



**MVA PICH**

MPI, PGAS and Hybrid MPI+PGAS Library

# Designing Efficient Shared Address Space Reduction Collectives for Multi-/Many-cores

Jahanzeb Hashmi, Sourav Chakraborty, Mohammadreza Bayatpour,  
Hari Subramoni and DK Panda

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

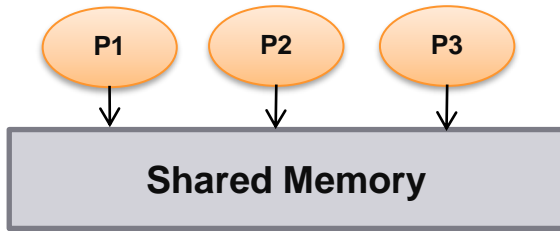
Network Based Computing Laboratory (NBCL)

The Ohio State University

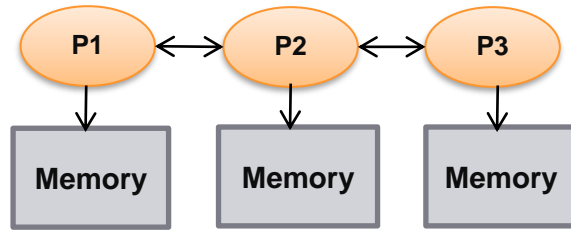
# Outline

- Introduction and Motivation
- Background
  - Shared-memory Communication
  - Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Reduction Collectives MPI\_Allreduce / MPI\_Reduce
- Performance Evaluation and Analysis
- Concluding Remarks

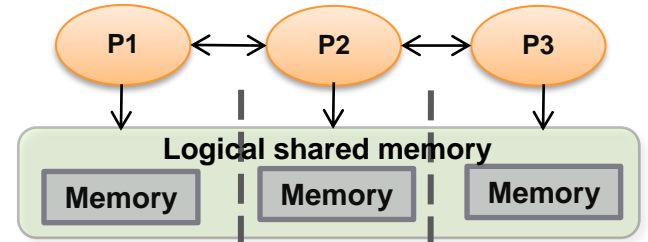
# Parallel Programming Models Overview



Shared Memory Model  
SHMEM, DSM



Distributed Memory Model  
MPI (Message Passing Interface)



Partitioned Global Address Space (PGAS)  
Global Arrays, UPC, Chapel, X10, CAF, ...

- Programming models provide abstract machine models
- Models can be mapped on different types of systems
  - e.g. Distributed Shared Memory (DSM), MPI within a node, etc.
- Programming models offer various communication primitives
  - Point-to-point (between pair of processes/threads)
  - Remote Memory Access (directly access memory of another process)
  - **Collectives (group communication)**

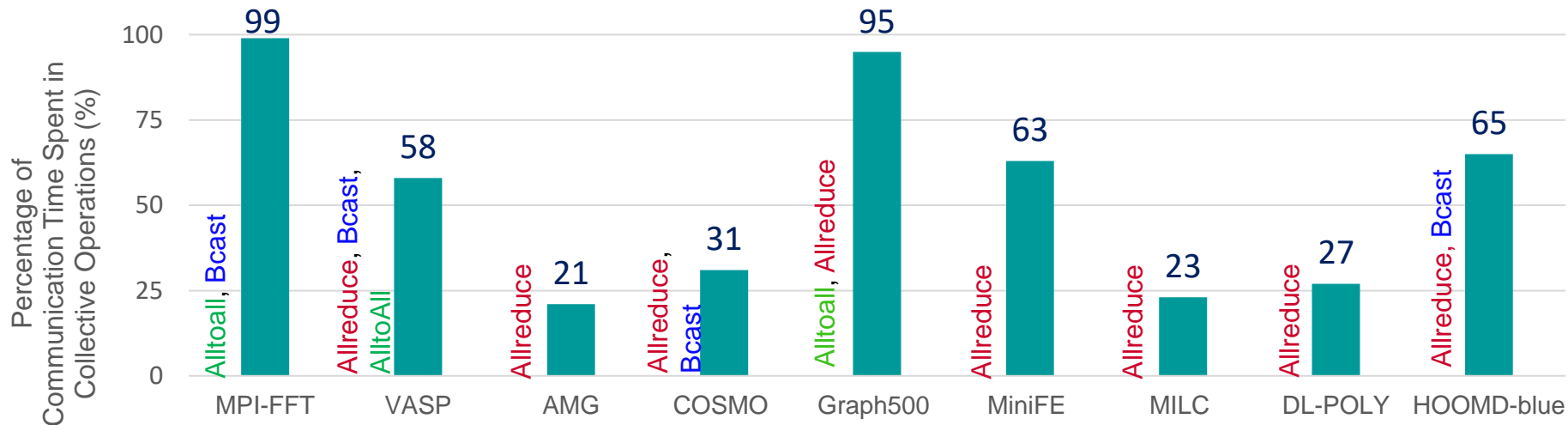
# Diversity in HPC Architectures



	Knights Landing (KNL)	Xeon	OpenPower
Clock Speed	Low	High	Very High
Core count	High (64-72)	Low (8-16)	Low (8-12)
Hardware Threads	Medium (4)	Low (1-2)	High (8)
Multi-Socket	No	Yes	Yes
Max. DDR Channels	6	4	8
HBM/MCDRAM	Yes	No	No

Dense Nodes  $\Rightarrow$  More Intra Node Communication

# Why Collective Communication Matters?



- HPC Advisory Council (HPCAC) MPI application profiles
- Most application profiles showed majority of time spent in collective operations
- Optimizing collective communication directly impacts scientific applications leading to accelerated scientific discovery

Courtesy: <http://www.hpcadvisorycouncil.com>

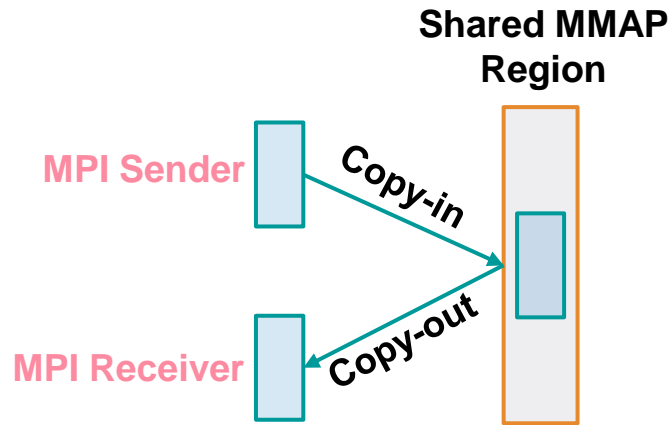
# Broad Challenges in MPI due to Architectural Diversity

- **Can we exploit high-concurrency and high-bandwidth offered by modern architectures?**
  - better resource utilization → high throughput → faster communication performance
  - Computation and communication offloading
- **Can we design “zero-copy” and contention-free MPI communication primitives?**
  - Memory copies are expensive on many-cores
  - “Zero-copy” (kernel-assisted) designs are Contention-prone

# Outline

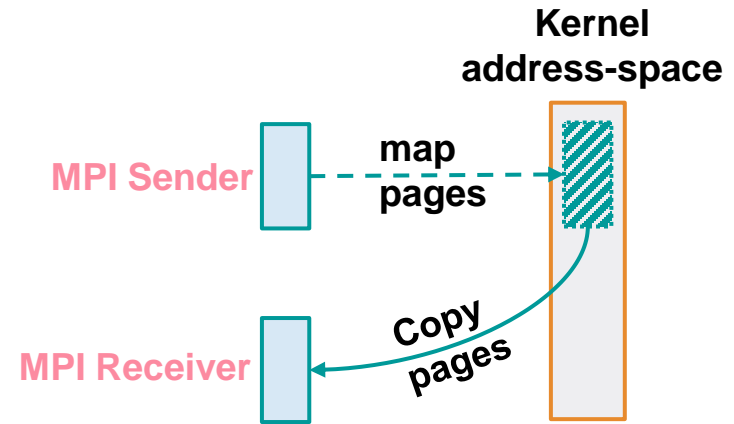
- Introduction and Motivation
- Background
  - Shared-memory Communication
  - Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Reduction Collectives MPI\_Allreduce / MPI\_Reduce
- Performance Evaluation and Analysis
- Concluding Remarks

# Intra-node Communication Designs in MPI



## Shared Memory – SHMEM

Requires two copies  
No system call overhead  
Better for Small Messages



## Kernel-Assisted Copy

System call overhead  
Requires single(a.k.a “zero”) copy  
Better for Large Messages



# A Variety of Available Zero-copy Mechanisms

	LiMIC	KNEM	CMA	XPMEM
Permission Check	Not Supported	Supported	Supported	Supported
Availability	Kernel Module	Kernel Module	Included in Linux 3.2+	Kernel Module
Memcpy invocation	Kernel-space	Kernel-space	Kernel-space	User-space

