MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

Accelerating MPI Message Matching and Reduction Collectives For Multi-/Many-core Architectures

Mohammadreza Bayatpour, Hari Subramoni, D. K. Panda

Department of Computer Science and Engineering
The Ohio State University

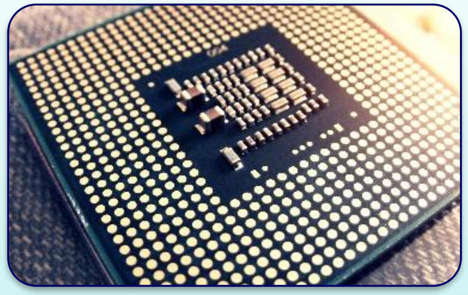
MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

Adaptive and Dynamic Design for MPI Tag Matching

M. Bayatpour, H. Subramoni, S. Chakraborty and D. K. Panda

Department of Computer Science and Engineering
The Ohio State University

Current Trends in HPC



Supercomputing systems scaling rapidly

- Multi- and Many-core architectures
- High-performance Interconnects

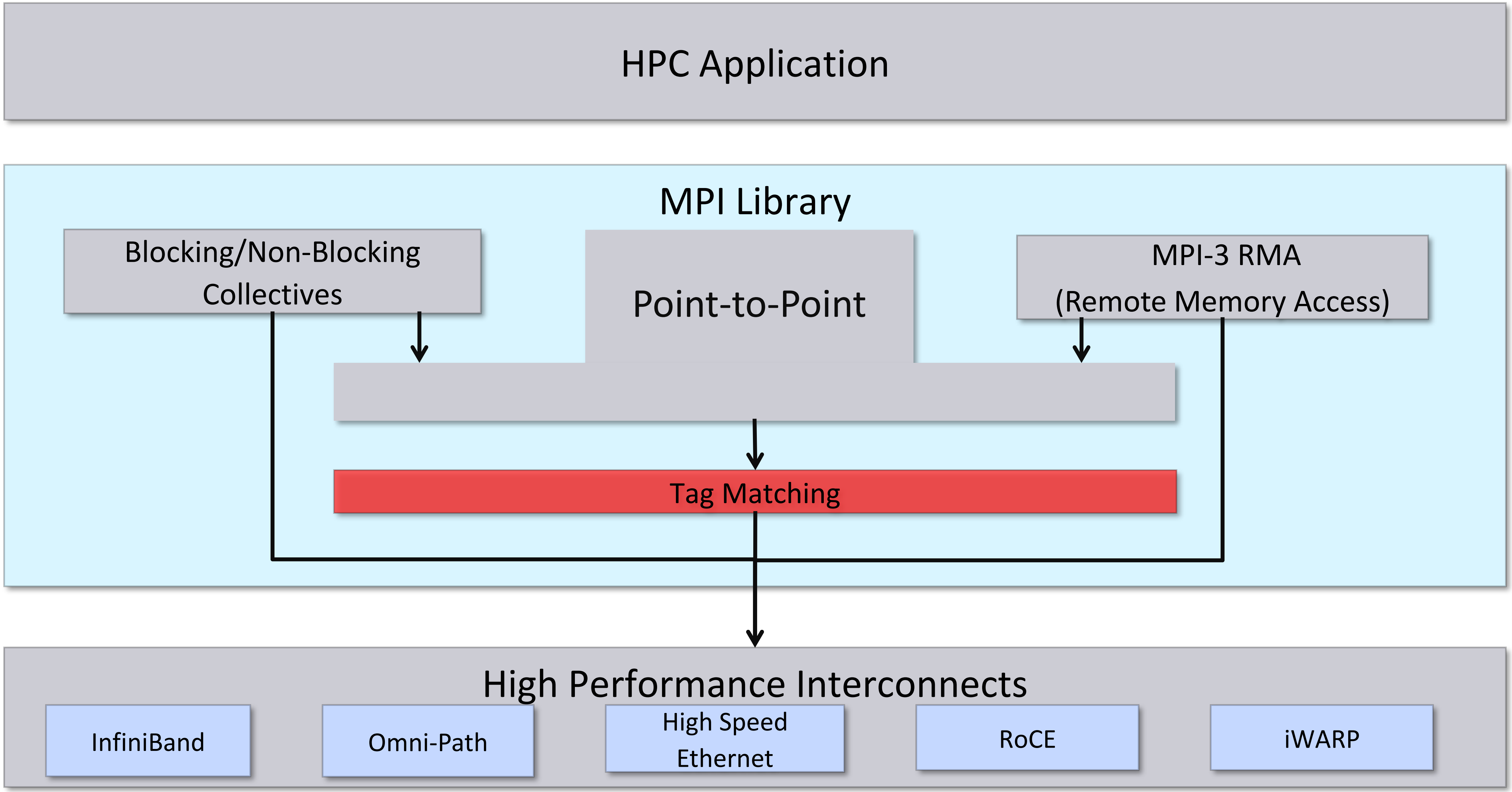
InfiniBand and Omni-Path are popular HPC Interconnects

- Low-latency and High-bandwidth
- 192 systems (39%) in Jun'17 Top500 use IB

MPI used by vast majority of HPC applications

- Helping applications scale to thousands of cores
- Large systems exposing new scalability issues

Components of an MPI Library



MPI Tag Matching 101

- On the receiver side, one needs to match the incoming message with the message that was posted by receiver
- Three parameters should match
  - Context id, Source Rank, Tag
  - Wildcards (MPI_ANY_SRC, MPI_ANY_TAG) introduce additional complexity
- Two kinds of the queues are involved in the receiver side
  - Posted queue
  - Unexpected queue

Search Time Analysis of the Default Double Linked List Design

- Most MPI libraries use double linked list for unexpected and posted queues
- Message to be removed could be in any position of the queue
  - Removal time in the best case is O(1) and in the average case is linear O(N)
- Tag matching is in the critical path for point-to-point based operations
- Number of the processes in a job is increasing
  - Future extreme-scale systems are expected to have millions of cores*
  - Multithreaded programming models
- All can push the search functions to go deeper in the lists
  - Impose significant overhead on the performance

* Thakur R, Balaji P, Buntinas D, Goodell D, Gropp W, Hoefler T, Kumar S, Lusk E, Träff JL. MPI at Exascale. Proceedings of SciDAC. 2010 Jul;2:14-35.

Proposed Adaptive Design

- Based on the Bin-based and default simple double linked list scheme
- Three phases
  - Starts with the default design
  - Observes the communication pattern for each process during the runtime
  - If all the conditions are held, it begins to convert the default scheme to the Bin-based scheme
- Each process can have its own scheme
  - Some may stay at the default scheme, some may need to convert to bin-based scheme

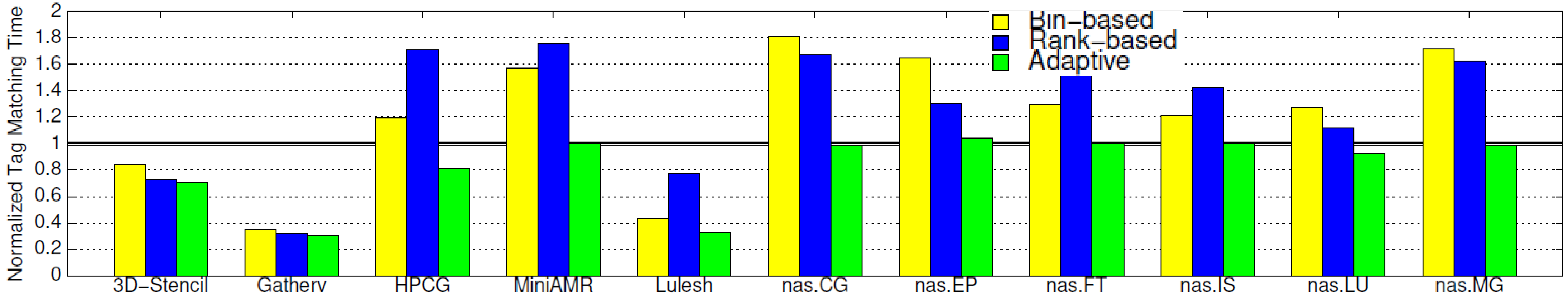
Proposed Adaptive Design (Cont'd)

- For each of the posted and unexpected queues, we consider the following thresholds
  - Number of the calls to the tag matching functions in the library (CALLS_NUM)
  - The average number of queue look-up attempts per CALLS_NUM (MACTCH_ATTMPS)
- Each process maintains both during the runtime
- If both thresholds are crossed
  - Adaptive design changes from the double linked list scheme to the bin-based scheme

Proposed Adaptive Design (Cont'd)

- Currently, conversion is one way from default to bin-based scheme and may occur only one time through the entire runtime
- These thresholds are fixed through entire runtime and they are configurable
  - We have tuned them based on empirical analysis using OSU micro benchmarks
- We consider two possible sizes for NUM_BINS
  - ¼ JOB_SIZE and ½ JOB_SIZE
  - Based on MATCH_ATTMPS, we decide which one to choose

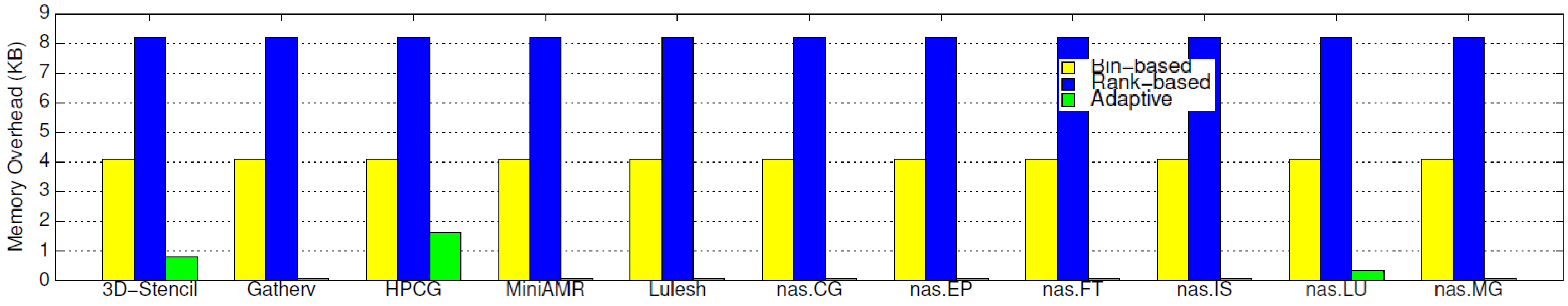
Summary of Tag Matching Performance



(b) Total Tag Matching Time, Normalized to Default (Lower is Better)

- Comparison of different designs/benchmarks at 512 processes on RI
- Adaptive design shows the best performance

Summary of Memory Consumed for Tag Matching



(a) Memory Overhead per Process, Compared to Default (Lower is Better)

- Comparison of different designs/ benchmarks at 512 processes on RI with default design
- Adaptive design shows minimal memory overhead

MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

Scalable Reduction Collectives with Data Partitioning-based Multi-Leader Design

M. Bayatpour, S. Chakraborty , H. Subramoni, X. Lu, and D. K. Panda

Department of Computer Science and Engineering
The Ohio State University
Presented at Supercomputing 2017

MPI Reduction Collectives 101

- Convenient abstraction to implement group communication operations
- Widely used across various scientific domains
  - Owing to their ease of use and performance portability
- One of the most popular collective operations: MPI_Allreduce
  - 37% of communication time
- MPI_Allreduce reduces values from all processes and distribute the result back to all processes

Existing Designs for MPI_Allreduce

- Hierarchical strategy
  - A two-level approach
    - Intra-node reduction by root + inter-node Allreduce
    - Computations are done by the root process of each node

- Tree-based strategies
  - Recursive Doubling
    - Based on point-to-point operation
    - High parallelism for computation
      - All the process are involved in computation
    - Pairs distance doubles after each step
    - Log (P*) steps

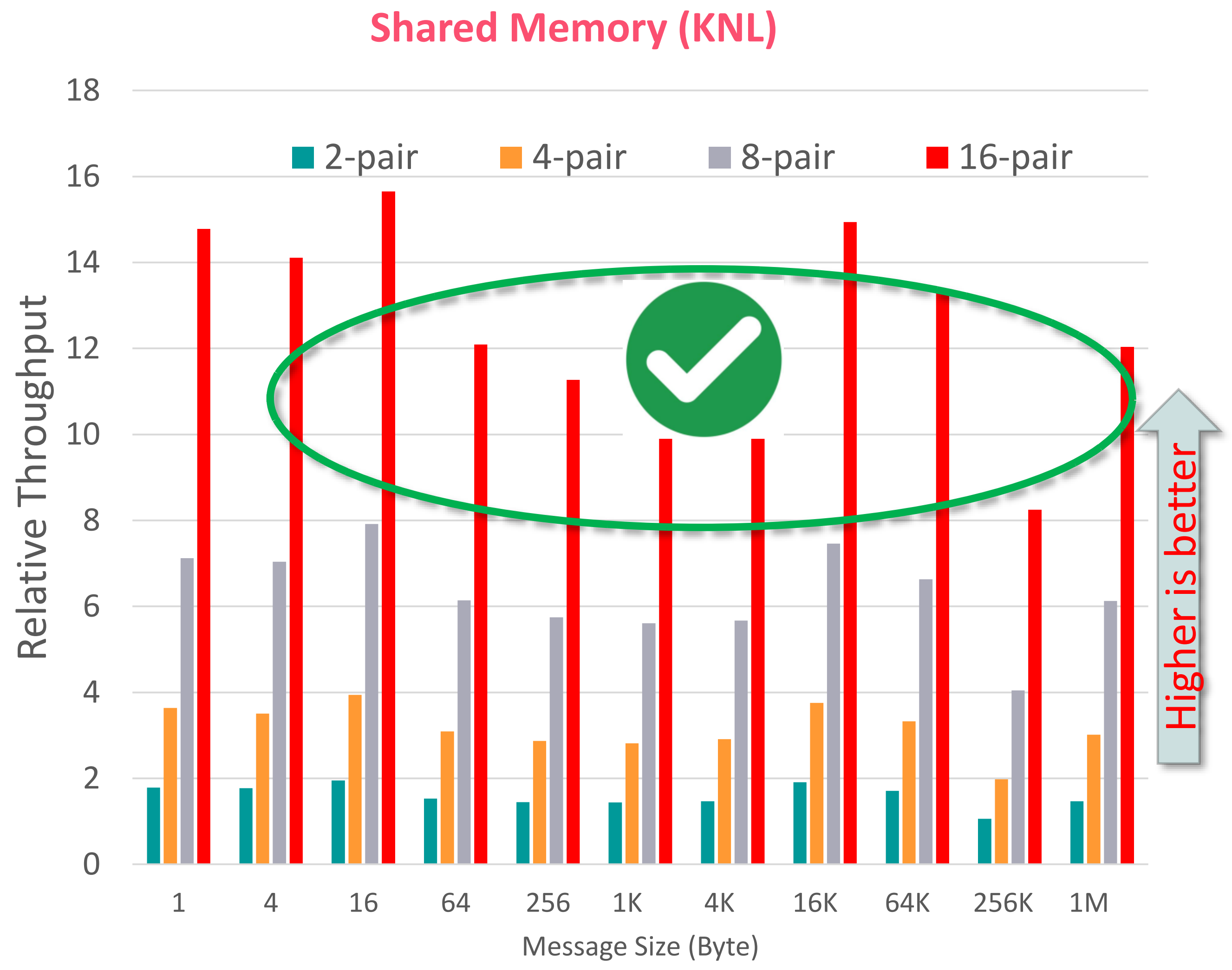
* Bloch et al. Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction

