NETWORK-BASED COMPUTING LABORATORY

MPI Alltoall Personalized Exchange on GPGPU Clusters: Design Alternatives and Benefits

Ashish Kumar Singh, Sreeram Potluri, **Hao Wang**,
Krishna Kandalla, Sayantan Sur, and Dhabaleswar K. Panda

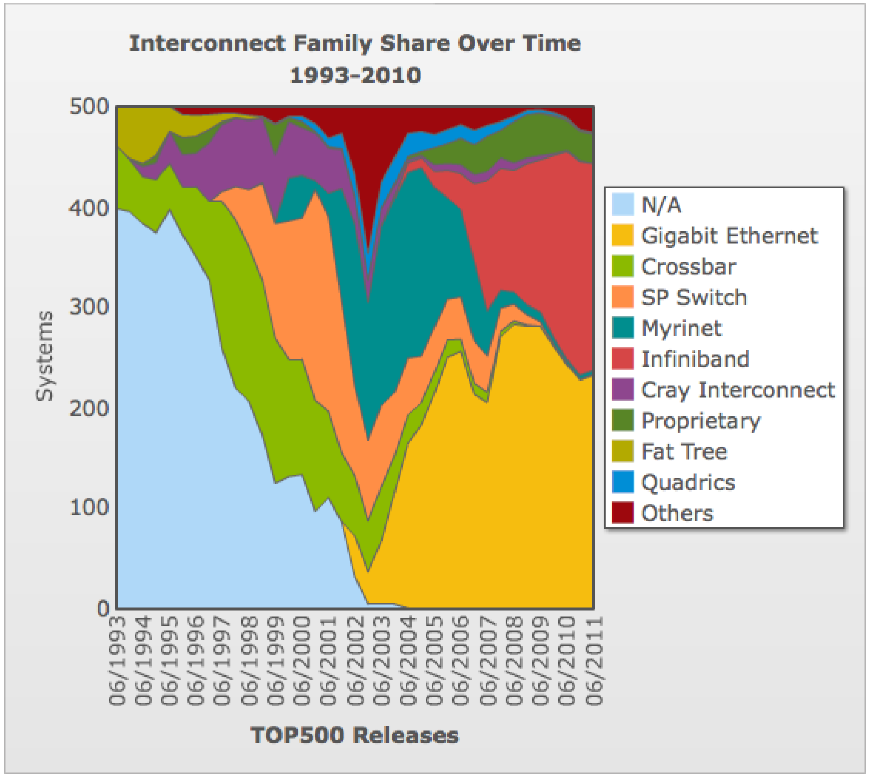
Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University, USA

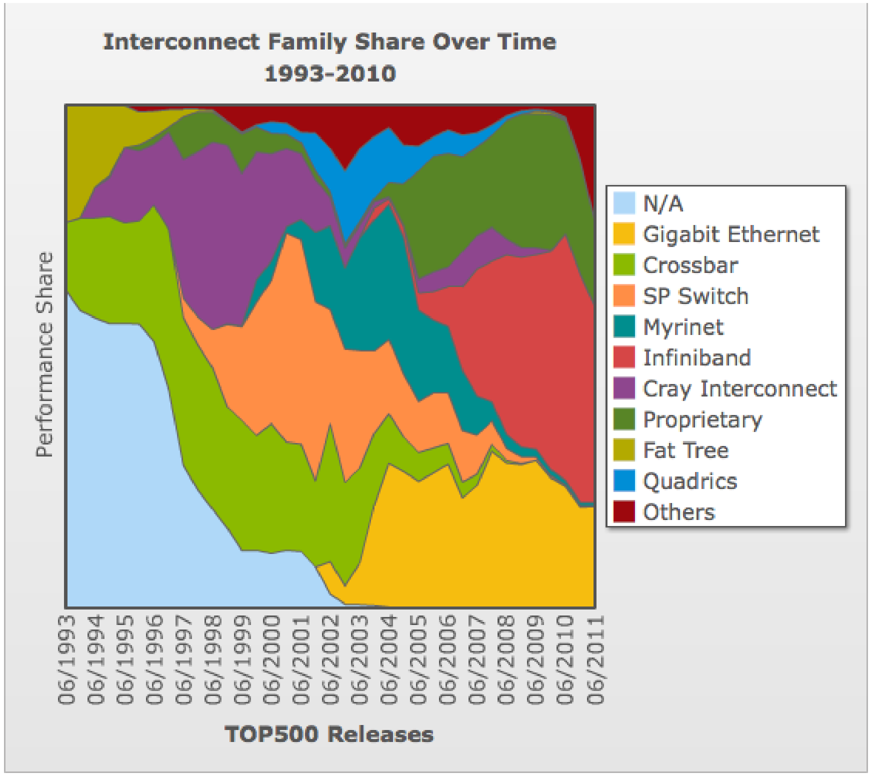
OHIO STATE

Outline

- Introduction
- Problem Statement
- Design Considerations
- Our Solution
- Performance Evaluation
- Conclusion and Future Work