## MPI Library Support

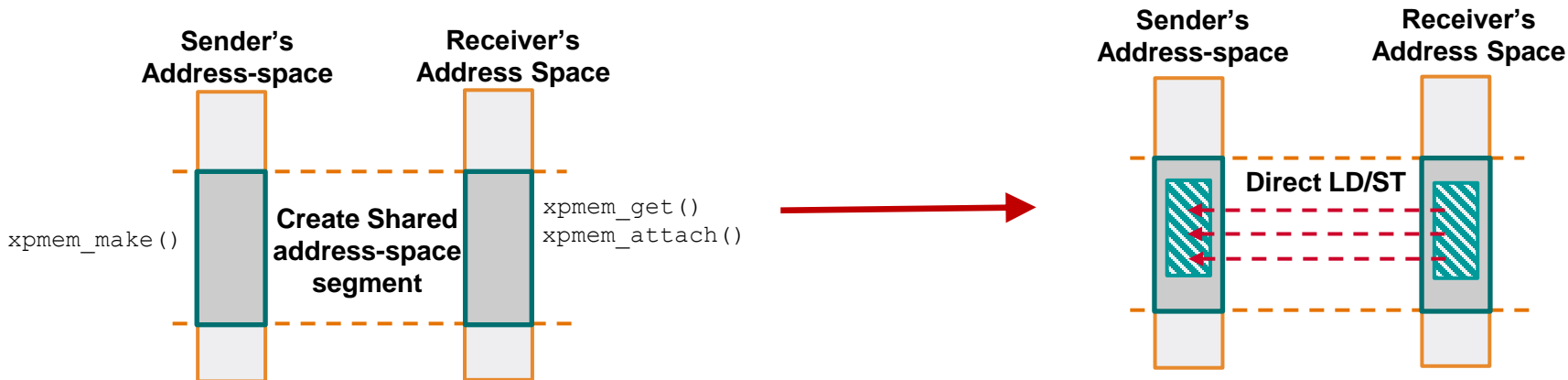
	LiMIC	KNEM	CMA	XPMEM
MVAPICH2	✓	x	✓	✓ (upcoming release)
OpenMPI	x	✓	✓	✓
Intel MPI	x	x	✓	x
Cray MPI	x	x	✓	✓

# Outline

- Introduction and Motivation
- Background
  - Shared-memory Communication
  - Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Reduction Collectives MPI\_Allreduce / MPI\_Reduce
- Performance Evaluation and Analysis
- Concluding Remarks

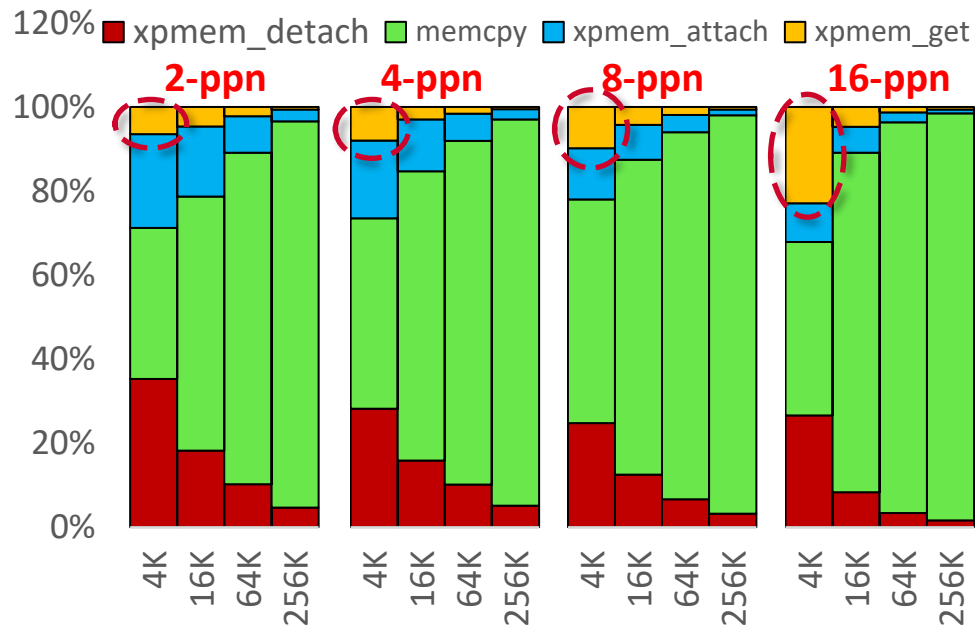
# Shared Address-space based Communication

- XPMEM (<https://github.com/hjelmnx1/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 sender process calls “`xpmem_make()`” to create a shared segment
  - Segment information is then shared with the receiver
- The receiver process calls “`xpmem_get()`” followed by “`xpmem_attach()`”
- The receiver process can directly read/write on the remote process’ memory



# Quantifying the Registration Overheads of XPMEM

- XPMEM based one-to-all latency benchmark
  - Mimics rooted collectives
- A process needs to attach to remote process before memcpy
- Up to **65%** time spent in XPMEM registration for short message (4K)
- Increasing PPN increases the cost of `xpmem_get()` operation
  - Lock contention
  - Pronounced at small messages



Relative costs of XPMEM API functions for different PPN using one-to-all communication benchmark on a single dual-socket Broadwell node with 14 cores.

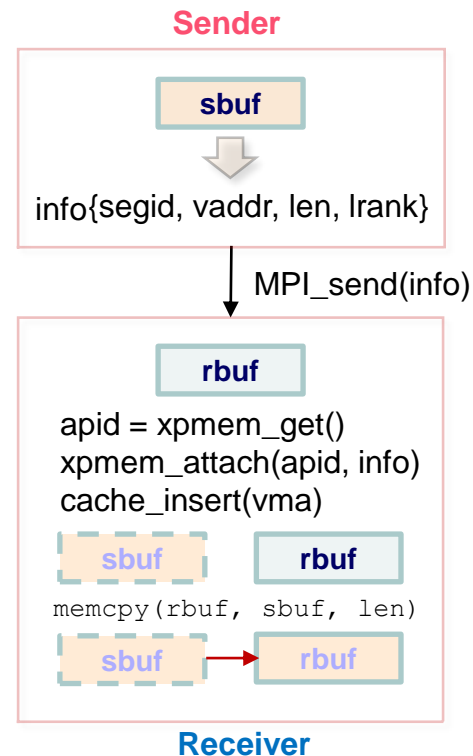
*How can we alleviate the overheads posed by XPMEM registration and improve the performance of shared address-space based communication primitives?*



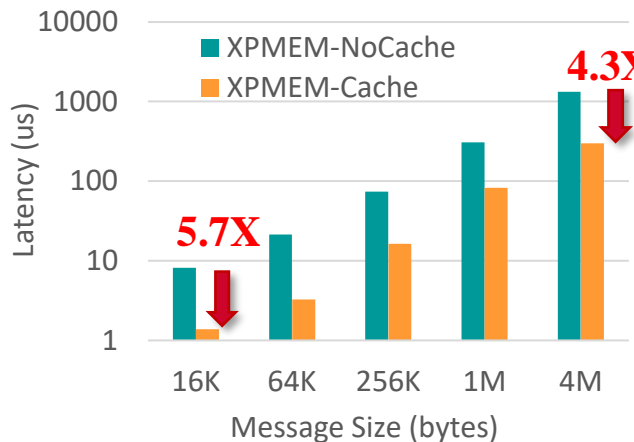
*Registration Cache!*

# Registration Cache for XPMEM based Communication

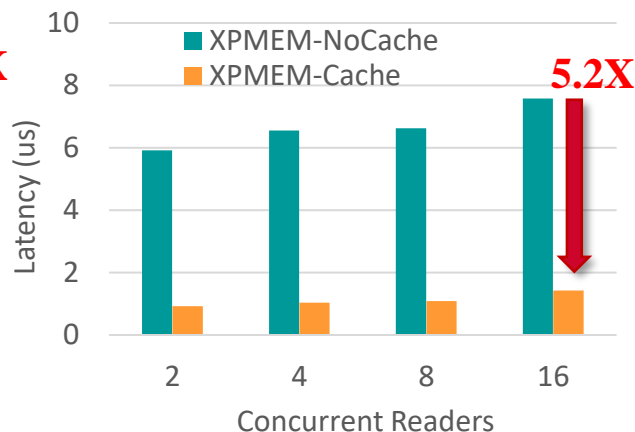
- Remote pages that are *attached* are kept in an AVL tree
  - One tree per remote peer
  - Insertion and lookup in  $O(\log n)$  time
- First miss, attach remote VMA and cache locally
  - Later accesses are found in registration cache
- Lazy memory deregistration principle
  - Deregister pages only at *finalize* or when capacity-miss occurs (FIFO)
- MPI operations using same buffer do not incur XPMEM registration overheads
  - Performance is only limited by the memcopy