Relative Throughput of Different Architectures

- Using OSU Micro benchmark suite*
- “Multiple Bandwidth Test”
  - Back-to-back messages
    - Sent to a pair before waiting for receive
- Evaluates the aggregate unidirectional bandwidth between multiple pairs of processes
- 1) Xeon + IB, 2)Xeon + Omni-Path, and 3) KNL + Omni-Path

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

Communication Characteristics of Modern Architectures: Intra-node Communication

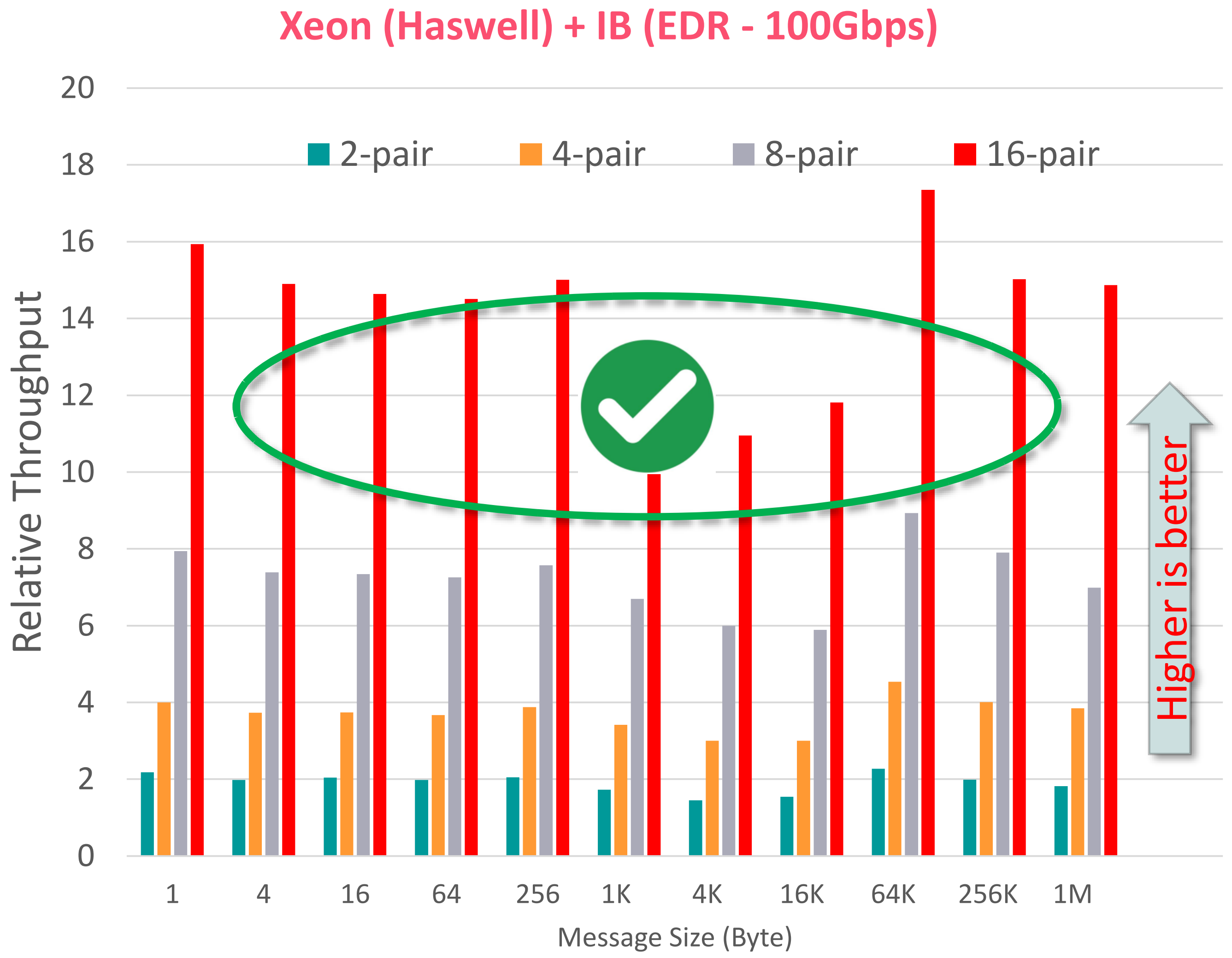


Multiple pair test vs. one pair test

- The relative throughput very close to the number of pairs

- Support many concurrent intra-node communication

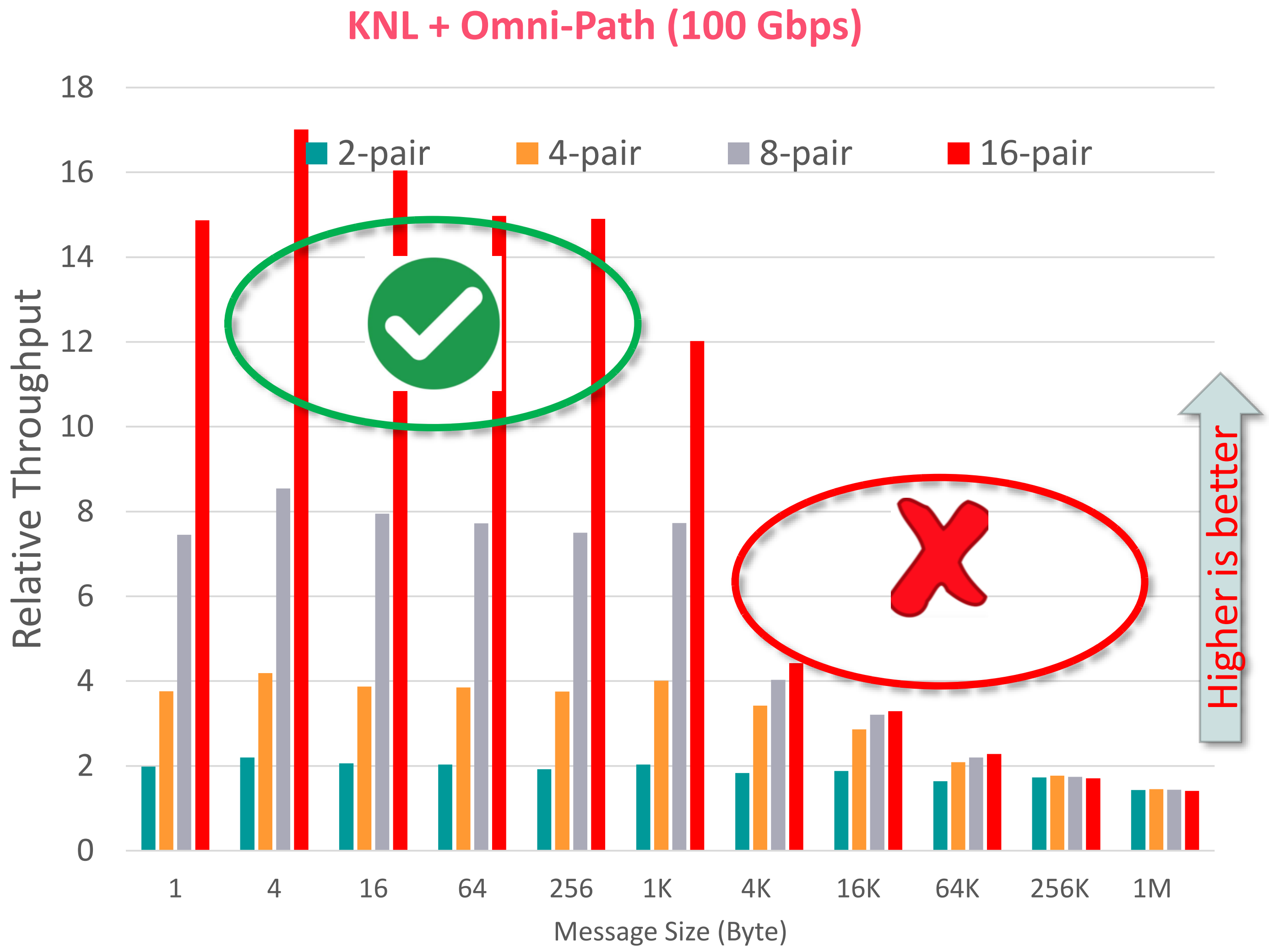
Communication Characteristics of Modern Architectures: InfiniBand Interconnect



Multiple pair test vs. one pair test

- The relative throughput close to the number of communicating processes per node
- Support many concurrent intra-node communication

Communication Characteristics of Modern Architectures: Omni-Path Interconnect



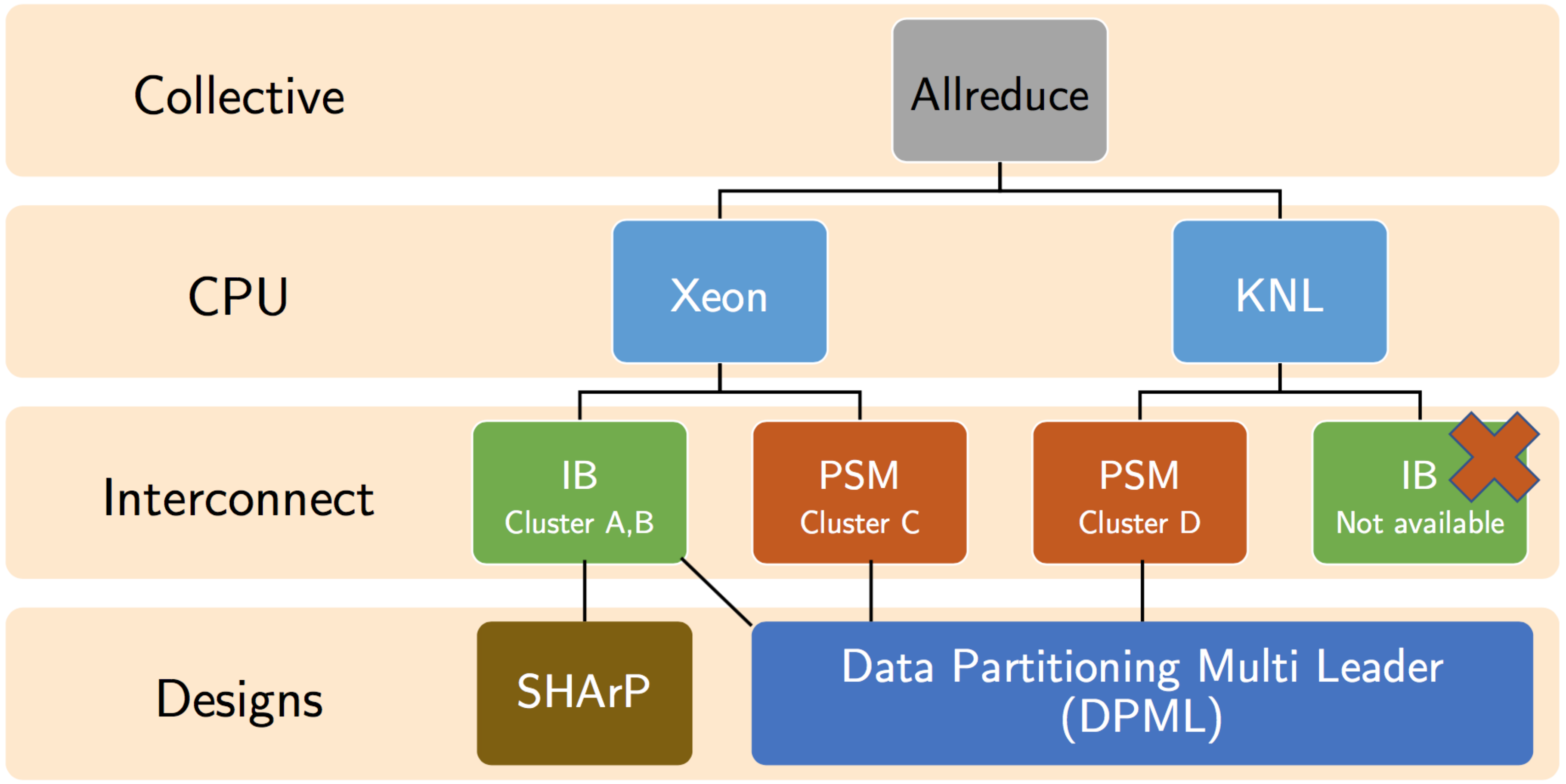
Multiple pair test vs. one pair test

- The relative throughput of one for large messages
- Supports many concurrent communications for small and medium message range
- Similar behavior observed for Xeon + Omni-Path

Performance limitations of Existing Designs for MPI_Allreduce

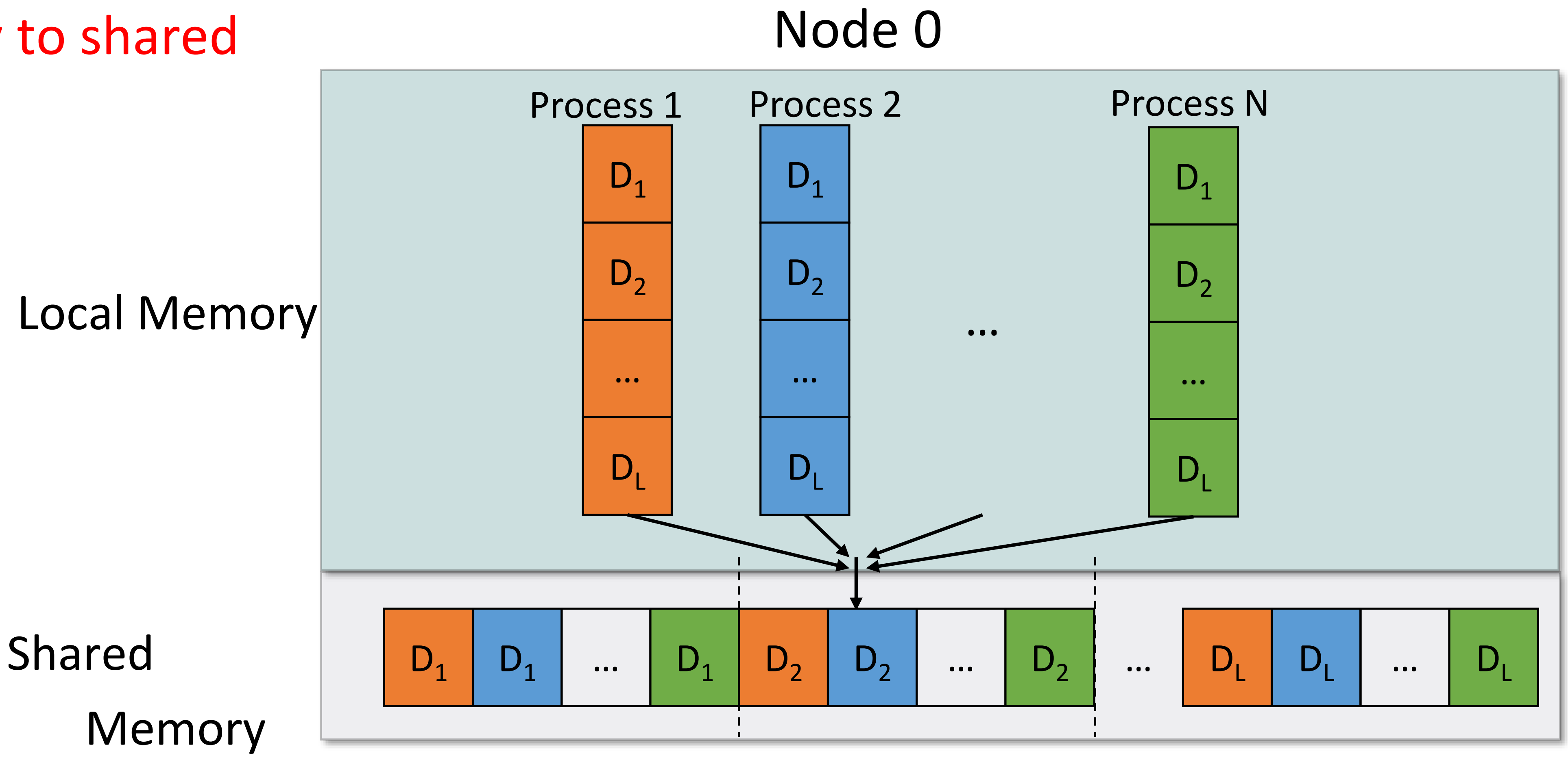
- Does not take advantage of large number of cores and high concurrency in communication
- Does not take advantage of shared memory collectives
  - Needs kernel support for zero-copy communication for large messages in same node
- Too many inter-node communication for large PPNs
- Limited performance due to extra QPI transfers
- Limited computing power of switches limits its performance for medium and large message ranges

