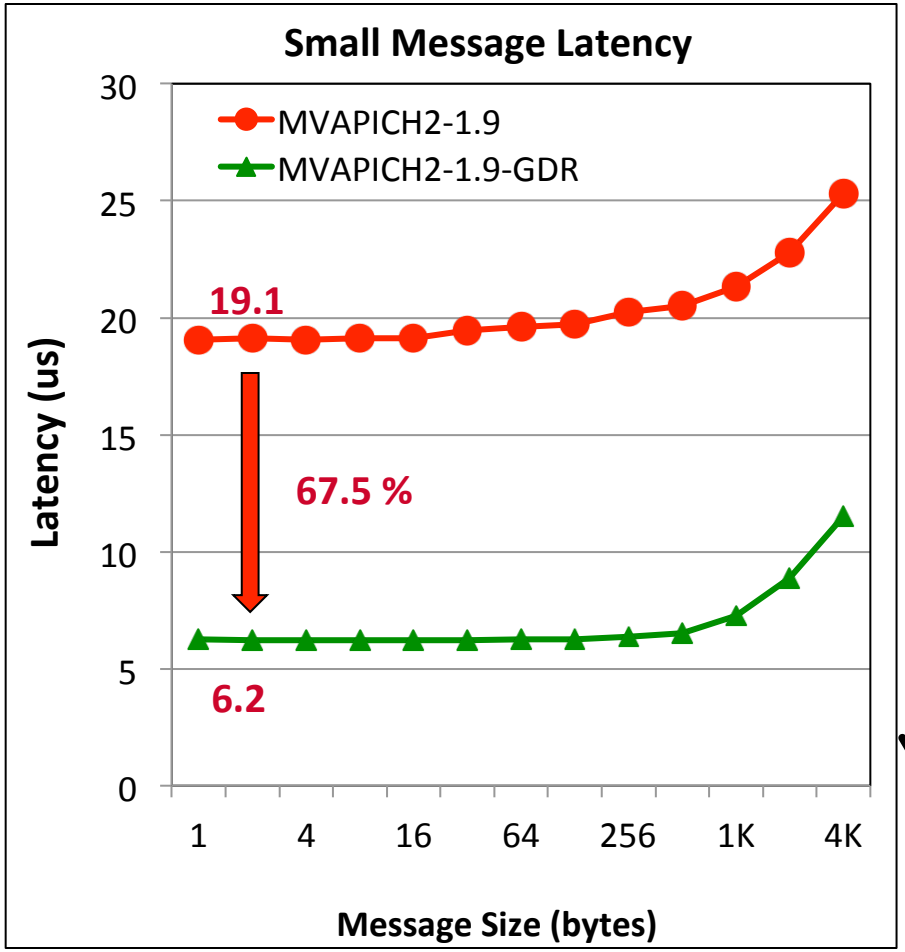
- Peer2Peer (P2P) bottlenecks on Sandy Bridge
- MVAPICH2
  - Provides a hybrid design
  - Takes advantage of GDR for all small messages and for large writes to GPU
  - Uses host-based buffered design in current MVAPICH2 for large reads
  - Works around the bottlenecks transparently



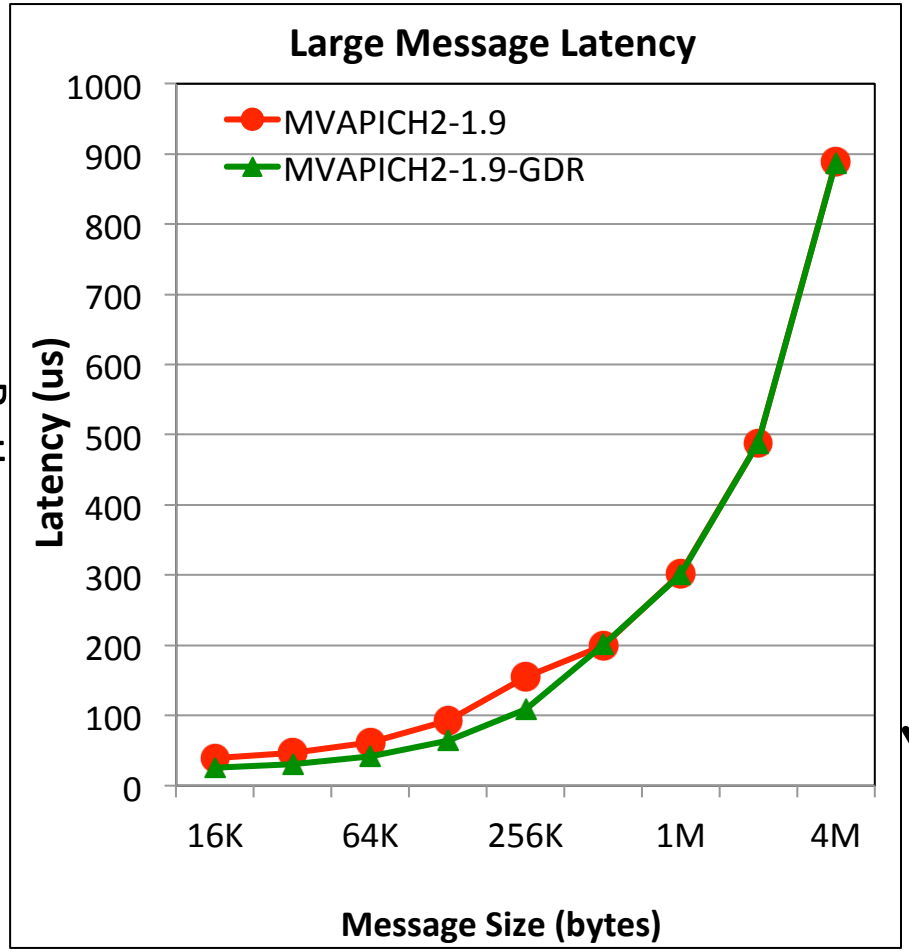
P2P write: 5.2 GB/s
P2P read: < 1.0 GB/s

