MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

# Design and Characterization of Shared Address Space MPI Collectives on Modern Architectures

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

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

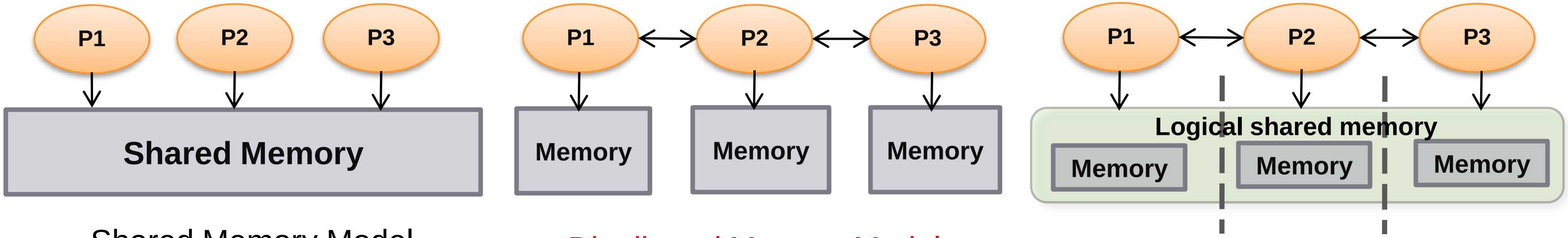
Network Based Computing Laboratory (NBCL)

The Ohio State University

# Outline

- Introduction and Motivation
- Background
  - Shared-memory vs. Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Collectives
- Performance Evaluation and Analysis
  - Contrasting different Collectives Designs
  - Comparison with other MPI libraries
  - Scaling Two-level designs via XPMEM
- 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, …

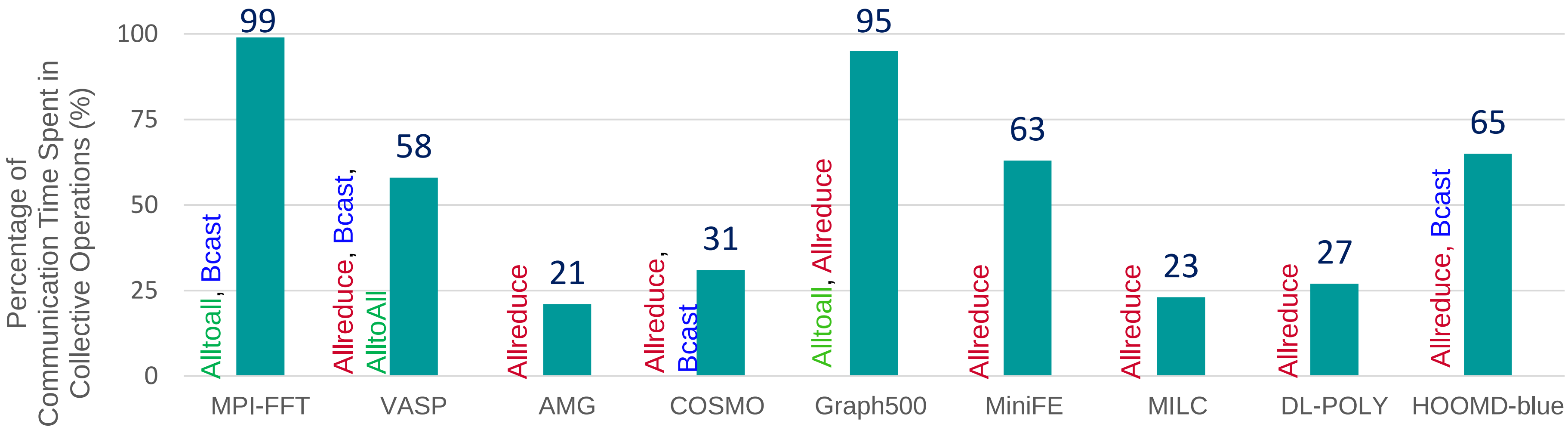
- Programming models provide abstract machine models
- Models can be mapped on different types of systems
  - e.g. Distributed Shared Memory (DSM), MPI within a node, etc.
- Programming models offer various communication primitives
  - Point-to-point (between pair of processes/threads)
  - Remote Memory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