InfiniBand Clusters in TOP500



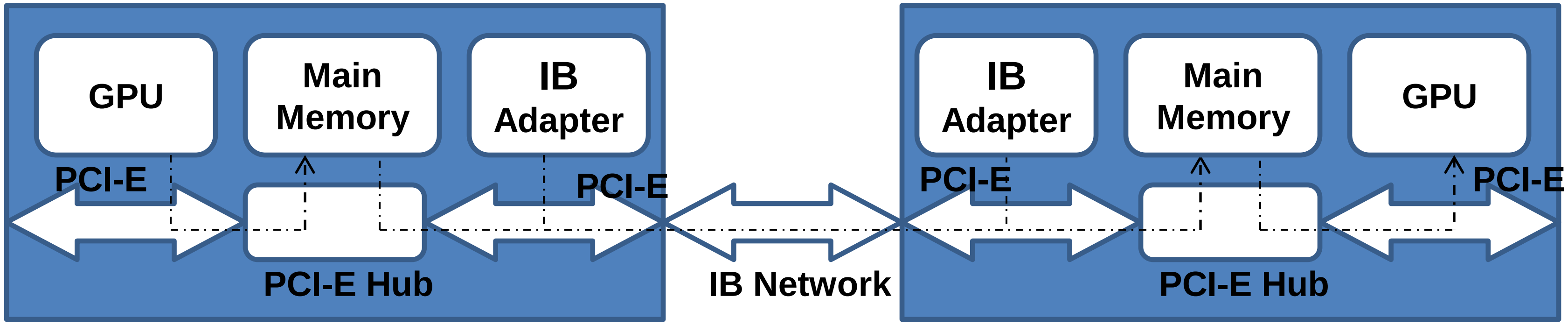


- Percentage share of InfiniBand is steadily increasing
- 41% of systems in TOP500 using InfiniBand (June ’11)
- 61% of systems in TOP100 using InfiniBand (June ‘11)

GPGPUs and Infiniband

- GPGPUs are becoming an integral part of high performance system architectures
- 3 of the 5 fastest supercomputers in the world use GPGPUs with Infiniband
  - TOP500 list features Tianhe-1A at #2, Nebulae at # 4 and Tsubame at # 5.
- Programming:
  - CUDA or OpenCL on GPGPUs
  - MPI on the whole system
- Manage memory issue
  - Prof. Van de Geijn just mentioned memory management is an issue, and the data granularity is important

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 78,000 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

MVAPICH2-GPU: GPU-GPU using MPI

- Is it possible to optimize GPU-GPU communication with MPI?
  - *H. Wang, S. Potluri, M. Luo, A. K. Singh, S. Sur, D. K. Panda, "MVAPICH2-GPU: Optimized GPU to GPU Communication for InfiniBand Clusters", ISC'11, June, 2011*
  - Support GPU to remote GPU communication using MPI
  - P2P and One-sided were improved
  - Collectives can directly get benefits from p2p improvement
- How to handle non-contiguous data in GPU device memory?
  - *H. Wang, S. Potluri, M. Luo, A. K. Singh, X. Ouyang, S. Sur, D. K. Panda, "Optimized Non-contiguous MPI Datatype Communication for GPU Clusters: Design, Implementation and Evaluation with MVAPICH2", Cluster'11, Sep., 2011 (Thursday, TP6-A,1:30 PM)*
  - Support GPU-GPU non-contiguous data communication (P2P) using MPI
  - Vector datatype and SHOC benchmark are optimized
- How to optimize collectives with different algorithms?
  - In this paper, MPI_Alltoall on GPGPUs cluster is optimized

MPI_Alltoall

- Many scientific applications spend much execution time in MPI_Alltoall:
  - P3DFFT, CPMD
- Heavy communication in MPI_Alltoall
  - $O(N^2)$ communication for N processes
- Different MPI_Alltoall algorithms:
  - Related with message size, process number, etc.
- What will happen if the data is in GPU device memory?

Outline

- Introduction
- Problem Statement
- Design Considerations
- Our Solution
- Performance Evaluation
- Conclusion and Future Work

Problem Statement

- High start-up overheads in accessing small and medium data inside GPU device memory:
  - Start-up time: the time to move the data from GPU device memory to host main memory, and vice versa
- Hard to optimize GPU-GPU Alltoall communication at the application level:
  - CUDA and MPI expertise is required for efficient data movement
  - Existing Alltoall optimizations are implemented in MPI library
  - Optimizations are dependent on hardware characteristics, like latency