Design Outline



DPML Design Phases

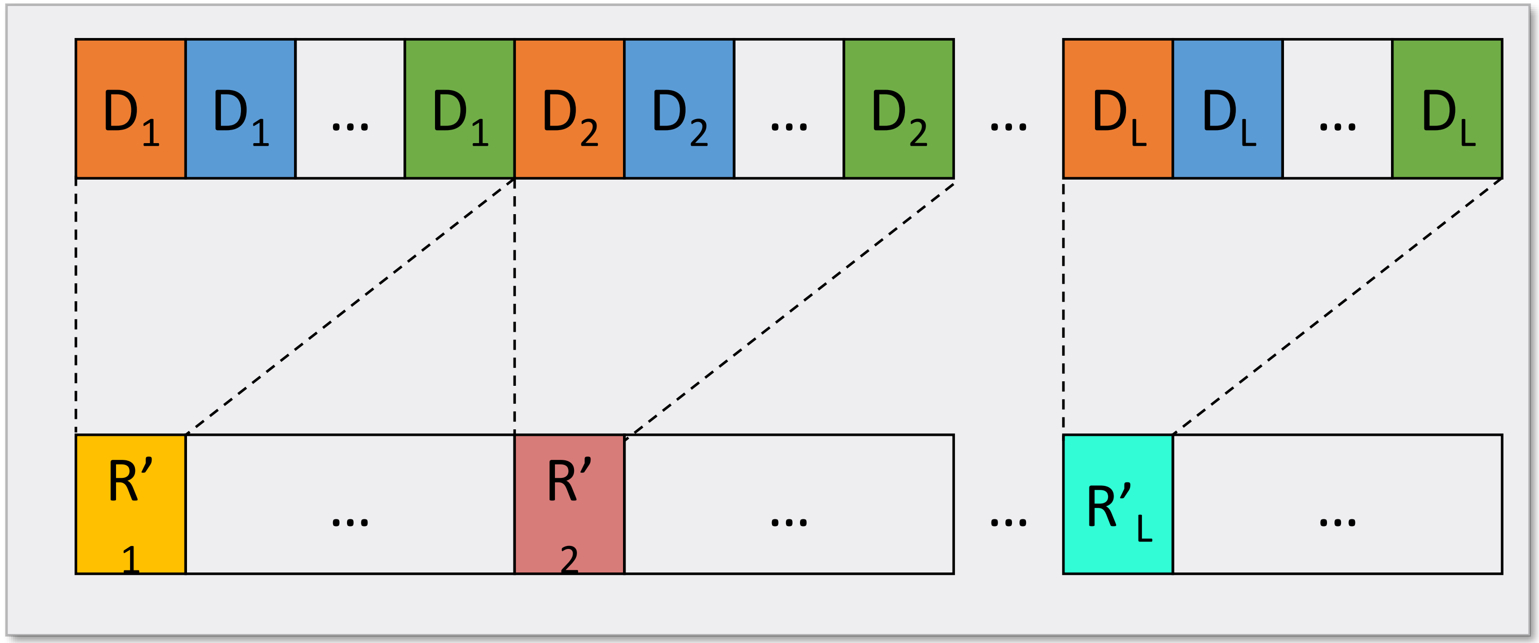
- Phase 1: Copy to shared Memory



DPML Design Phases

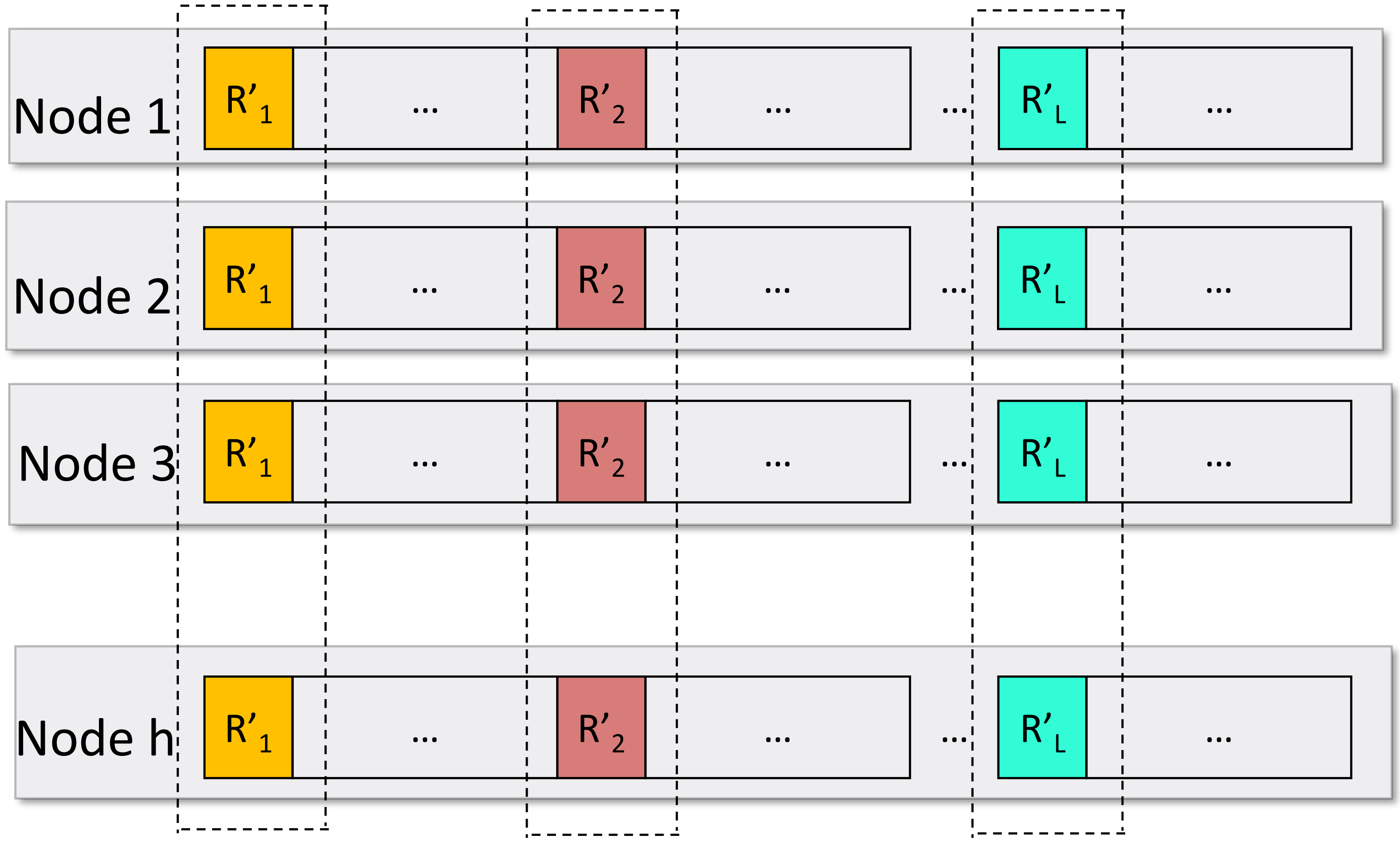
- Phase 1: Copy to shared Memory
- Phase 2: Parallel Intra-node reduction by the leaders

Shared Memory



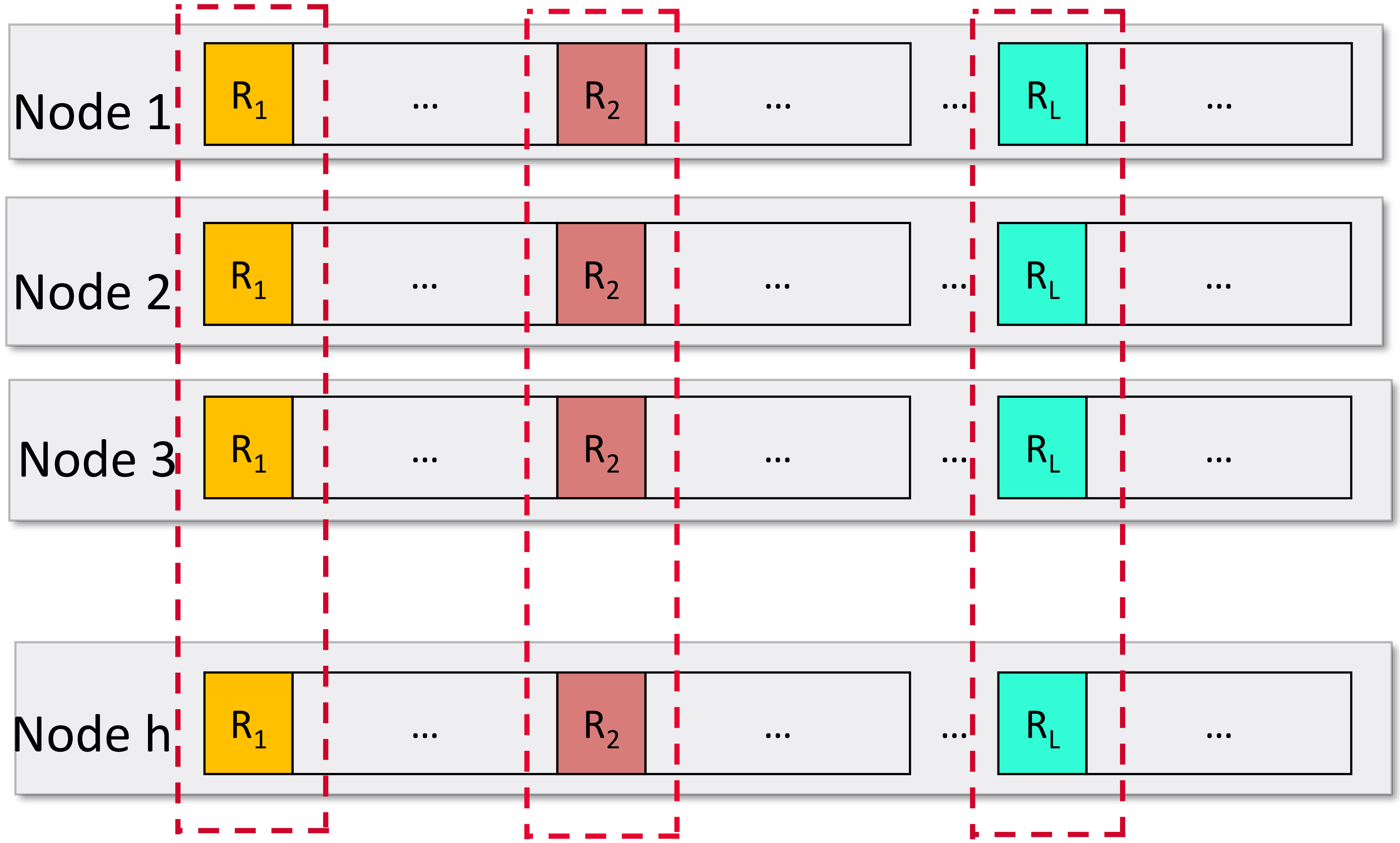
DPML Design Phases

- Phase 1: Copy to shared Memory
- Phase 2: Parallel Intra-node reduction by the leaders



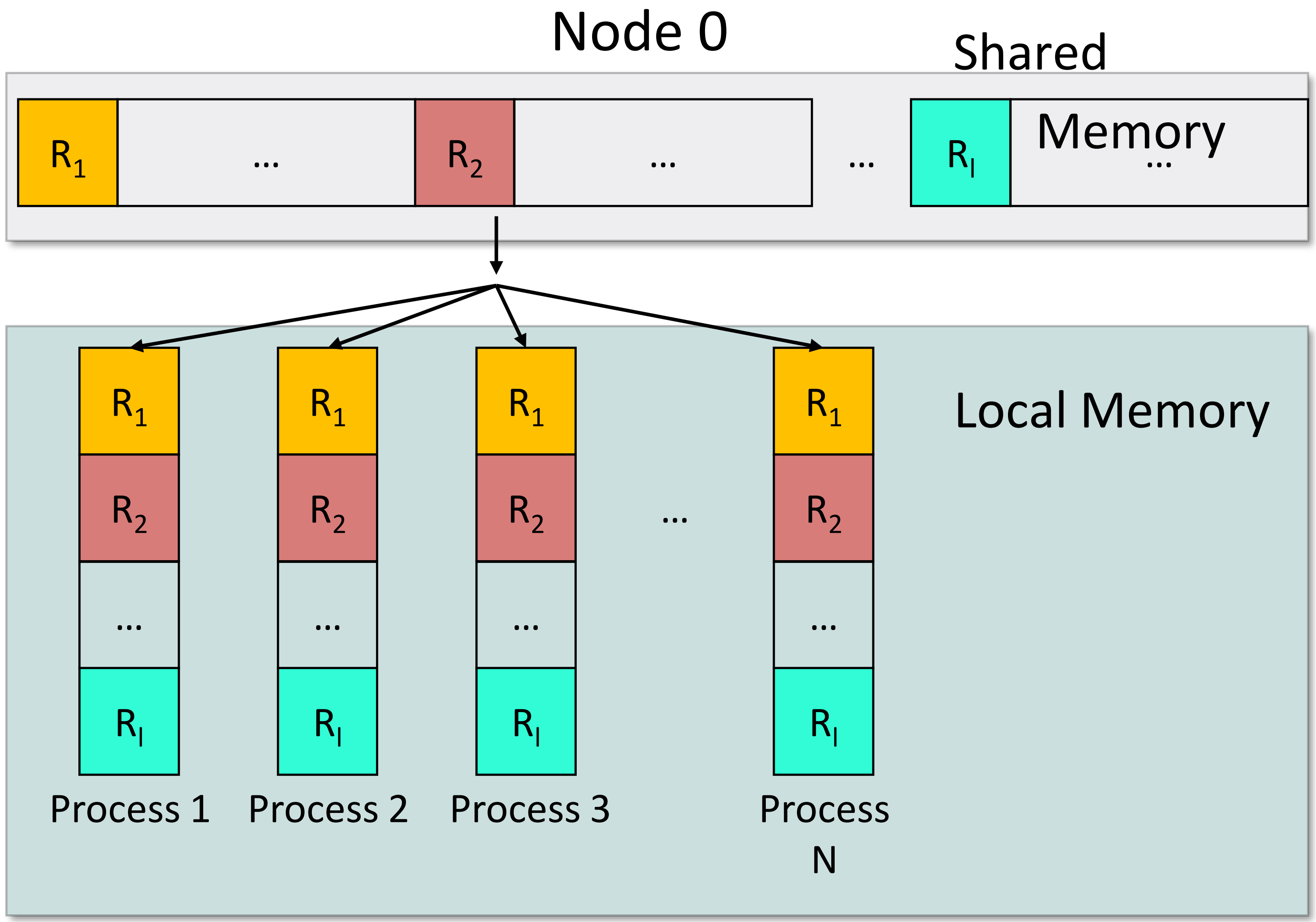
DPML Design Phases

- Phase 1: Copy to shared Memory
- Phase 2: Parallel Intra-node reduction by the leaders
- Phase 3: Parallel Inter-node Allreduce by the leaders with same index

