Designing In-network Computing Aware Reduction Collectives in MPI

Presentation at the 30th IEEE Hot Interconnects symposium (Hoti '23)

Bharath Ramesh, Goutham Kalikrishna Reddy Kuncham, Kaushik Kandadi Suresh, Rahul Vaidya, Nawras Alnaasan, Mustafa Abduljabbar, Aamir Shafi, Hari Subramoni and Dhabaleswar K. (DK) Panda

The Ohio State University

ramesh.113@osu.edu

Follow us on

https://twitter.com/mvapich

Outline

- **Introduction**
- Background
- Motivation
- Problem Statement and Contributions
- Design
  - Overview
  - Registration cache design
  - Analyzing impact of chunking reductions
  - Proposed Allreduce design
- Results
- Conclusion and Future work

Introduction: Drivers of Modern HPC Cluster Architectures

Multi-/Many-core Processors

High Performance Interconnects – InfiniBand
<1usec latency, 200-400Gbps Bandwidth>

Accelerators
high compute density, high performance/watt
>9.7 TFlop DP on a chip

SSD, NVMe-SSD, NVRAM

- Multi-core/many-core technologies
- Remote Direct Memory Access (RDMA)-enabled networking (InfiniBand, RoCE, Slingshot)
- Solid State Drives (SSDs), Non-Volatile Random-Access Memory (NVRAM), NVMe-SSD
- Accelerators (NVIDIA GPGPUs)

Frontier

Fugaku

Summit

Lumi

MPI Reduction collectives and In-network Computing

- Reduction collectives (such as MPI_Allreduce) are important for HPC/DL
  - Involve both compute and communication
- Using CPUs everywhere leads to sub-optimal scale-up and scale-out efficiency
  - Motivates the need for offloading common operations away from the CPU to allow the CPU to perform other useful tasks
- In-network compute allows offloading operations to network devices
  - Switches are a good candidate due to high bandwidth and ability to reduce data on-the-fly eliminating redundancy
  - High scale-out efficiency and network topology awareness
  - Frees up CPU cycles for other operations

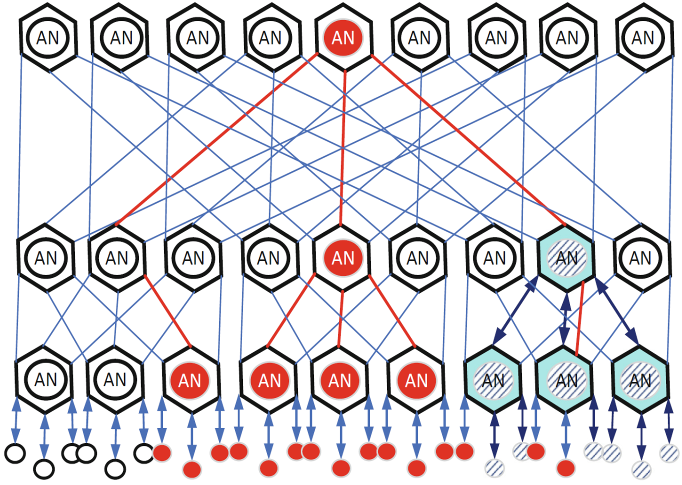
Outline

- Introduction
- **Background**
- Motivation
- Problem Statement and Contributions
- Design
  - Overview
  - Registration cache design
  - Analyzing impact of chunking reductions
  - Proposed Allreduce design
- Results
- Conclusion and Future work

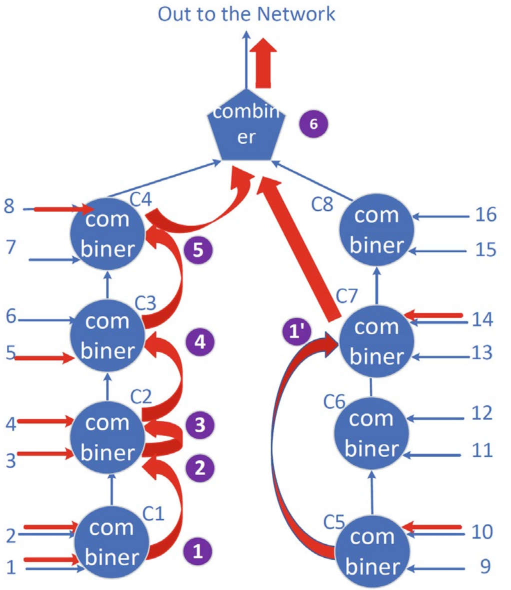
What is SHARP?