GPUDirect RDMA(GDR) Designs in MVAPICH2: Performance

GPU-GPU Internode MPI Latency



Better

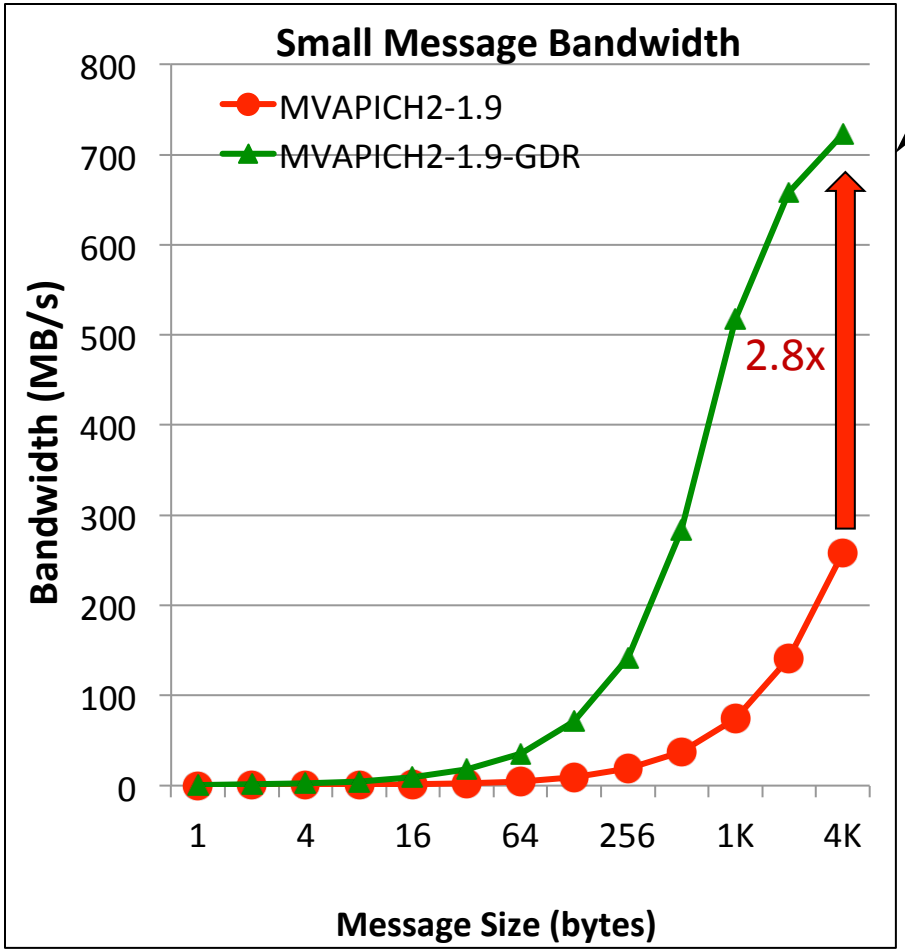


Better

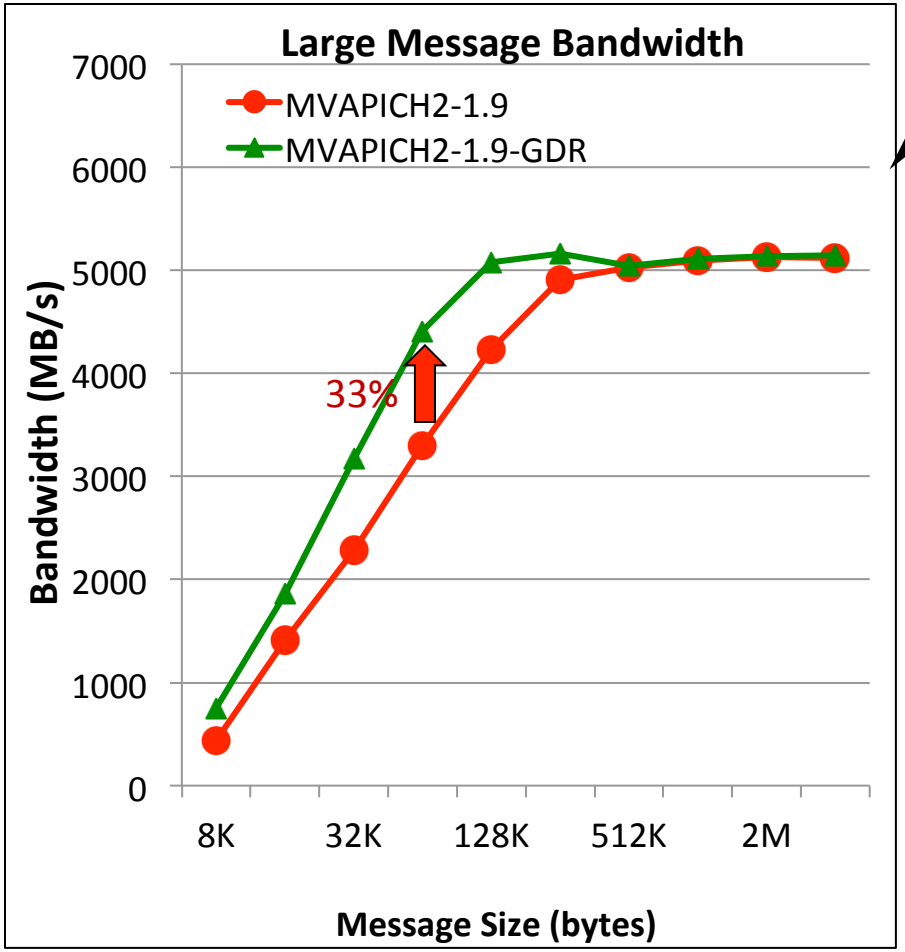
Based on MVAPICH2-1.9
Intel Sandy Bridge (E5-2670) node with 16 cores
NVIDIA Telsa K20c GPU, Mellanox ConnectX-3 FDR HCA
CUDA 5.5, OFED 1.5.4.1 with GPU-Direct-RDMA Patch

GPUDirect RDMA(GDR) Designs in MVAPICH2: Performance

GPU-GPU Internode MPI Uni-Directional Bandwidth



Better



Better

Based on MVAPICH2-1.9
Intel Sandy Bridge (E5-2670) node with 16 cores
NVIDIA Telsa K20c GPU, Mellanox ConnectX-3 FDR HCA
CUDA 5.5, OFED 1.5.4.1 with GPU-Direct-RDMA Patch

GPUDirect RDMA (GDR) Designs in MVAPICH2: Tuning

Parameter	Significance	Default	Notes
MV2_USE_GPUDIRECT	• Enable / Disable GDR-based designs	0 (Disabled)	• Always enable
MV2_GPUDIRECT_LIMIT	• Controls messages size until which GPUDirect RDMA is used	8 KByte	• Tune for your system • GPU type, host architecture and CUDA version: impact pipelining overheads and P2P bandwidth bottlenecks

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 1.9 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

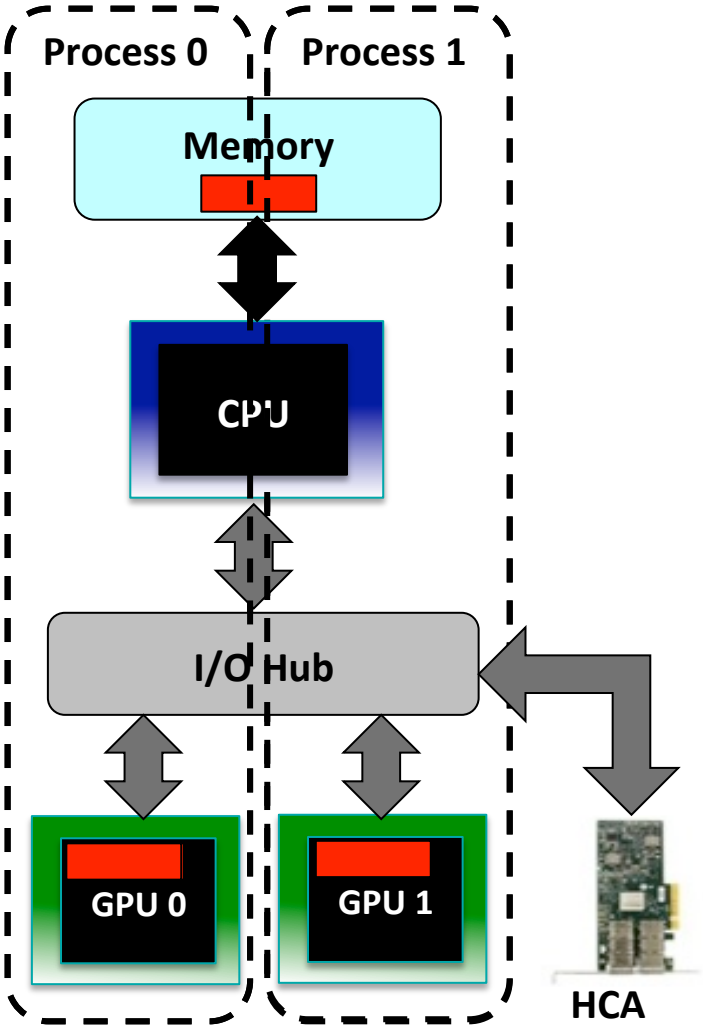
System Requirements for GPUDirect RDMA (GDR)

- OFED 1.5.4.1 is required to install GPUDirect RDMA extensions (support with MLNX OFED 2.0 to be added in near future)
- NVIDIA enables GDR only on their Tesla and Quadro lines of GPUs
- GDR works with Mellanox adapters in both IB and RoCE modes
- GPU and IB card should be connected to the same socket or I/O hub
  - CUDA GPUDirect RDMA driver has known limitations when they are on different sockets - not recommended
  - MVAPICH2 supports selection of HCA on multi-rail clusters through the MV2_PROCESS_TO_RAIL_MAPPING parameter
  - Example: MV2_PROCESS_TO_RAIL_MAPPING=0:1 binds process 0 to HCA 0 and process 1 to HCA 1

Outline

- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
    - Pipelined data movement
    - GPUDirect RDMA with Mellanox IB Adapters
    - CUDA IPC for multi-GPU clusters
    - MPI Collective Communication
    - MPI Datatype Processing
  - Usage: Build, Initialization and Cleanup
  - Support for MPI + OpenACC
  - CUDA and OpenACC extensions in OMB