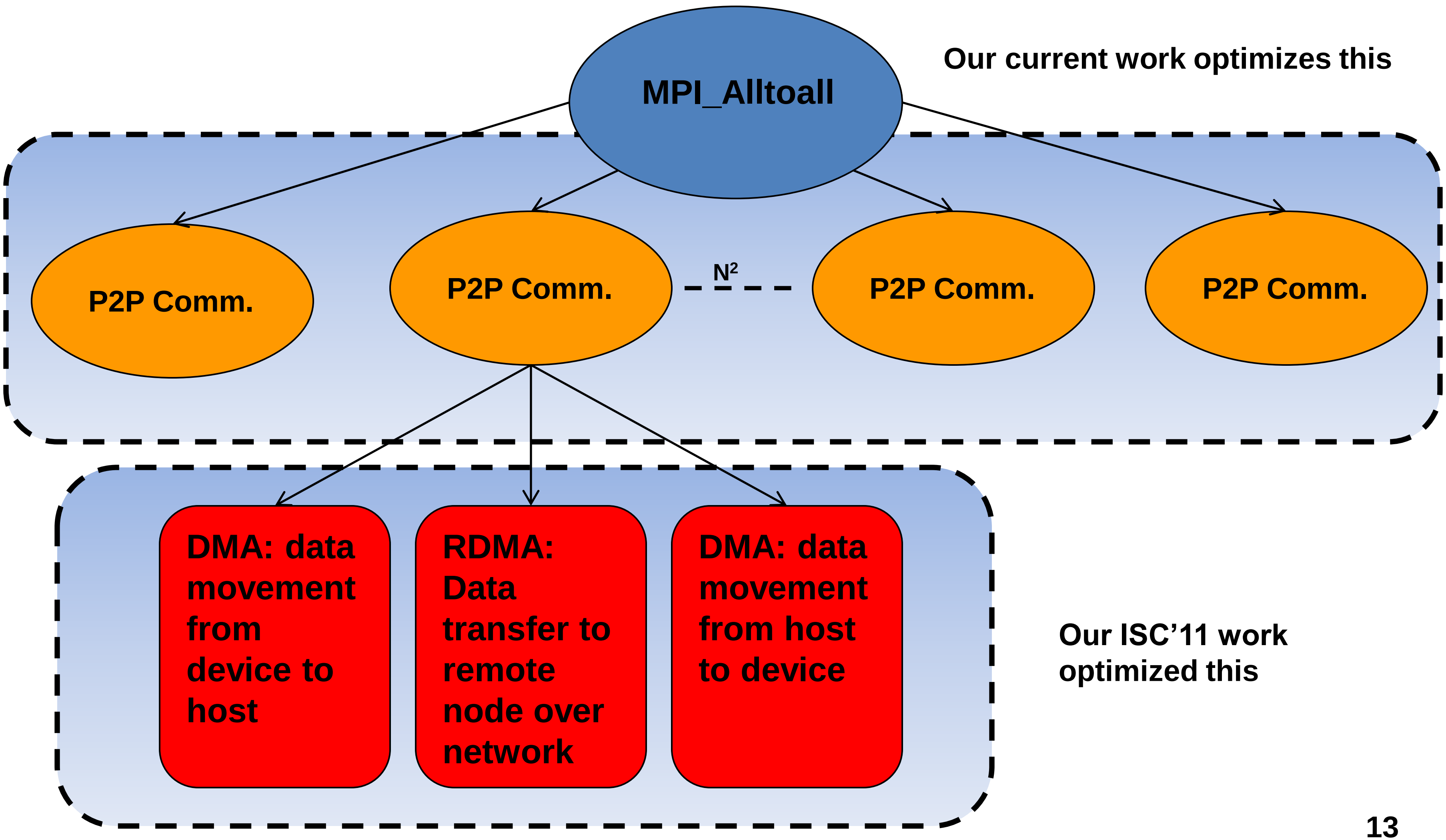
Outline

- Introduction
- Problem Statement
- Design Considerations
- Our Solution
- Performance Evaluation
- Conclusion and Future Work

Alltoall Algorithms

- Hypercube algorithm (Bruck's) proposed by Bruck et. al, for small messages
  - requires (logN) steps, for N processes
  - additional data movement in the local memroy
- Scattered destination (SD) algorithm for medium messages
  - a linear implementation of Alltoall personalized exchange operation
  - uses non-blocking send/recv to overlap data transfer on network
- Pair-wise exchange (PE) algorithm for large messages
  - network contention (SD) becomes the bottleneck, switch to PE
  - uses blocking send/recv; in any step, a process communicates with only one source and one destination

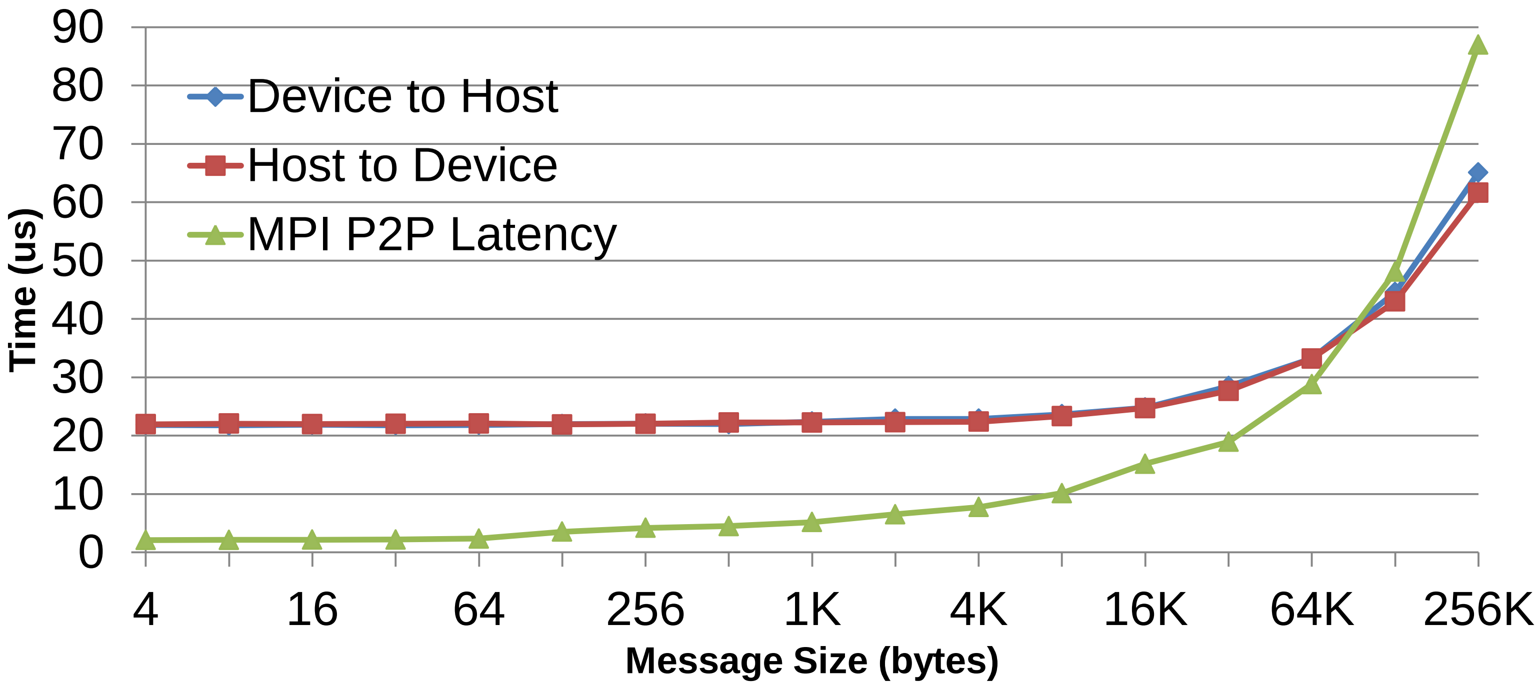
Design Considerations



Design Considerations

- Message size
  - not enough to consider data movement in local memory (Bruck's)
  - Start-up overhead must be considered
- Network transfer
  - not enough to overlap different p2p transfer on networks (SD)
  - data movement between device and host (DMA) can be overlapped with data transfer (RDMA) in each peer on networks
- Network contention
  - blocking send/recv (in PE) will harm the overlapping (DMA and RDMA)
  - possible to overlap DMA and RDMA on multiple channels until the network contention dominates the performance again