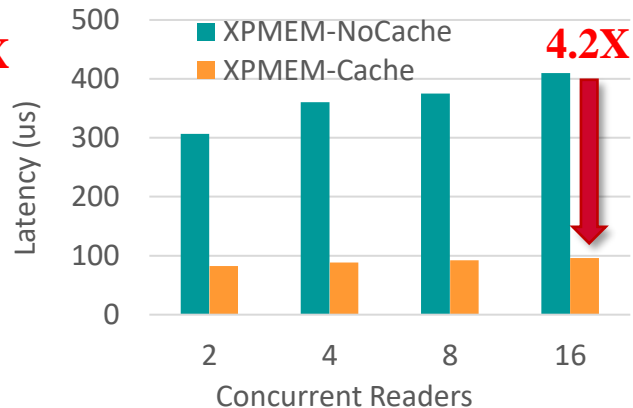
# Impact of Registration Cache on the Performance of XPMEM based Communication



Two-process latency  
at varying messages



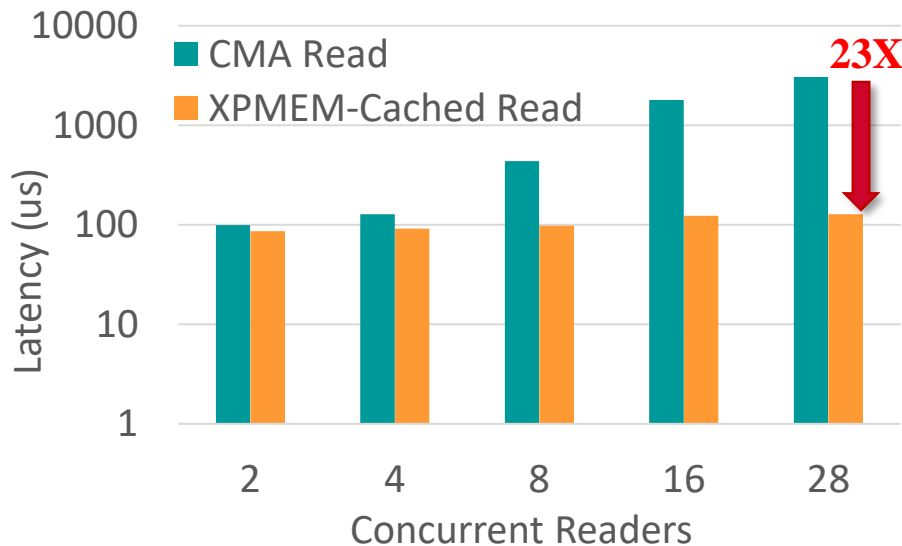
Multi-process latency  
at 16KB message



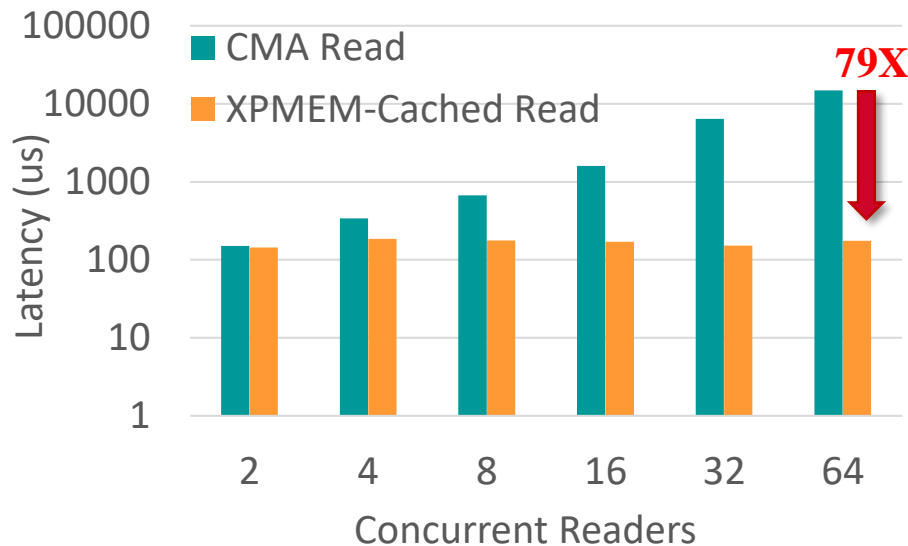
Multi-process latency  
at 1MB message

- Registration cache mitigates the overhead of XPMEM registration of remote memory segments
  - At first miss, remote pages are attached and cached
- Look-up in registration cache cost  $O(\log n)$  time due to AVL tree based design
- Benefits are more pronounced at small to medium message size

# Performance of XPMEM and CMA based Communication



Broadwell (2-socket, 14-core)



KNL (68-core, cache-mode)

- Latency comparison of CMA and XPMEM based “read” on a pair-wise *one-to-all* communication pattern at 1MB message size
- CMA based reads suffer from **process-level lock-contention** inside the kernel
- XPMEM based reads do not have locking overheads and thus show significantly scalable performance



# Outline

- Introduction and Motivation
- Background
  - Shared-memory Communication
  - Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- **Designing XPMEM based Reduction Collectives MPI\_Allreduce / MPI\_Reduce**
- Performance Evaluation and Analysis
- Concluding Remarks

# Current Designs for MPI Collectives

- Send/Recv based collectives
  - Rely on the implementation of MPI point-to-point primitives
  - Handshake overheads for each rendezvous message transfer
- Direct Shared-memory based MPI collectives
  - Communication between pairs of processes realized by copying message to a shared-memory region (**copy-in / copy-out**)
- Direct Kernel-assisted MPI collective e.g., CMA, LiMIC, KNEM
  - Can perform direct “*read*” or “*write*” on the user buffers (**zero-copy**)
  - Performance relies on the communication pattern of the collective
- Use two-level designs for inter-node

# Towards Truly Zero-copy Reductions

- Existing work on direct collectives that are based on CMA, LiMIC, KNEM, do not offer zero-copy for reduction implementations
  - Remote data is required to be copied to local memory first
  - Extra copies detrimental to collectives performance
- Can we design “zero-copy” reduction collectives using shared address-space paradigm?
  - Shared address-space based MPI\_Allreduce and MPI\_Reduce designs for MPI
  - Multi-leader design for inter-node scaling

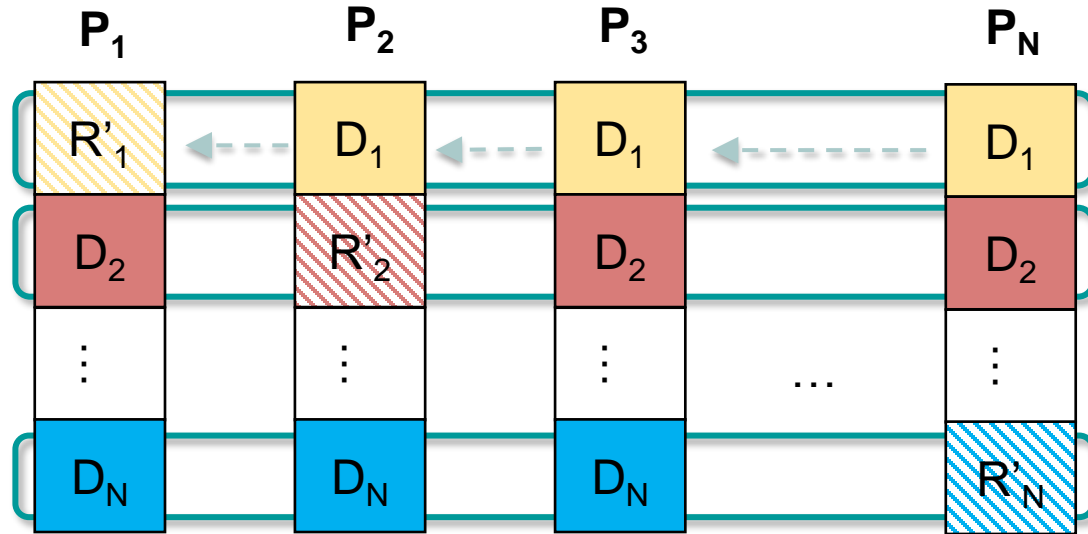
# Shared Address-space (XPMEM-based) Reduction Collectives

- Offload reduction computation and communication to peer MPI ranks
  - Every Peer has direct “load/store” access to other peer’s buffers
  - Multiple leader ranks independently carry-out reductions for intra-and inter-node phases in parallel
  - All peers remain busy and exploit high concurrency of the architecture
- True “zero-copy” design for Allreduce and Reduce
  - No copies require during the entire duration of Reduction operation
  - Scalable to multiple nodes via multi-leader schemes
- No contention overheads due to proposed registration cache design
  - memory copies happen in “user-space”

# Shared Address-space based MPI\_Allreduce

- Every process in the communicator exchanges sendbuf / recvbuf memory segment Information with other processes
  - Application buffers are registered with XPMEM and cached in Registration Cache
- XPMEM based MPI\_Allreduce
  - Step-1: Parallel Intra-node Partitioned *Reduce*
  - Step-2: Parallel Inter-node Partitioned *Allreduce*
  - Step-3: Parallel Intra-node Partitioned *Bcast*
- Similar approach for MPI\_Reduce as well with minor differences
  - Final Bcast step of Allreduce is not performed
  - Final Results needs to be delivered to the “root” process
    - Use one extra point-to-point Send / Recv if “root” is arbitrary

