# CUDA Kernel based Collective Reduction Operations on Large-scale GPU Clusters

Ching-Hsiang Chu, Khaled Hamidouche, Akshay Venkatesh, Ammar Ahmad Awan and Dhabaleswar K. (DK) Panda

**Speaker: Sourav Chakraborty**

Network-based Computing Laboratory

Department of Computer Science and Engineering

The Ohio State University

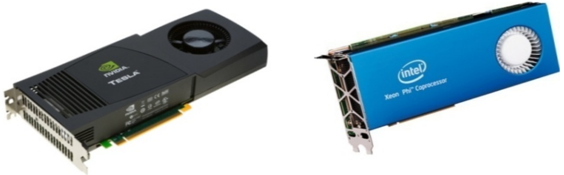
# Outline

- Introduction
- Proposed Designs
- Performance Evaluation
- Conclusion

# Drivers of Modern HPC Cluster Architectures

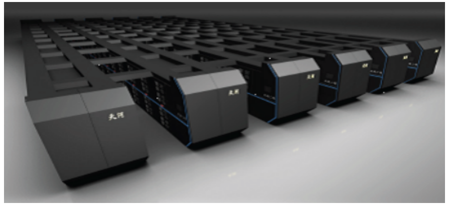
Multi-core Processors

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



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

- Multi-core processors are ubiquitous
- InfiniBand very popular in HPC clusters
- Accelerators/Coprocessors becoming common in high-end systems
- Pushing the envelope for Exascale computing