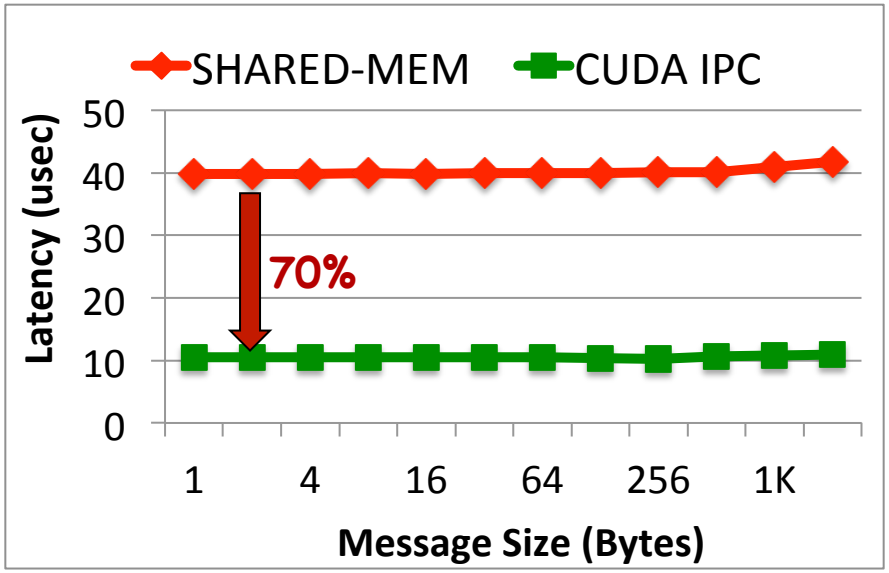
Multi-GPU Configurations



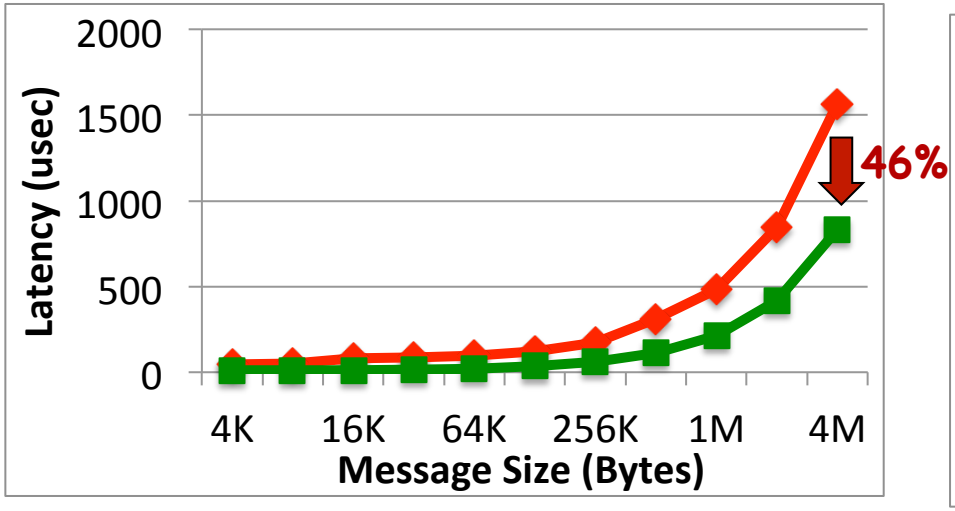
- Multi-GPU node architectures are becoming common
- Until CUDA 3.2
  - Communication between processes staged through the host
  - Shared Memory (pipelined)
  - Network Loopback [asynchronous)
- CUDA 4.0 and later
  - Inter-Process Communication (IPC)
  - Host bypass
  - Handled by a DMA Engine
  -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

Designs in MVAPICH2 and Performance

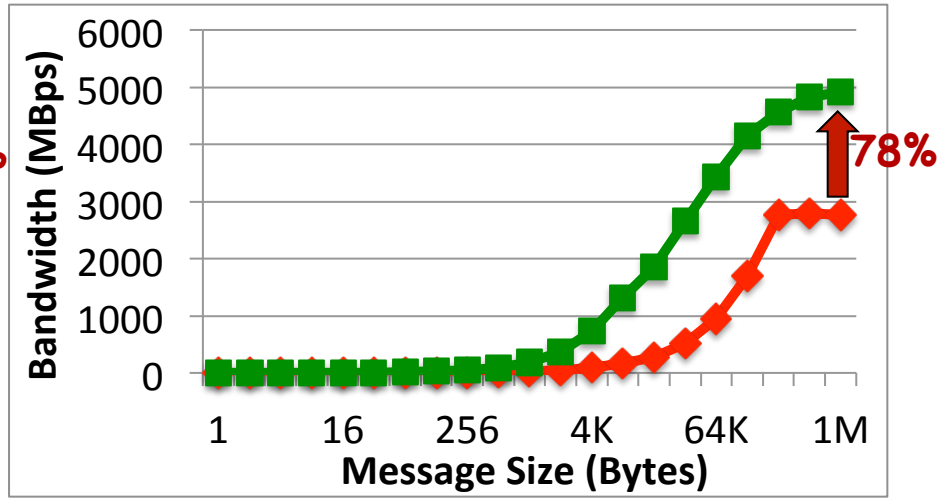
- MVAPICH2 takes advantage of CUDA IPC for MPI communication between GPUs
- Hides the complexity and overheads of handle creation, exchange and mapping
- Available in standard releases from MVAPICH2 1.8



Intranode osu_latency small



Intranode osu_latency large

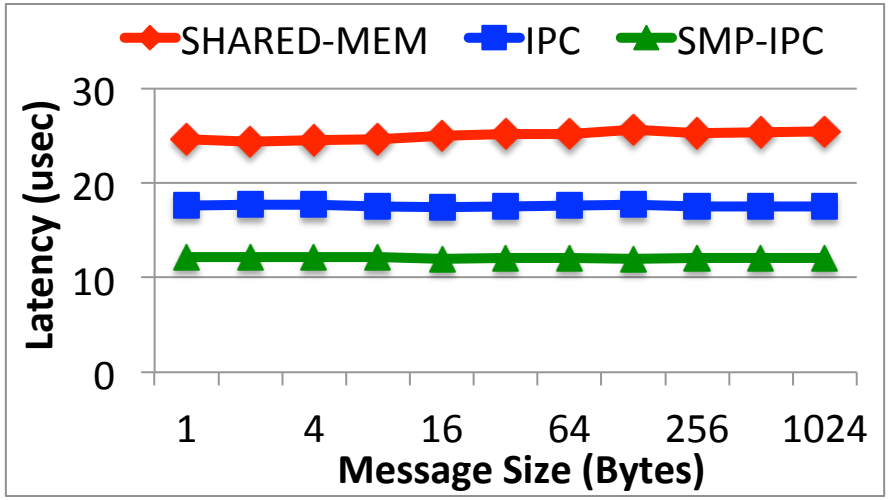


Intranode osu_bw

Runtime Parameters

- Works between GPUs within the same socket or IOH

Parameter	Significance	Default	Notes
MV2_CUDA_IPC	• Enable / Disable CUDA IPC-based designs	1 (Enabled)	• Always leave set to 1
MV2_CUDA_SMP_IPC	• Enable / Disable CUDA IPC fastpath design for short messages	0 (Disabled)	• Benefits Device-to-Device transfers • Hurts Device-to-Host/Host-to-Device transfers • Always set to 1 if application involves only Device-to-Device transfers