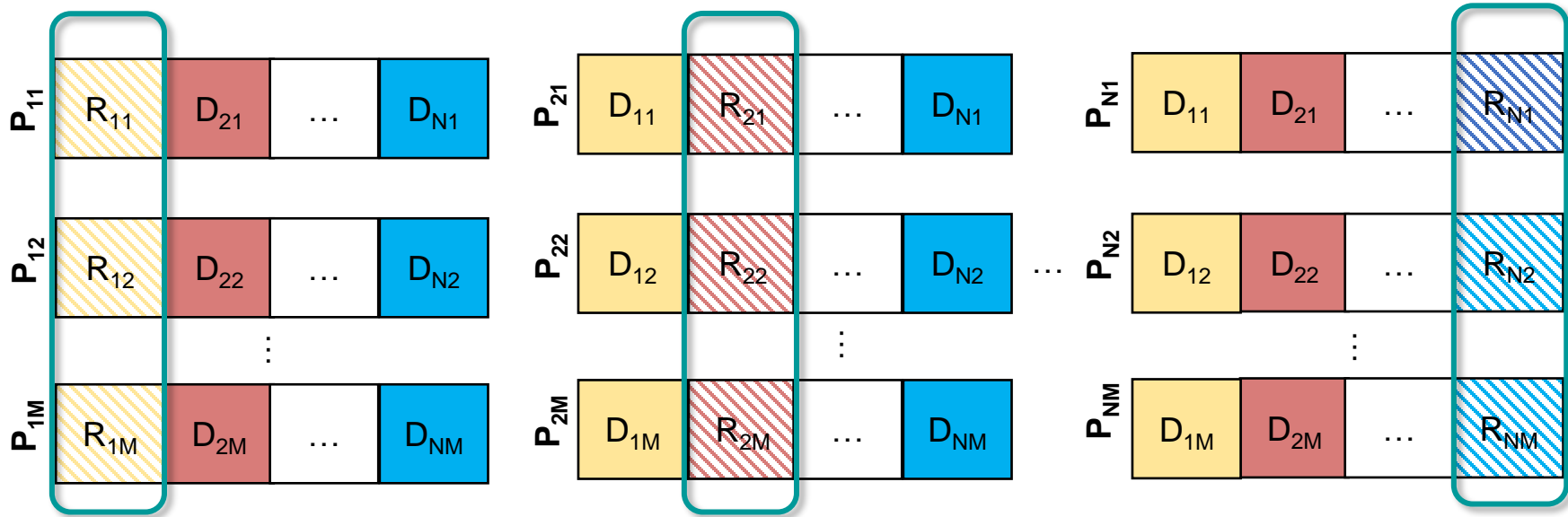
# Step-1: Parallel Intra-node Partitioned Reduce



## Concurrent Intra-Node Reduction by all the Processes on Data Partitions with Same Index

- All intra-node processes ( $n$ ) participate in intra-node reduce phase
- Each  $P_i$  performs reduce operation on  $D_i$  partition of all  $(N-1)$  intra-node processes
- Each  $P_i$  stores the partial reduction result at  $i$ -th partition of its local receive buffer

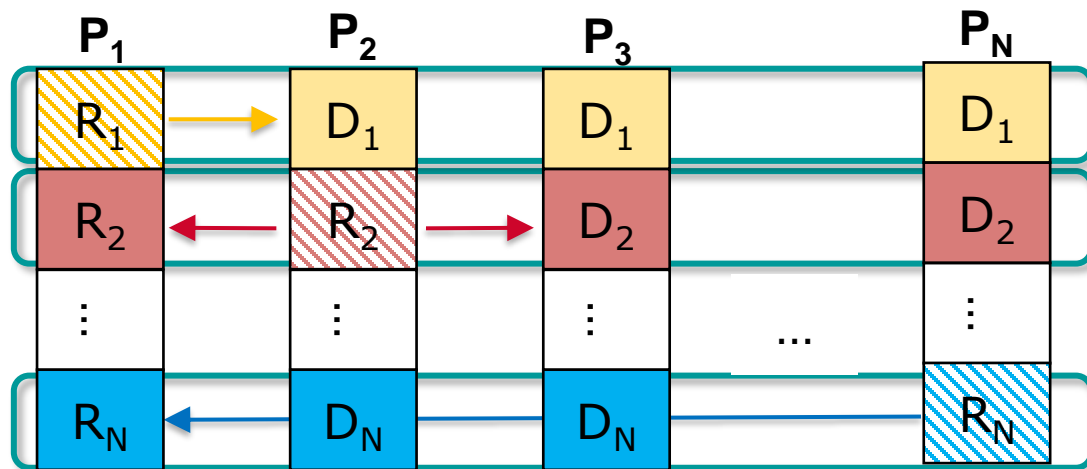
## Step-2: Parallel Intra-node Partitioned Allreduce



### Concurrent Inter-Node Allreduce by all the Processes on Same Index of Data Partitions

- $n$  intra-node processes become leaders for inter-node Allreduce operation across  $m$ -nodes
- Each  $P_{ij}$ , ( $i \in n, j \in m$ ), performs inter-node Allreduce on the partially reduce chunk of data
- The result of inter-node Allreduce is directly stored at the corresponding partition of each leader's receive buffer

## Step:3 Parallel Intra-node Partitioned Bcast



### Copy Local Chunk (full result) to N-1 Processes (Bcast)

[Full result shown for  $P_1$  only, but same for others as well]

- Finally, all intra-node processes ( $n$ ) Broadcast their chunk (fully-allreduced) to all other ( $N-1$ ) processes
  - Copy local chunk to  $D_i$  location of  $N-1$  intra-node processes
- An intra-node barrier is ensued to ensure the completion of Allreduce operation



# Outline

- Introduction and Motivation
- Background
  - Shared-memory Communication
  - Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Reduction Collectives MPI\_Allreduce / MPI\_Reduce
- **Performance Evaluation and Analysis**
- **Concluding Remarks**

# Evaluation Methodology and Cluster Testbeds

Hardware Specification of Cluster Testbeds

Specification	Xeon	Xeon Phi	OpenPOWER
Processor Family	Intel Broadwell	Knights Landing	IBM POWER-8
Processor Model	E5 2680v4	KNL 7250	PPC64LE
Clock Speed	2.4 GHz	1.4 GHz	3.4 GHz
No. of Sockets	2	1	2
Cores Per Socket	14	68	10
Threads Per Core	1	4	8
RAM (DDR)	128 GB	96 GB	256 GB
Interconnect	IB-EDR (100G)	IB-EDR (100G)	IB-EDR (100G)

- Proposed designs, implemented on MVAPICH2, is called MVPIACH2-XPMEM
- Compared against default MVPAPICH2-2.3, Intel MPI 2017, OpenMPI v3.0.0, Spectrum MPI v10.1.0.2
- OSU Microbenchmarks, MiniAMR kernel, and AlexNet DNN Training using CNTK

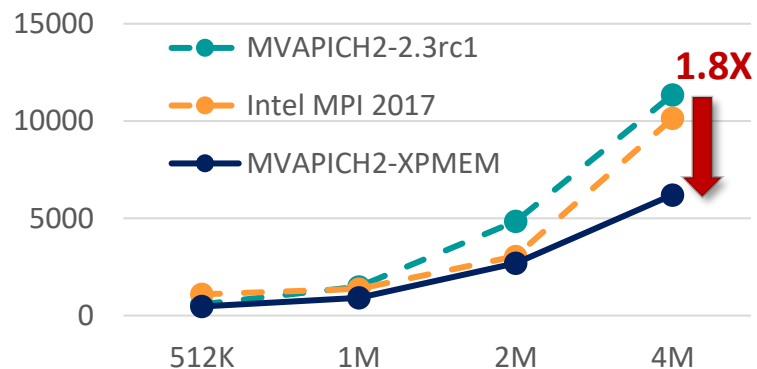
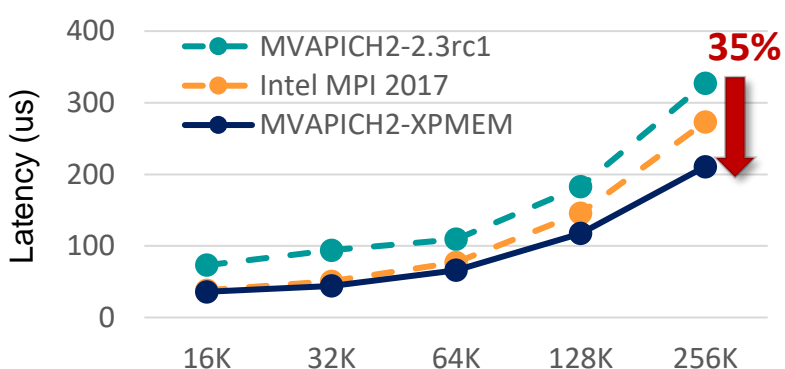
# Overview of the MVAPICH2 Project