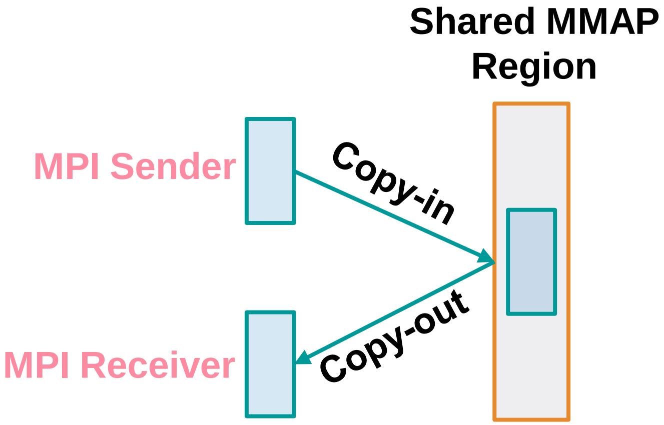
# Broad Challenges in MPI due to Architectural Diversity

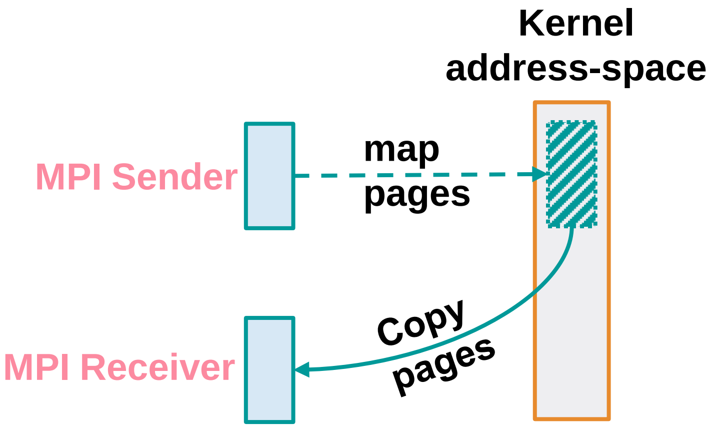
- **Can we exploit high-concurrency and high-bandwidth offered by modern architectures?**
  – better resource utilization → high throughput → faster communication performance
  – Computation and communication offloading
- **Can we design "zero-copy" and contention-free MPI communication primitives?**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

# Outline

- Introduction and Motivation
- Background
  - Shared-memory vs. Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Collectives
- Performance Evaluation and Analysis
  - Contrasting different Collectives Designs
  - Comparison with other MPI libraries
  - Scaling Two-level designs via XPMEM
- 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

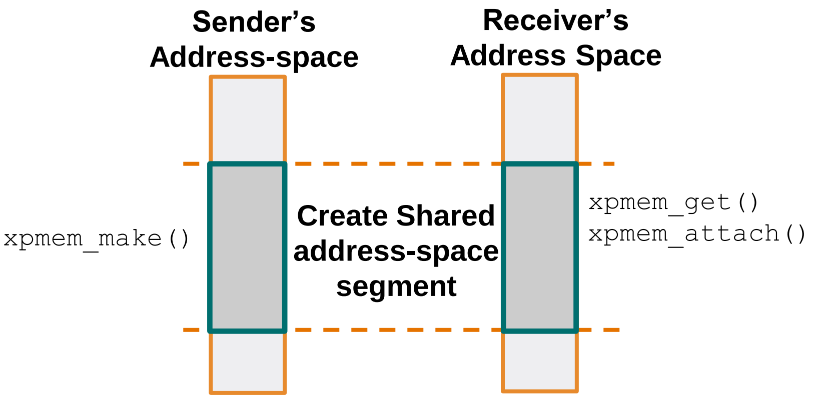
System call overhead
Requires single(a.k.a "zero") copy
Better for Large Messages

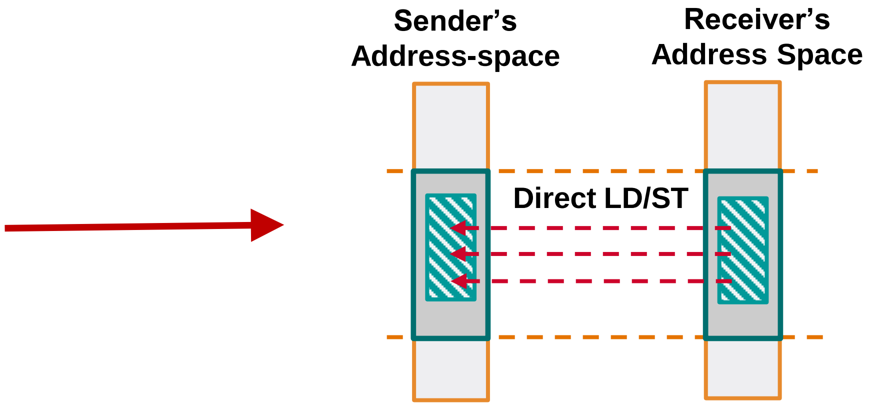
# Outline

- Introduction and Motivation
- Background
  - Shared-memory vs. Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Collectives
- Performance Evaluation and Analysis
  - Contrasting different Collectives Designs
  - Comparison with other MPI libraries
  - Scaling Two-level designs via XPMEM
- Concluding Remarks