- Scalable Hierarchical Aggregation and Reduction Protocol
- Advantages
  - Progress, offload computation and communication
  - Focuses on low latency for small messages, maximal bandwidth utilization for large messages
  - Hierarchical Aggregation in a logical tree providing low latency for small messages - also known as a low latency tree (LLT)
  - Streaming aggregation with pipelined ring-based algorithms for large messages, called Streaming Aggregation Tree (SAT)
- In-Network computing

SHARP Reduction trees and Streaming Aggregation (SAT)



Aggregation Tree



Switch-level reduction (radix 16)

Images taken from Graham, Richard et al. Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)™ Streaming-Aggregation Hardware Design and Evaluation. DOI : 10.1007/978-3-030-50743-5_3 (https://link.springer.com/content/pdf/10.1007/978-3-030-50743-5_3.pdf)

Outline

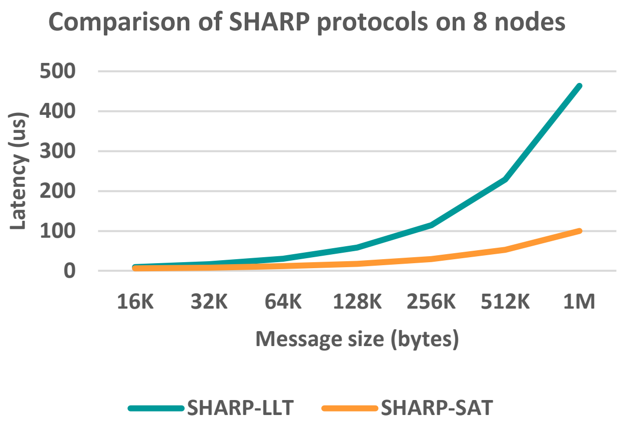
- Introduction
- Background
- **Motivation**
- Problem Statement and Contributions
- Design
  - Overview
  - Registration cache design
  - Analyzing impact of chunking reductions
  - Proposed Allreduce design
- Results
- Conclusion and Future work

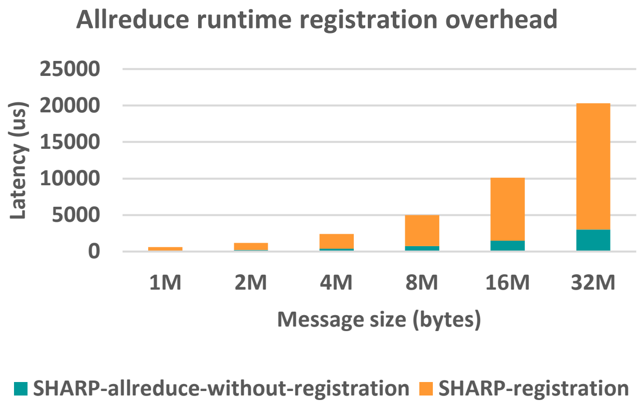
Limitations of state-of-the-art schemes for large message reduction collectives

- Prior work on reduction collectives with SHARP
  - Used leader-based schemes that had a reduction, followed by a SHARP operation and finally a broadcast
  - Not suitable for message sizes >= 8K
- Single-copy schemes are very efficient for large message data movement
  - XPMEM allows remote process to have load/store access through address space mapping
- Using Sharp SAT in MPI has a few limitations and bottlenecks that need to be addressed for achieving good scale-out performance
- Motivates the need for large message reduction designs that combine advantages of SHARP and single-copy schemes like XPMEM

Motivation

- SHARP SAT provides excellent bandwidth with close to point-to-point latency
- Registration involves pinning pages to memory (like InfiniBand registration)
  - Overhead increases significantly with increase in message size
  - Requires a cache that avoids expensive calls to sharp_coll_reg_mr
- Switch resources are limited
  - Causes bottlenecks when scaling up on modern CPUs with hundreds of cores
  - The SHARP runtime places limits to manage resources
- Motivates need for designs that are aware of SHARP runtime capabilities, overcome bottlenecks and scale-up efficiently for many processes per node