- High Performance open-source MPI Library for InfiniBand, Omni-Path, Ethernet/iWARP, and RDMA over Converged Ethernet (RoCE)
  - MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.1), 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 2900 organizations in 86 countries**
  - **More than 469,000 (> 0.46 million) downloads from the OSU site directly**
  - Empowering many TOP500 clusters (Nov '17 ranking)
    - **1st, 10,649,600-core (Sunway TaihuLight) at National Supercomputing Center in Wuxi, China**
    - 9th, 556,104 cores (Oakforest-PACS) in Japan
    - 12th, 368,928-core (Stampede2) at TACC
    - 17th, 241,108-core (Pleiades) at NASA
    - 48th, 76,032-core (Tsubame 2.5) at Tokyo Institute of Technology
  - 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**

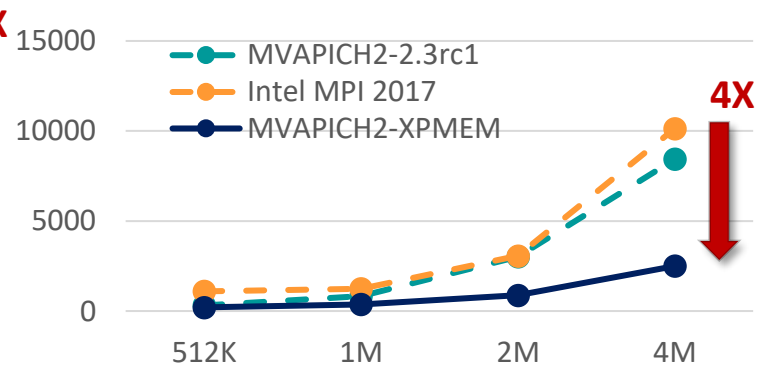
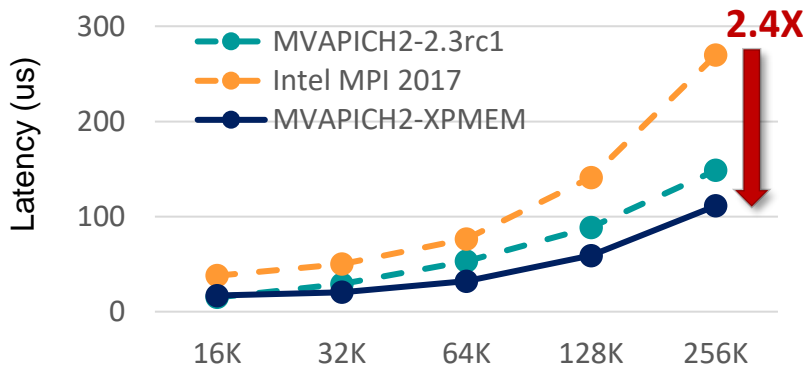


# Micro-benchmark Evaluation on Broadwell Cluster

OSU\_Allreduce

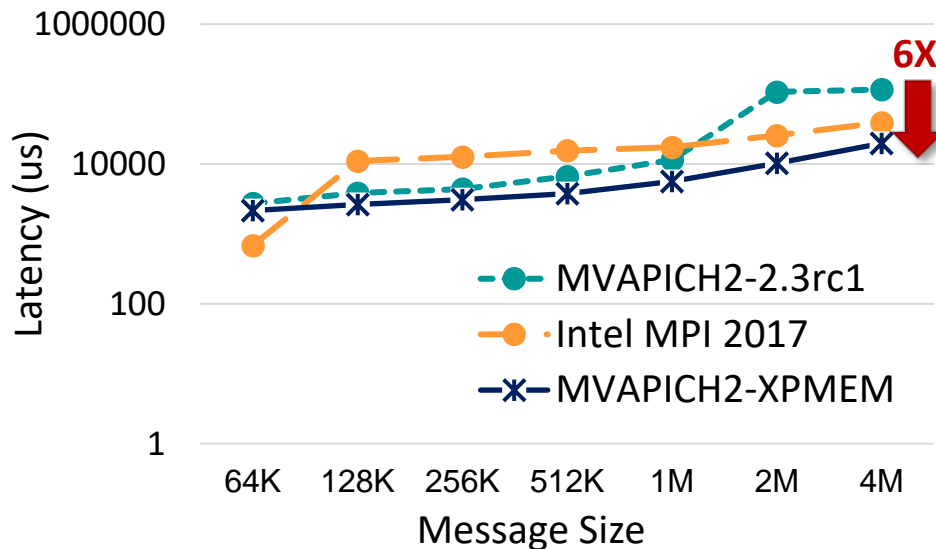


OSU\_Reduce

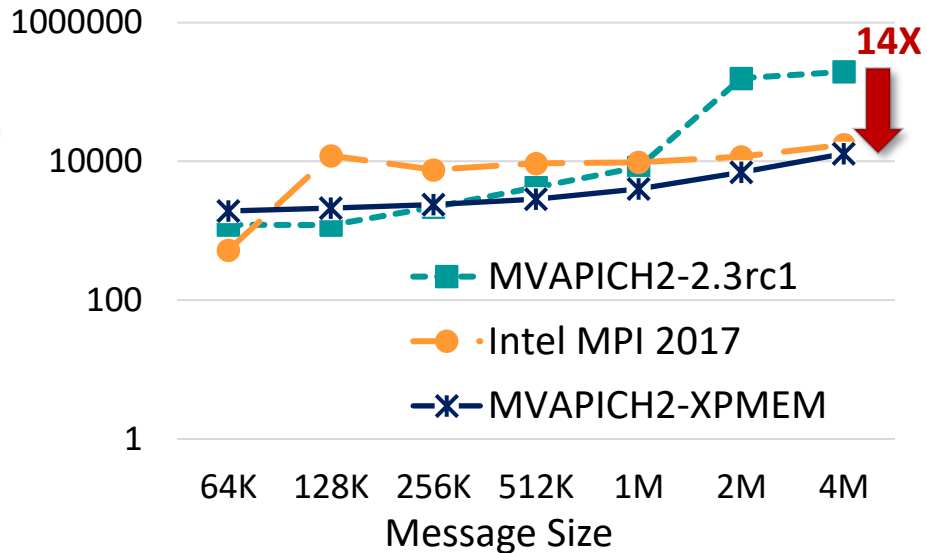


- 16 nodes, 256 processes of dual-socket Broadwell system
- Up to **1.8X** improvement for 4MB AllReduce and **4X** improvement for 4MB Reduce

# Micro-benchmark Evaluation on KNL Cluster



OSU\_Allreduce (KNL 256 procs)

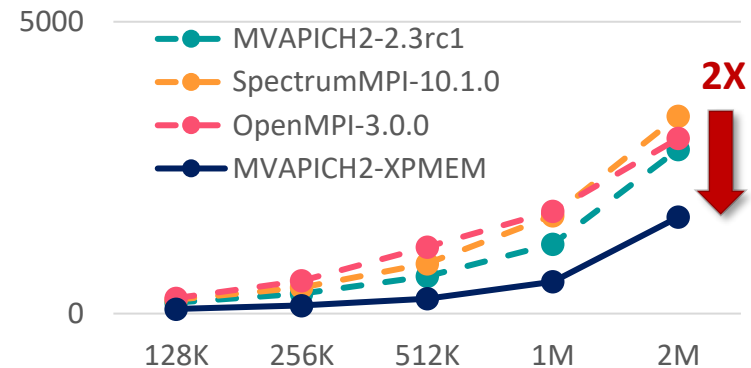
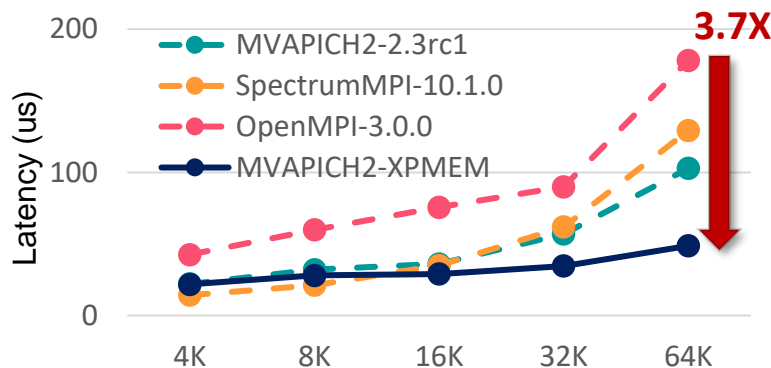


OSU\_Reduce (KNL 256 procs)