*Tianhe – 2*

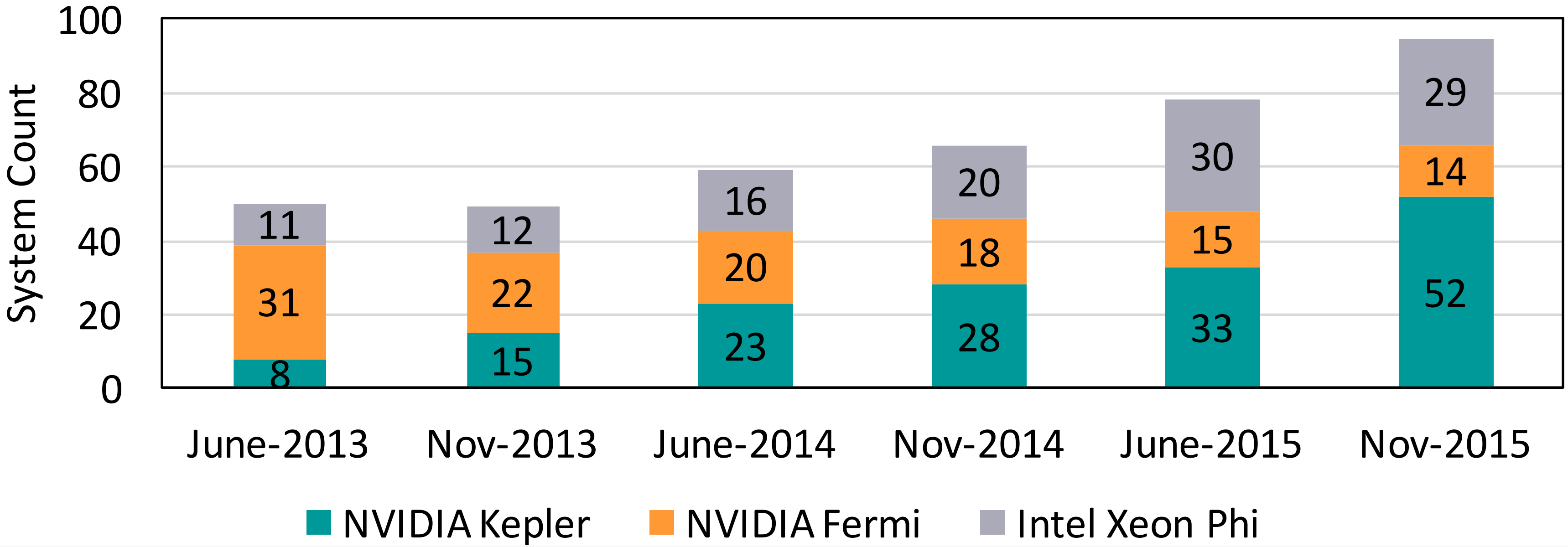
*Titan*

*Stampede*

*Tianhe – 1A*

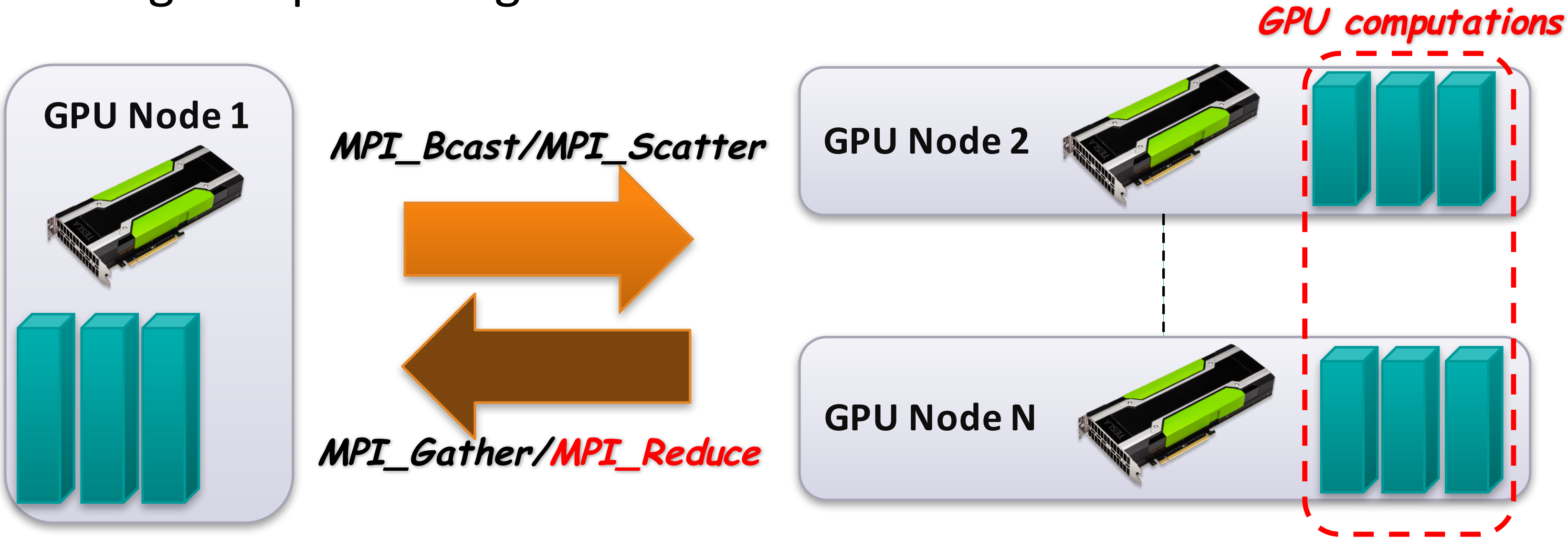
# Accelerators in HPC Systems

- Growth of Accelerator-enabled clusters in the last 3 years
  - 22% of Top 50 clusters are boosted by NVIDIA GPUs in Nov’15
  - From Top500 list (http://www.top500.org)



# Motivation – Collectives in Applications

- Scientific parallel applications spend a considerable amount of time in collective communication operations
  - E.g. Deep learning frameworks such as Caffe

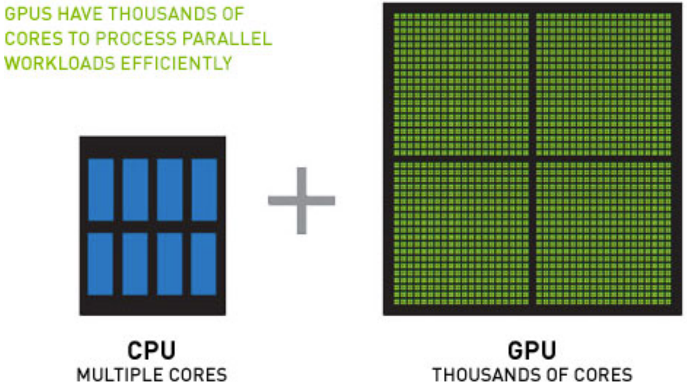


# Motivation - Collective Reduction Operations

- Scientific parallel applications spend a considerable amount of time in collective communication operations
  - **Pure communication collectives**: *Broadcast, Gather, Scatter…*
  - **Compute-oriented collectives**: *Reduce, Allreduce, Scan*
  - *Communication part is highly optimized*
- Compute-oriented collectives operations are not fully optimized for GPU clusters
  - ***CPU is doing all the works***
  - ***GPU resources are not fully utilized***

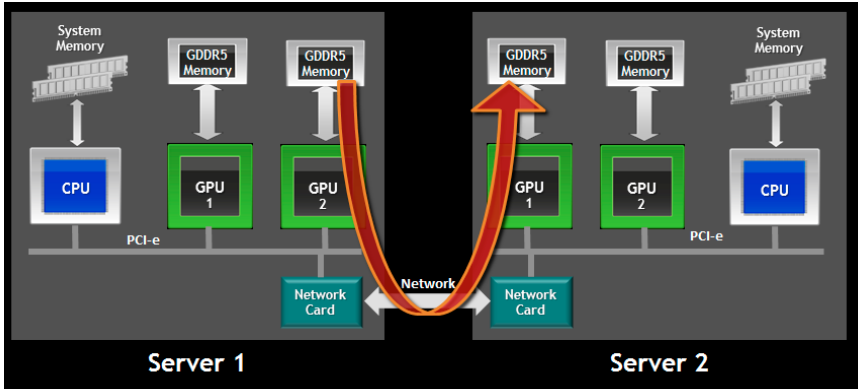
# Motivation – Powerful GPU Resources

- **Fast computation**
  - Massive parallelism


GPUS HAVE THOUSANDS OF CORES TO PROCESS PARALLEL WORKLOADS EFFICIENTLY
CPU
MULTIPLE CORES
GPU
THOUSANDS OF CORES

http://www.nvidia.com/object/gpu-accelerated-computing.html

- **Efficient communication**
  - NVIDIA GPUDirect RDMA



https://developer.nvidia.com/gpudirect

- **GPU features are not being utilized for all collectives**
- **Can we leverage these features to further optimize the compute-oriented collectives on GPU clusters?**

# Problem Statement

- **How** to eliminate explicit data movements between Host and GPU memories?
  - *cudaMemcpy* call is expensive!
- **How** to handle the GPU-to-GPU communication after the computations finish?
- **When** to use GPU for compute-oriented collective operations?
  - Launching kernels bring overhead; ***How to minimize?***

# Overview

- Design a framework that exploits the CUDA kernels to efficiently handle compute-oriented collectives
- Propose extensions to the existing collective algorithms to be GPU-Aware compute-oriented algorithms
  - ***MPI_Reduce, MPI_Allreduce and MPI_Scan***
- Detailed analysis and evaluation using different GPU systems including a Cray CS-Storm system.