Start-up Overhead



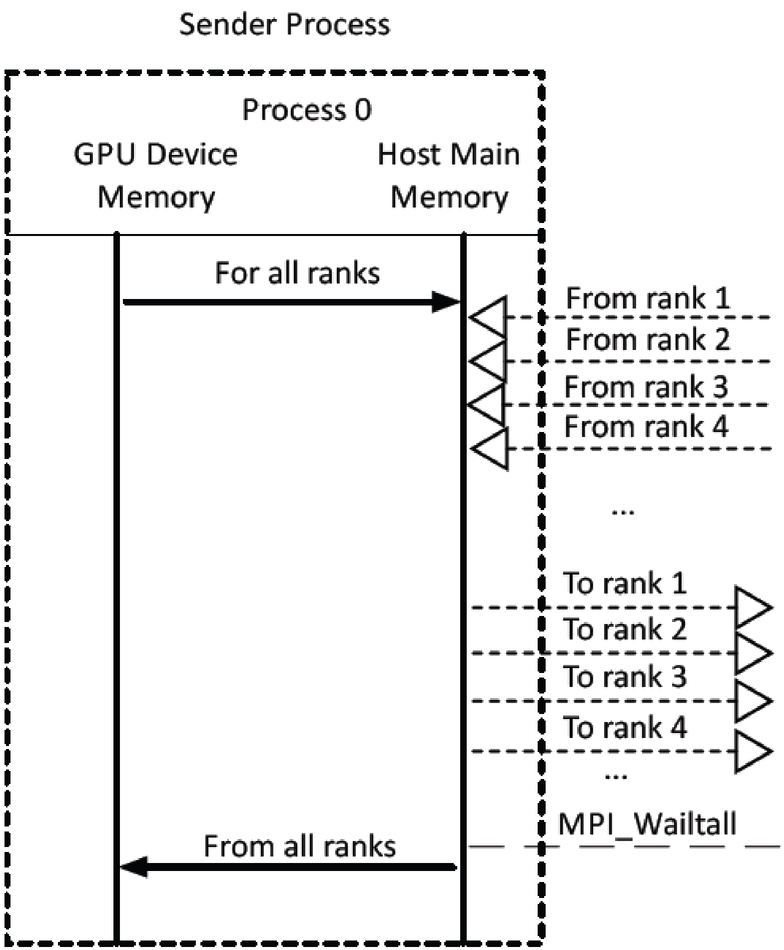
- Data movement cost (GPU and host) remains constant until a threshold
  - 16 KB is the threshold in our cluster
  - compared with MPI p2p latency, start-up cost dominates GPU-GPU performance at small and medium datasize

Outline

- Introduction
- Problem Statement
- Design Considerations
- Our Solution
- Performance Evaluation
- Conclusion and Future Work

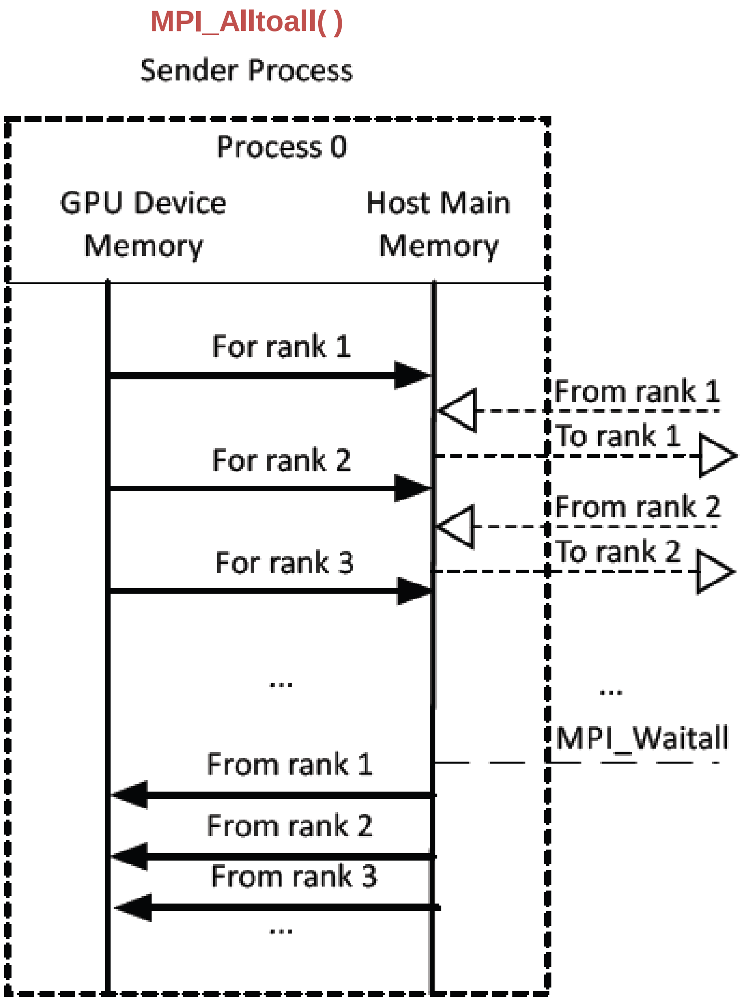
No MPI Level Optimization

cudaMemcpy( ) + MPI_Alltoall( ) + cudaMemcpy( )



