Efficient Large Message Broadcast using NCCL and CUDA-Aware MPI for Deep Learning

Ammar Ahmad Awan, Khaled Hamidouche, Akshay Venkatesh, and Dhabaleswar K. Panda

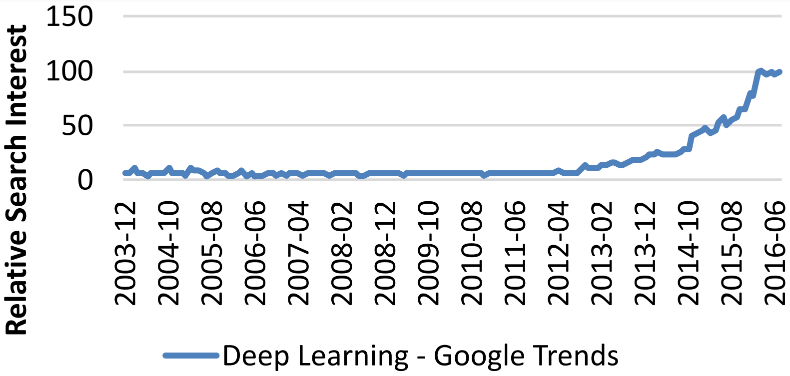
Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University, Columbus, OH, U.S.A

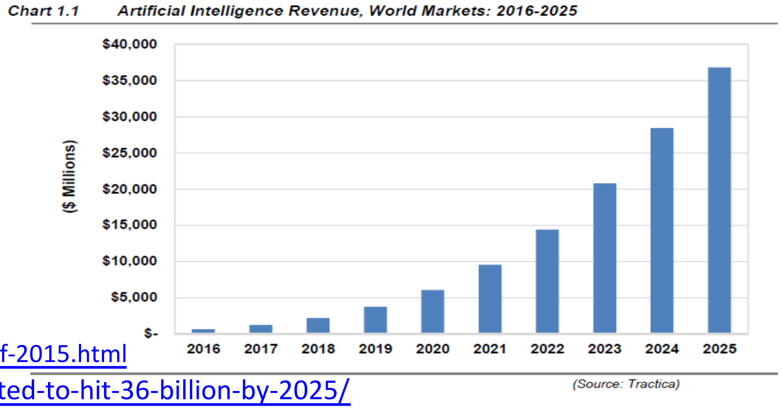
Outline

- Introduction
  - Deep Learning
  - CUDA-Aware MPI
  - NCCL
- Research Challenges
- Proposed Design
  - Hierarchical Communication
  - Implementation Details
- Performance Evaluation
- Conclusion and Future Work

Deep Learning Resurgence

- Deep Learning is going through a resurgence
  - Excellent accuracy for deep/convolutional neural networks
  - Public availability of versatile datasets like MNIST, CIFAR, and ImageNet
  - Widespread popularity of accelerators like Nvidia GPUs
- DL frameworks and applications
  - Caffe, **Microsoft CNTK**, Google TensorFlow, and many more..
  - Most of the frameworks are exploiting GPUs to accelerate training
  - Diverse range of applications – Image Recognition, Cancer Detection, Self-Driving Cars, Speech Processing etc.



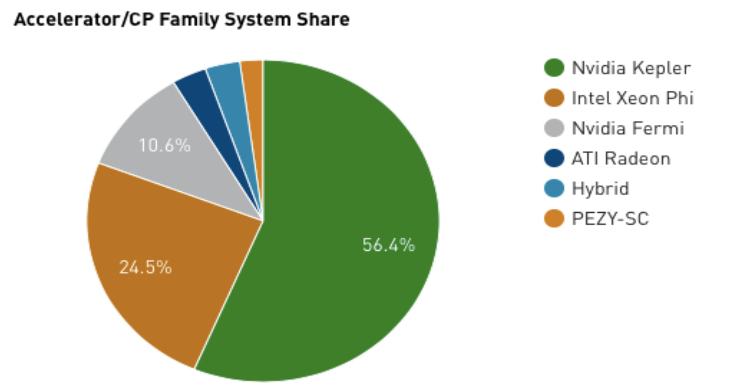


http://www.computervisionblog.com/2015/11/the-deep-learning-gold-rush-of-2015.html

https://www.top500.org/news/market-for-artificial-intelligence-projected-to-hit-36-billion-by-2025/