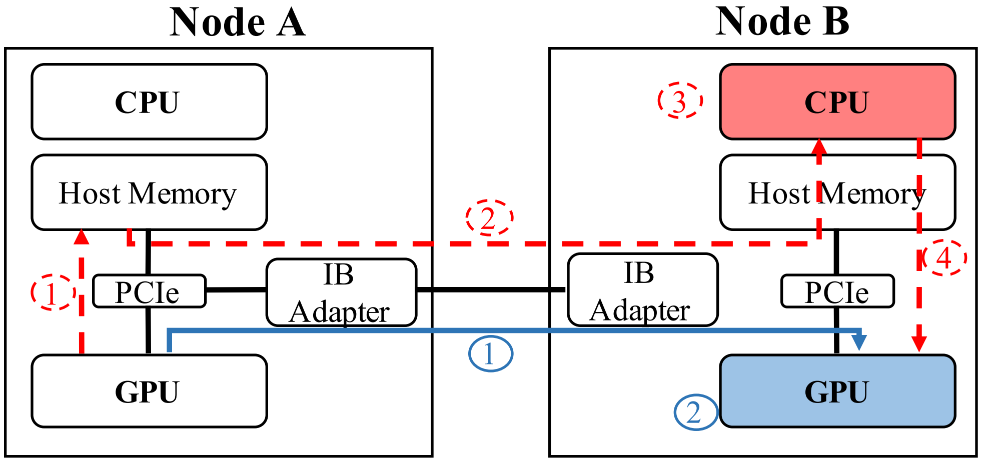
# Outline

- Introduction
- Proposed Designs
- Performance Evaluation
- Conclusion

# Design Consideration

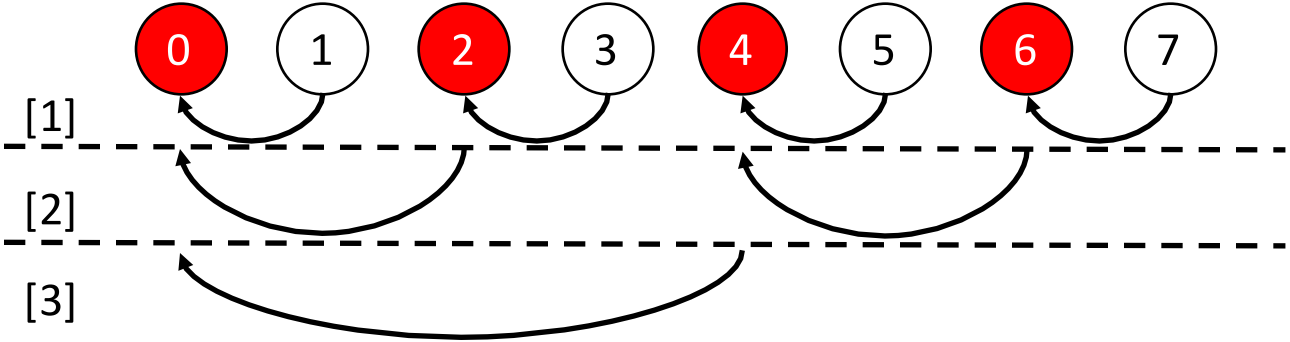
- **Existing designs**
  1. Explicit copy the data from GPU to host memory **Expensive!**
  2. Host-to-Host communication to remote processes **Fast**
  3. Perform computation on CPU **Good for small data** **Relative slow for large data**
  4. Explicit copy the data from host to GPU memory **Expensive!**

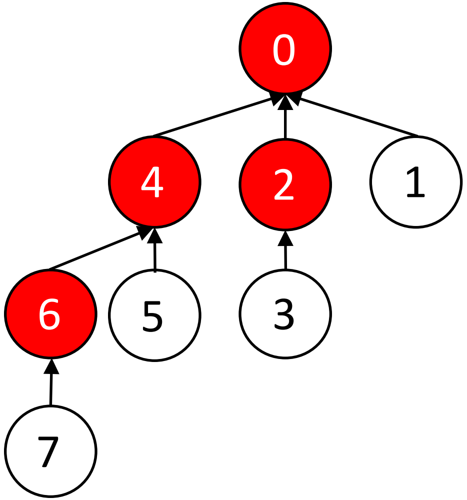
- **Proposed designs**
  1. GPU-to-GPU communication
     - **NVIDIA GPUDirect RDMA (GDR)**
     - Pipeline through host for large msg
  2. Perform computation on GPU
     - Efficient CUDA kernels



# K-nomial MPI_Reduce

- Tree-based K-nomial algorithm
  - Only the **non-leaf** nodes perform reduction operation
- Pros & Cons
  - Load balance, Efficient/scalable communication
  - Higher average latency





# Cost Analysis

- **Host-based Binomial-Reduce (Default)**

*Constant variant of tree initialization*

*Expensive **cudaMemcpy**, before/after reduction op.*

$$\lceil \log_2 n \rceil \times (\epsilon \times Comm_{Host}(M) + Comp_{Host}(M)) + 2 \times Copy(M)$$

*Fast Host-Host Comm.*

*Relatively slow computation on CPU*

*Message Size*

- **GPU-based Binomial-Reduce (BR-DD)**

PU

$\lceil l$ … $)$

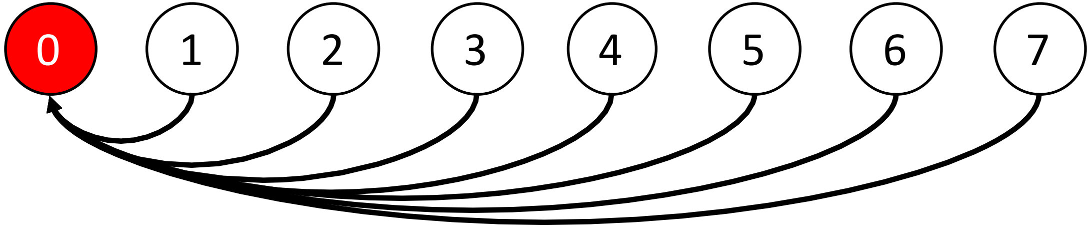
Higher cost of communication and kernel overhead, but fast GPU computation compensates it