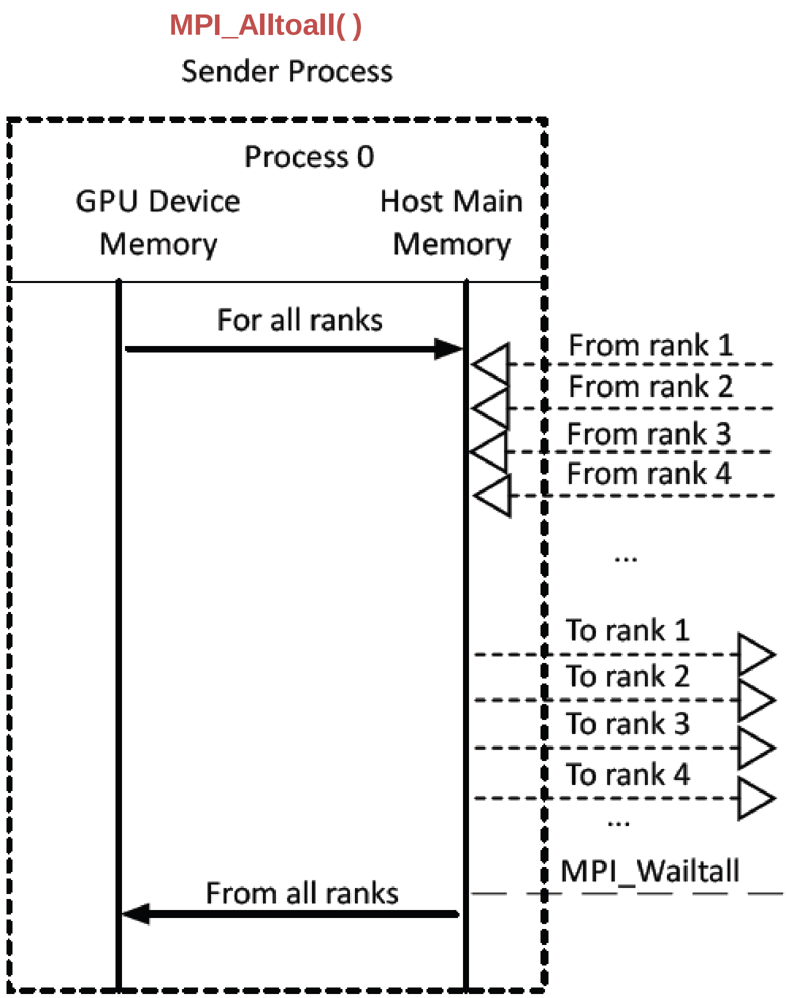
- No MPI level optimization:
  - can be implemented at user level
  - doesn't requires any changes in MPI library
- Reduce programming productivity:
  - adds extra burden on programmer to manage data movement and corresponding buffers
  - hard to overlap DMA and RDMA to hide memory transfer latency since MPI_Alltoall() is blocking

Point-to-Point Based



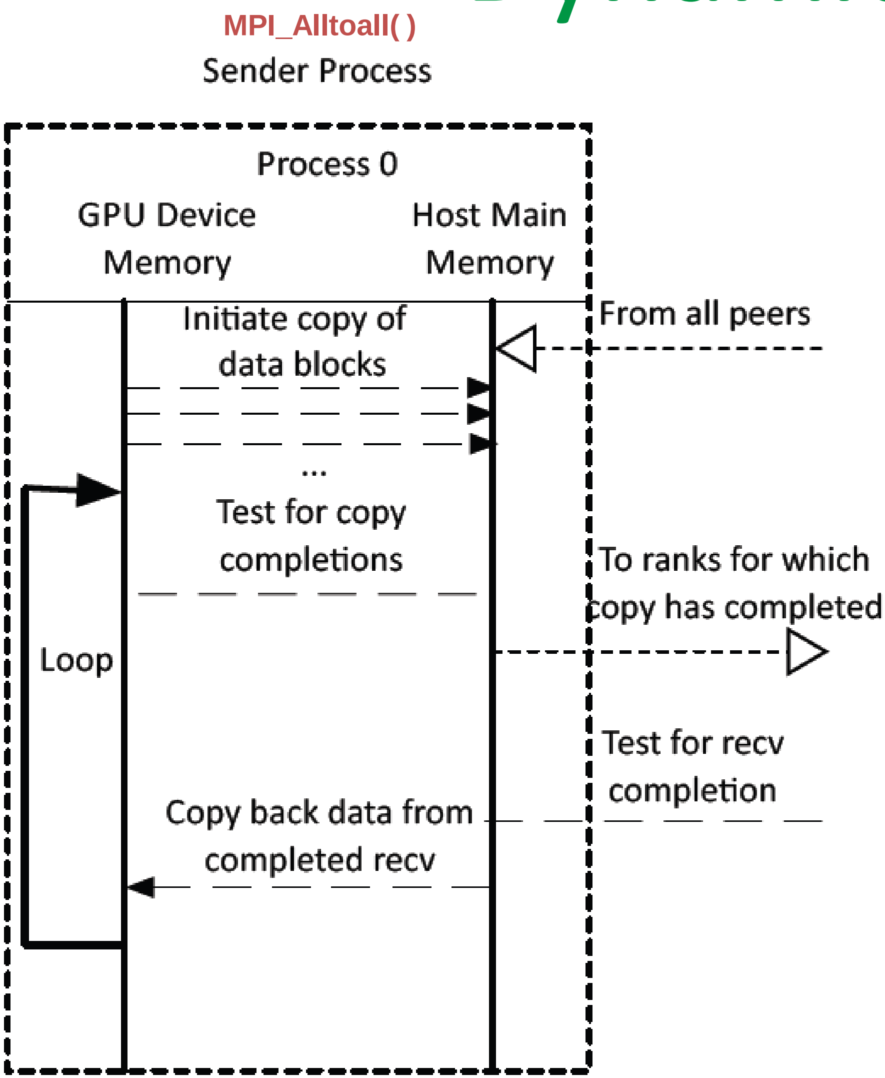
- Basic way to enable collectives for GPU memory
  - for each p2p channel, moves the data between device and host, and uses send/recv interfaces
  - handle GPU-to-GPU transfer with Send/Recv interfaces
- High start-up overhead to move data between device and host (for small and medium data)