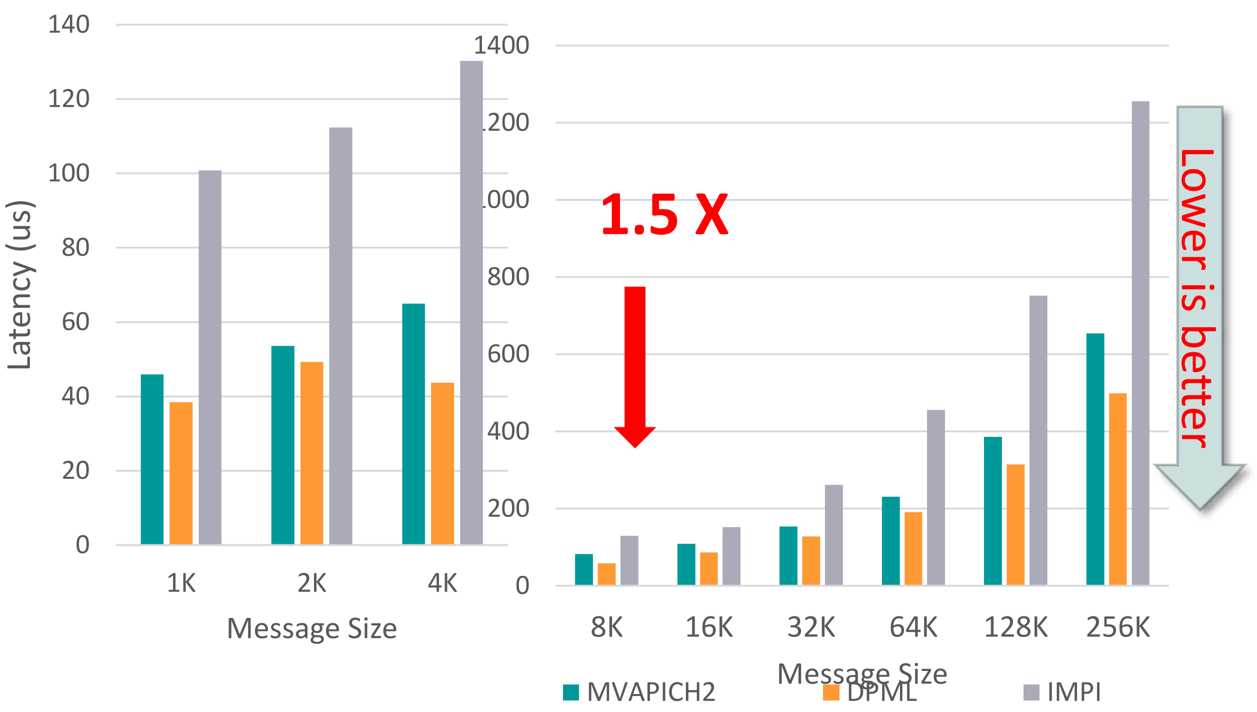


DPML Design Phases

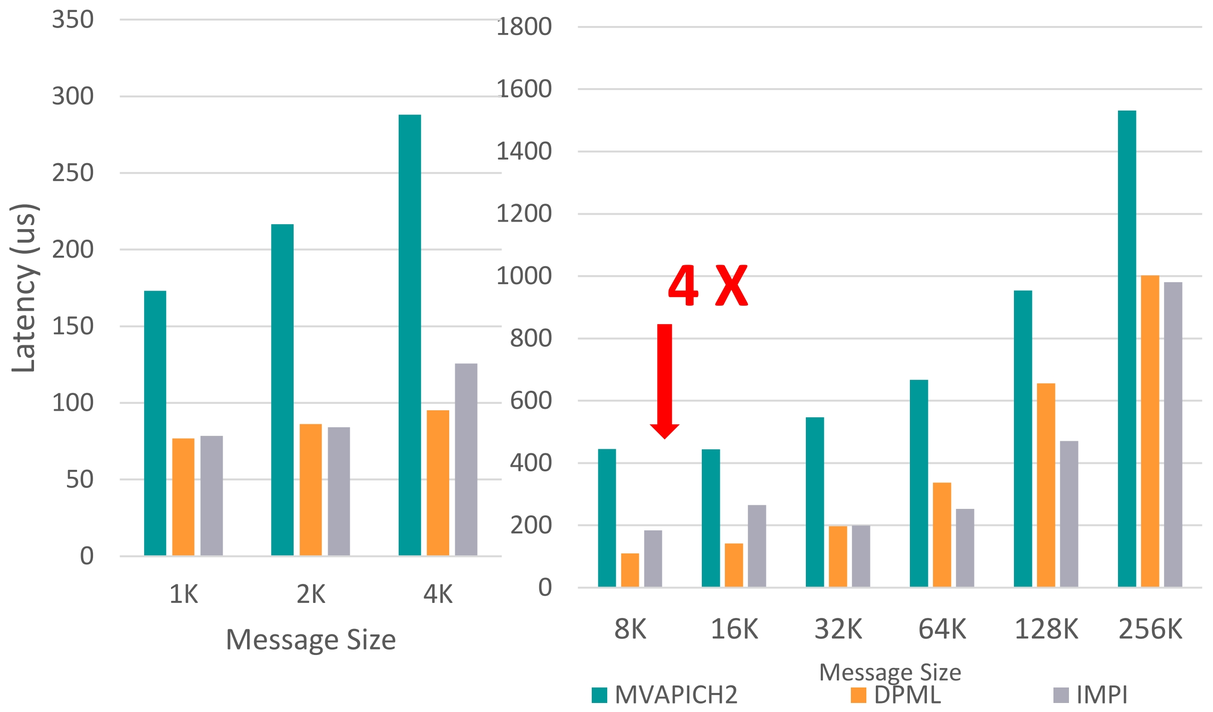
- Phase 1: Copy to shared Memory
- Phase 2: Parallel Intra-node reduction by the leaders
- Phase 3: Parallel Inter-node Allreduce by the leaders with same index
- Phase 4: Parallel distribution of Allreduce results to local buffers



Performance of MPI_Allreduce On Omni-Path



XEON + Omni-Path (64 Nodes, 28 PPN*)

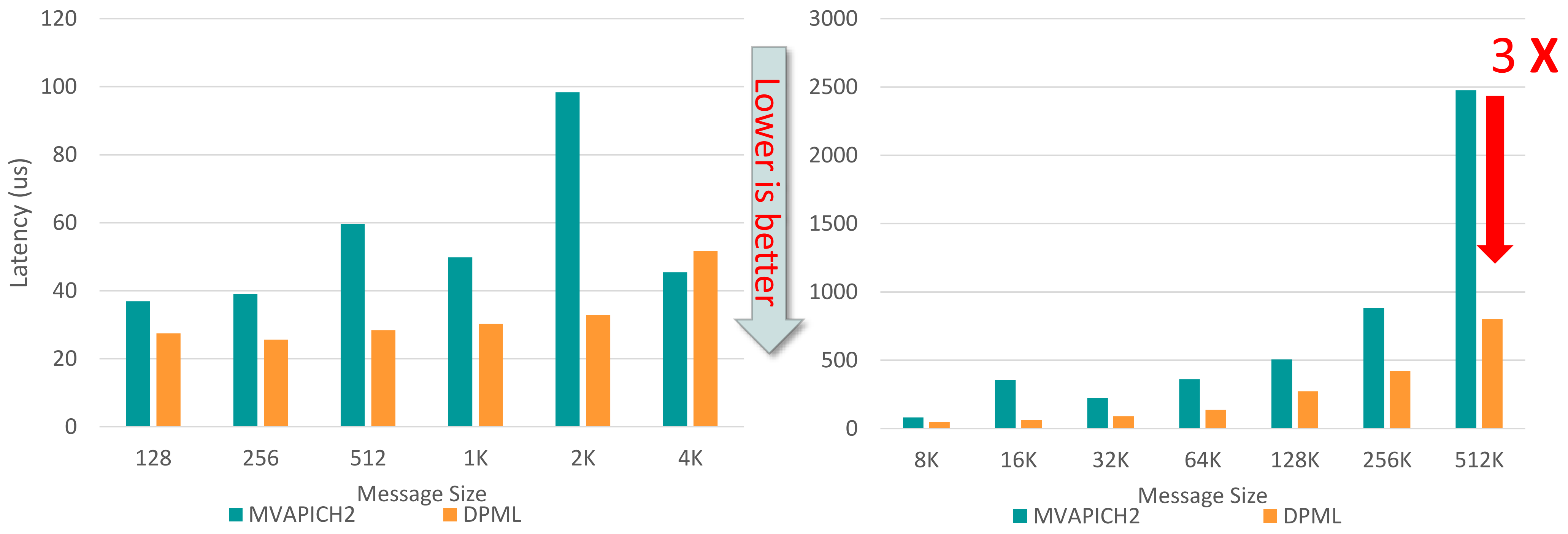


KNL + Omni-Path (32 Nodes, 32 PPN)

- DPML always outperform MVAPICH2 for all medium and large message range
- DPML outperform IMPI in medium message range
- High parallelism of DPML benefits KNL more than XEON

*Processes Per Node

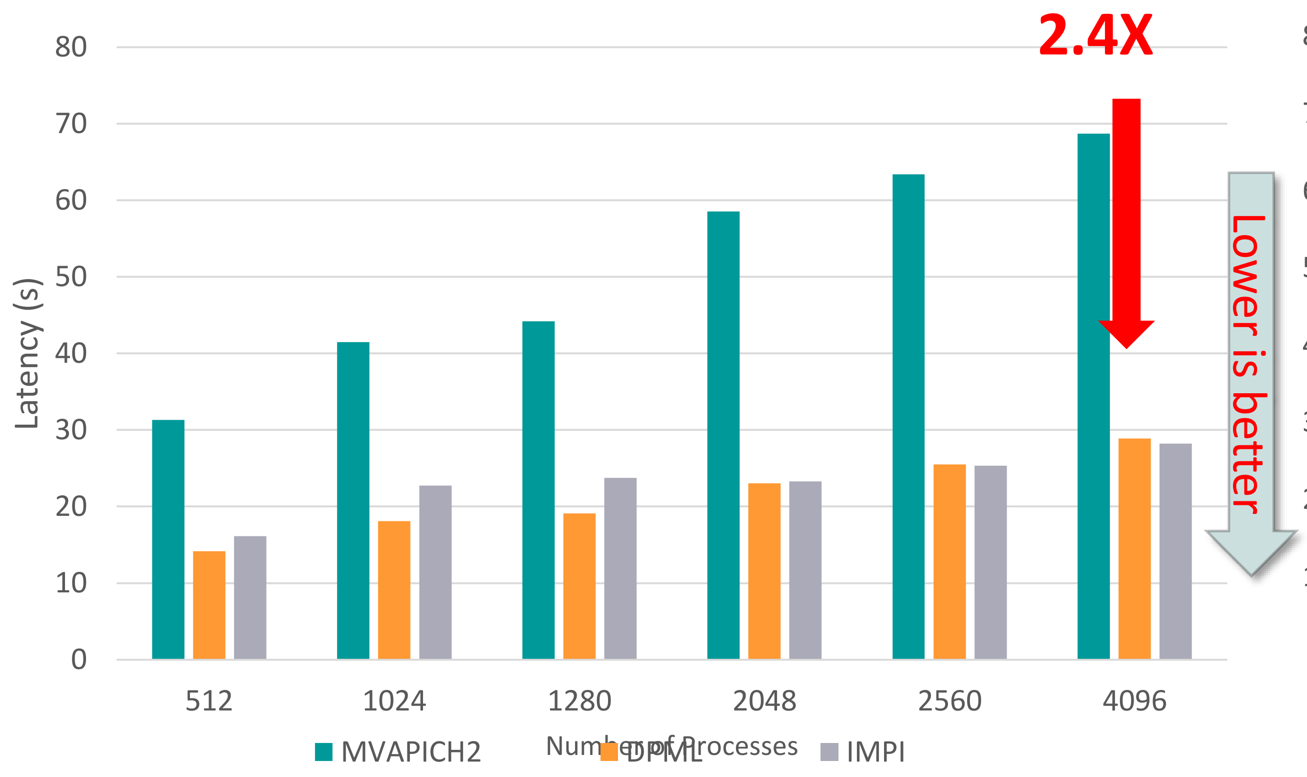
Performance of MPI_Allreduce On InfiniBand



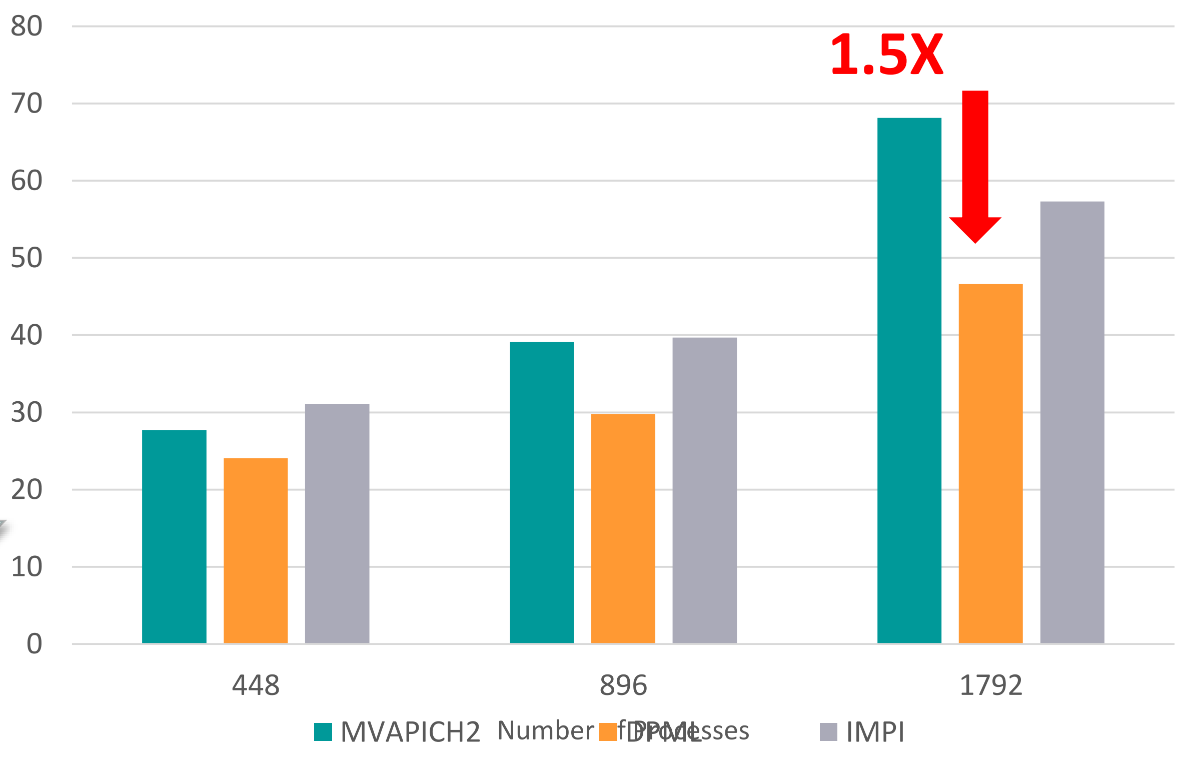
XEON + IB (64 Nodes, 28 PPN)

- DPML outperform MVAPICH2 for most of the medium and large message range
  - With 512K bytes, 3X improvement of DPML
- Higher benefits of DPML as the message size increases

Performance Benefits for MiniAMR Application



KNL + Omni-Path (32 PPN)



XEON + Omni-Path (28 PPN)

- For MiniAMR Application with 4096 processes, DPML can reduce the latency by 2.4X on KNL + Omni-Path cluster
- On XEON + Omni-Path, with 1792 processes, DPML can reduce the latency by 1.5X