Outline

- Introduction
- Background
- Motivation
- **Problem Statement and Contributions**
- Design
  - Overview
  - Registration cache design
  - Analyzing impact of chunking reductions
  - Proposed Allreduce design
- Results
- Conclusion and Future work

Problem Statement and Contributions

- **Problem Statement - Can we propose an algorithm for large message allreduce that overcomes bottlenecks and resource constraints in the SHARP runtime by making efficient use of node and network level resources?**
- Contributions
  - Identify registration overheads involved in the use of SHARP streaming aggregation for large messages and propose solutions to address them
  - Analyze the impact of chunking reductions when using streaming aggregation for different message sizes to empirically determine ways to overlap intra-node reductions with SHARP-based reductions
  - Propose an algorithm for large allreduce that utilizes SAT and CPUs efficiently
  - Evaluate the proposed design by comparing it against state-of-the-art MPI libraries

Outline

- Introduction
- Background
- Motivation
- Problem Statement and Contributions
- **Design**
  - **Overview**
  - Registration cache design
  - Analyzing impact of chunking reductions
  - Proposed Allreduce design
- Results
- Conclusion and Future work

Proposed Design Overview

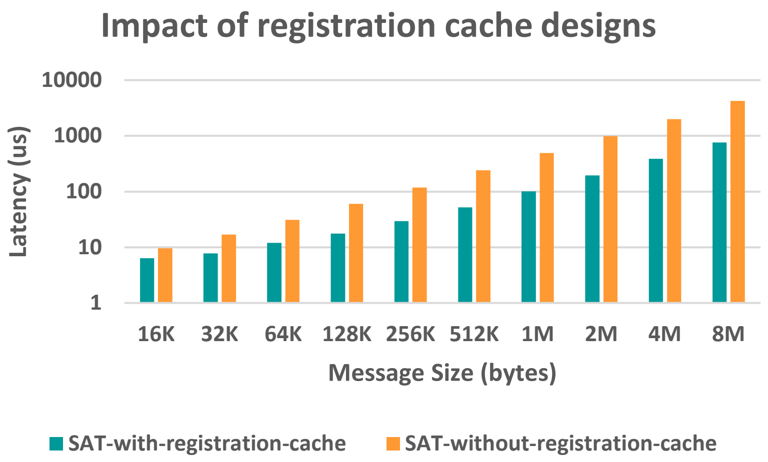
- Use a registration cache to amortize registration costs in the SHARP runtime
- Designate a “leader” process on each node to interact with SHARP
- Chunk buffer into PPN (number of processes per node) chunks and reduce to a single buffer belonging to the leader process
  - Uses XPMEM for load/store access
  - All processes perform local reductions, but only leader process calls the SHARP runtime
  - Once local reductions are complete, leader calls a non-blocking MPI_Allreduce
    - Perfect overlap of intra-node and inter-node steps
  - Local reduction happens in batches for good network utilization
  - Final result broadcast within the node

Outline

- Introduction
- Background
- Motivation
- Problem Statement and Contributions
- **Design**
  - Overview
  - **Registration cache design**
  - Analyzing impact of chunking reductions
  - Proposed Allreduce design
- Results
- Conclusion and Future work

Registration cache design

- Use an AVL tree or similar, to store buffer addresses
  - O(log n) insertion/query time
  - If buffer entry exists, directly get registration information from cache
- Up to 5.6X reduction in latency