Deep Learning, GPUs, and HPC

- The ImageNet - Large Scale Visual Recognition Challenge (ILSVRC)
  – 90% of the ImageNet teams used GPUs in 2014*
  – DL models like AlexNet, GoogLeNet, and VGG are used
  – A natural fit for DL due to the throughput-oriented nature
- Nvidia GPUs are the main driving force for faster training of DL models
  – Can use Nvidia Kepler and/or Pascal architecture
  – DGX-1 (dedicated DL super-computer)
  – Titan series (lower precision but a good fit for DL)
- 63 / 500 Top HPC systems use Nvidia GPUs – www.top500.org

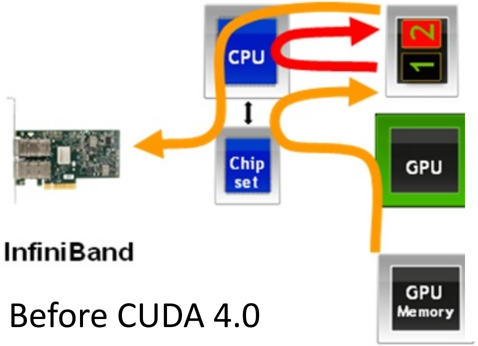


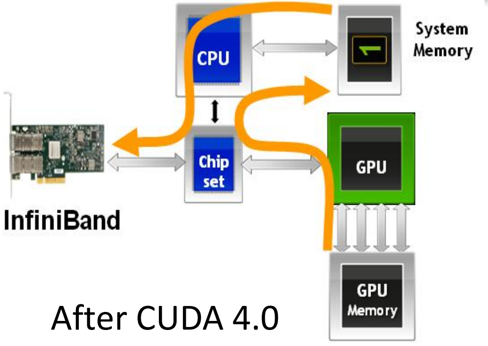
*https://blogs.nvidia.com/blog/2014/09/07/imagenet/

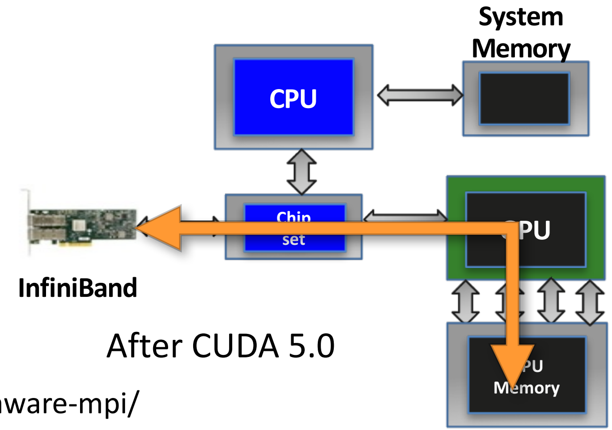
https://www.microway.com/hpc-tech-tips/deep-learning-frameworks-survey-tensorflow-torch-theano-caffe-neon-ibm-machine-learning-stack/

CUDA-Aware MPI

- Before CUDA 4.0, 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
- After CUDA 4.0, 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**
- After CUDA 5.0 (GDR), 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



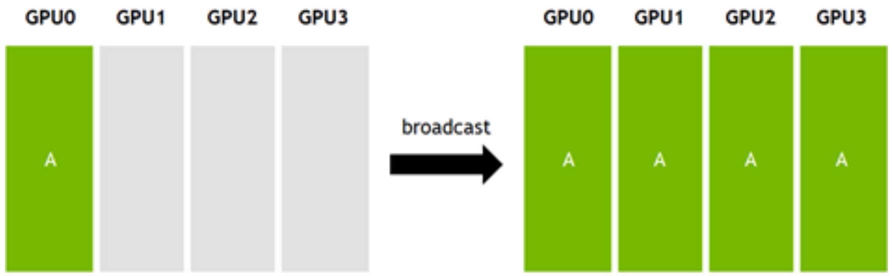




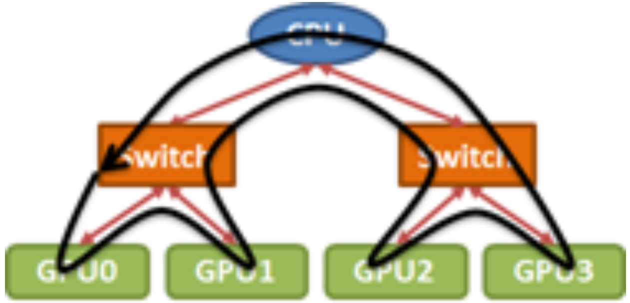
https://devblogs.nvidia.com/parallelforall/introduction-cuda-aware-mpi/

NCCL Communication Library

- Collective Communication with a caveat!
  - GPU buffer exchange
  - **Dense Multi-GPU** systems
    (Cray CS-Storm, DGX-1)
  - MPI-like – but not MPI standard compliant