SALaR: Scalable and Adaptive Designs for Large Message Reduction Collectives

M. Bayatpour, J. Hashmi,
S. Chakraborty, H. Subramoni, P. Kousha, and D. K. Panda

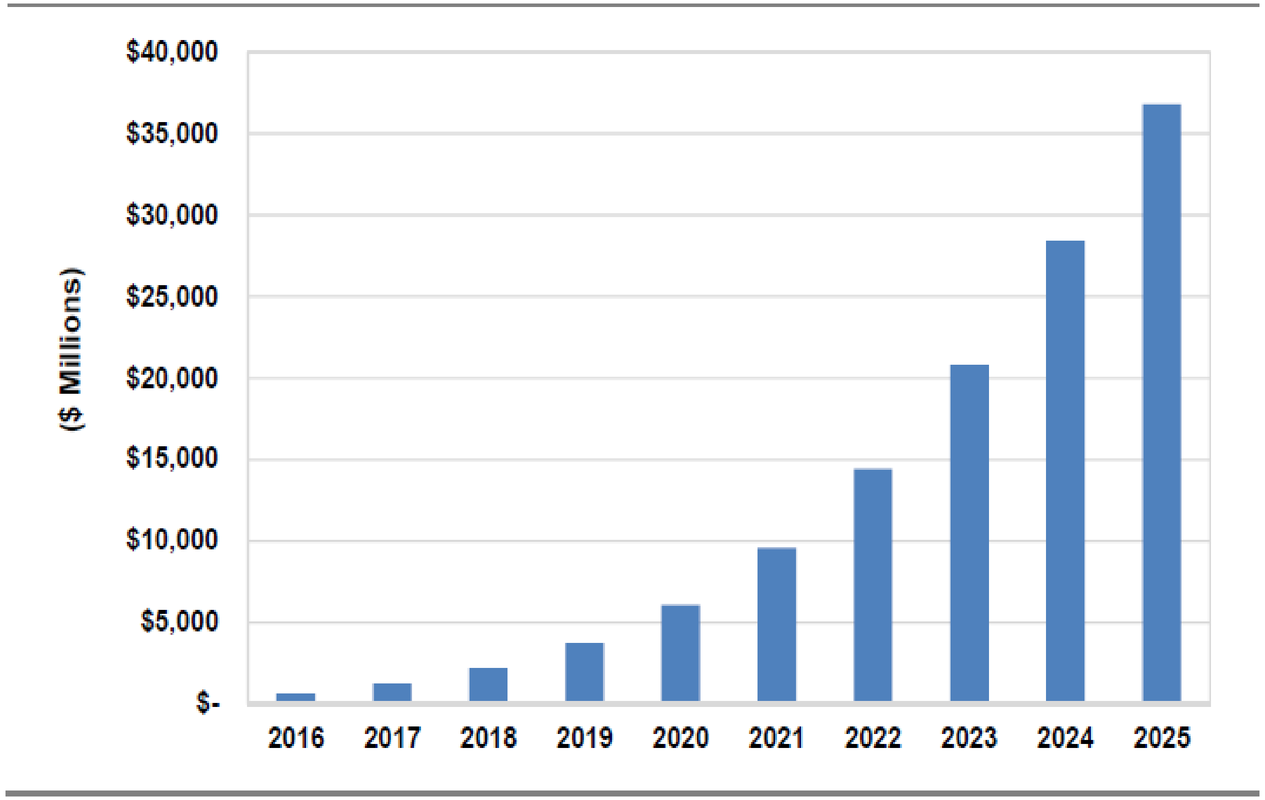
{bayatpour.1, hashmi.29, chakraborty.52, subramoni.1, kousha.2, panda.2}
@osu.edu

Department of Computer Science and Engineering
The Ohio State University
Presented at IEEE Cluster 2018

Deep Learning (DL) Frameworks and Trends

- Renewed interest in DL
  - Deep Neural Networks (DNNs)
- Tensorflow, CNTK and many more
- Excellent accuracy for deep/convolutional neural networks
- Diverse applications – Image Recognition, Cancer Detection, Self-Driving Cars, Speech Processing etc.

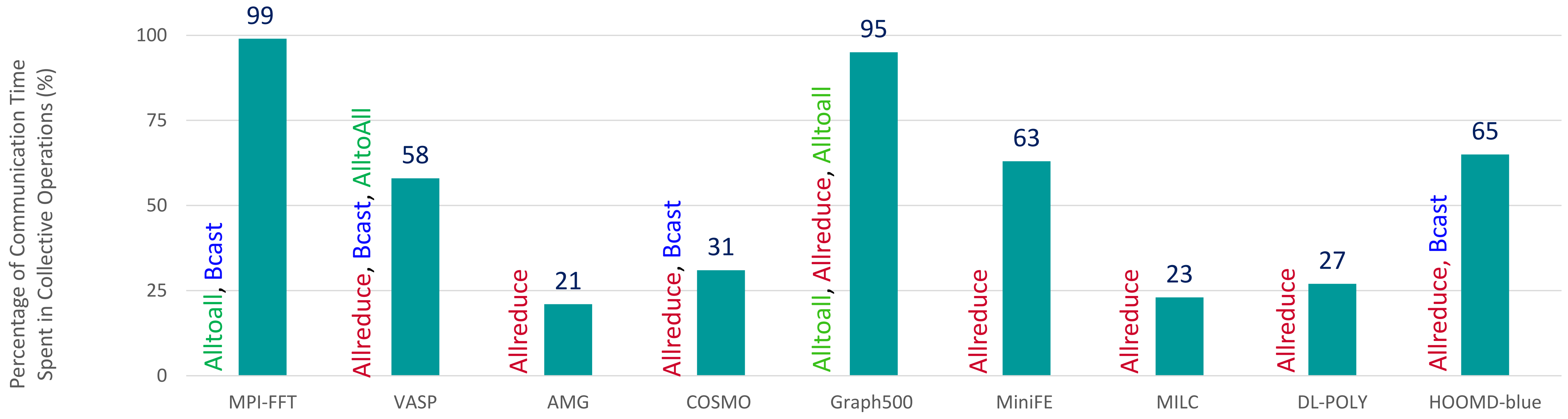
Chart 1.1 Artificial Intelligence Revenue, World Markets: 2016-2025



(Source: Tractica)

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

Why Collective Communication Matters?

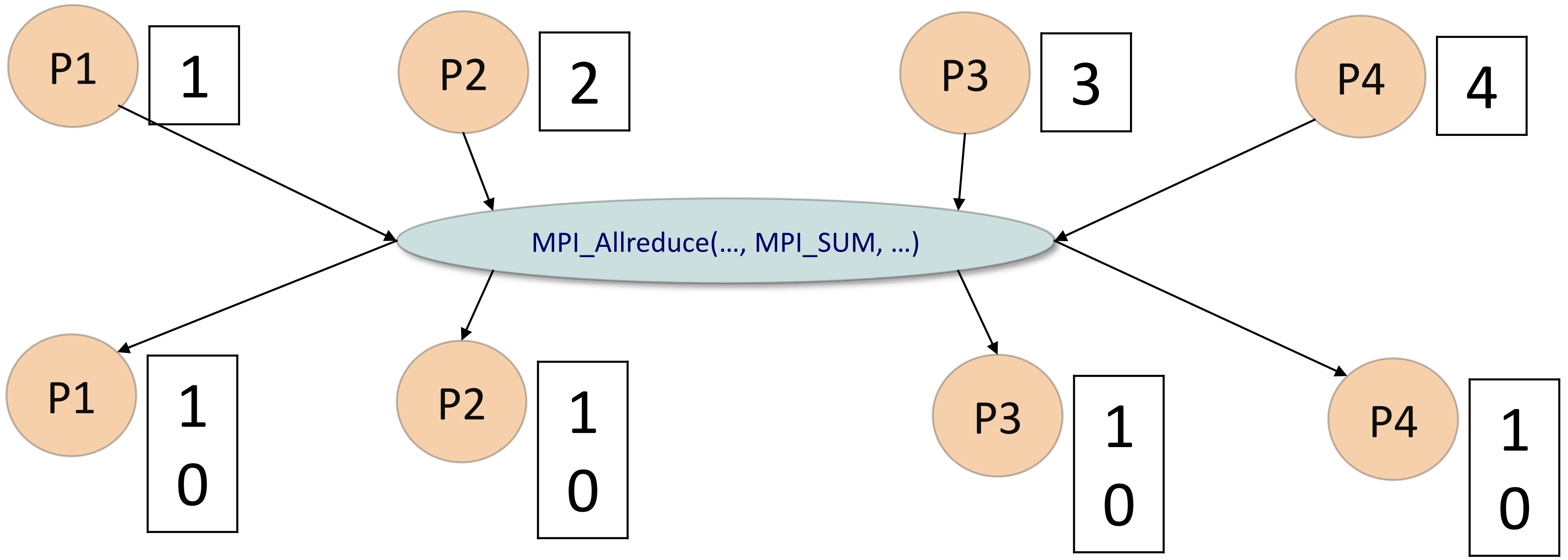


- Convenient abstraction to implement group communication
- Most application profiles showed majority of time spent in collective operations
- Optimizing collective communication directly impacts scientific applications leading to accelerated scientific discovery

http://www.hpcadvisorycouncil.com

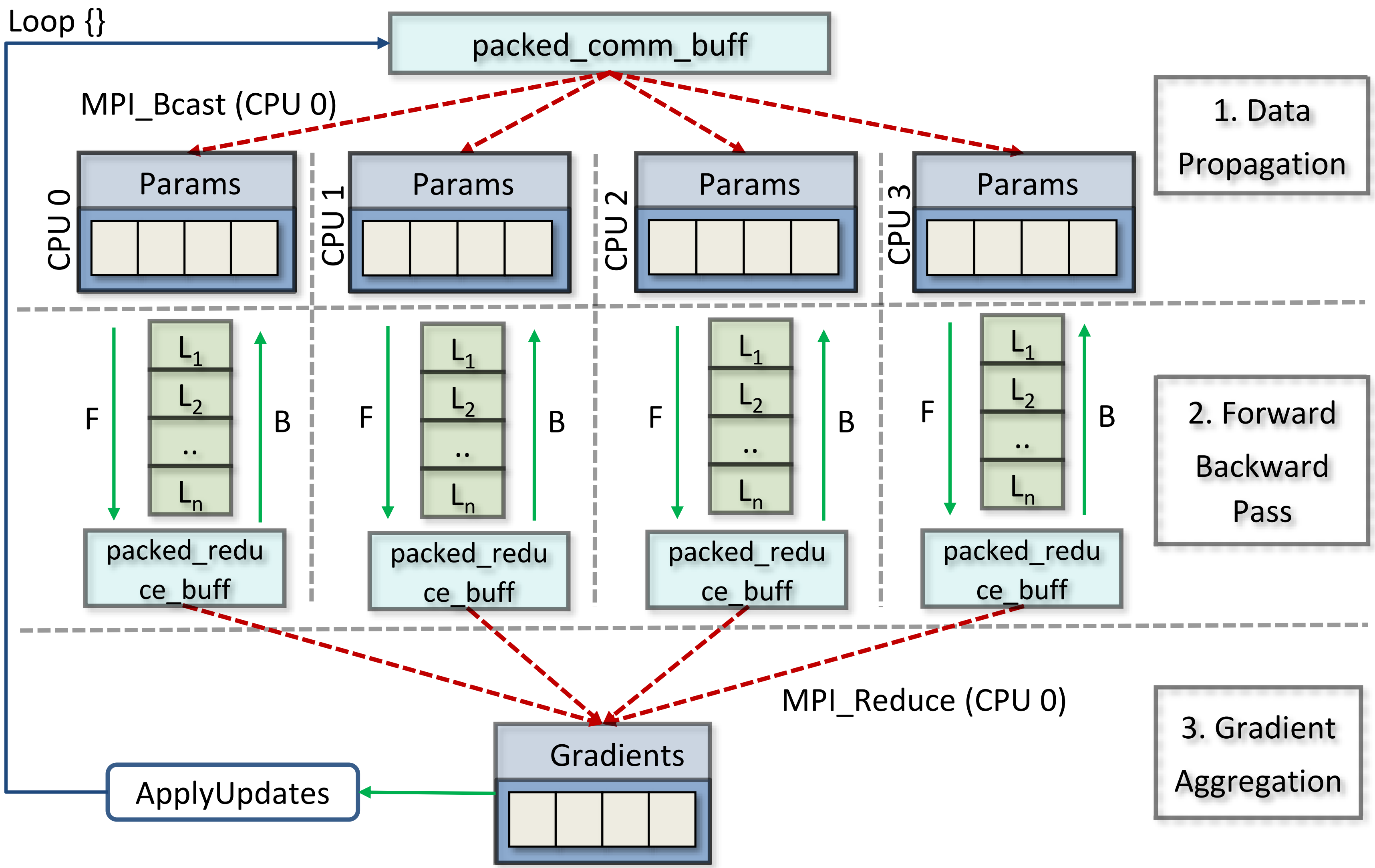
MPI Allreduce Collective

- MPI_Allreduce – Walkthrough Example



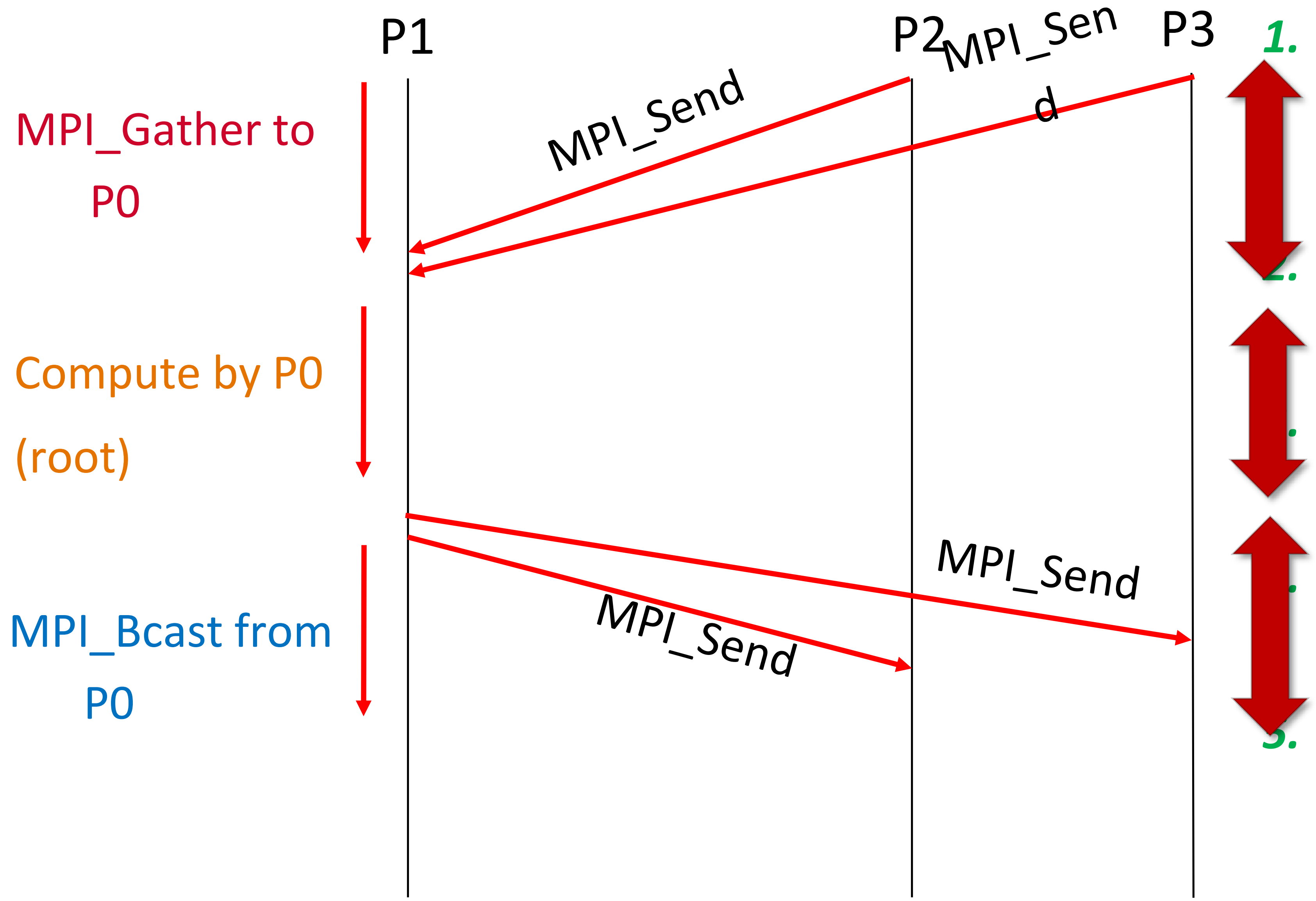
MPI Collectives used in Deep Learning

- **MPI_Bcast** – required for DNN parameter exchange
- **MPI_Reduce** – needed for gradient accumulation from multiple solvers
- **MPI_Allreduce** –Reduce followed by a Broadcast can be realized as one Allreduce
- ***Allreduce is the major collective operation in Deep Learning***



A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (PPoPP '17)

Naive Allreduce Design



Initiation overhead (Copy, ..)