# Shared Address-space based Communication

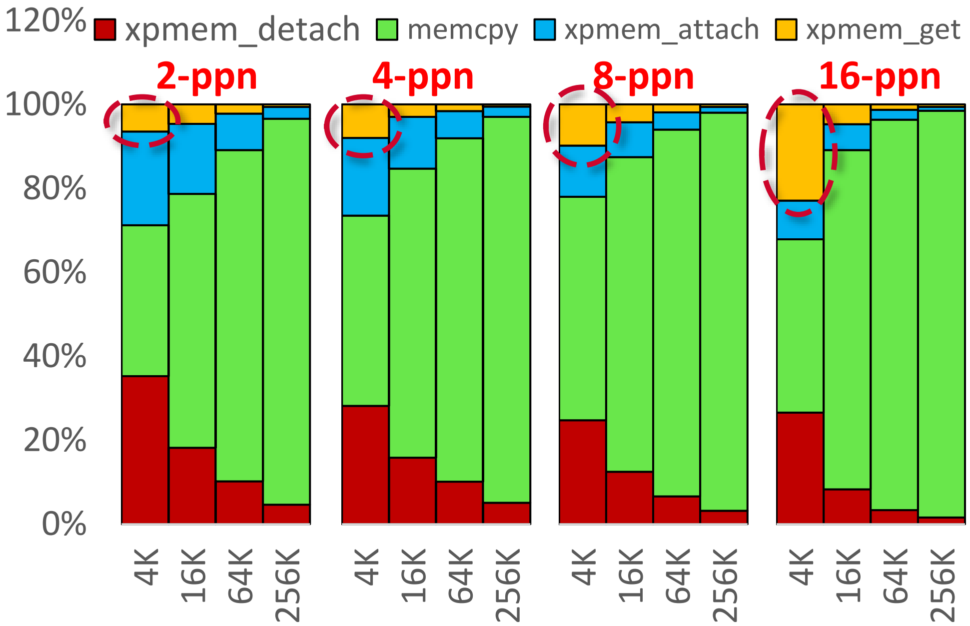
- XPMEM (https://gitlab.com/hjelmn/xpmem) --- "Cross-partition Memory"
  - Mechanisms for a process to "*attach*" to the virtual memory segment of a remote process
  - Consists of a user-space API and a kernel module
- The sender process calls "`xpmem_make()`" to create a shared segment
  - Segment information 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 <u>one-to-all latency</u> 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.

# 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

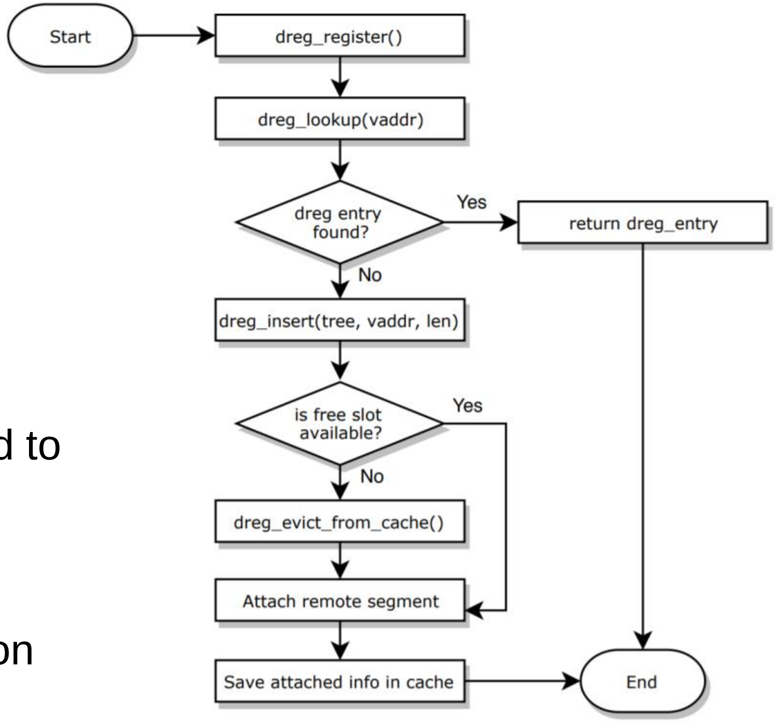
| | LiMIC | KNEM | CMA | XPMEM |
|---|---|---|---|---|
| MVAPICH2 | √ | x | √ | √ (upcoming release) |
| OpenMPI | x | √ | √ | √ |
| Intel MPI | x | x | √ | x |
| Cray MPI | x | x | √ | √ |