- NCCL (pronounced Nickel)
  - Open-source Comm. Library by Nvidia
  - Topology-aware, ring-based (linear) collective communication library for GPUs
  - Divide bigger buffers to smaller chunks
  - Good performance for large messages
    - Kernel-based threaded copy (Warp-level Parallel) instead of cudaMemcpy



https://devblogs.nvidia.com/parallelforall/fast-multi-gpu-collectives-nccl/

Research Challenges: Overview

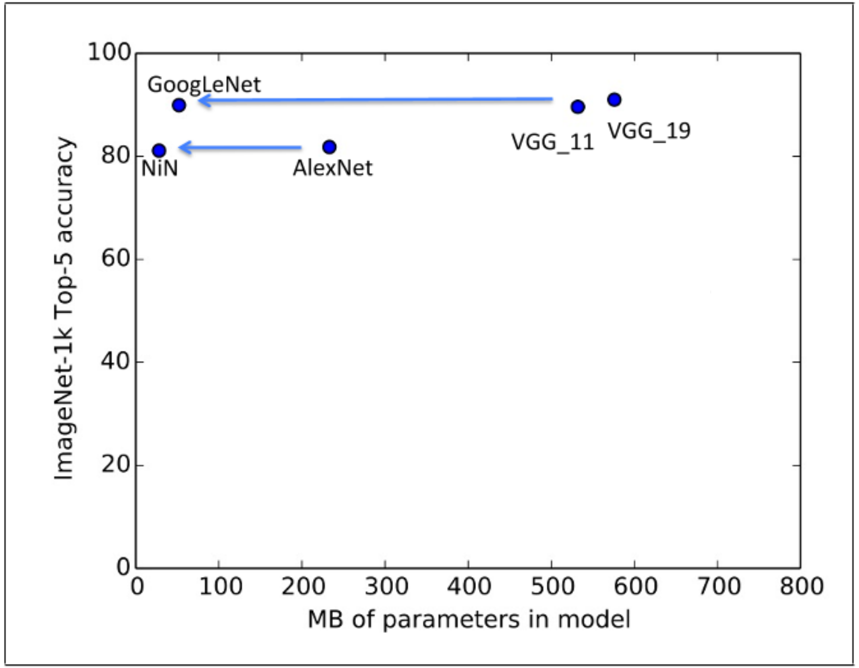
- What are the **new design challenges** brought forward by modern **Deep Learning (DL)** frameworks?

- How can we design **efficient** and **scalable** communication of **very large** GPU buffers for upcoming multi-GPU nodes?

- Can we exploit a **GPU-only** single-node communication library like **NCCL** to **scale out** on multi-GPU nodes of next-generation clusters?

Research Challenges: Details

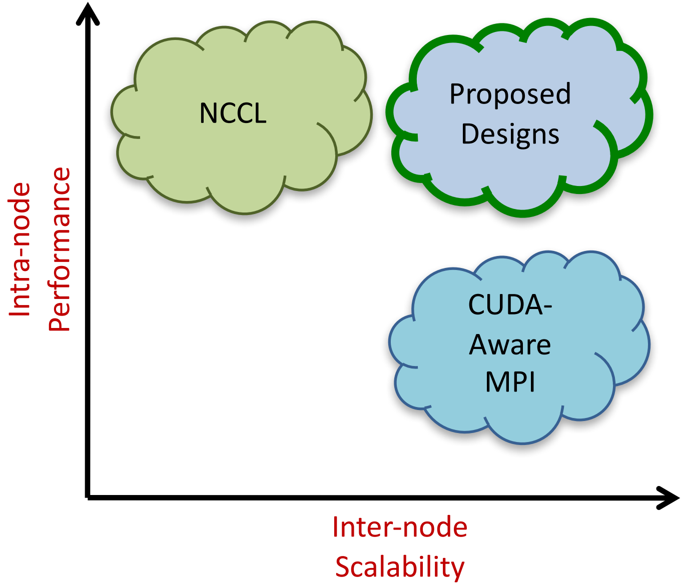
- What are the new requirements brought forward for **MPI_Bcast** by modern DL frameworks?
  1. **Very Large** Buffer Sizes – Order of Megabytes
  2. **Broadcast** of model parameters before each iteration
  3. **GPU-only** buffers in most cases
- Are there **libraries** that can improve collective communication **performance** for GPU buffers?
  - Yes, NCCL Library
- What are the **issues** in using **just NCCL**?
  - Single Process – Multiple Threads => Single node only
- Can MPI runtimes take advantage of CUDA-Awareness and NCCL in tandem?
  - Yes, **hierarchical** communication in MPI runtimes can be exploited to **scale out**