All Computation is done by P0 and others are idle

P0 outgoing link is getting oversubscribed

1. *Load-balancing the computation and network resources*
2. *Overlap of communication and computation*
3. *Avoiding data staging and data copies*
4. *Avoiding unnecessary synchronization*
5. *Heuristic based adaptive design*

Performance limitations of Existing Designs for MPI_Allreduce

1. Load-balancing the computation and network resources
2. Overlap of communication and computation
3. Avoiding data copies and data staging
4. Avoiding the unnecessary synchronization overheads
5. Heuristic based adaptive design

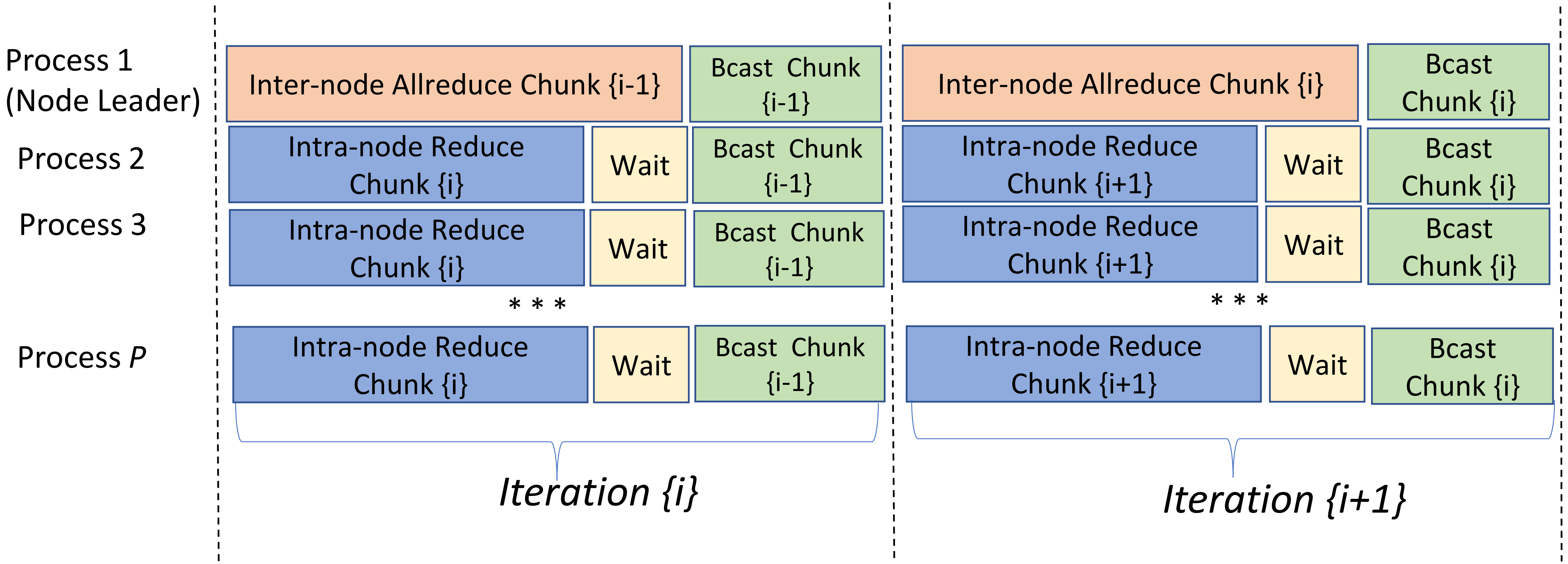
State-of-the-art Allreduce Designs	Feature being used				
	1	2	3	4	5
Baidu-Allreduce [a]	✓	✓	✗	✗	✗
Linear Pipelining [b]	✓	✓	✗	✗	✗
Reduce-scatter followed by Allgather [c,d]	✓	✗	✗	✗	✗
Segmented Ring [e]	✓	✓	✗	✗	✗
XPMEM-based Reduction [f]	✗	✗	✓	✗	✗
Proposed "**SALaR**"	✓	✓	✓	✓	✓

Research Contribution

- Designing high-performance Allreduce
  - Pipelined design for efficient overlap of computation and communication
  - Exploiting process Shared Address Space based truly zero-copy intra-node reduction
  - One-sided inter-node communication to reduce synchronizations
  - Efficient load-balanced inter-node communication
  - Heuristic based adaptive design
- Modeling the proposed design
- Improved the AlexNet training time on CNTK by up to 46%
- Reduced the latency of osu_allreduce by up to 5X at scale

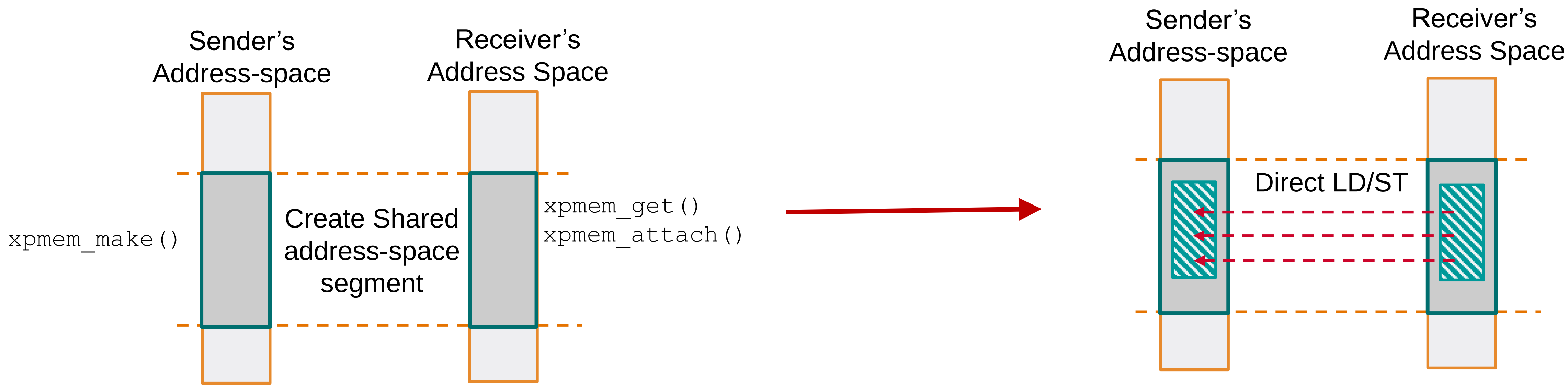
Proposed SALaR Design

- The input vector is splitted into smaller chunks
- At each iteration, the inter-node Allreduce operation of give chunk is overlapped with the intra-node reduction of successive chunk
- Timeline of the Processes in Node 0



Background: Shared Address-space based Communication

- XPMEM (https://github.com/hjelmn/xpmem) --- "Cross-partition Memory"
  - Mechanisms for a process to "*attach*" to the virtual memory segment of a remote process
  - Consists of a user-space API and a kernel module
- The receiver process can directly read/write on the remote process' memory

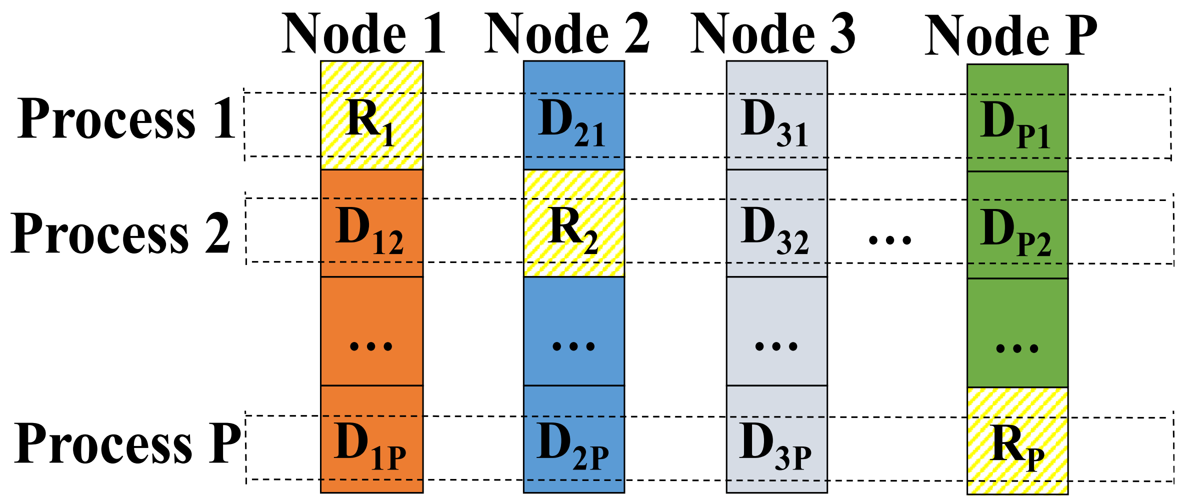


Proposed Inter-node Allreduce Design

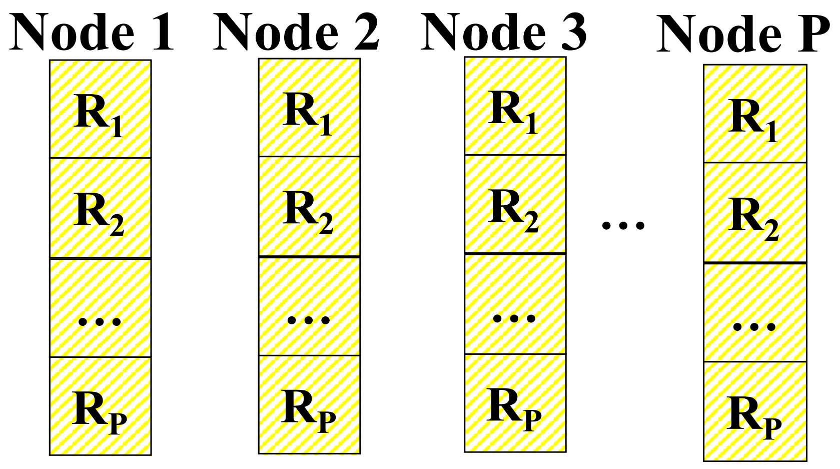
- An efficient one-sided based Allreduce design
  - Performing local reduction during Allreduce reduces the availability of the receiver to respond handshakes quickly
  - Avoids the unnecessary synchronization between the leaders
- Phase-1 (Setup Phase)
  - Buffer registration and RDMA key/address exchange
  - By taking advantage of registration cache, overhead of step-1 is visible only for the first touch to a buffer

Proposed Inter-node Allreduce Design (cont'd)

- Phase-2
  - The input vector (Chunk[i]) is divided into P chunks
    - (P = inter-node job size)
  - Each process is responsible for calculating the Allreduce results of its corresponding chunk
- Phase-3
  - Allgather all the chunks to get the final full results



Phase-2

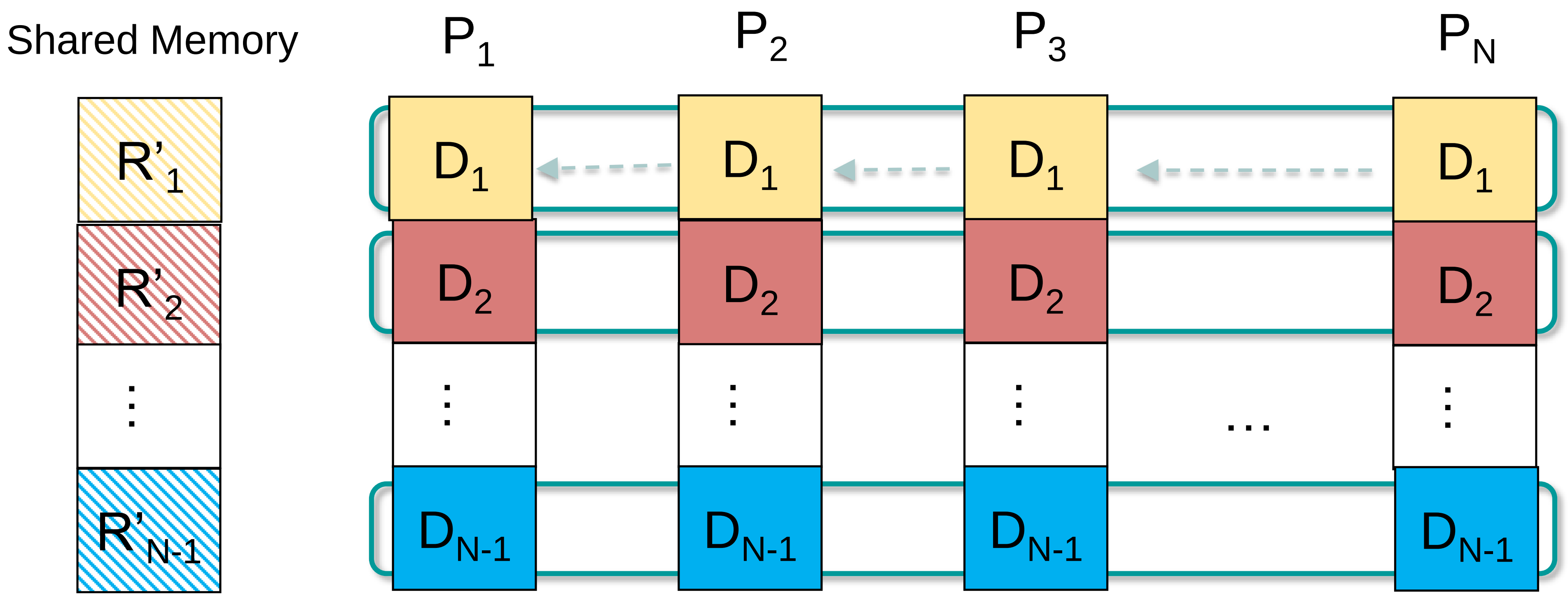


Phase-3

Outline

- Introduction
- Motivation
- Contributions
- Proposed Designs
  - Design Optimizations
  - Modeling
- Experimental Results
- Conclusions & Future Work