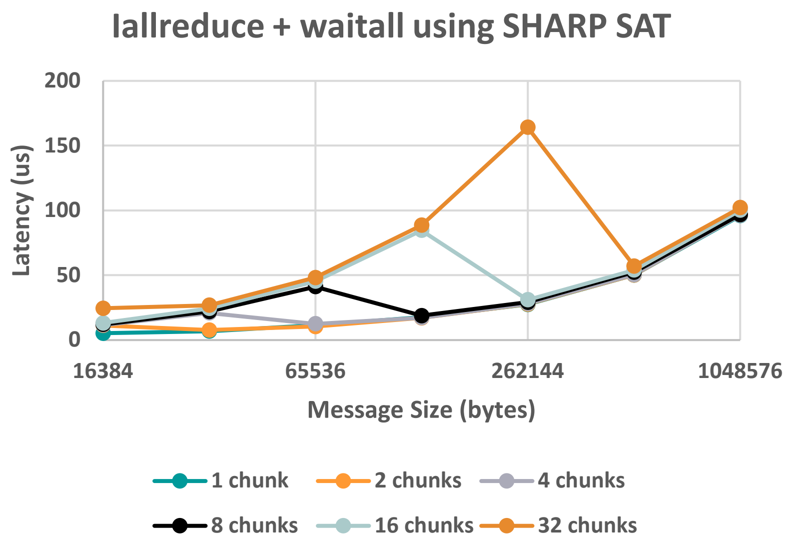


Outline

- Introduction
- Background
- Motivation
- Problem Statement and Contributions
- **Design**
  - Overview
  - Registration cache design
  - **Analyzing impact of chunking reductions**
  - Proposed Allreduce design
- Results
- Conclusion and Future work

Analyzing impact of chunking iallreduce operations

- Measure impact of a message sent using one call to the SHARP library vs multiple calls
- Given a message size M and number of chunks C, call non-blocking SHARP allreduce C times (of size M/C each) followed by waitall
- Indirect measure of overlap at the network level
- Splitting into chunks of size >= 16384 gives the same latency (independent of num_chunks)
  - Can be overlapped with reductions within the node

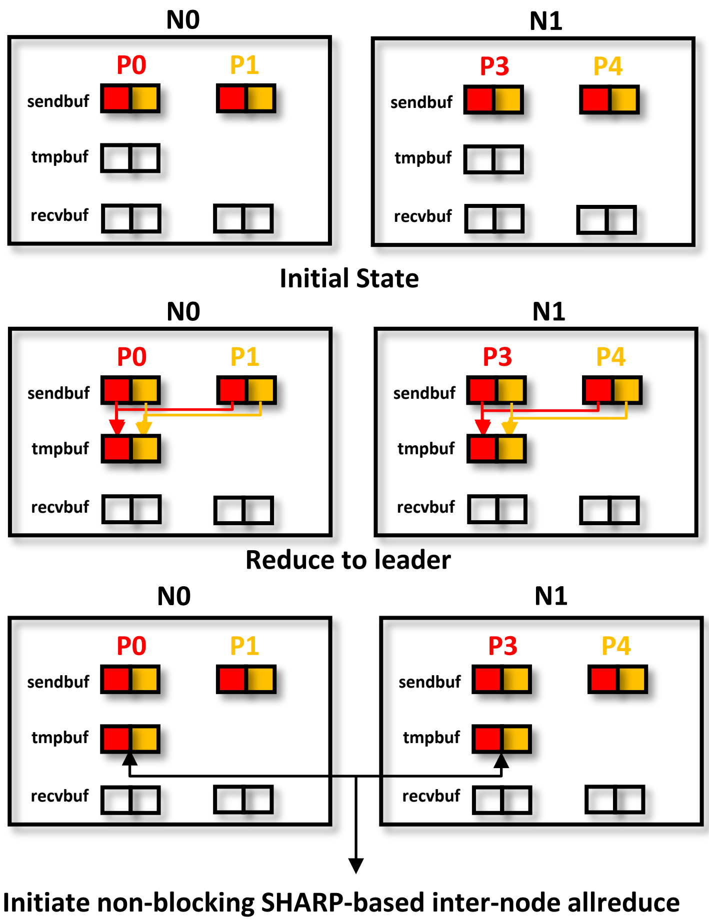


Outline

- Introduction
- Background
- Motivation
- Problem Statement and Contributions
- **Design**
  - Overview
  - Registration cache design
  - Analyzing impact of chunking reductions
  - **Proposed Allreduce design**
- Results
- Conclusion and Future work

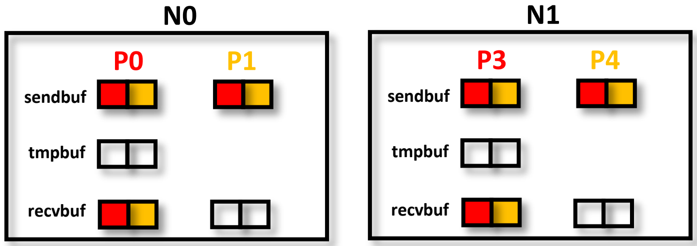
Proposed Allreduce Design

- First process on each node is designated as leader
- Before reduction, exchange buffer information using shared memory (for XPMEM load/stores)
- Process i reduces the ith chunk from every process and stores to tmpbuf at leader
- At the end of this step, leader on every node has the reduced result for the current phase
- Leader process initiates non-blocking inter-node SHARP allreduce
- Use "request" objects to track progress of SHARP Allreduce operations