http://arxiv.org/abs/1511.00175

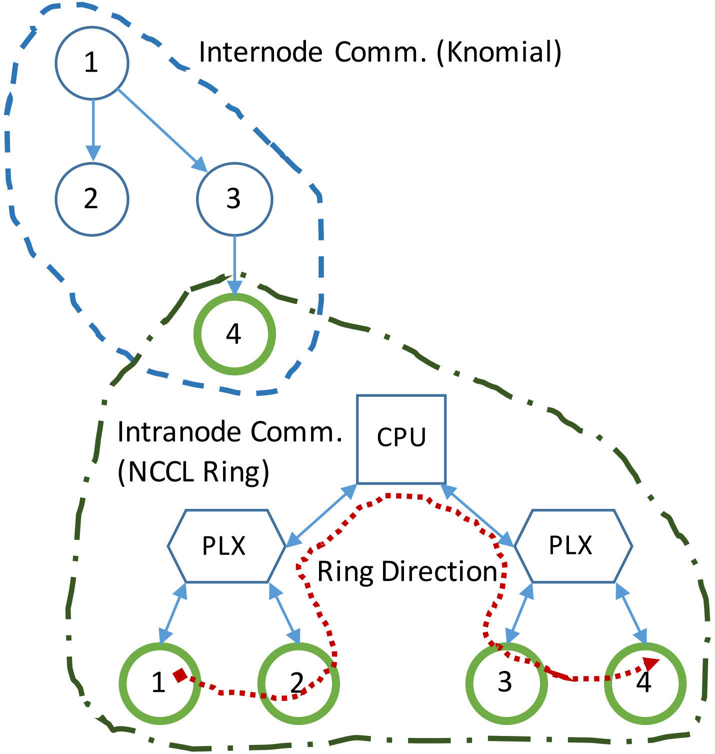
Design Space for Broadcast

- CUDA-Aware MPI provides excellent performance for small and medium message sizes
- NCCL has overhead for small messages but provides excellent performance for large messages
- Can we have designs that provide good performance for internode communication and internode scalability?



Proposed Design: Overview

- We design and implement MPI_Bcast
  - Augment different MPI_Bcast algorithms
  - Exploit hierarchical designs for scalable inter-node communication
  - In tandem, efficiently exploit NCCL for intra-node communication.
  - Using NCCL for the appropriate message range to get optimized performance for large messages
  - Exploit tuning to provide overall best performance for all message sizes
- Study and analyze the benefits of the proposed designs through a comprehensive performance evaluation using a micro-benchmark and a popular DL framework called CNTK.



Proposed Design: Details

1. Communicator Creation (MPI_Init time)
   – NCCL Communicators inside MPI Communicators
2. Communicator Caching (Successive Calls)
   – Do not create the communicators if we have them available => Creation is expensive!
3. First, exploit internode broadcast to exchange b/w nodes
4. Then, use ncclBcast for intranode GPUs where it performs better than existing intranode broadcast
5. Cleanup communicators (Some minor details)

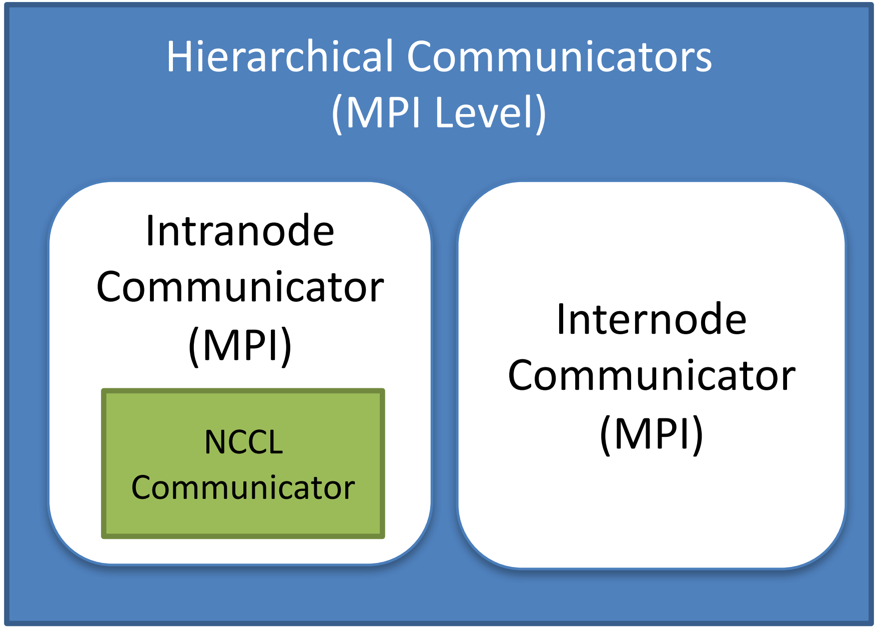
Communicator Creation/Caching

1. Create a NCCL communicator on the first MPI_Bcast call

```
ncclCommInitRank
    (&nccl_comm,
    my_local_size,
    nccl_commId,
    my_local_id);
```