Static Staging



- Reduce the number of DMA operations:
  - merge all ranks' data to one package, and move between device and host
- Compared with no MPI level method, only MPI_Alltoall needed
  - similar performance
  - better programming productivity
- Problem:
  - aggressively merge all ranks' data into one large package maybe increase the latency

Dynamic Staging



- Group data
  - group data based on a threshold
  - use non-blocking function to move data between device and host
- Pipeline
  - overlap DMA data movement between host and device and RDMA transfer on network
- Hard to implement at user level
  - MPI_Alltoall is a blocking function
  - hardware latency dependent

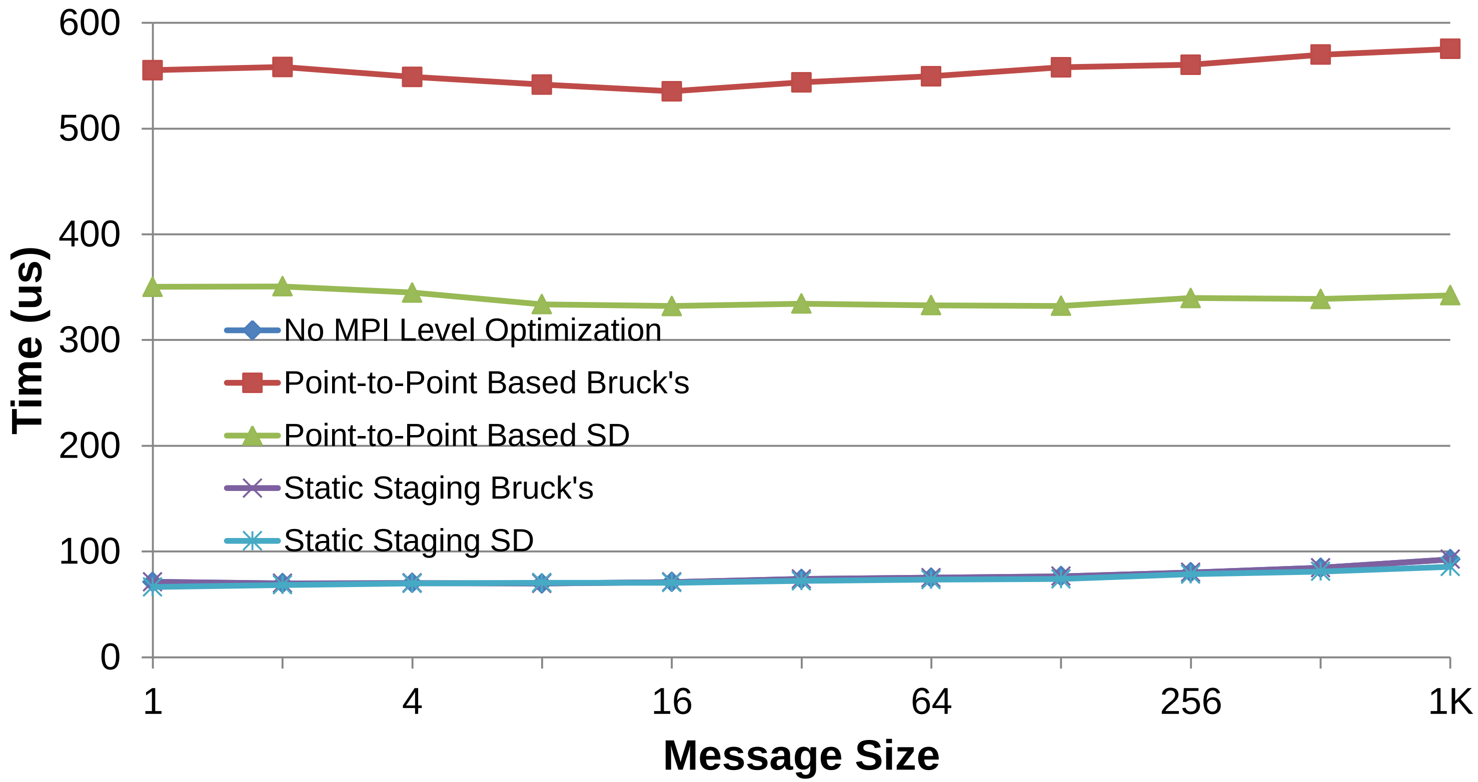
Outline

- Introduction
- Problem Statement
- Design Considerations
- Our Solution
- Performance Evaluation
- Conclusion and Future Work