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

***Registration Cache!***

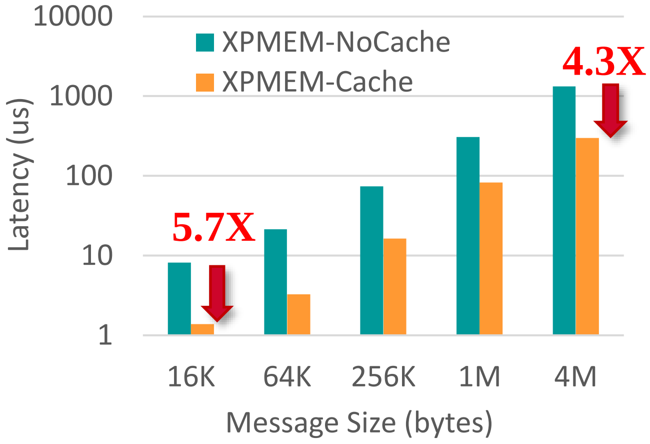
# Registration Cache for XPMEM based Communication

- Per-rank AVL tree maintains remote attached pages
- Lazy memory de-registration principle
  - Detach pages only in *MPI_Finalize()* or when capacity-miss occurs (FIFO)
  - MPI operations using same buffer do not incur XPMEM registration overheads
- Multiple calls to malloc/free on the remote buffers lead to invalid mappings
  - Linux memory allocator maintains memory pools
  - Access to attached buffer which has been freed on remote rank, is considered invalid
- Interception of malloc/free calls to invalidate remote mappings

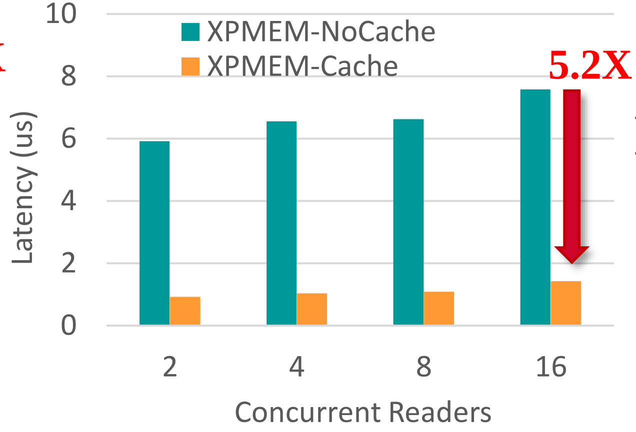



A high-level flow of the proposed Dynamic Registration Cache

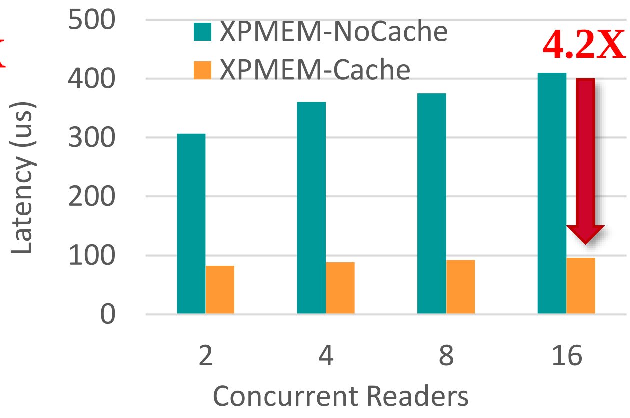
# Impact of Registration Cache on the Performance of XPMEM based Point-to-point Communication