2. Communicator creation is expensive!

- Save it for successive MPI_Bcast calls



NCCL Broadcast

- MPI applications can use nccl<Collective>() style functions with a CUDA stream argument
- ncclBcast() →

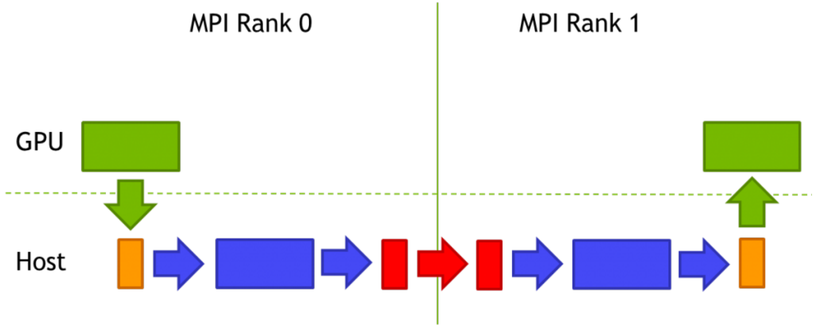
```
ncclResult_t
    ncclBcast(void*          buffer,
              int            count,
              ncclDataType_t datatype,
              int            root,
              ncclComm_t     comm,
              cudaStream_t   stream);
```

Hierarchical Communication

3. Hierarchical Design
   - Inter-node communication
     - Wide array of algorithms have been proposed and used
       - Binomial Tree
       - Knomial Tree
       - Scatter-Ring Allgather
       - Scatter-RecursiveDoubling Allgather
     - **STG-COLL**: Staged collectives – Copies between Device and Host buffers
     - **GDR-COLL**: GPUDirectRDMA (GDR) collectives – Direct communication using Device buffers
   - Intra-node communication
     - **STG-COLL**: Can use shared-memory communication when buffers are on the host
     - **NCCL**: (ring-based) communication for GPU buffers

Internode: STG-COLL and GDR-COLL

- STG-COLL:
  - Exploit the pipelined staging support via the Host memory
- GDR-COLL:
  - Use GPUDirect RDMA (GDR) in the correct fashion avoiding the P2P bottleneck
  - Use CUDA IPC where appropriate



STG-COLL: Copies between Host and GPU

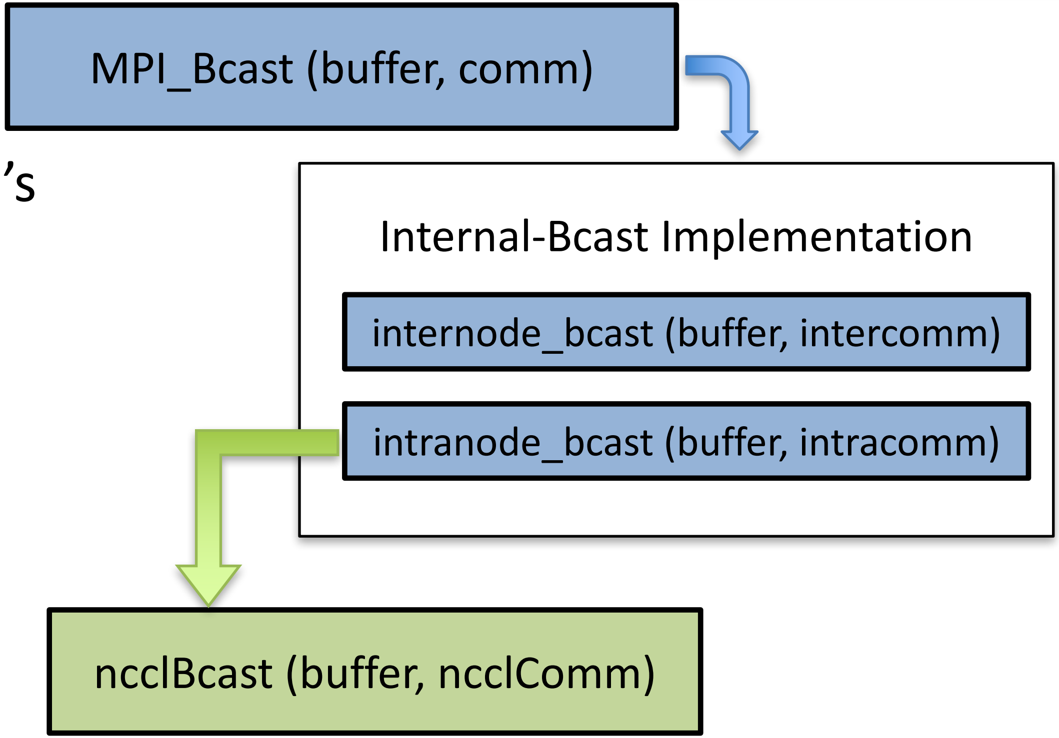
https://devblogs.nvidia.com/parallelforall/introduction-cuda-aware-mpi/