➔ **Good for large messages**

*verhead of launching CUDA kernels (~10us)*

# Gather-first MPI_Reduce

- Gather-first algorithm
  - Root gathers all the data and perform the computation
    - Since only root needs the final result
- Pros & Cons
  - Low computation overhead
  - Poor scalability



# Cost Analysis

- **Host-based Gather and Reduce (GR-H-HH)**

$$(n-1)\times(Comm_{Host}(M) + Comp_{Host}(M)) + 2\times Copy(M)$$

- **Host-based Gather, GPU-based Reduce (GR-HH)**

$$\underline{(n-1)}\times(Comm_{Host}(M) + Overhead_{GPU}(M) + Comp_{GPU}(M) + 2\times Copy(M))$$

*Could suffer scalable issue ➔ Good for small messages or small scale*

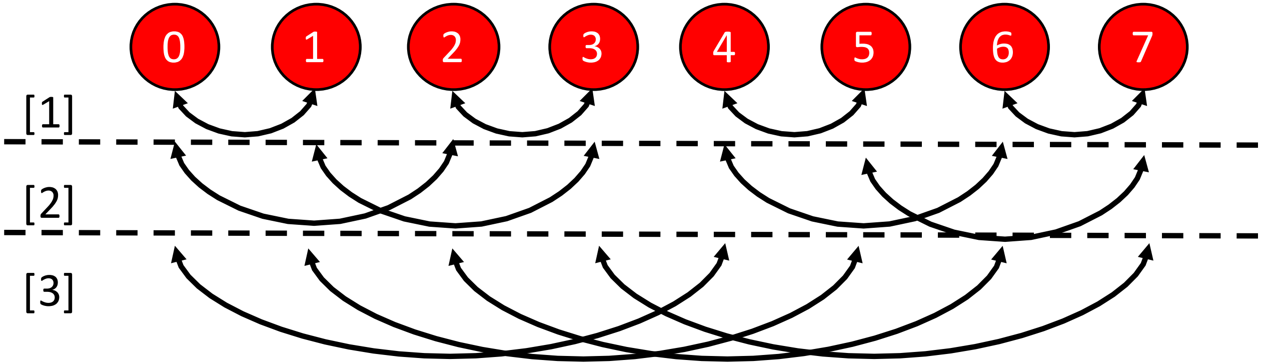
- **GPU-based Gather and Reduce (GR-DD)**

$$(n-1)\times Comm_{GDR}(M) + \underline{Overhead_{GPU}(M)} + Comp_{GPU}(M)$$

*Less affect from CUDA kernel launching overhead ➔ Good for small messages*

# GPU-based MPI_Allreduce and MPI_Scan

- Recursive doubling algorithm
  - Every processor needs to perform computation
- Pros & Cons
  - Load balance, Efficient/scalable communication
  - Higher average latency



# Cost Analysis

- GPU-based Recursive Doubling (RD-DD)

$$\lceil \log_2 n \rceil \times (\epsilon \times Comm_{GDR}(M) + Overhead_{GPU}(M) + Comp_{GPU}(M))$$

*Same as BD-DD MPI_Reduce*

- GPU-based Binomial-Reduce-Broadcast (GBRB-DD)

Higher cost of communication and kernel overhead, but fast GPU computation compensates it

→ **Good for large messages**

# Alternative and Extended Designs

| Communication | Computation | Design | Algorithm | Benefit |
|---|---|---|---|---|
| Host<->Host | CPU | **BR-H-HH (Default)** | Binomial-Reduce | *Large scale, small messages* |
| | | **RD-H-HH (Default)** | Recursive doubling | |
| | | **GR-H-HH** | Gather-Reduce | *Small scale, small messages* |
| | GPU | **GR-HH** | | |
| Host<->Device (GDR) | | **GR-HD / GR-DH** | | |
| Device<->Device (GDR) | | **GR-DD** | | |
| | | **BR-DD** | Binomial-Reduce | Large messages for any scale |
| | | **BRB-DD** | Binomial-Reduce-Bcast | |
| | | **RD-DD** | Recursive doubling | |
| Host<->Device (GDR) | | **RD-HD/RD-DH** | | |