Two-process latency at varying messages



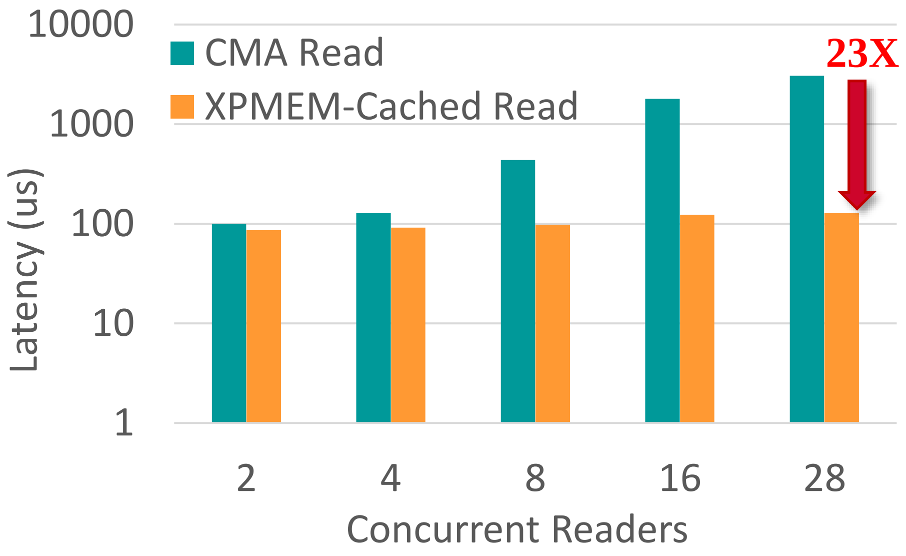

Multi-process latency at 16KB message



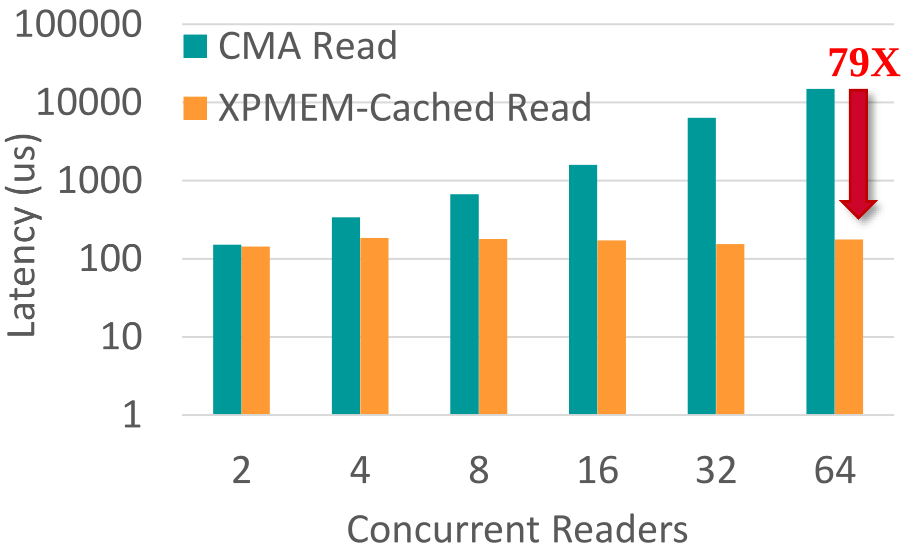

Multi-process latency at 1MB message

- 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 CMA vs. XPMEM (with reg-cache) based one-to-all 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 vs. Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Collectives
- Performance Evaluation and Analysis
  - Contrasting different Collectives Designs
  - Comparison with other MPI libraries
  - Scaling Two-level designs via XPMEM
- Concluding Remarks

# Existing 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

# Design Overview of XPMEM based Direct MPI Collectives

- All ranks in communicator call xpmem_make() to generate segment id
- All ranks in communicator exchange buffer, len, and segment id information
- All ranks in communicator attach to remote buffers of the peer ranks
- After attachment, direct load/store access is permitted
- An intra-node barrier is enforced to ensure correctness and ordering
- Finally, a direct XPMEM collective implementation is called e.g., Bcast

```
/* Share vaddr with peer ranks */
Exchange_buffer_addresses();                          ▷ Step-1
  /* Create remote buffers mapping */;                ▷ Step-2
foreach rem_rank in SMP rank list do
    if rank ≠ local then
        Dreg_entry d;
        /* Find in local registration cache */
        d ← AVL_lookup(rem_rank, rbuf, len);
        if found then
            return d;
        else
            /* create remote page mappings */
            d ← XPMEM_Attach(rbuf, len);
            /* Cache dreg entry in local tree */
            AVL_insert(d, avl_roots[rem_rank]);
            return d ;
        end
    end
end
synchronize();                                        ▷ Step-3
/* Call direct Load/Store based algorithm */
MV2_XPMEM_Direct_coll*(...);                          ▷ Step-4
```