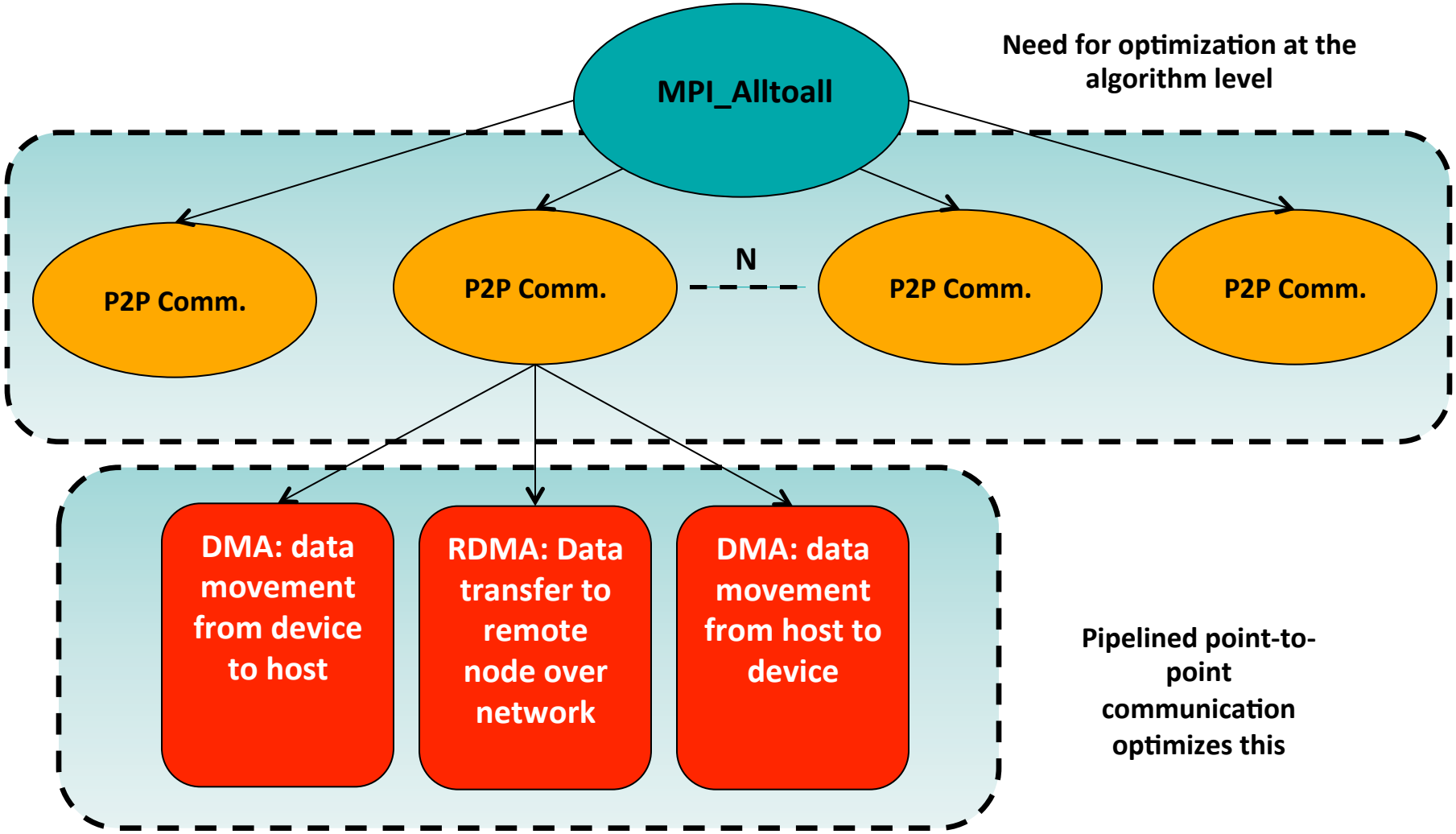


Intranode osu_latency small

Outline

- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
    - Pipelined data movement
    - GPUDirect RDMA with Mellanox IB Adapters
    - CUDA IPC for multi-GPU clusters
    - MPI Collective Communication
    - MPI Datatype Processing
  - Usage: Build, Initialization and Cleanup
  - Support for MPI + OpenACC
  - CUDA and OpenACC extensions in OMB

Optimizing Collective Communication



Optimizations in MVAPICH2: Overview

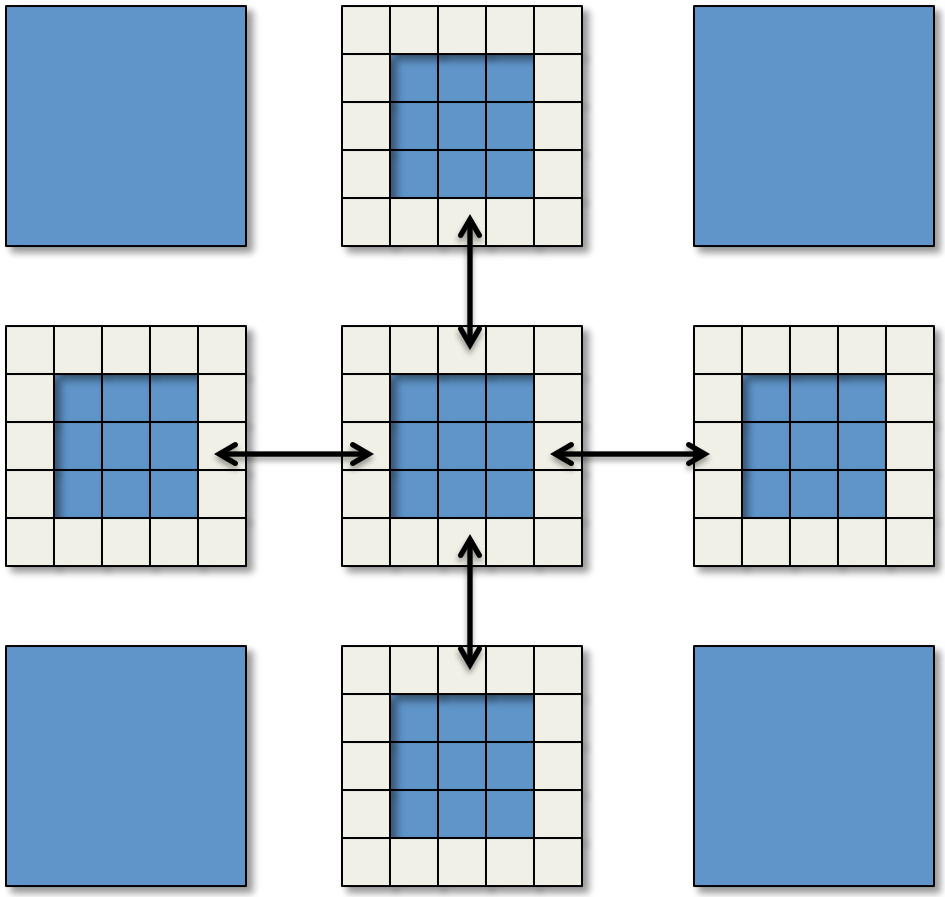
- Optimizes data movement at the collective level for small messages
- Pipelines data movement in each send/recv operation for large messages
- Several collectives have been optimized
  - Bcast, Gather, Scatter, Allgather, Alltoall, Scatterv, Gatherv, Allgatherv, Alltoallv
- Collective level optimizations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

Outline

- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
    - Pipelined data movement
    - GPUDirect RDMA with Mellanox IB Adapters
    - CUDA IPC for multi-GPU clusters
    - MPI Collective Communication
    - MPI Datatype Processing
  - Usage: Build, Initialization and Cleanup
  - Support for MPI + OpenACC
  - CUDA and OpenACC extensions in OMB