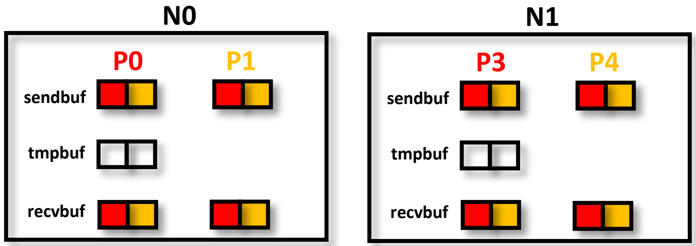


Proposed Allreduce Design – Continued

- For large buffers, the intra-node reduction and inter-node phases are run multiple times
  - Reduction of large buffers is time consuming
  - Done in multiple phases for good network utilization
  - Chunk size if tuned to get perfect overlap of intra-node and inter-node operations
- Leader waits for non-blocking allreduces to complete after all runs of the first two phases are done
- Perform and intra-node broadcast to get final result



After Waitall



After Broadcast

Outline

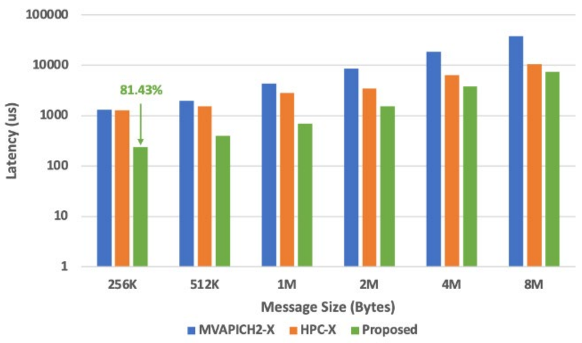
- Introduction
- Background
- Motivation
- Problem Statement and Contributions
- Design
  - Overview
  - Registration cache design
  - Analyzing impact of chunking reductions
  - Proposed Allreduce design
- **Results**
- Conclusion and Future work

Experimental setup

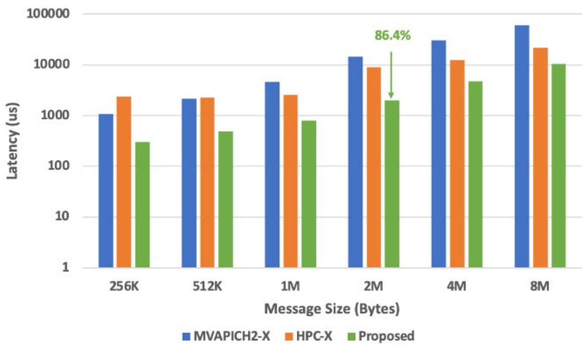
Cluster	MRI	HPCAC
Processor model	AMD EPYC 7713	Intel(R) Xeon(R) Gold 6138
Max Clock speed	3.72GHz	2GHz
Number of sockets	2	2
Cores per socket	64	20
RAM	256GB	196GB
Interconnect	NVIDIA HDR-200 with Quantum 2 switches	NVIDIA HDR-200 with Quantum 2 switches
MPI libraries	MVAPICH2-X, HPC-X	MVAPICH2-X, HPC-X

MPI_Allreduce – 2 nodes

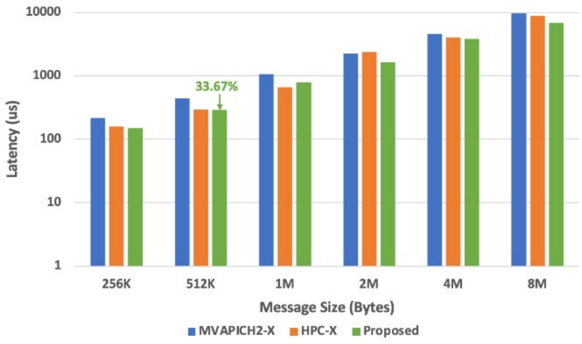
- Increased parallelism by using multiple processes and SHARP for reduction
- Up to 81.43% over state-of-the-art for 32PPN and 86.4% for 64PPN on MRI
- Up to 33.67% over state-of-the-art for 16PPN and 60% for 32PPN on HPCAC
- Increased number of page faults leads to decreased benefits at 1M (Needs to be investigated further)