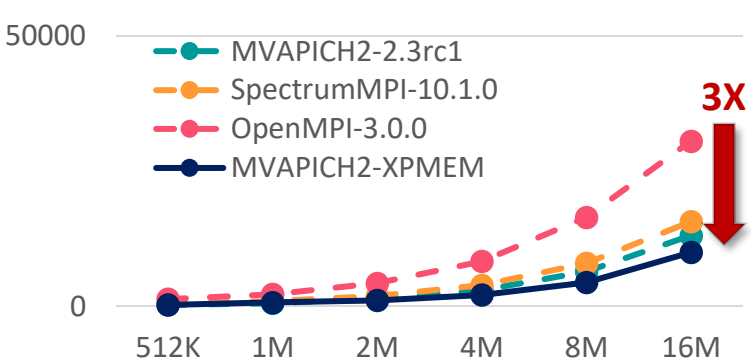
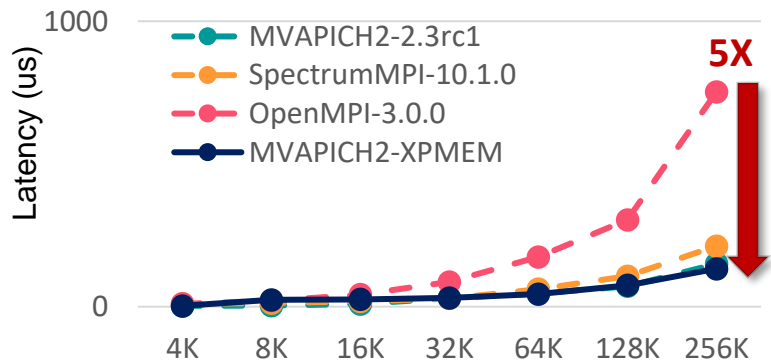
- 4 x KNL 7250 in cache-mode with XPMEM based reduction collectives
- **6X and 14X** improvement over Intel MPI 2017 on XPMEM based Allreduce and Reduce respectively, on 4MB message size

# Micro-benchmark Evaluation on OpenPOWER Cluster

OSU\_Allreduce



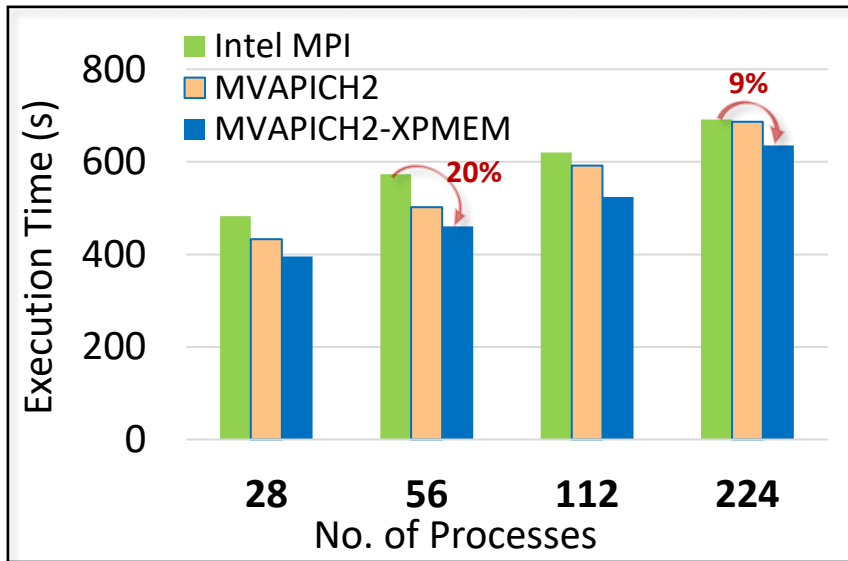
OSU\_Reduce



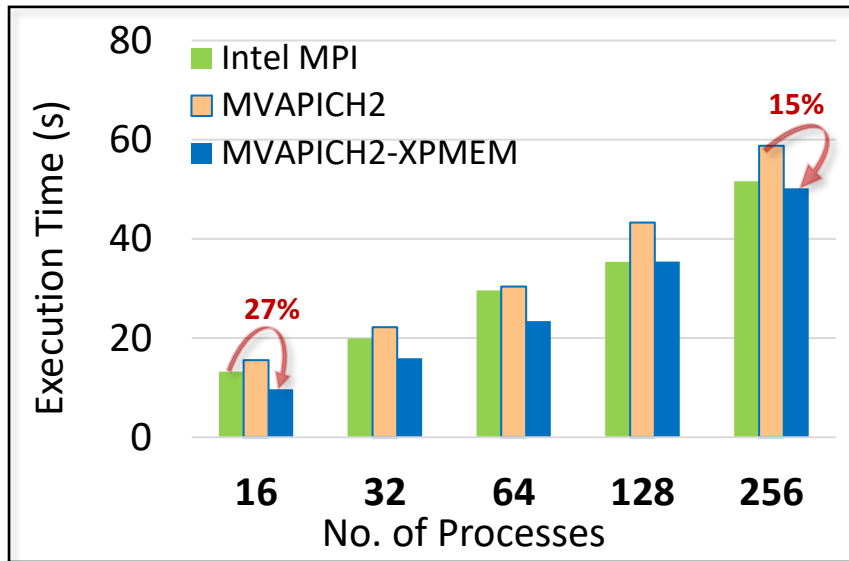
- Two POWER8 dual-socket nodes each with 20 ppn
- Up to **2X** improvement for Allreduce and **3X** improvement for Reduce at 4MB message

# Application Performance of MPI\_Allreduce on Broadwell

CNTK AlexNet Training  
(B.S=default, iteration=50, ppn=28)

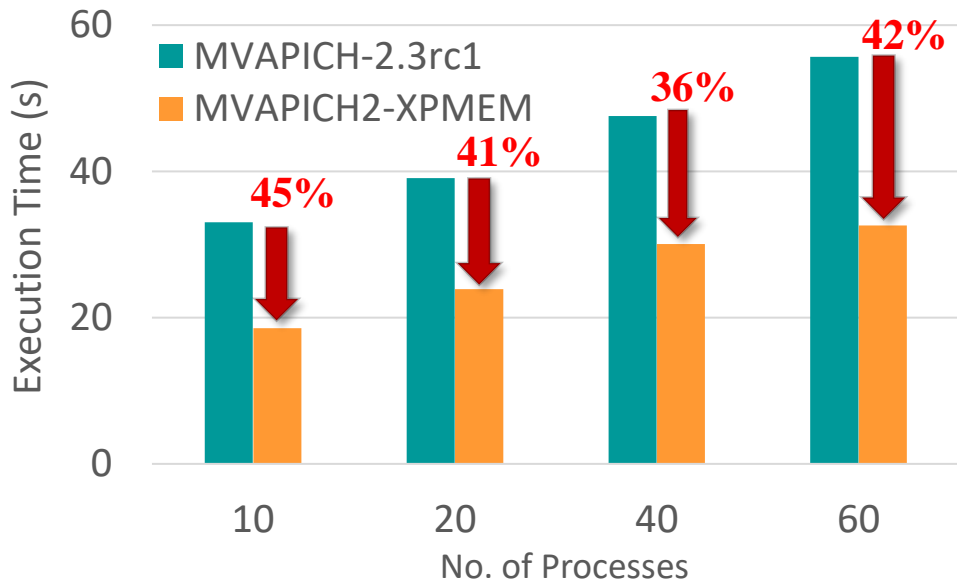


MiniAMR (dual-socket, ppn=16)



- Up to **20%** benefits over IMPI for CNTK DNN training using AllReduce
- Up to **27%** benefits over IMPI and up to **15%** improvement over MVAPICH2 for MiniAMR application kernel

# miniAMR using XPMEM-based AllReduce on OpenPOWER Cluster



OpenPOWER (weak-scaling, 3 nodes, ppn=20)

- miniAMR application execution time comparing MVAPICH2-2.3rc1 and optimized All-Reduce design
  - MiniAMR application for weak-scaling workload on up to three POWER8 nodes.
  - Up to 45% improvement over MVAPICH2-2.3rc1 in mesh-refinement time



# Outline

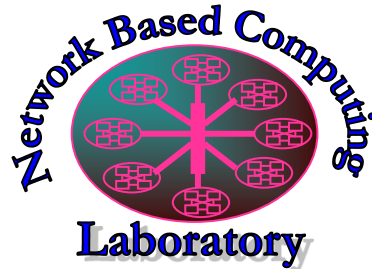
- Introduction and Motivation
- Background
  - Shared-memory Communication
  - Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Reduction Collectives MPI\_Allreduce / MPI\_Reduce
- Performance Evaluation and Analysis
- **Concluding Remarks**

# Concluding Remarks

- Characterized the performance trade-offs involved in designing Shared address-space based communication in MPI
  - Registration cache based schemes to overcome performance bottlenecks
- Design and Implementation of “*true zero-copy*” reduction collectives in MPI
  - Demonstrated the performance benefits of new MPI\_Allreduce and MPI\_Reduce designs on Xeon, Xeon Phi, and OpenPOWER architecture
- Demonstrated the efficacy of the proposed solutions at micro-benchmarks as well as wide range of applications
  - AMR kernel, Neural Network Training, micro-benchmark
  - Significant speedup over existing designs in prevalent MPI libraries such as MVPAICH2, OpenMPI, IntelMPI, and SpectrumMPI
- We plan to expand to designs to other collectives and evaluate other architectures e.g., ARM