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

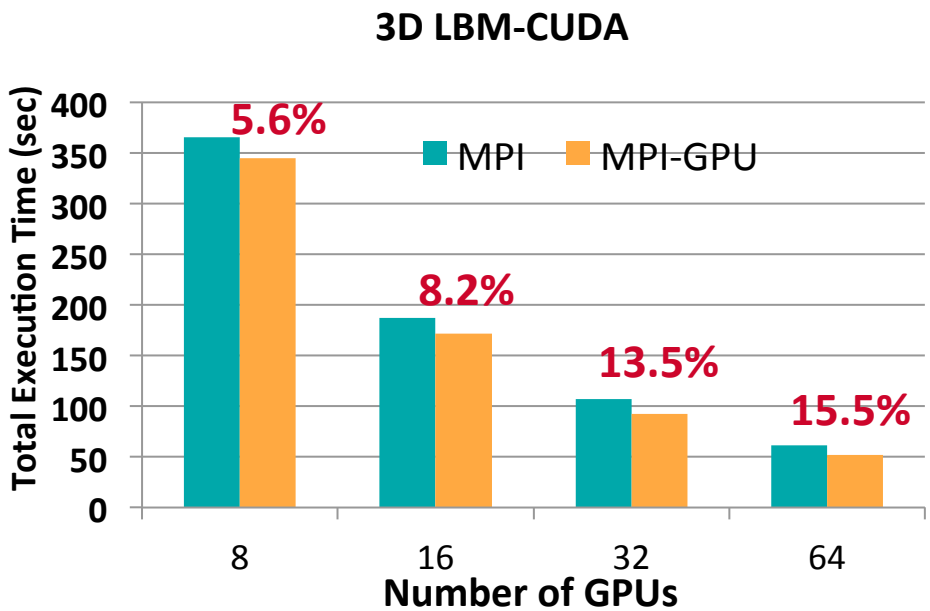
At Sender:

```
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
  - Efficiently move data between nodes using RDMA
  - In progress - currently optimizes *vector* and *hindexed* datatypes
  - Transparent to the user

H. Wang, S. Potluri, D. Bureddy, C. Rosales and D. K. Panda, GPU-aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation, IEEE Transactions on Parallel and Distributed Systems, Accepted for Publication.

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

- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
    - Pipelined data movement
    - GPUDirect RDMA with Mellanox IB Adapters
    - CUDA IPC for multi-GPU clusters
    - MPI Collective Communication
    - MPI Datatype Processing
  - Usage: Build, Initialization and Cleanup
  - Support for MPI + OpenACC
  - CUDA and OpenACC extensions in OMB

Enabling Support for MPI Communication from GPU Memory

- Configuring the support
  - --enable-cuda –with-cuda=<path-to-cuda-installation>
  - With PGI compilers
    - PGI does not handle an asm string leading to linker issues with CUDA kernels in MVAPICH2 library
    - --enable-cuda=basic –with-cuda=<path-to-cuda-installation>

- Enabling the support at runtime
  - Disabled by default to avoid pointer checking overheads for Host communication
  - Set MV2_USE_CUDA=1 to enable support of communication from GPU memory

GPU Device Selection, Initialization and Cleanup

- MVAPICH2 1.9 and earlier versions require GPU device to be selected before MPI_Init
  - To allocate and initialize required GPU resources in MPI Init
  - Node rank information is exposed by the launchers (mpirun_rsh or hydra) as MV2_COMM_WORLD_LOCAL_RANK

```
int local_rank = atoi(getenv("MV2_COMM_WORLD_LOCAL_RANK"));
cudaSetDevice (local_rank% num_devices)
```

- MVAPICH2 2.0 removes this restriction
  - Does GPU resource initialization dynamically
  - Applications can select the device before or after MPI_Init

- CUDA allocates resources like shared memory files which require explicit cleanup
  - cudaDeviceReset or cuCtxDestroy
  - Applications have to do this after MPI_Finalize to allow MVAPICH2 runtime to deallocate resources

Outline

- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
    - Pipelined data movement
    - GPUDirect RDMA with Mellanox IB Adapters
    - CUDA IPC for multi-GPU clusters
    - MPI Collective Communication
    - MPI Datatype Processing
  - Usage: Build, Initialization and Cleanup
  - Support for MPI + OpenACC
  - CUDA and OpenACC extensions in OMB

OpenACC

- OpenACC is gaining popularity
- Several sessions during GTC
- A set of compiler directives (#pragma)
- Offload specific loops or parallelizable sections in code onto accelerators

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

- Routines to allocate/free memory on accelerators

```
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

```
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

```
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

- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
    - Pipelined data movement
    - GPUDirect RDMA with Mellanox IB Adapters
    - CUDA IPC for multi-GPU clusters
    - MPI Collective Communication
    - MPI Datatype Processing
  - Usage: Build, Initialization and Cleanup
  - 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: Alltoall, Gather and Scatter
- 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
- Available in an integrated manner with MVAPICH2 stack

Configuring, Building and Running OMB

- Configuring with GPU support
  - Enabling CUDA support: *"--enable-cuda –with-cuda-include=<path-to-CUDA-headers> --with-cuda-libraries=<path-to-CUDA-libraries>"*
  - Enabling OpenACC support: *"--enable-openacc"*
  - Both enabled in integrated version when library is build with CUDA support
- Running the benchmarks
  - *"pt2pt-benchamark [options] [RANK0 RANK1]"*
  - *"collective-benchmark [options]"*
- Option to select between CUDA and OpenACC: *"-d [cuda|openacc]"*
- Parameters to select location of buffers
  - Pt2pt benchmarks support selection at each rank: *"D D, D H, H D, H H"*
  - The –d option enables use of device buffers in collective benchmarks

Device Selection

- MVAPICH2 1.8, 1.9 and other MPI libraries require device selection before MPI_Init
- This restriction has been removed with MVAPICH2 2.0a
- OSU micro-benchmarks still selects device before MPI_Init for backward compatibility
- Uses node-level rank information exposed by launchers
- We provide a script which exports this node rank information to be used by the benchmark
  - Can be modified to work with different MPI libraries without modifying the benchmarks themselves
  - Sample script provided with OMB : "get_local_rank"

```
export LOCAL_RANK=$MV2_COMM_WORLD_LOCAL_RANK
exec $*
```

Examples

Consider two GPU nodes: *n1 and n2 each with two GPUs*

Measure internode GPU-to-GPU latency using CUDA

mpirun_rsh –np 2 n1 n2 MV2_USE_CUDA=1 ./get_local_rank ./osu_latency D D

mpirun_rsh –np 2 n1 n2 MV2_USE_CUDA=1 ./get_local_rank ./osu_latency –d cuda D D

Measure internode GPU-to-GPU latency using OpenACC

mpirun_rsh –np 2 n1 n2 MV2_USE_CUDA=1 ./get_local_rank ./osu_latency – d openacc D D

Measure internode GPU-to-Host latency using CUDA

mpirun_rsh –np 2 n1 n2 MV2_USE_CUDA=1 ./get_local_rank ./osu_latency D H

Examples

Measure intranode GPU-to-GPU latency

mpirun_rsh –np 2 n1 n1 MV2_USE_CUDA=1 ./get_local_rank ./osu_latency D D

Measure MPI_Alltoall latency between GPUs with CUDA

mpirun_rsh –np 4 n1 n1 n2 n2 MV2_USE_CUDA=1 ./get_local_rank ./osu_alltoall –d CUDA