Performance Evaluation

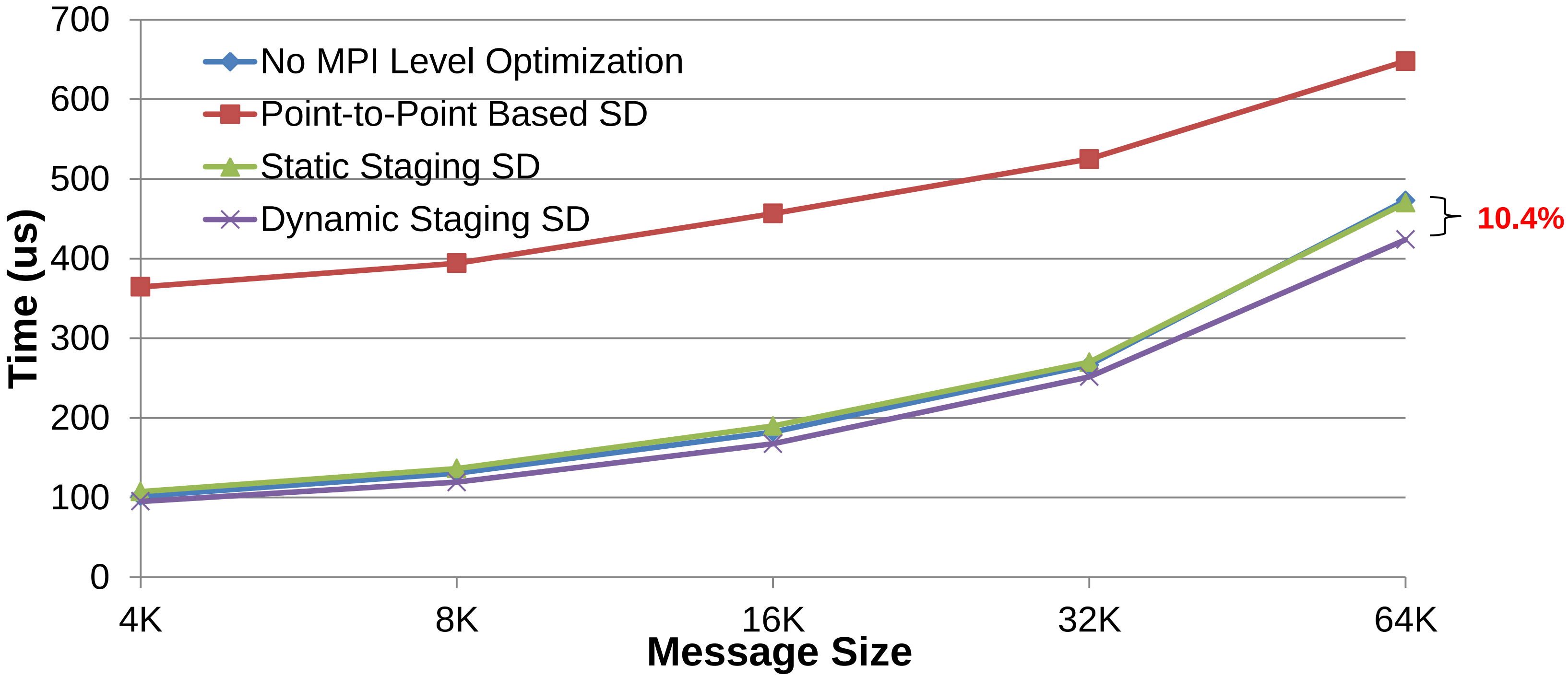
- Experimental environment
  - NVIDIA Tesla C2050
  - Mellanox QDR InfiniBand HCA MT26428
  - Intel Westmere processor with 12 GB main memory
  - MVAPICH2 1.6, CUDA Toolkit 4.0
- OSU Micro-Benchmarks
  - The source and destination addresses are in GPU device memory
- Run one process per node with one GPU card (8 nodes)

Alltoall Latency Performance (small)



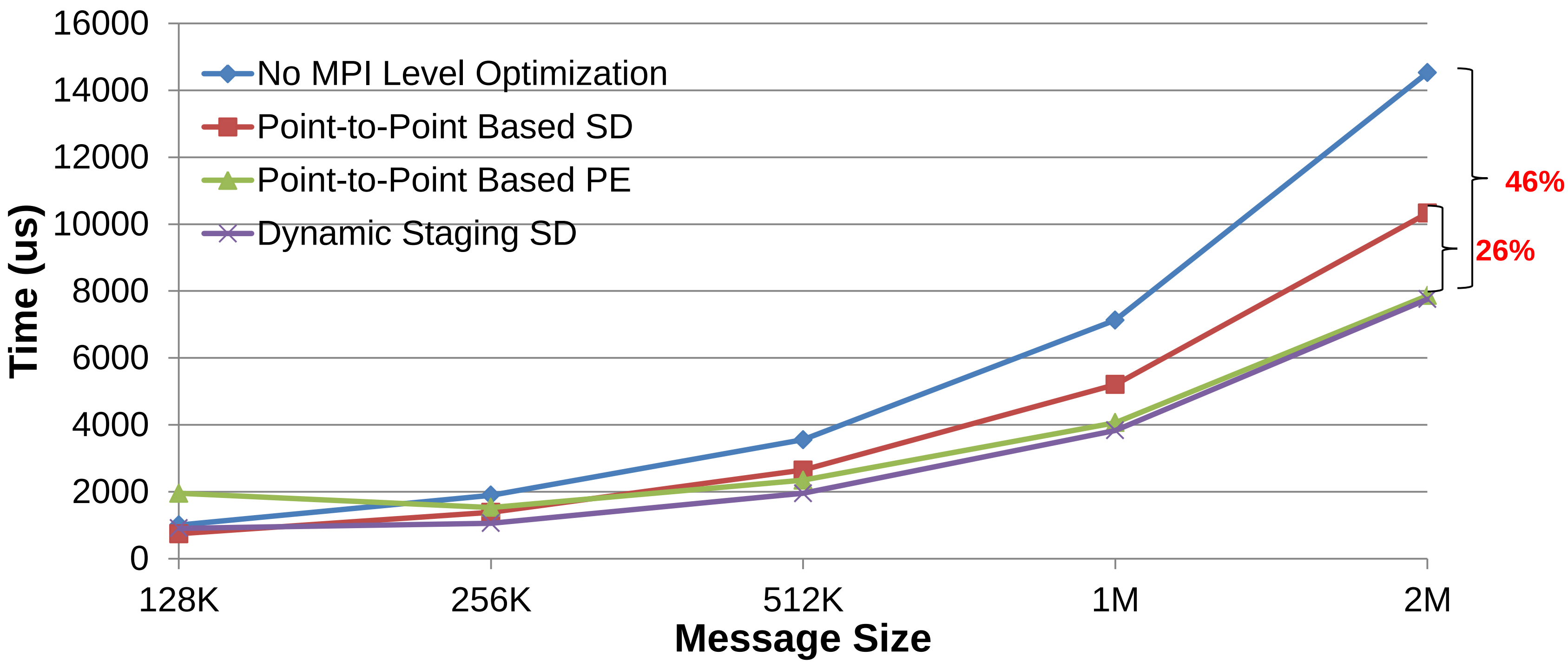
- High start-up overhead in P2P Based algorithms
- Static Staging method can overcome high start-up overhead
  - performs only slightly better than No MPI Level implementation
- We didn't group small data size to enable pipeline between DMA and RDMA