Intranode: STG-COLL or NCCL

- STG-COLL:
  - When the buffer is copied to the Host, shared memory collectives can be used directly
- Proposed (NCCL):
  - Sort of GDR-COLL – because we operate on the GPU buffers directly.
  - No copies between Host and GPU
  - STG-COLL is slow for large because of copying overhead
  - NCCL is throughput-oriented so it works much better
    - No copies are involved so lesser overhead

The Actual MPI_Bcast Operation

4. Making the actual **MPI_Bcast** call
   - Call the implementation's internal functions
   - Perform the internode phase
   - Perform the intranode phase



5. Cleanup at MPI_Finalize

Overview of MVAPICH2 Project

- High Performance open-source MPI Library for InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE
  - MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.0), Started in 2001, First version available in 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
  - Support for InfiniBand Network Analysis and Monitoring (OSU INAM) since 2015
  - **Used by more than 2,675 organizations in 83 countries**
  - **More than 391,000 (> 0.39 million) downloads from the OSU site directly**
  - Empowering many TOP500 clusters (Jun '16 ranking)
    - 12th ranked 519,640-core cluster (Stampede) at TACC
    - 15th ranked 185,344-core cluster (Pleiades) at NASA
    - 31st 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 (12th in Jun'16, 462,462 cores, 5.168 Plops)

Experimental Setup

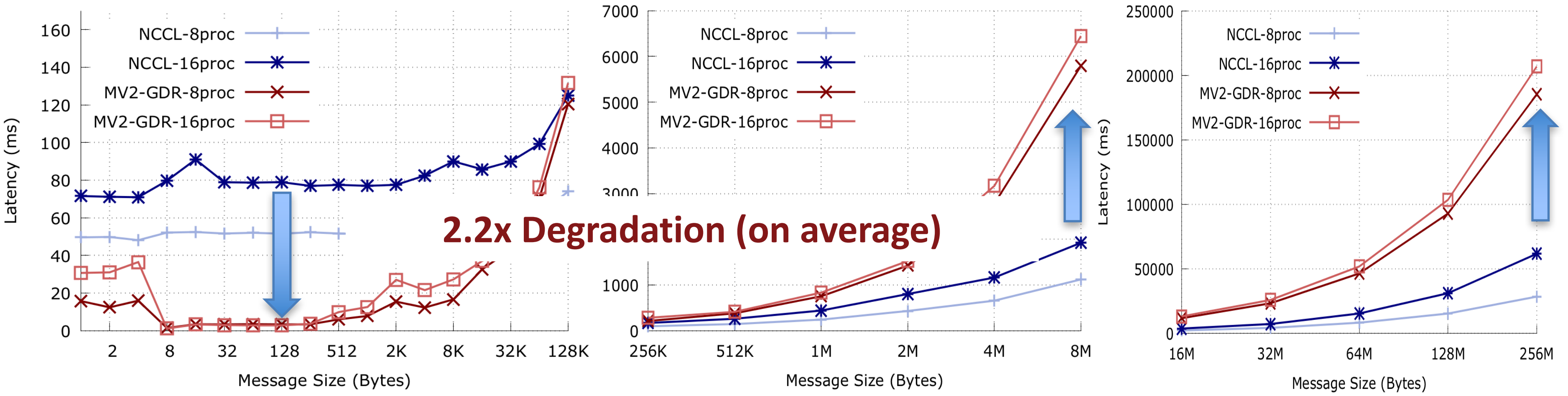
- We have performed all experiments on a Cray CS-Storm based GPU cluster called KESCH located at the Swiss National Supercomputing Center
- Multi-GPU dense cluster: 12 hybrid nodes, each node contains 8 NVIDIA K-80 GK210GL GPUs
- 4 K-80 cards are connected per socket
- 16 CUDA devices (or GPUs) in one node
- Dual-socket Intel Xeon CPUs
- Connect-IB FDR Interconnect