Outline

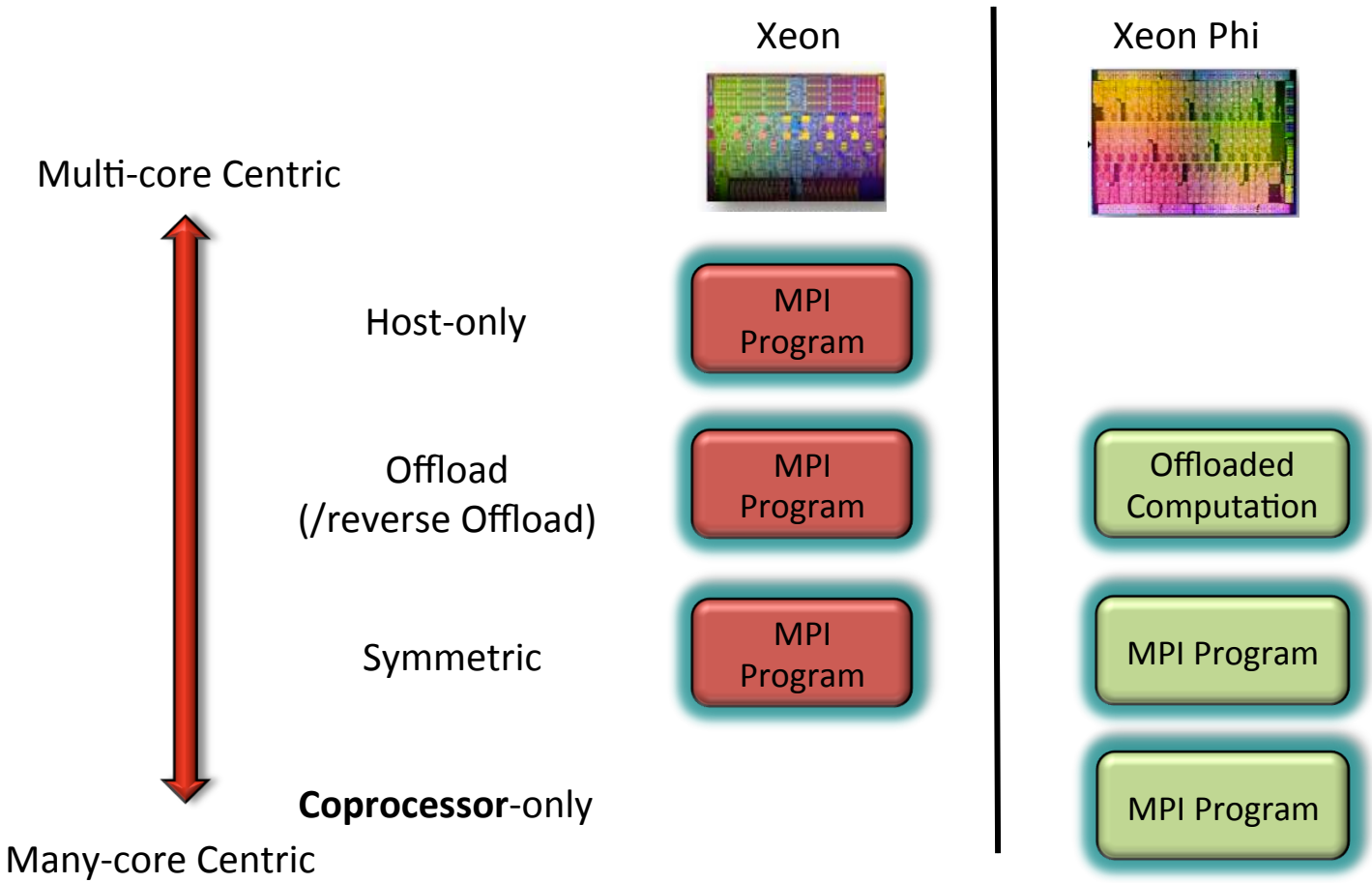
- MVAPICH2 for NVIDIA GPU Clusters
  - MVAPICH2-GPU Project
  - Designs: Overview, Performance and Tuning
  - Usage: Build, Initialization and Cleanup
  - Support for MPI + OpenACC
  - CUDA and OpenACC extensions in OMB
- MVAPICH2 for Intel Xeon Phi Clusters
- Conclusion

Intel Many Integrated Core Architecture

- **High compute power with high performance per watt**
  - **5110P – >1TFlop – 225 Watts**
- **X**86 compatibility – **reuse** the strong Xeon software eco-system
  - **Programming models, l**ibraries, tools and applications
- Xeon Phi, first product line based on MIC
  - **A**lready powering several Top500 systems
  - Tianhe-2@NUDT/China (1), Stampede@TACC (6), Conte@Purdue(28), Discover@NASA(65)

MPI Applications on MIC Clusters

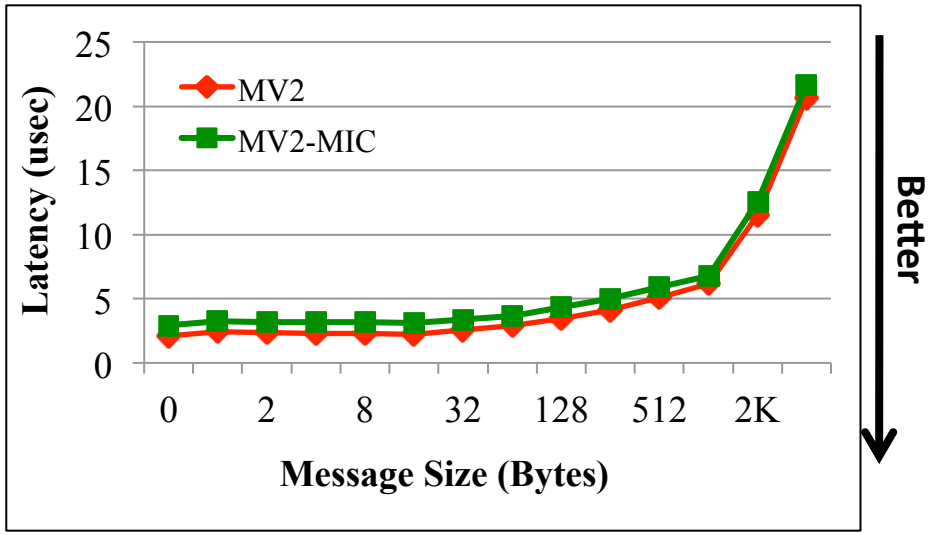
- Flexibility in launching MPI jobs on clusters with Xeon Phi



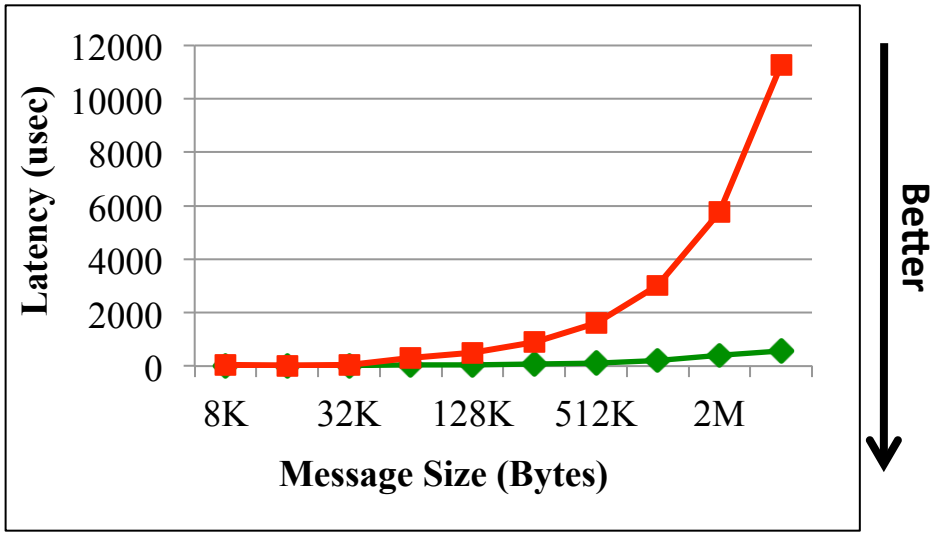
MVAPICH2-MIC: Optimized MPI communication on Xeon Phi Clusters

- MPI libraries run out of box (or with minor changes) on the Xeon Phi
- Critical to optimize the runtimes for better performance
  - Tune existing designs
  - Designs using lower level features offered by MPSS
  - Designs to address system-level limitations
  - Hide the complexity from the user
- Initial version of MVAPICH2-MIC is available – improved MVAPICH2 library for Xeon Phi clusters with InfiniBand.
  - Supports all modes of usage – host-only, offload, coprocessor-only and symmetric
  - Improved shared memory communication channel
  - SCIF-based designs for improved communication within MIC and between MICs and Hosts
  - Proxy-based design to work around bandwidth limitations on Sandy Bridge platform
- Available on Stampede.
- Working with other sites for deployment.