Alltoall Latency Performance (medium)



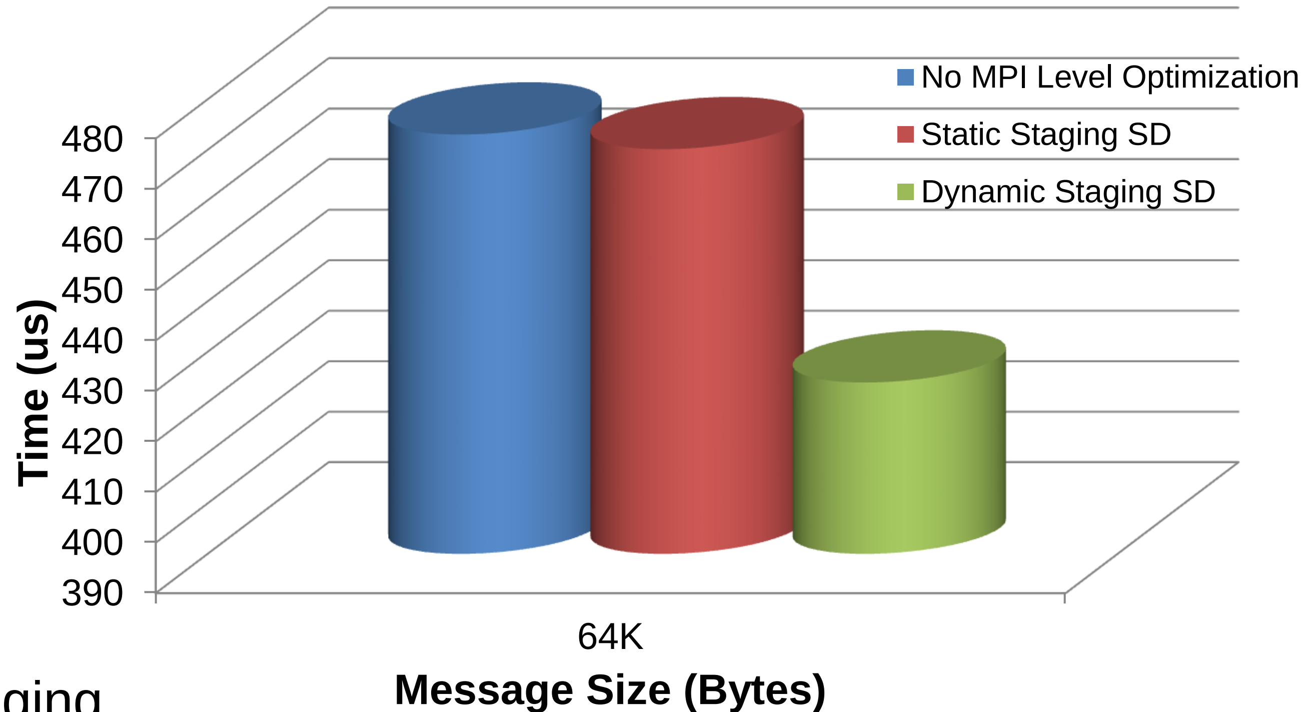
- P2P Based SD lost performance because of multiple times data movement between device and host
- Without pipeline design, No MPI Level Optimization method can't hide DMA data movement latency with RDMA data transfer
- *Up to 10.4% improvement from Dynamic Staging SD over No MPI Level Optimization method*

Alltoall Latency Performance (large)



- P2P Based
  - Pipeline is enabled for each P2P channel (ISC'11); better than No MPI Level Optimization
- Dynamic Staging
  - not only overlap DMA and RDMA for each channel, but also for different channels
  - *up to 46% improvement for Dynamic Staging SD over No MPI Level Optimization*
  - *up to 26% improvement for Dynamic Staging SD over P2P Based method SD*

Staging Benefit



- Static staging
  - Move data for all ranks in one package can't get better performance beyond a threshold
- Dynamic Staging
  - group data for in a threshold size package (128KB)
  - overlap DMA and RDMA for all channels

Outline

- Introduction
- Problem Statement
- Design Considerations
- Our Solution
- Performance Evaluation
- Conclusion and Future Work

Conclusion and Future Work

- MPI_Alltoall optimizations on GPU clusters (MVAPICH2-GPU)
  - support GPU to GPU alltoall communication with MPI_Alltoall; improve the programming productivity
  - resolve high start-up overhead between device and host for small and medium datasize
  - improve alltoall performance through Dynamic Staging method
  - get up to 46% latency improvement of Dynamic Staging compared with No MPI Level Optimization method

- Future work
  - integrate this design into MVAPICH2 future releases
  - improve applications' performance (3DFFT and CPMD)
  - investigate other collectives performance with MVAPICH2-GPU

Thank You!

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

Network-Based Computing Laboratory

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

MVAPICH Web Page

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