Performance Evaluation

- Micro-benchmark: Used osu_bcast from the OSU Microbenchmarks (OMB) suite
- Application: Microsoft CNTK
  - CUDA-Aware version called CA-CNTK*
  - Uses MPI_Bcast on large GPU buffers

* Dip Sankar Banerjee, Khaled Hamidouche and Dhabaleswar Panda; Re-designing CNTK Deep Learning Framework on Modern GPU Enabled Clusters; **to be presented** at 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Luxembourg 12-15 December 2016

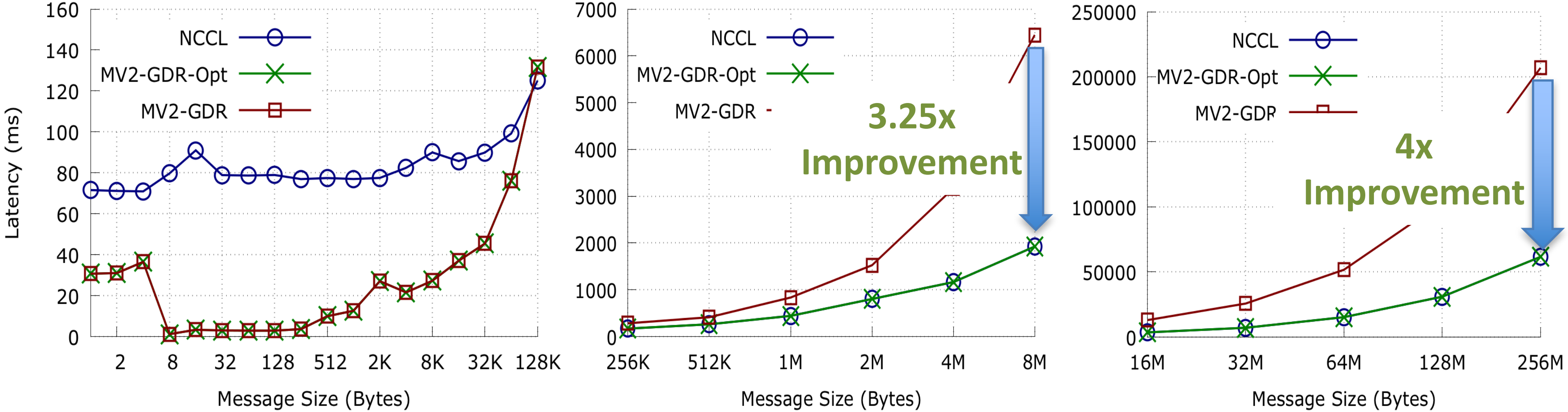
Current State: NCCL vs. MV2-GDR (8 and 16 GPUs)



- For small messages (up to 64K), NCCL suffers up to **2.2x degradation** for both 8 and 16 GPU cases while MV2-GDR has excellent performance

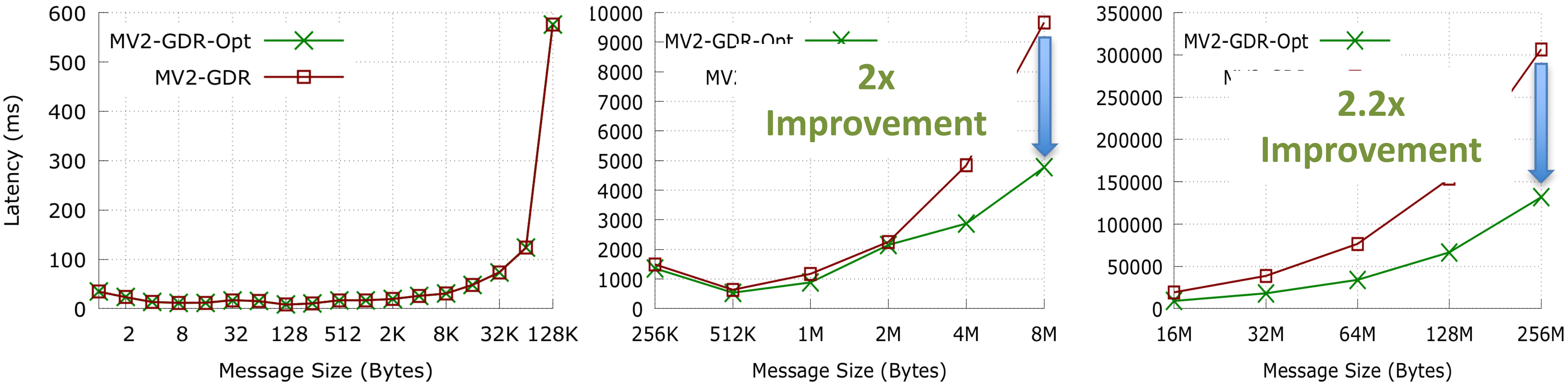
- For medium and large messages, the trend is reversed!
- NCCL performs much better and MV2-GDR suffers up to **2.2x degradation** for both 8 and 16 GPU cases