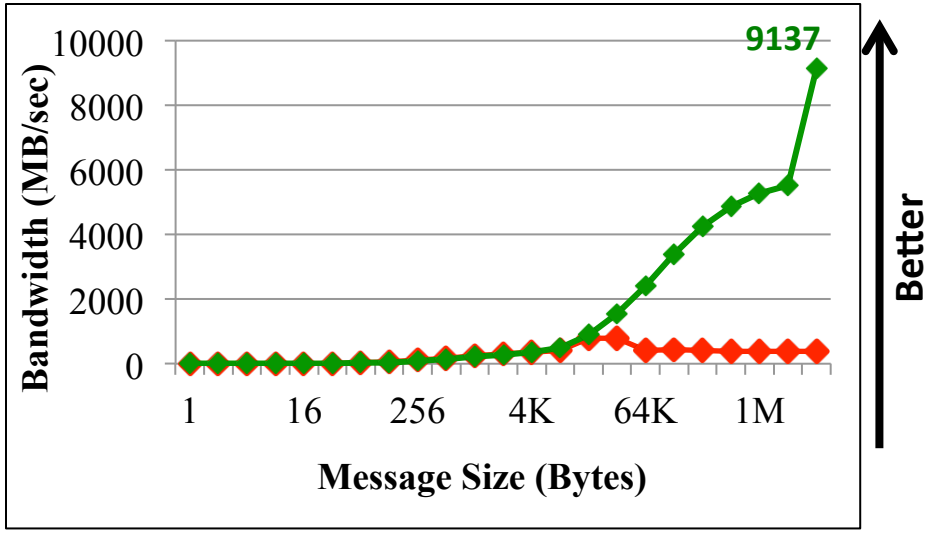
Intra-MIC - Point-to-Point Communication



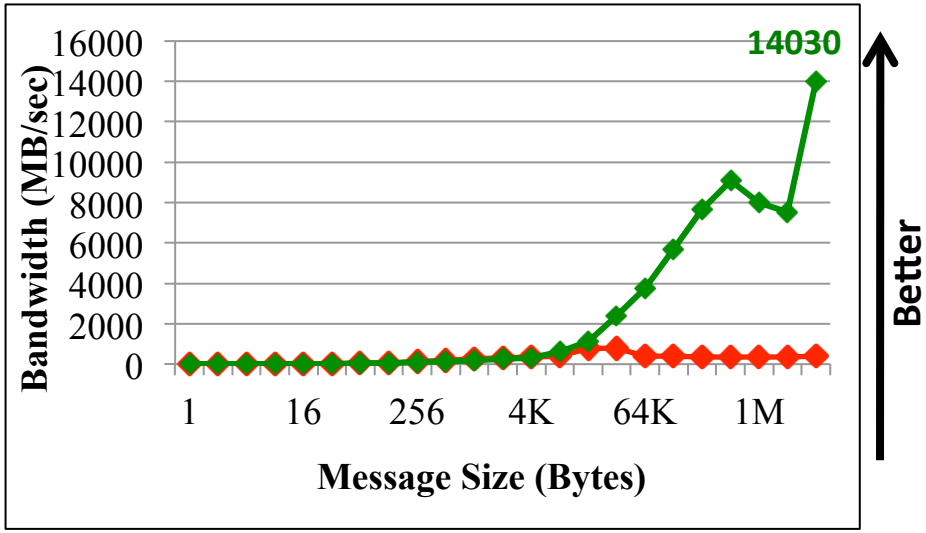
osu_latency (small)



osu_latency (large)

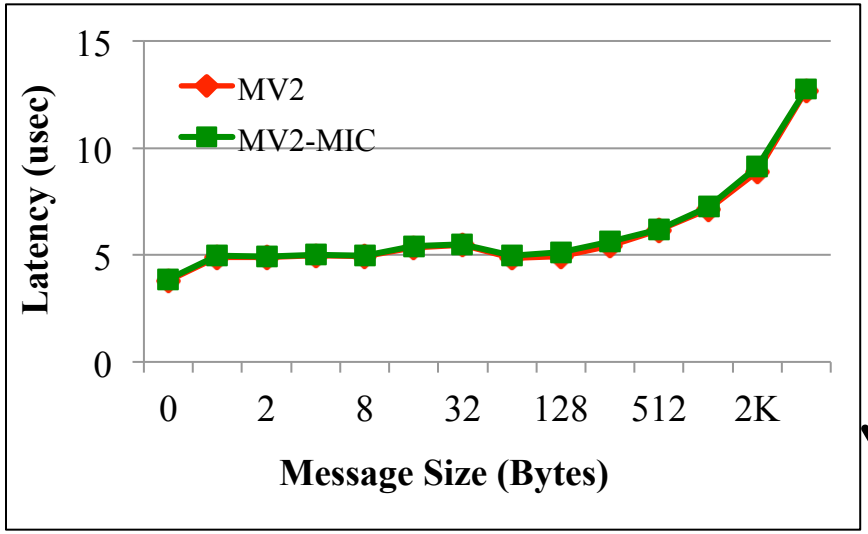


osu_bw



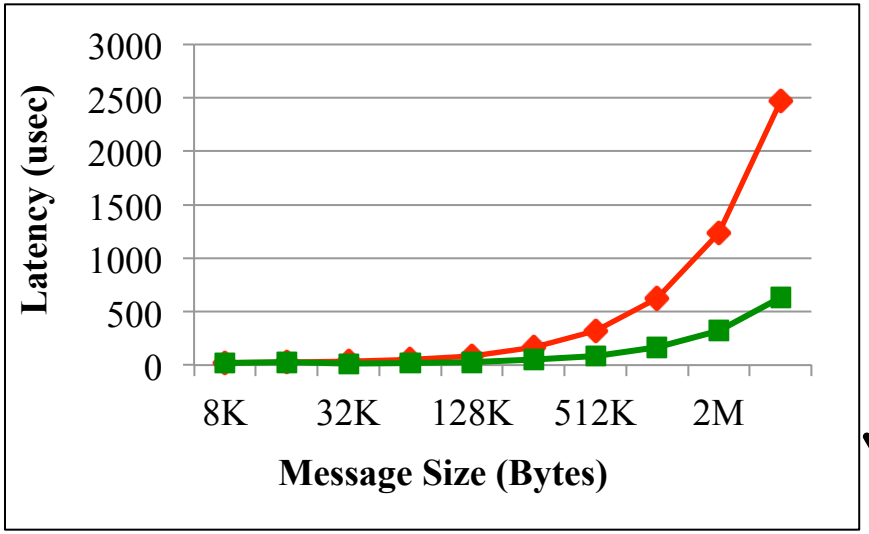
osu_bibw

IntraNode Host-MIC - Point-to-Point Communication



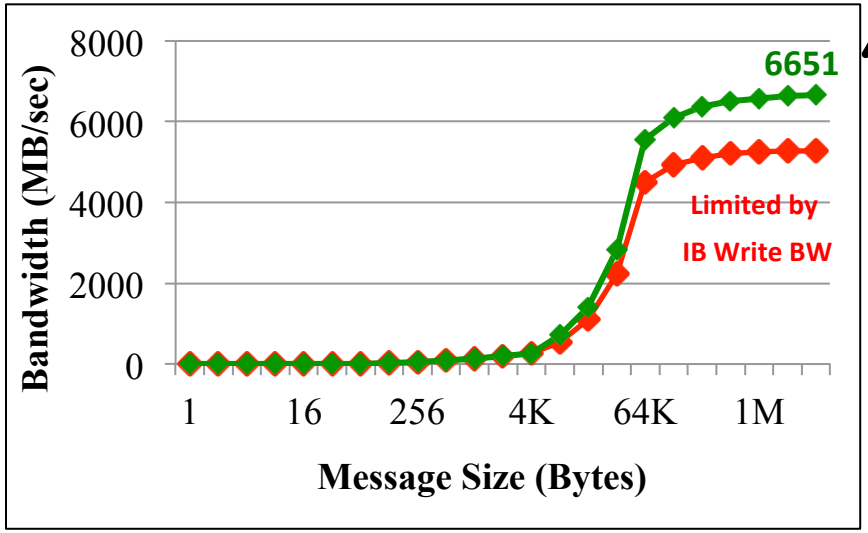
Better

osu_latency (small)