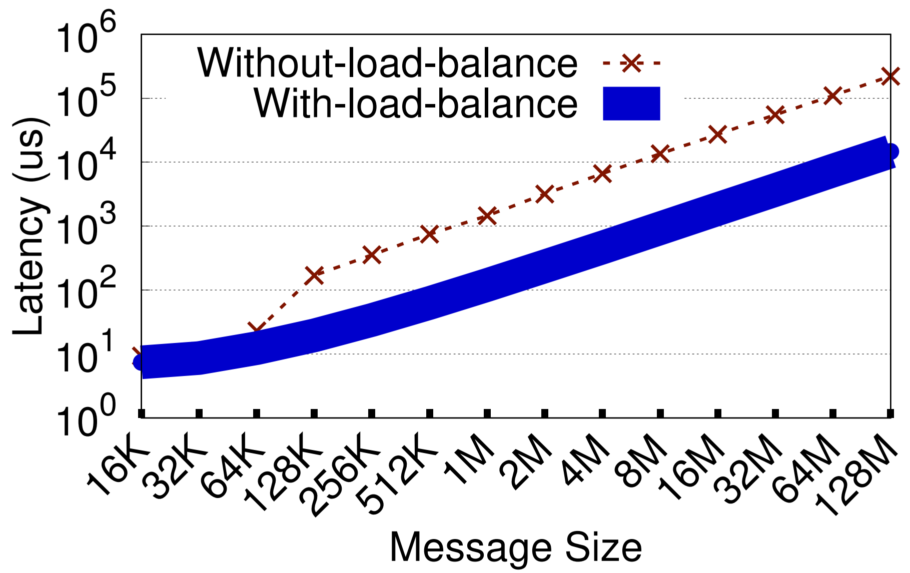
Intra-node XPMEM-based Reduce Design



- Using XPMEM to avoid the extra copy overhead
- There are N processes per node
- Only Non-leader processes (P2, P3, …) are computing the intra-node
    - P1 is busy with inter-node operation

Load-balancing the InfiniBand Links

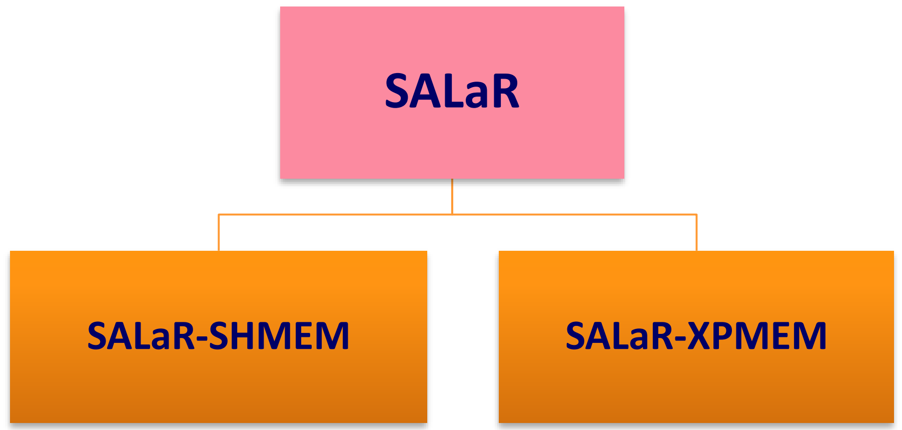
- To prevent the link to be choked, no two processes should access same remote buffer at the same time
- We use a cyclic pattern to orchestrate the data-transfer
- Process k, in the *i* th iteration, accesses the remote buffer of process (k+i)%P (P=communicator size)



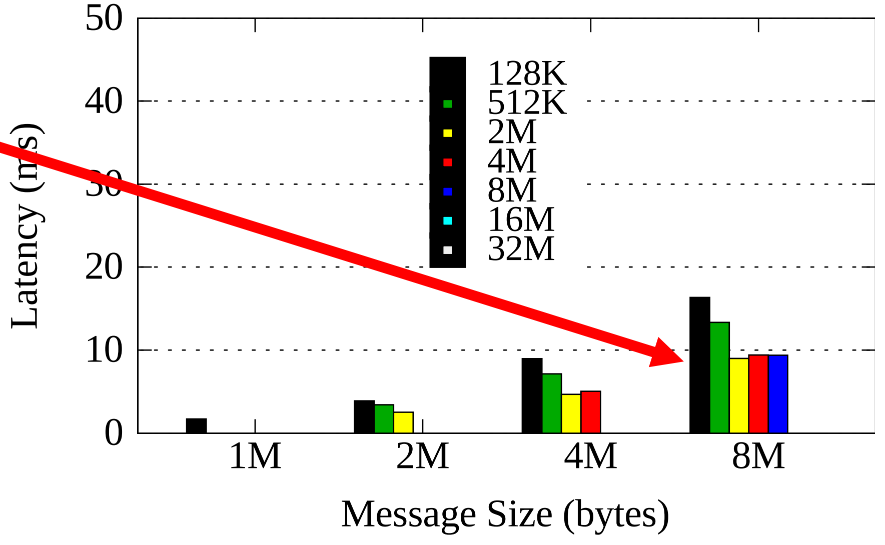
Impact of load-balancing, latency of a link during SALaR-Inter, 16 nodes

Summary of Proposed SALaR Designs

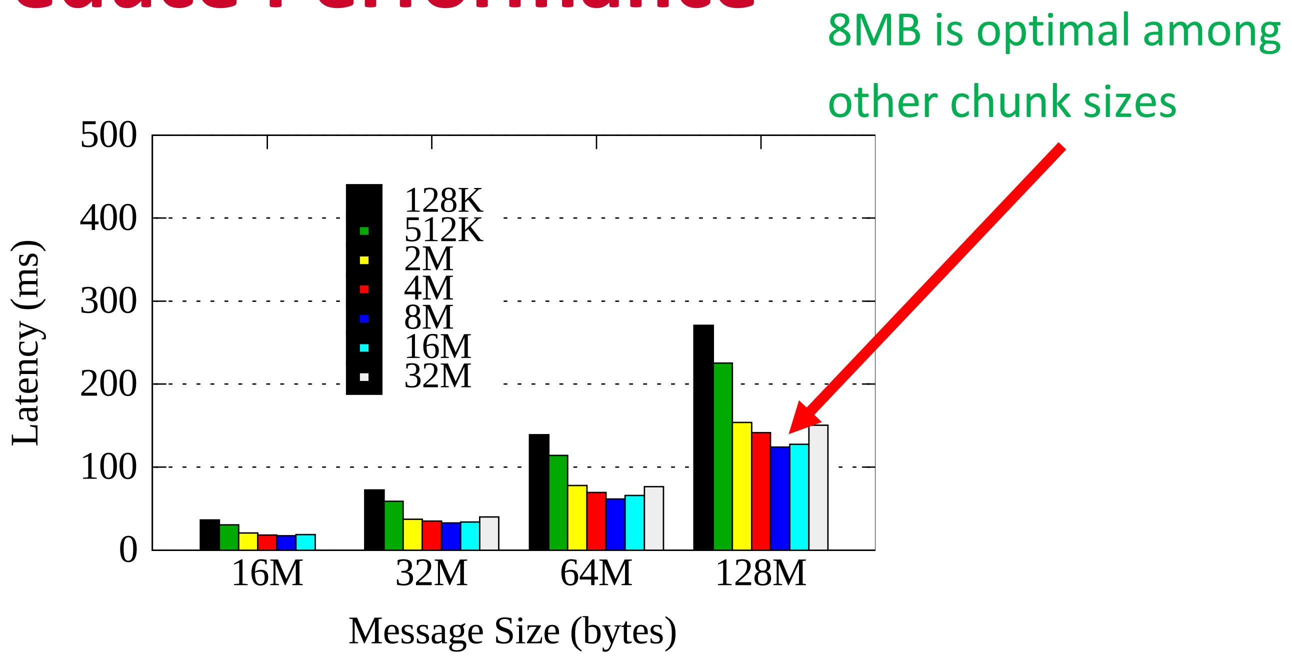
- SALaR-XPMEM
  - Efficient Pipeline of Inter-node Allreduce with Intra-node Reduce
  - Uses XPMEM as intra-node zero copy mechanism
- SALaR-SHMEM
  - In case of lack of XPMEM module, shared memory is being used as the intra-node mechanism



Impact of Chunk Size on Allreduce Performance



SALaR-XPMEM



SALaR-XPMEM (Larger Messages)

Latency of MPI_Allreduce on 224 processes and 28 processes per node on Cluster A

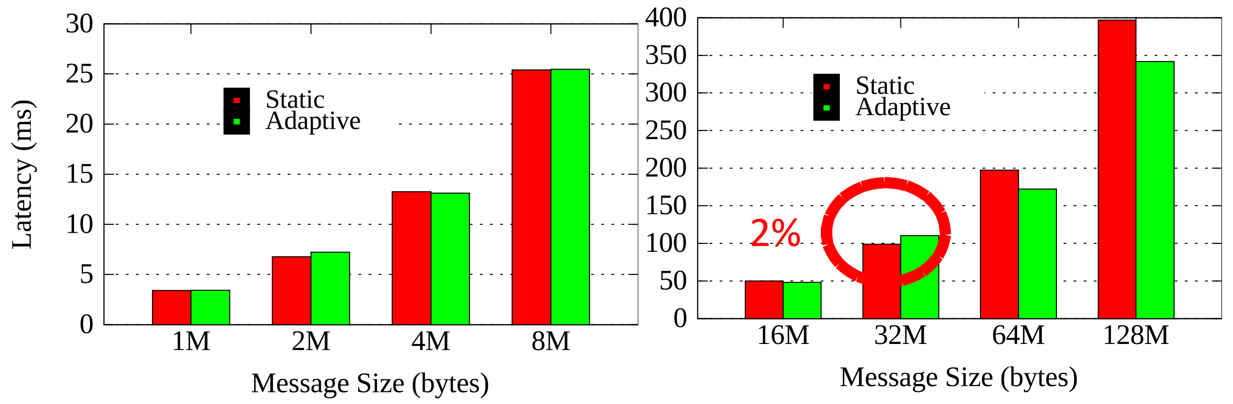
- Selecting the proper chunk size can have a big impact on the performance
- Different chunk is optimal for each message range

Adaptive and Dynamic Chunk Size Selection

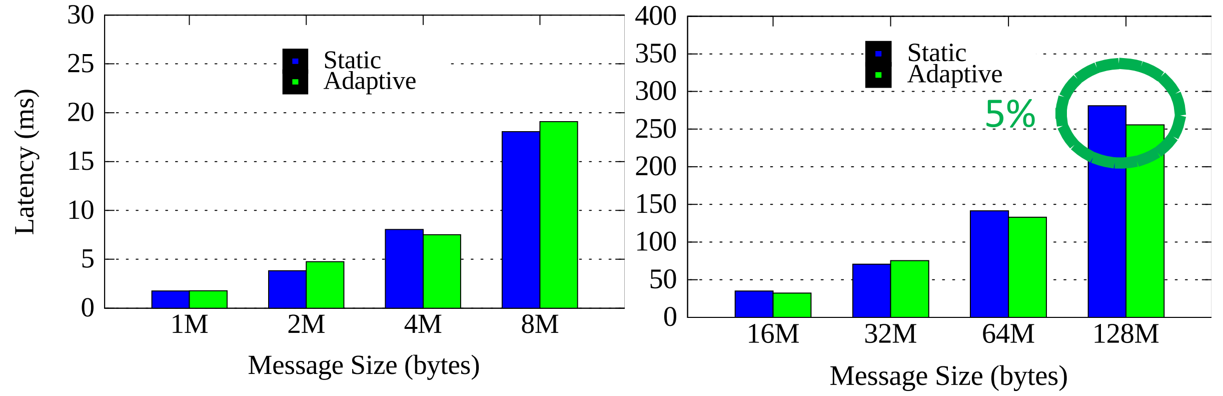
- The optimal chunk size depends on many factors
  - System configuration, job size, message size, PPN
  - Static tuning is a costly operation for large messages
- Select the appropriate chunk size for a particular message range using:
  - Comparison to previous calls latency and the performance model of SALaR
    - Performance model will be introduced

Impact of Heuristic based Design on Allreduce Performance

- Adaptive design is close and in some cases, even has better performance compared to the Static version
- Effectively removes the hassle of static tuning



SALaR-SHMEM design on 896 processes on Cluster A



SALaR-XPMEM designs 896 processes on Cluster A

Outline

Modeling the Proposed SALaR Design

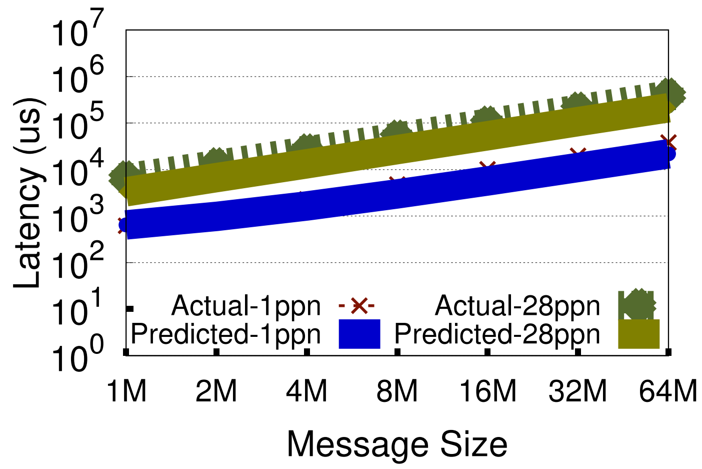
$$T_{total-allreduce}(l) =$$
$$I \times max\{T_{inter}, T_{intra}\}(l/I) + I \times T_{intra-bcast}(l/I) =$$
$$l \times max\{(\frac{n-1}{n})max\{G, C\} + \frac{G}{K} + \frac{C}{nK} + \frac{C'I}{l}, \frac{p}{p-1}(G' + C)\} + G'l$$

In order to simplify the model, if we assume that $n \gg 1$, $p \gg 1$, $K = 1$, and $C' \simeq 0$, then we can have:

$$T_{total-allreduce}(l) =$$
$$\boxed{max\{max\{G, C\} + G, (G' + C)\}l + G'l}$$

- Based on LogGP modeling framework
- At scale, the total latency does not heavily dependent of number of processes
    - Shows the scalability of the design

Symbol	Description
n	Number of Nodes
p	Number of Processes Per lode
l	Size of the input vector
M	Size of the transferred vector in each iteration
G	Gap per inter-node byte transfer
G'	Gap per intra-node byte transfer
C	Computation time per byte
K	Number of the inter-node chunks
I	Number of the intra-node chunks



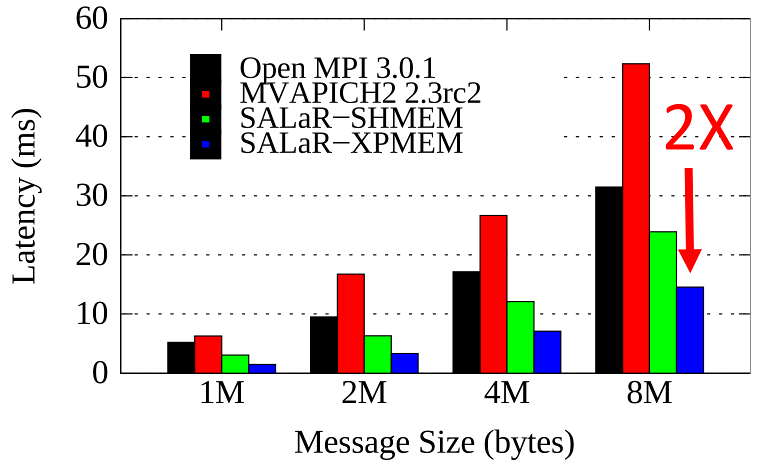
Validation of Allreduce model on 8 nodes on cluster A. (G = 0.0000841, G0 = 0.0003077, C = 0.0001835)