# Outline

- Introduction
- Proposed Designs
- Performance Evaluation
- Conclusion

# Overview of the MVAPICH2 Project

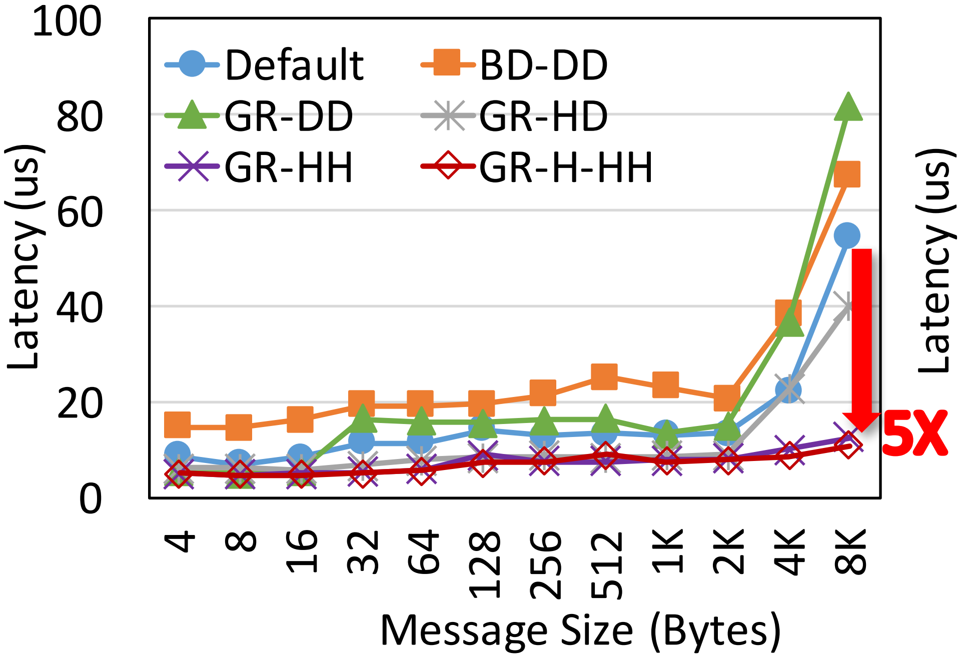
- High Performance open-source MPI Library for InfiniBand, 10-40Gig/iWARP, and RDMA over Converged Enhanced Ethernet (RoCE)
  - MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.0), Available since 2002
  - MVAPICH2-X (MPI + PGAS), Available since 2011
  - **Support for GPGPUs (MVAPICH2-GDR)** and MIC (MVAPICH2-MIC), Available since 2014
  - Support for Virtualization (MVAPICH2-Virt), Available since 2015
  - Support for Energy-Awareness (MVAPICH2-EA), Available since 2015
  - **Used by more than 2,550 organizations in 79 countries**
  - **More than 360,000 (> 0.36 million) downloads from the OSU site directly**
  - Empowering many TOP500 clusters (Nov '15 ranking)
    - 10th ranked 519,640-core cluster (Stampede) at TACC
    - 13th ranked 185,344-core cluster (Pleiades) at NASA
    - 25th ranked 76,032-core cluster (Tsubame 2.5) at Tokyo Institute of Technology and many others
  - Available with software stacks of many vendors and Linux Distros (RedHat and SuSE)
  - http://mvapich.cse.ohio-state.edu
- Empowering Top500 systems for over a decade
  - System-X from Virginia Tech (3rd in Nov 2003, 2,200 processors, 12.25 TFlops) ->
  - Stampede at TACC (10th in Nov'15, 519,640 cores, 5.168 Plops)

# Experimental Environments

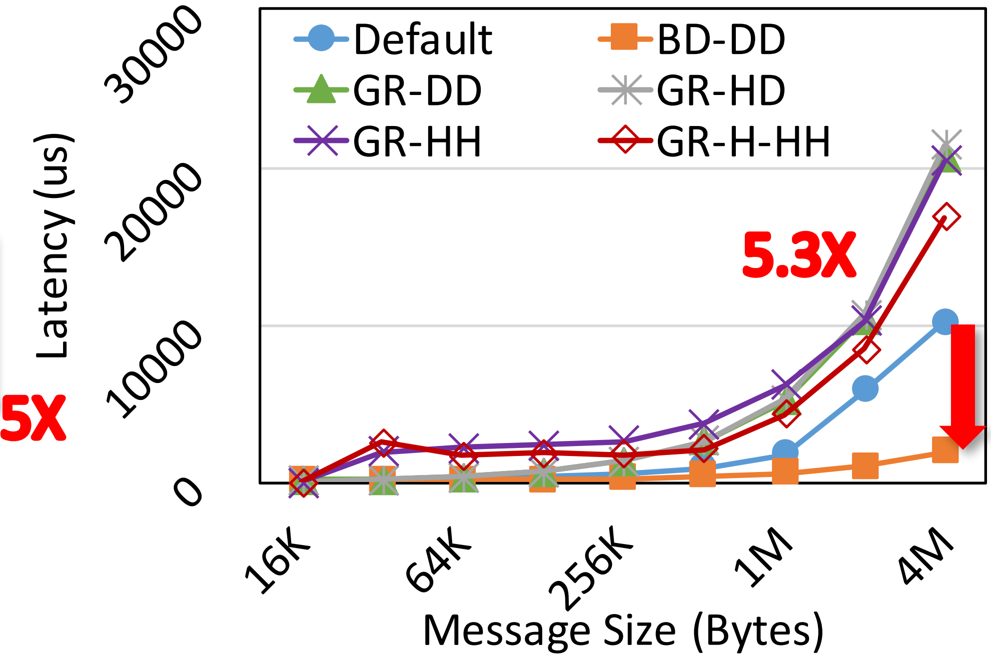
1. Wilkes cluster @ University of Cambridge
   - 2 NVIDIA K20c GPUs per node
   - Used for inter-node experiments
     - Up to 32 GPU nodes
2. CSCS cluster @ Swiss National Supercomputing Centre
   - Cray CS-Storm system
   - 8 NVIDIA K80 GPUs per node ( = 16 NVIDIA K40 GPUs per node)
   - Used for intra-node experiments
     - Up to 96 GPUs over 11 nodes