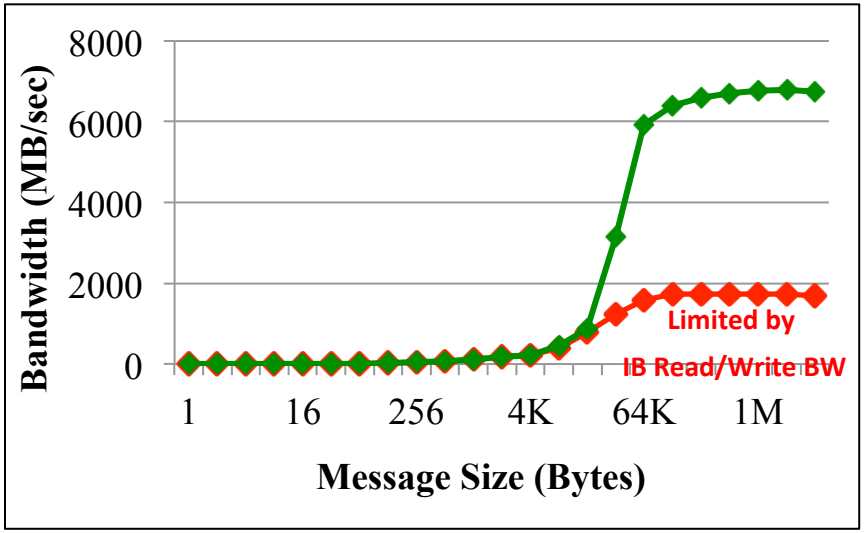
Better

osu_latency (large)



Better

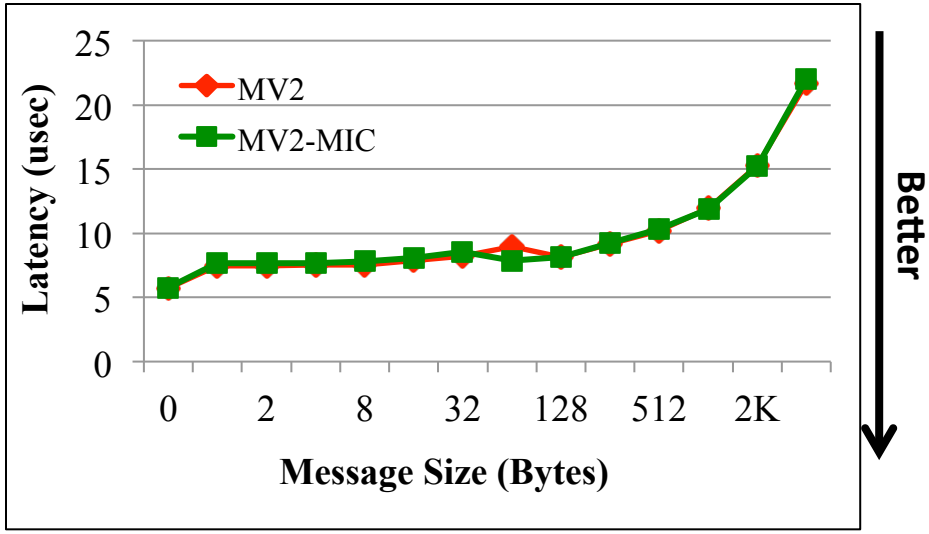
osu_bw



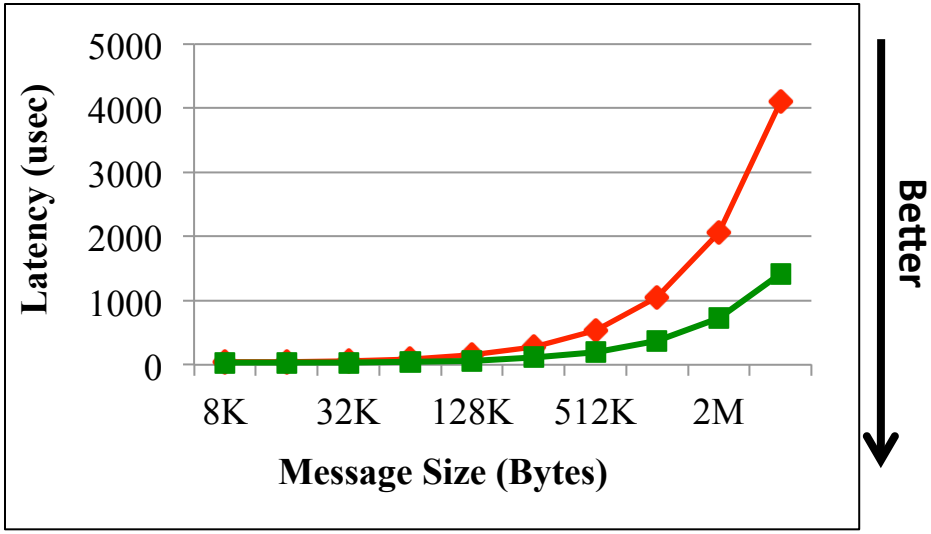
Better

osu_bibw

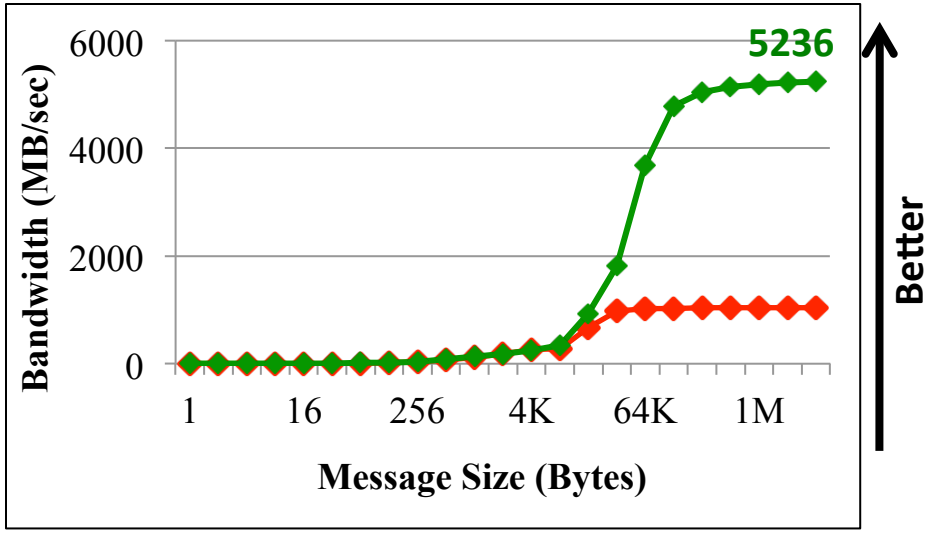
InterNode MIC-to-MIC Point-to-Point Communication



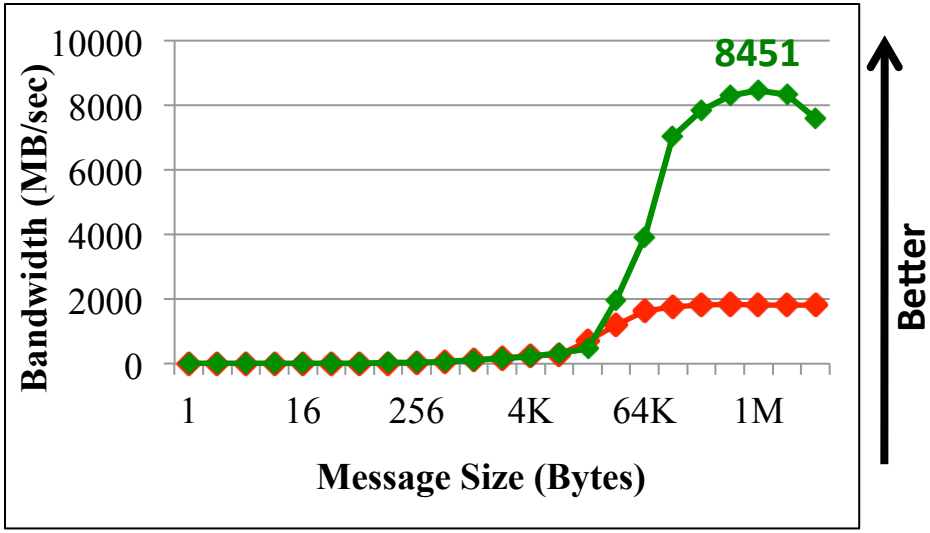
osu_latency (small)



osu_latency (large)



osu_bw



osu_bibw

Summary

- MVAPICH2 evolving to efficiently support MPI communication on heterogeneous clusters with GPU and Xeon Phi
- 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