# Thank You!

[hashmi.29@osu.edu](mailto:hashmi.29@osu.edu)



Network-Based Computing Laboratory

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



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



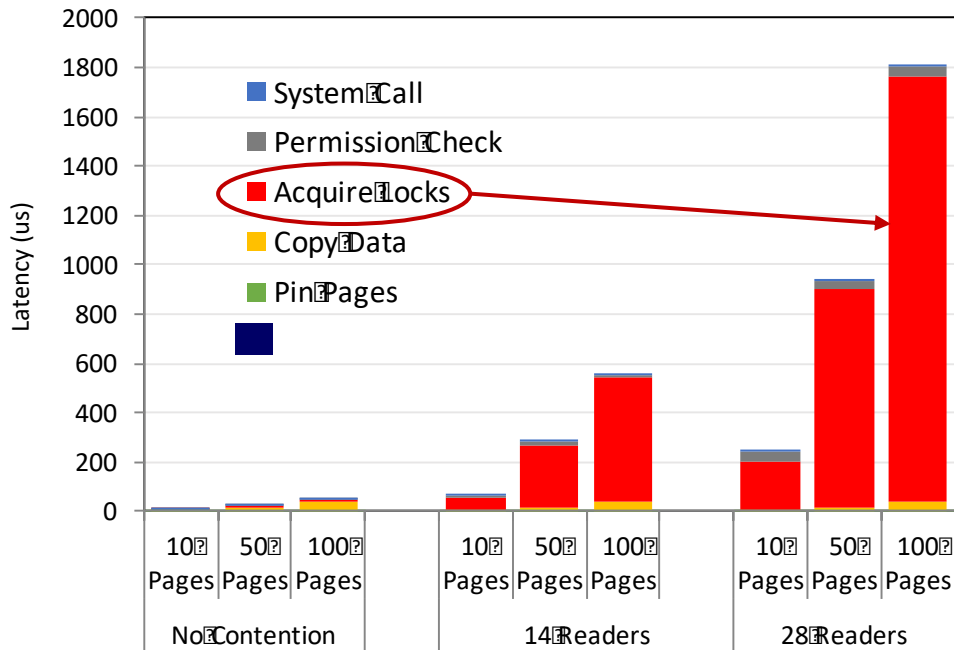
High-Performance  
Big Data

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



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

# Breakdown of a CMA Read operation

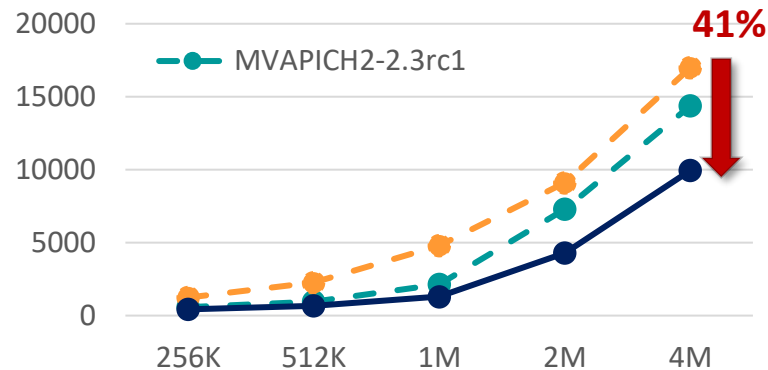
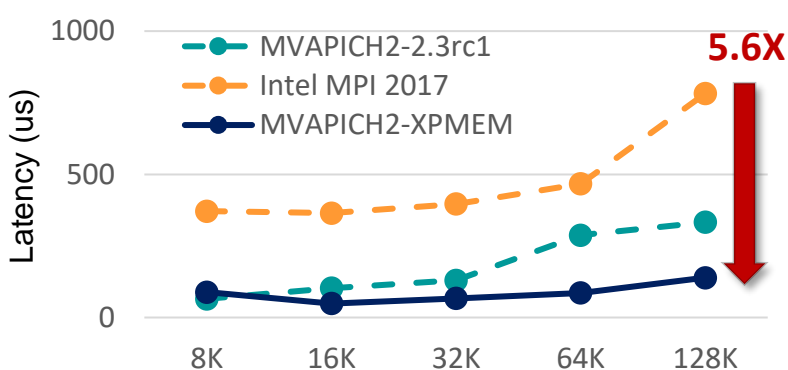


- CMA relies on `get_user_pages()` function
- Takes a page table lock on the target process
- Lock contention increases with number of concurrent readers
- **Over 90% of total time spent in lock contention**
- One-to-all communication on Broadwell, profiled using `ftrace`

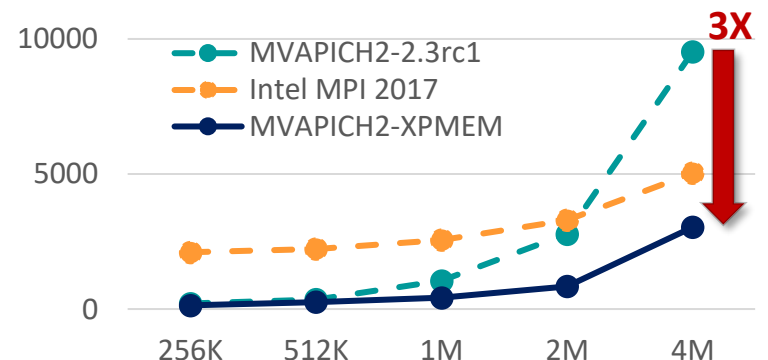
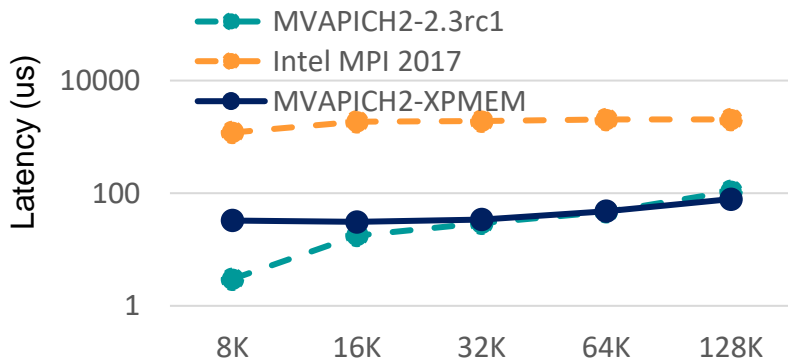
- Lock contention is the root cause of performance degradation
- Present in other kernel-assisted schemes such as KNEM, LiMiC as well

# Scalability Evaluation on Broadwell Cluster

OSU\_Allreduce

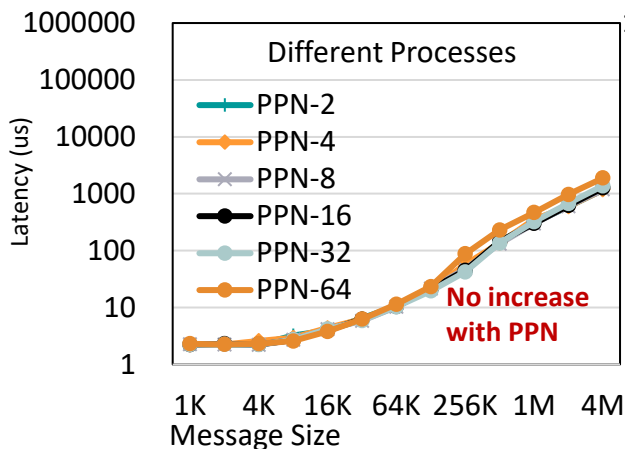


OSU\_Reduce

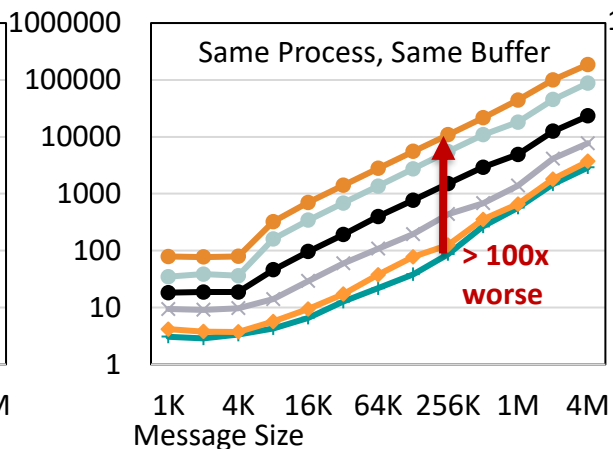
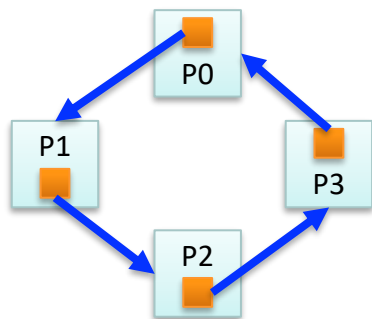