# Evaluation - MPI_Reduce @ Wilkes (32 GPUs)

**Gather-first** approaches win for small messages

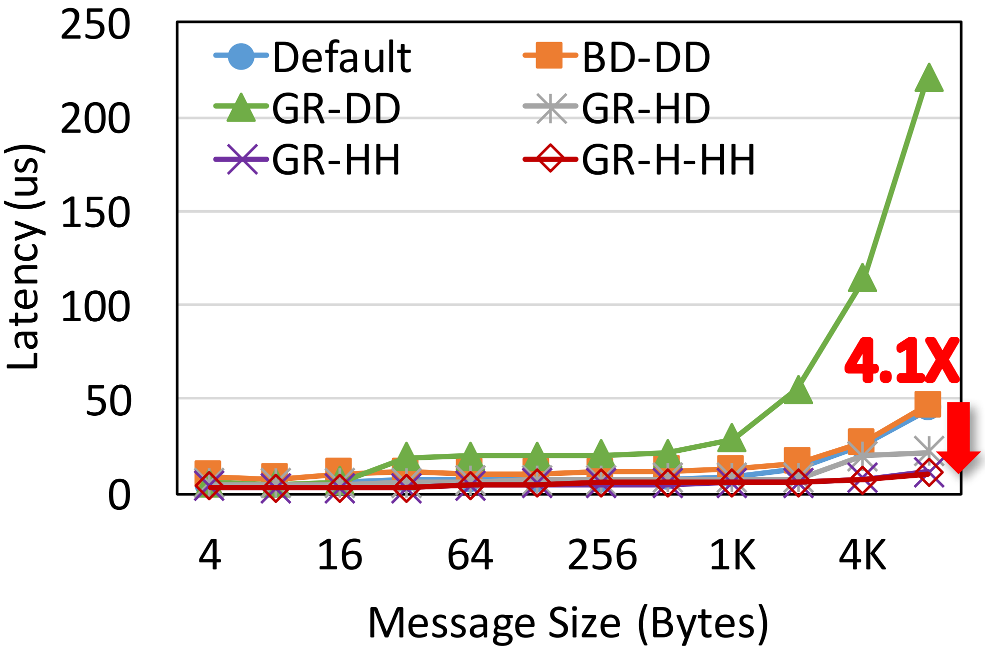


**K-nomial GPU-based** approach wins for large messages

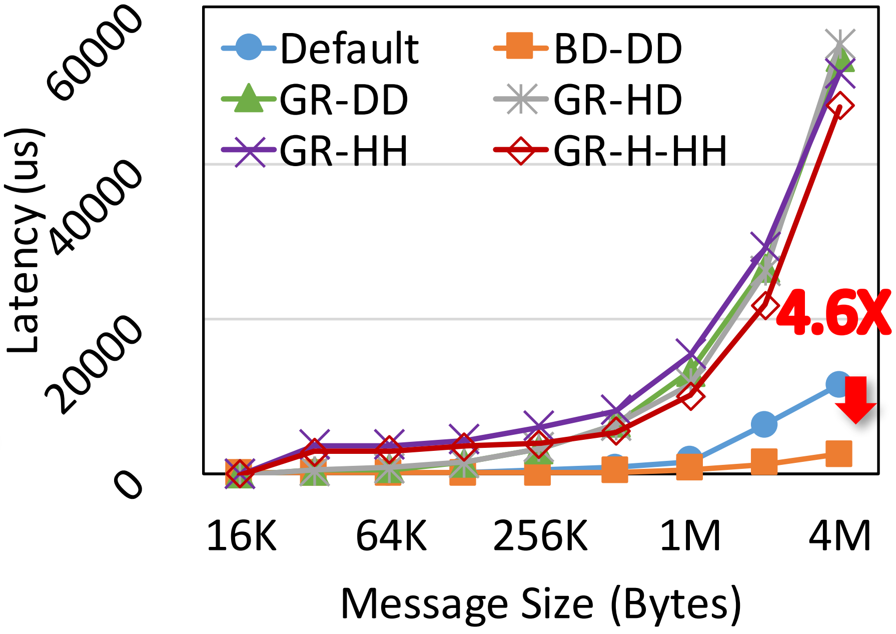


# Evaluation - MPI_Reduce @ CSCS (96 GPUs)

**Gather-first** approaches win for small messages

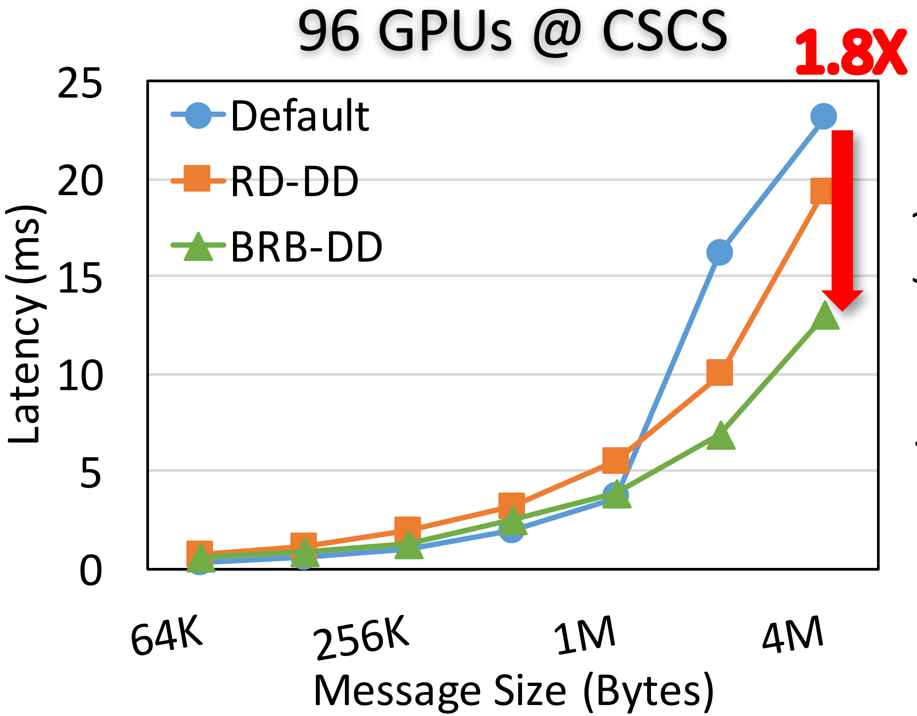


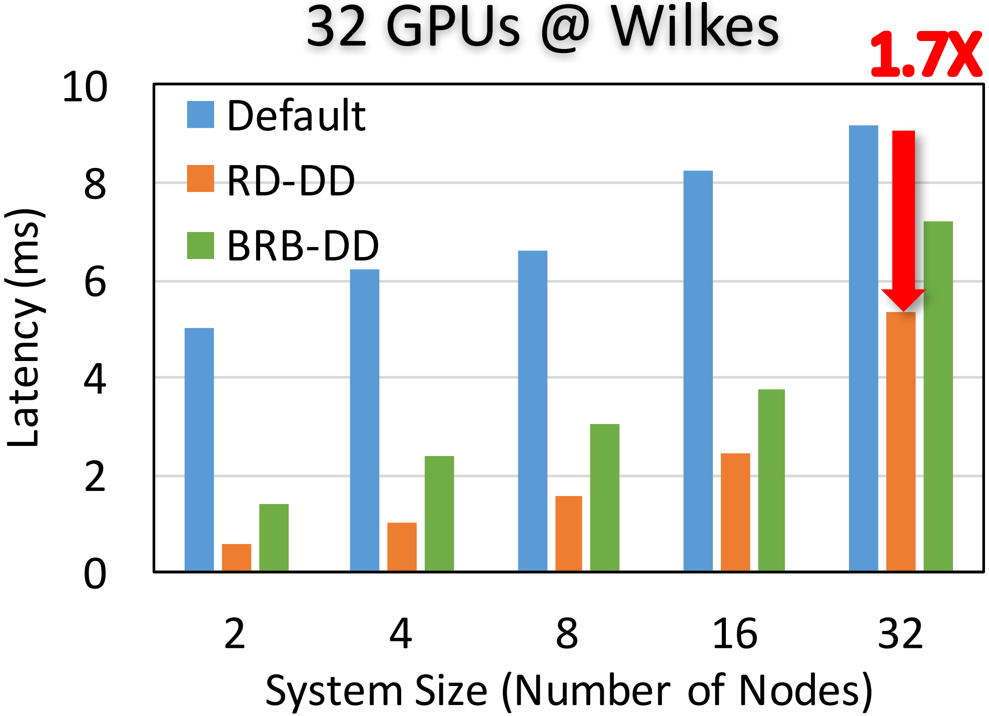
**K-nomial GPU-based** approach win for large messages