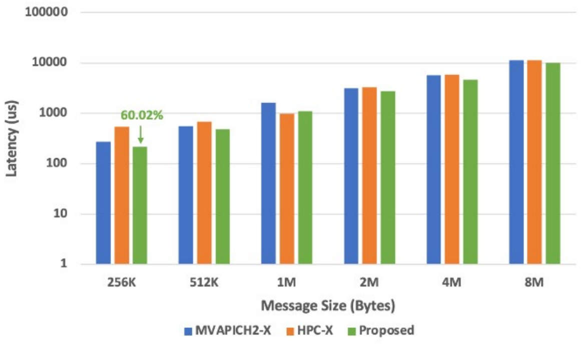
MRI - 32PPN



MRI - 64PPN



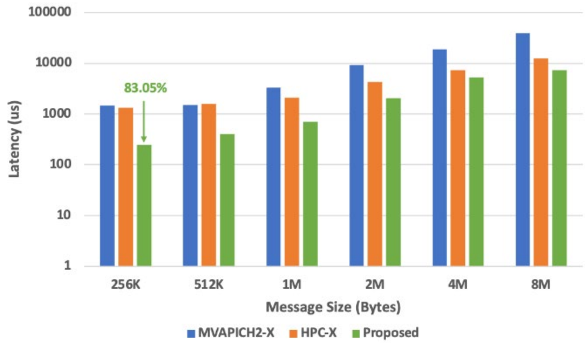
HPCAC - 16PPN



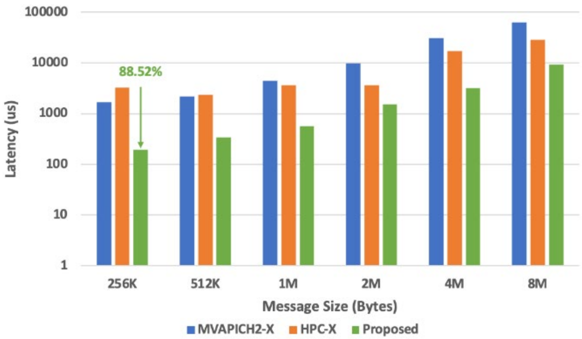
HPCAC - 32PPN

MPI_Allreduce – 4 nodes

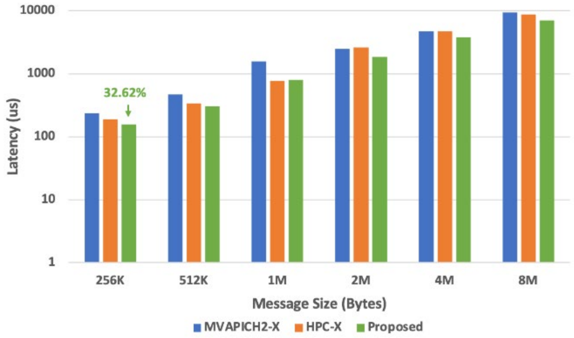
- Increased parallelism by using multiple processes and SHARP for reduction
- Up to 83.05% over state-of-the-art for 32PPN and 88.52% for 64PPN on MRI
- Up to 32.62% over state-of-the-art for 16PPN and 46.91% for 32PPN on HPCAC