- 32 nodes, 896 processes (28ppn) of dual-socket Broadwell system
- Up to **5.6X** improvement for 4MB AllReduce and **3X** improvement for 4MB Reduce

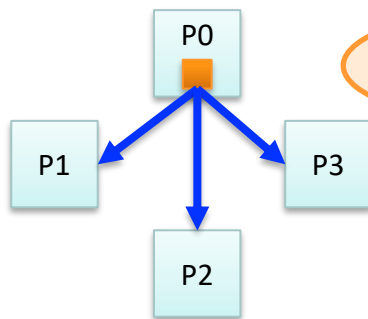
# Impact of Collective Communication Pattern on CMA Collectives



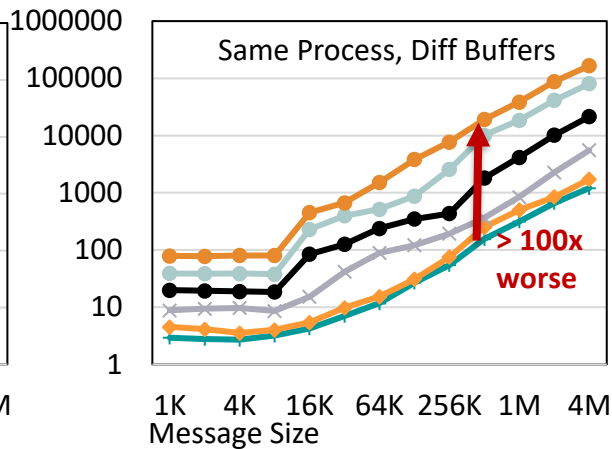
**All-to-All – Good Scalability**



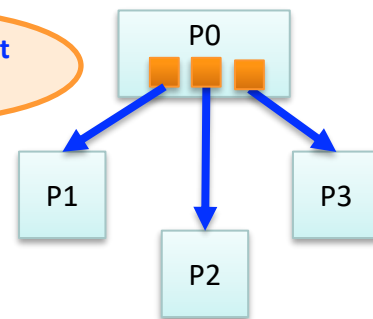
**One-to-All - Poor Scalability**



Contention is at Process level



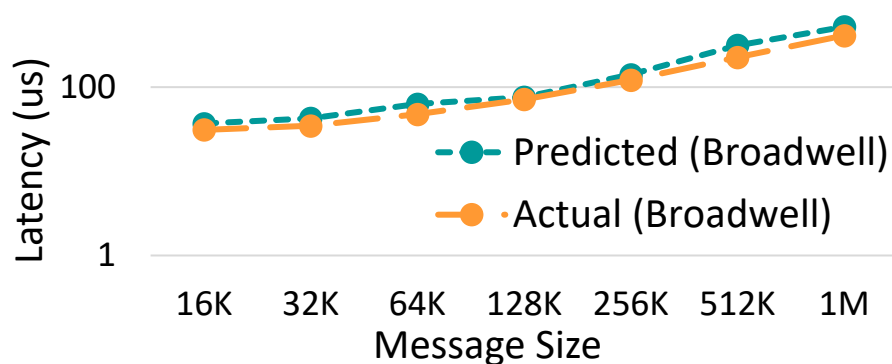
**One-to-All – Poor Scalability**



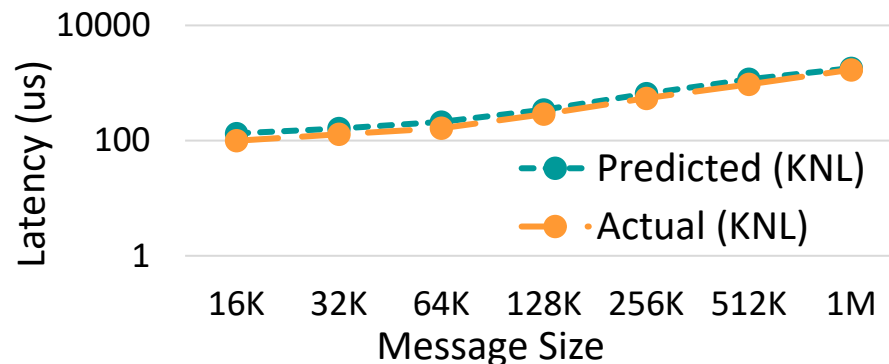
# Modeling and Validation of XPMEM based MPI\_Allreduce

$$\begin{aligned}
 T_{allreduce} &= T_{exchange} + T_{comp} + T_{comm} + T_{bcast} \\
 &= C + (p - 1) \left( \frac{v}{p} \right) c + \lceil \lg m \rceil \left( a + \frac{vb}{p} + \frac{vc}{p} \right) \\
 &\quad + (p - 1) \left( a' + b' \left( \frac{v}{p} \right) \right)
 \end{aligned}$$

M	V	$(v/p)*b'$	$(p-1)*\frac{(v/p)*c}{(v/p)*c}$	$(p-1)*(a'+(b'*(v/p)))$	Predicted	Observed
16K	8192	0.01	23.22	0.271	36.76	31.04



Broadwell (2-socket, 14-core)



KNL (68-core, cache-mode)

# Registration Cache Miss-rate Analysis on Various Benchmarks

Benchmark	MPI Processes	No. of Hits	No. of Misses
MiniAMR	256	10,322,520	30
osu_allreduce	224	223,668	432
osu_reduce	224	111,834	216

## Registration cache Hit/miss (per-process) analysis on Broadwell System

- Application kernels typically re-use same buffers for communication
  - High hit-rate for the registration cache due to temporal locality
- Tuning of registration cache parameters e.g., eviction policy, cache size etc.
  - FIFO performed better than LRU for a fixed sized cache
  - 4K as optimal cache size



# Supporting Programming Models for Multi-Petaflop and Exaflop Systems: Challenges

Application Kernels/Applications

Middleware

Programming Models

MPI, PGAS (UPC, Global Arrays, OpenSHMEM), CUDA, OpenMP, OpenACC, Cilk, Hadoop (MapReduce), Spark (RDD, DAG), etc.

Communication Library or Runtime for Programming Models

Point-to-point  
Communication

Collective  
Communication

Energy-  
Awareness

Synchronization  
and Locks

I/O and  
File Systems

Fault  
Tolerance

Networking Technologies

(InfiniBand, 40/100GigE,  
Aries, and Omni-Path)

Multi-/Many-core  
Architectures

Accelerators  
(GPU and FPGA)

Co-Design  
Opportunities  
and  
Challenges  
across Various  
Layers

Performance  
Scalability  
Resilience

# Architecture of MVAPICH2 Software Family

## High Performance Parallel Programming Models

Message Passing Interface  
(MPI)

PGAS  
(UPC, OpenSHMEM, CAF, UPC++)

Hybrid --- MPI + X  
(MPI + PGAS + OpenMP/Cilk)

## High Performance and Scalable Communication Runtime

### Diverse APIs and Mechanisms

Point-to-point  
Primitives

Collectives  
Algorithms

Job Startup

Energy-Awareness

Remote  
Memory  
Access

I/O and  
File Systems

Fault  
Tolerance

Virtualization

Active  
Messages

Introspection  
& Analysis

### Support for Modern Networking Technology (InfiniBand, iWARP, RoCE, Omni-Path)

#### Transport Protocols

RC

XRC

UD

DC

#### Modern Features

UMR

ODP

SR-IOV

Multi  
Rail

### Support for Modern Multi-/Many-core Architectures (Intel-Xeon, OpenPower, Xeon-Phi, ARM, NVIDIA GPGPU)

#### Transport Mechanisms

Shared  
Memory

CMA

IVSHMEM

XPMMEM\*

#### Modern Features

MCDRAM\*

NVLink\*

CAPI\*

\* Upcoming

# MVAPICH2 Software Family

## High-Performance Parallel Programming Libraries

MVAPICH2	Support for InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE
MVAPICH2-X	Advanced MPI features, OSU INAM, PGAS (OpenSHMEM, UPC, UPC++, and CAF), and MPI+PGAS programming models with unified communication runtime
MVAPICH2-GDR	Optimized MPI for clusters with NVIDIA GPUs
MVAPICH2-Virt	High-performance and scalable MPI for hypervisor and container based HPC cloud
MVAPICH2-EA	Energy aware and High-performance MPI
MVAPICH2-MIC	Optimized MPI for clusters with Intel KNC

## Microbenchmarks

OMB	Microbenchmarks suite to evaluate MPI and PGAS (OpenSHMEM, UPC, and UPC++) libraries for CPUs and GPUs
-----	--

## Tools

OSU INAM	Network monitoring, profiling, and analysis for clusters with MPI and scheduler integration
OEMT	Utility to measure the energy consumption of MPI applications