# Evaluation - MPI_Allreduce

Good Scalability

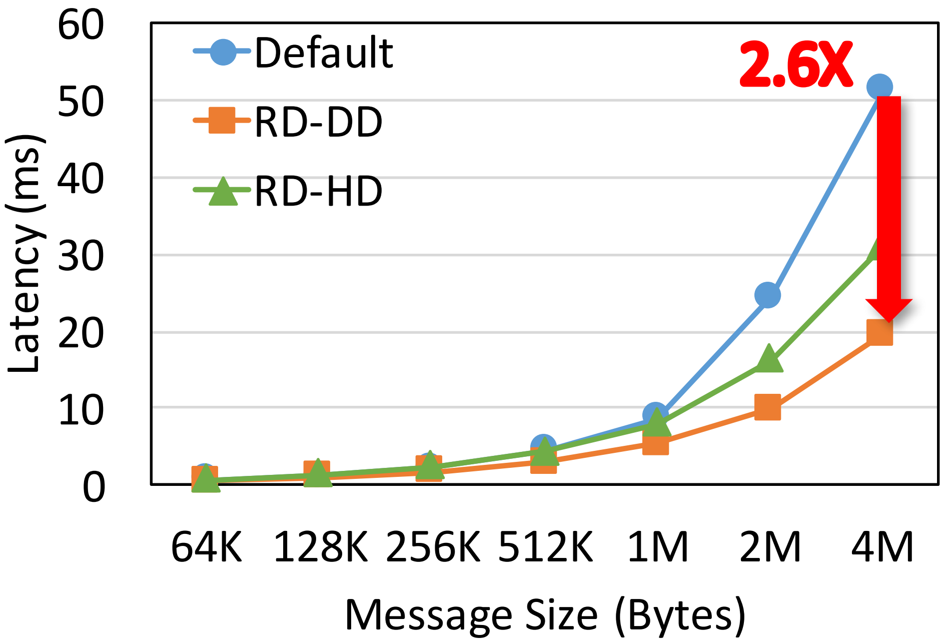




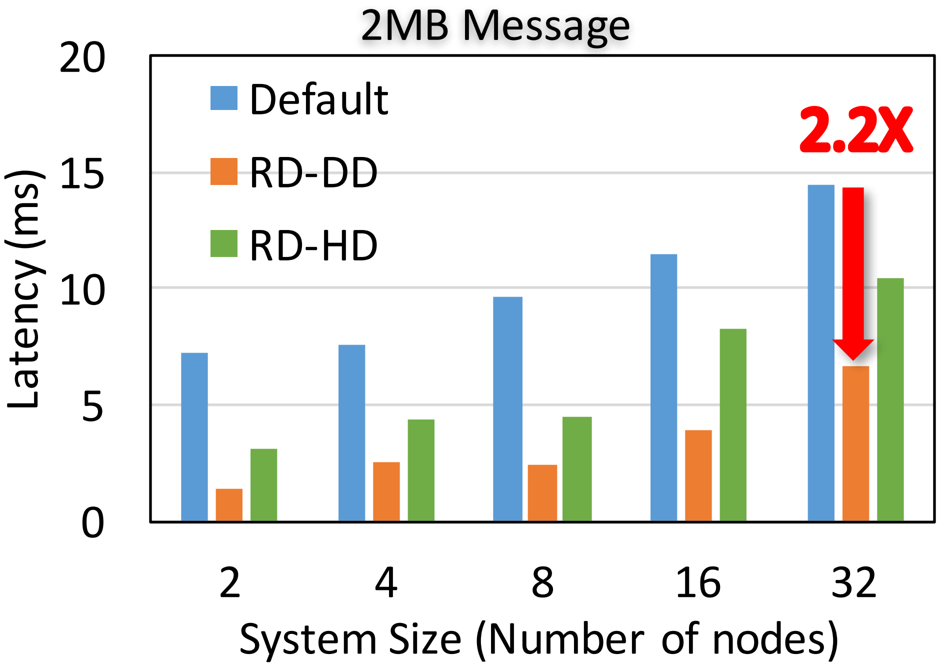
# Evaluation - MPI_Scan

Good Scalability

96 GPUs @ CSCS

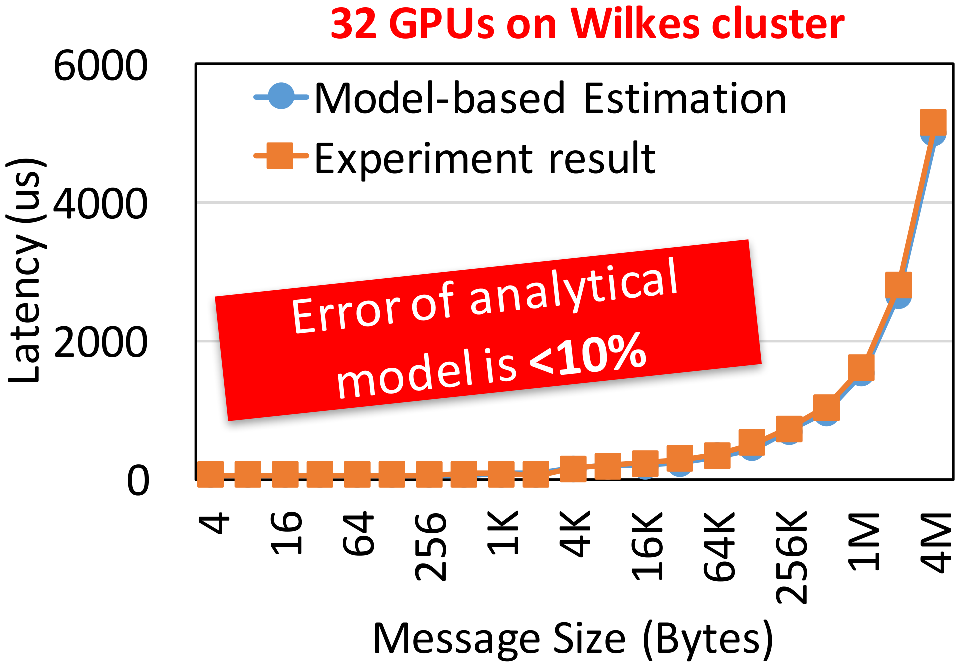


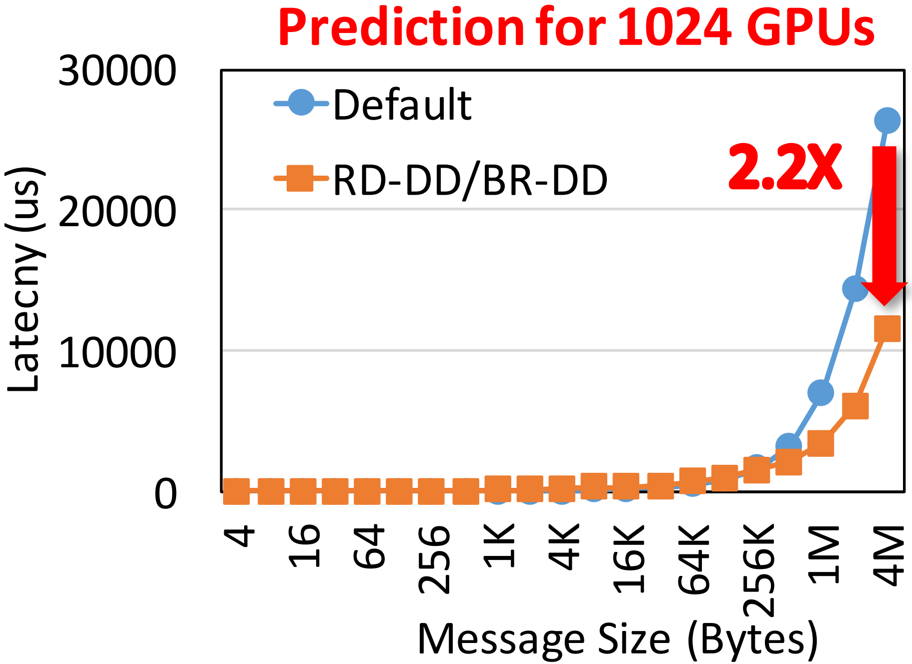
32 GPUs @ Wilkes



# Prediction

- Use the proposed analytical models to predict the performance for large scale GPU clusters





# Conclusion

- CUDA kernel based designs significantly improve the performance of compute-oriented collective operations
  - MPI_Reduce, MPI_Allreduce and MPI_Scan
  - CUDA kernels based reduction operations → Fast computation
  - GPUDirect feature → Efficient GPU-to-GPU communication
- Future work
  - Performing application-level evaluation
    - Deep learning frameworks such as Caffe
  - Will be included in the future release of MVAPICH2-GDR library

# Thank You!

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

The MVAPICH2 Project
http://mvapich.cse.ohio-state.edu/

# Reduction Operations on GPU

- CUDA Kernels
  - Example: Vector addition for MPI_SUM operation

```
template<class  T>
__global__  void vector_addition(T  *dst, T *src, size_t  count){
    int i = blockIdx.x  * blockDim.x  + threadIdx.x;
    for (; i < count; i += blockDim.x  * gridDim.x)
        dst[i]  += src[i];
}
```

**More information about optimizing your CUDA kernels:**

http://developer.download.nvidia.com/books/cuda-by-example/cuda-by-example-sample.pdf

http://http.developer.nvidia.com/GPUGems3/gpugems3_ch39.html