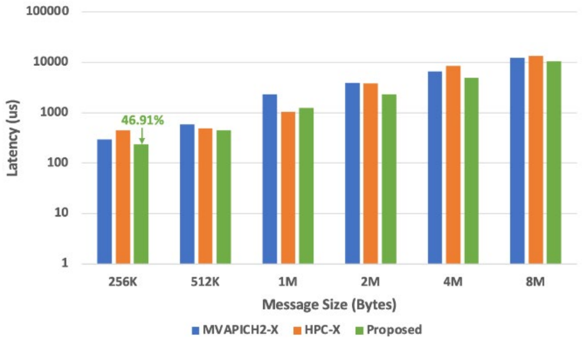
MRI - 32PPN



MRI - 64PPN



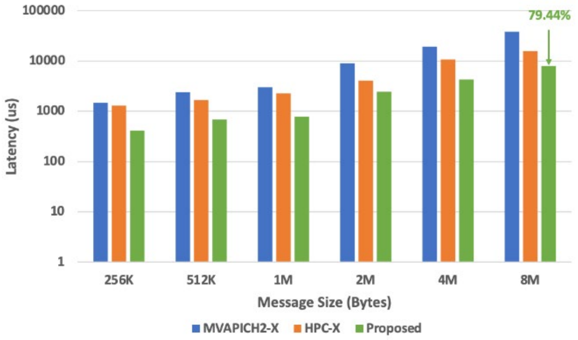
HPCAC - 16PPN



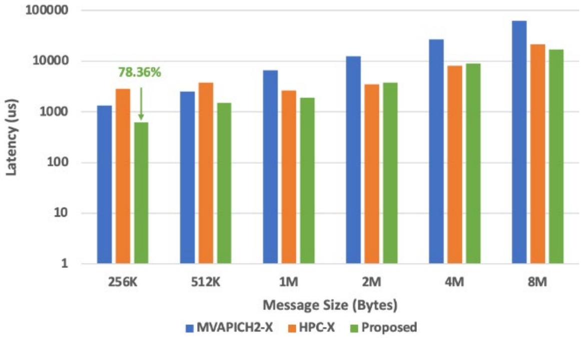
HPCAC - 32PPN

MPI_Allreduce – 8 nodes

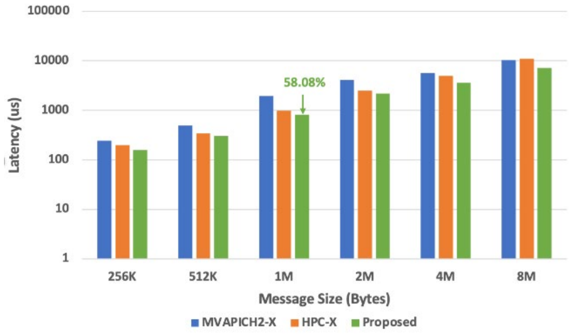
- Increased parallelism by using multiple processes and SHARP for reduction
- Up to 79.44% over state-of-the-art for 32PPN and 78.36% for 64PPN on MRI
- Up to 58.08% over state-of-the-art for 16PPN and 52.13% for 32PPN on HPCAC



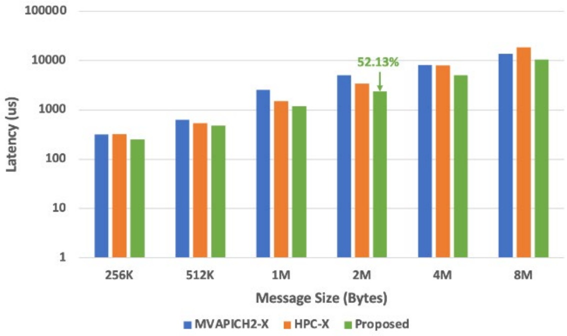
MRI - 32PPN



MRI - 64PPN



HPCAC - 16PPN



HPCAC - 32PPN

Outline

- Introduction
- Background
- Motivation
- Problem Statement and Contributions
- Design
  - Overview
  - Registration cache design
  - Analyzing impact of chunking reductions
  - Proposed Allreduce design
- Results
- **Conclusion and Future work**

Conclusion and Future Work

- SHARP runtime enables in-network offload with excellent bandwidth utilization
- Proposed designs overcome various bottlenecks by using a leader-based algorithm and streaming aggregation for large message reductions
  - Outperforms state-of-the-art by up to 86%
- Will be available in a future release of MVAPICH-plus
- Future work
  - Comprehensive application evaluation
  - Evaluating performance at larger scales
  - Exploring NUMA-awareness

THANK YOU!

Network-Based Computing Laboratory

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

The High-Performance MPI/PGAS Project
http://mvapich.cse.ohio-state.edu/

The High-Performance Big Data Project
http://hibd.cse.ohio-state.edu/

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/