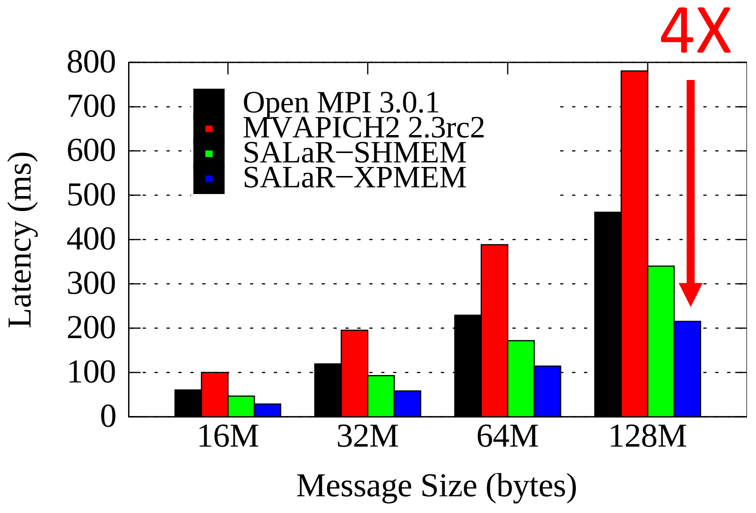
Experimental Setup

Hardware		Software	
Cluster A RI2	Cluster B Comet	MPI Benchmark	DL Frameworks
40 Dual socket Intel Xeon series CPUs 14-core Broadwell processors of 2.40 GHz	1944 Dell PowerEdge C6320 two- socket servers with 12-core Intel Xeon processors of 2.50 GHz	OSU Microbenchmarks v5.4.1	Microsoft Computational Network Toolkit (CNTK) v.2.3.1
Mellanox MT4115 EDR ConnectX-4 HCAs	Mellanox MT4099 FDR ConnectX-3 HCAs		Horovod: Uber implementation of Tensorflow v0.12.1

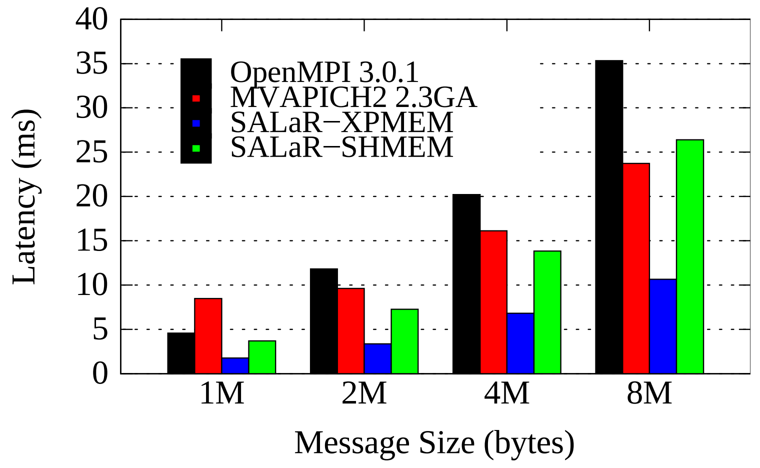
Performance Comparison of MPI_Allreduce

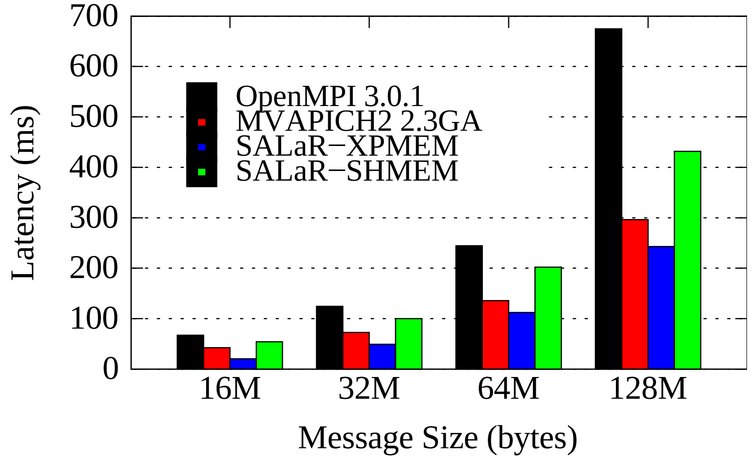
- Using osu_allreduce benchmark from OSU Microbenchmarks on Cluster A with 28 processes per node
- SALaR outperforms Open MPI and MVAPICH2 up to 2X and 4X
- In the latest release of MVAPICH2, we have incorporated some of similar SALaR ideas and enhanced the performance





448 Processes (Same as in the Paper)

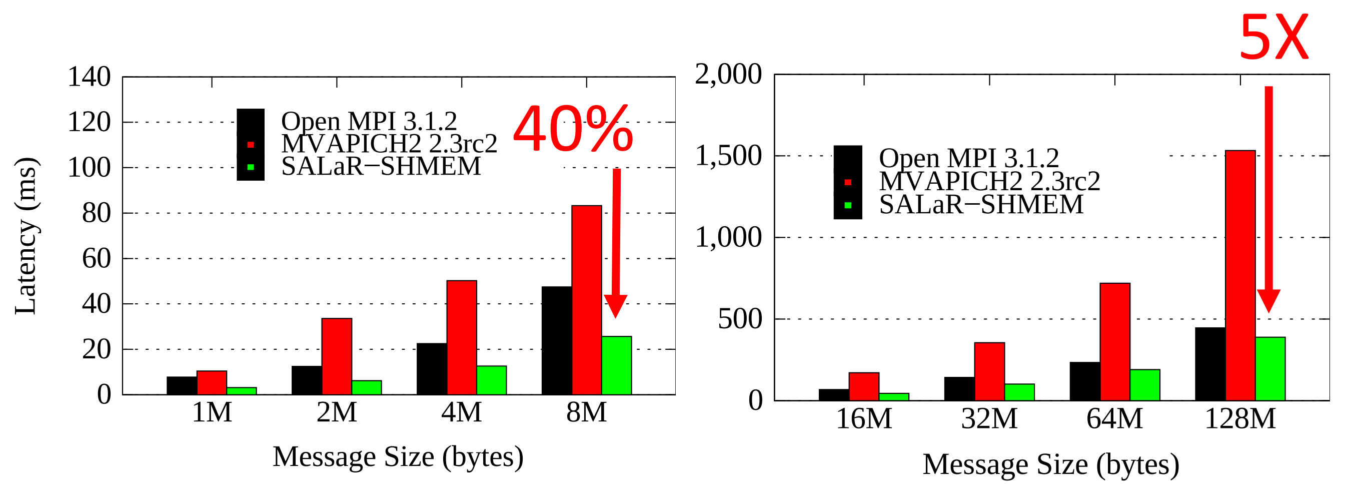




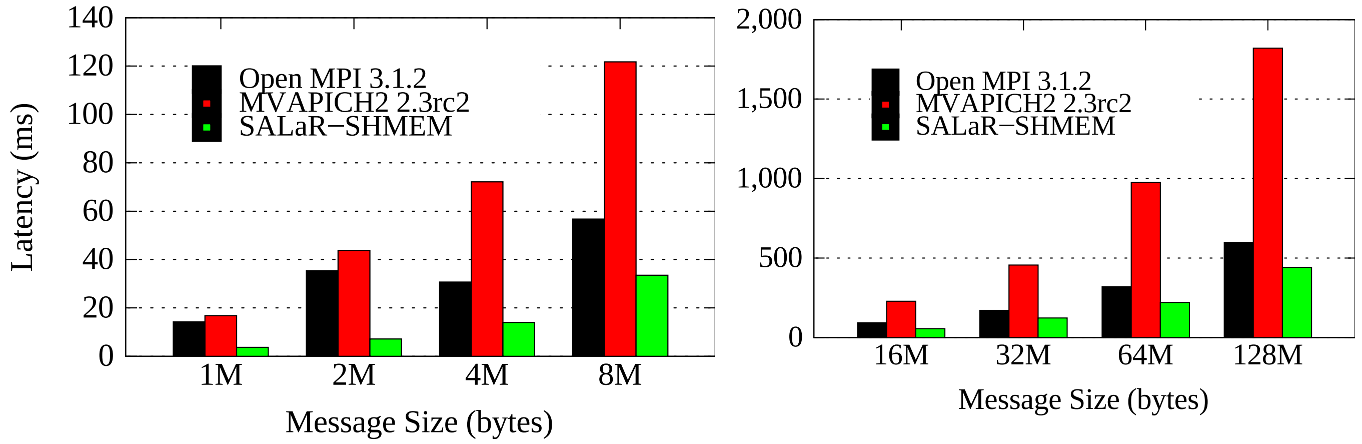
756 Processes (Latest Numbers)

Performance Comparison of MPI_Allreduce (cont'd)

- Using osu_allreduce benchmark from OSU Microbenchmarks on Cluster B with 24 processes per node
- SALaR outperforms Open MPI v3.1.2 and MVAPICH2 v2.3rc2 up to 40% and 5X respectively



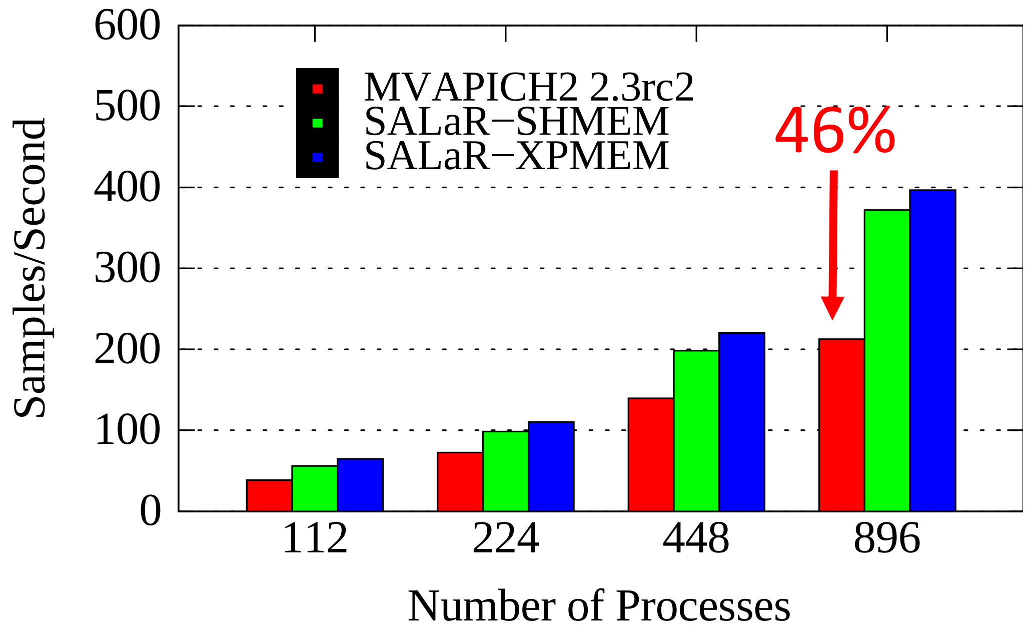
768 Processes on Cluster B



1536 Processes on Cluster B

Impact of SALaR Designs on CNTK

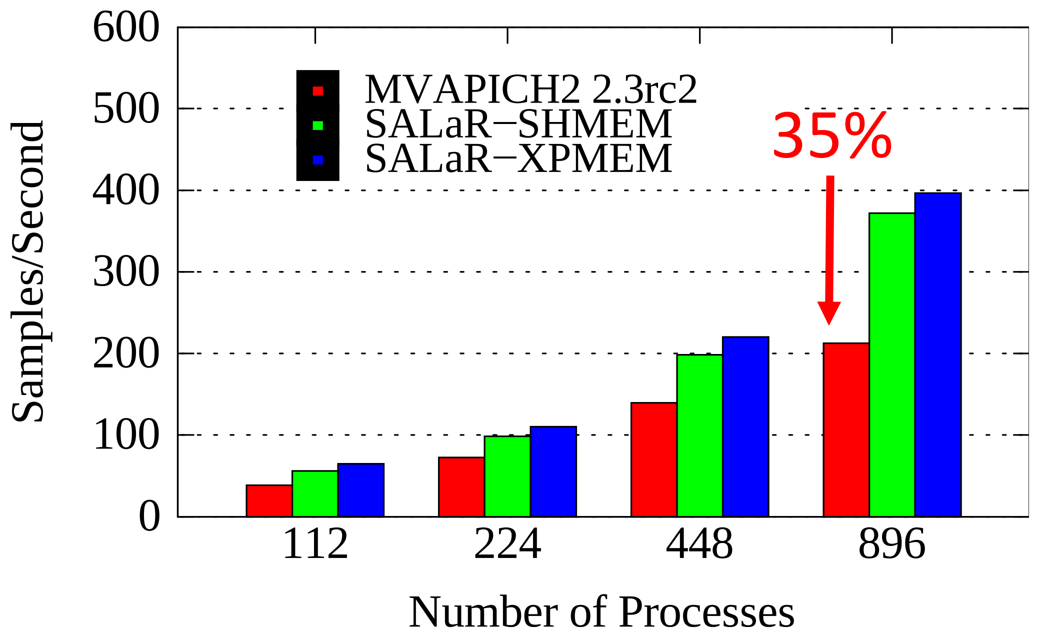
- CPU-based training AlexNet neural network ILSVRC2012 dataset from the ImageNet
- SALaR designs perform up to 46% better than the MVAPICH2 library at 896 processes
- Increasing the scale, the benefits of the proposed designs also increases



CNTK Samples per Second on Cluster A (higher is better)

Impact of SALaR Designs on TensorFlow

- CPU-based tf_cnn_benchmarks for distributed tests from TensorFlow Benchmarks (TF)
  - Training AlexNet neural network from the synthetic datasets
- 15% and 35% improvements in the number of images per second at 448 and 896 processes jobs
- Increasing the job size, the benefits of SALaR compared to MVAPICH2 keep increasing



TensorFlow Images per Second (higher is better)

Conclusions & Future Work

- Designed multi-leader based collective operations
  - Capable of taking advantage of high-end features offered by modern network interconnects
- Modeled and analyzed proposed design theoretically
- The benefits were evaluated on different architectures
- The DPML design is released as a part of MVAPICH2-X 2.3b! Check out:
  - http://mvapich.cse.ohio-state.edu/overview/#mv2X
- Studied the interplay between communication pattern of applications and different tag matching schemes
- Proposes, designed and implemented a dynamic and adaptive tag matching scheme capable to adapting dynamically to the communication characteristics of applications
- The adaptive approach opens up a new direction to design tag matching schemes for next-generation exascale systems

Conclusion and Future Work (cont'd)

- Proposed scalable and adaptive Allreduce design
  - Capable of taking advantage of high-end features offered by modern network interconnects and increased parallelism of Multi-/Many-core architectures
- Modeled and analyzed proposed design theoretically
- The benefits were evaluated on different architectures and Deep Learning frameworks
- Improved the AlexNet training time on CNTK by up to 46%
- Reduced the latency of osu_allreduce by up to 5X at scale
- In the future:
  - Exploring the SALaR for other collective operations
- The SALaR design will be as a part of MVAPICH2! Check out:
  - http://mvapich.cse.ohio-state.edu/

References

[a] Baidu Allreduce Design: https://github.com/baidu- research/baidu-allreduce

[b] Efficient communications in training large scale neural networks, Zhao et al, Thematic Workshops ACMMM2017

[c] MVAPICH2 2.3rc2

[d] Bandwidth optimal all-reduce algorithms for clusters of workstations, Patarasuk et al, Journal of Parallel and Distributed Comp '09
[e] OpenMPI 1.8.5 and later

[f] Designing Efficient Shared Address Space Reduction Collectives for Multi-/Many-cores, Hashmi et al, IPDPS '17

Thank you! Questions?