Comparison of MV2-GDR, MV2-GDR-Opt, and NCCL: 16 GPUs



- The proposed design (MV2-GDR-Opt) performs as good as MV2-GDR for small messages (up to 128K),
- For medium and large messages, MV2-GDR-Opt provides up to **4x improvement** over MV2-GDR

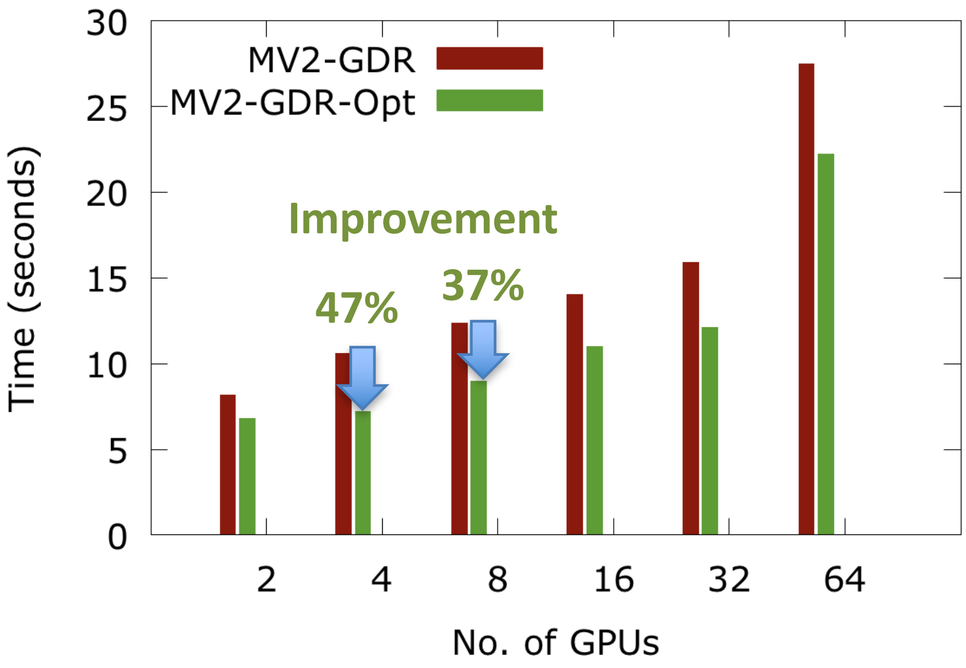
Comparison of MV2-GDR, MV2-GDR-Opt, and NCCL: 64 GPUs



- MV2-GDR-Opt) performs as good as MV2-GDR for small and medium messages (up to 2M)
- For large messages, MV2-GDR-Opt provides up to **2.2x improvement** over MV2-GDR

Application Performance: Microsoft CNTK (64 GPUs)

- Microsoft CNTK is a popular and efficient DL framework
- CA-CNTK is a CUDA-Aware version developed at OSU
- Proposed Broadcast provides up to **47 percent** improvement in Training time for the **VGG** network



MVAPICH

Conclusion

- Exponential growth in GPU-based Deep Learning frameworks that bring new requirements for MPI runtimes
- We proposed and implemented an efficient, scalable, and hierarchical design for MPI_Bcast to support DL frameworks.
- Proposed Designs provide
  - Efficient scale out up to 64 GPUs
  - Up to 47% improvement in training time for Microsoft CNTK framework
- Fundamental work that identifies challenges and opportunities for MPI runtimes that deal with next-generation DL frameworks and possibly HPDA applications
- Plan to make this work publicly available through a future MVAPICH2-GDR release

Future Work

- Exploit Optimizations for Dense GPU nodes with upcoming NVLink
- Towards Higher Performance (lower latency) and Scalability (> 256 GPUs)
- Evaluation with other DL frameworks

Thank You!

Ammar Ahmad Awan, Khaled Hamidouche, Akshay Venkatesh, and Dhabaleswar K. Panda

{awan.10, hamidouche.2, venkatesh.19, panda.2} @osu.edu

Network-Based Computing Laboratory

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

MVAPICH Web Page

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