High-level Overview of XPMEM base Direct MPI Collectives Implementation

# 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 3,000 organizations in 88 countries**
  - **More than 538,000 (> 0.5 million) downloads from the OSU site directly**
  - Empowering many TOP500 clusters (Nov '18 ranking)
    - 3rd ranked 10,649,640-core cluster (Sunway TaihuLight) at NSC, Wuxi, China
    - 14th, 556,104 cores (Oakforest-PACS) in Japan
    - 17th, 367,024 cores (Stampede2) at TACC
    - 27th, 241,108-core (Pleiades) at NASA and many others
  - Available with software stacks of many vendors and Linux Distros (RedHat, SuSE, and OpenHPC)
  - **http://mvapich.cse.ohio-state.edu**


**Partner in the upcoming TACC Frontera System**

- Empowering Top500 systems for over a decade

# Evaluation Methodology and Cluster Testbeds

Hardware Specification of Cluster Testbeds

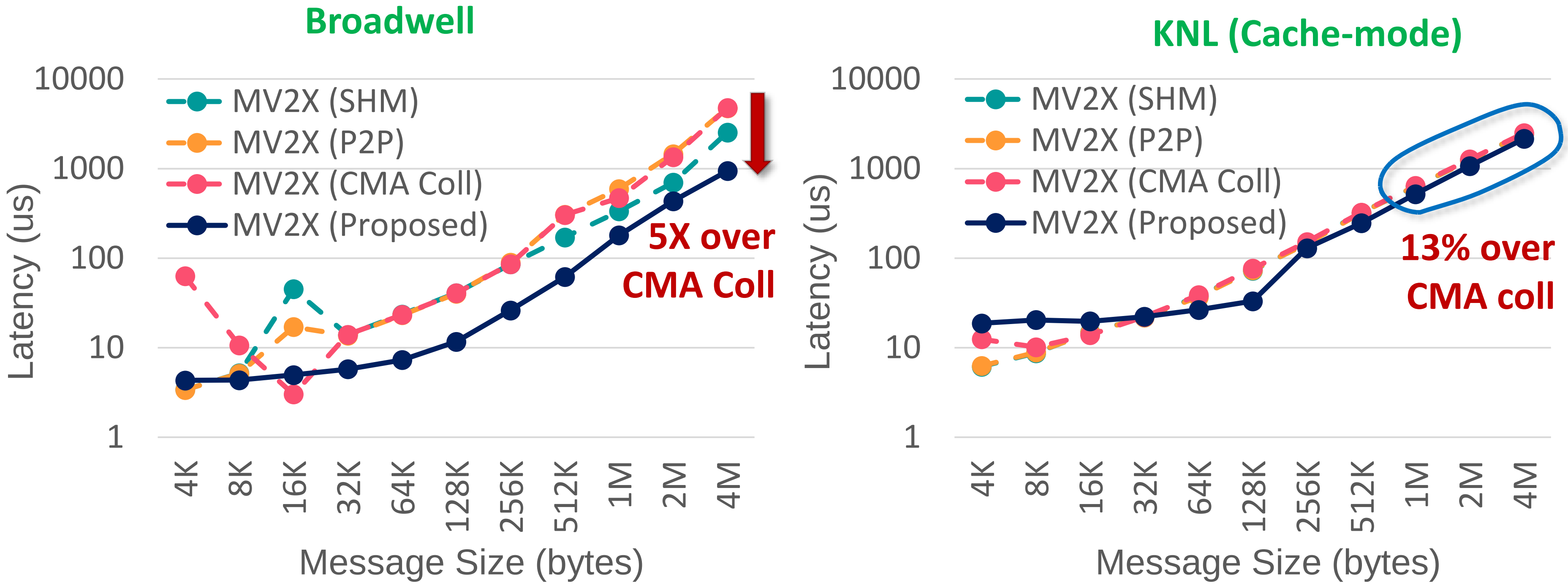
| Specification | Xeon | Xeon Phi |
|---|---|---|
| Processor Family | Intel Broadwell | Knights Landing |
| Processor Model | E5 2680v4 | KNL 7250 |
| Clock Speed | 2.4 GHz | 1.4 GHz |
| No. of Sockets | 2 | 1 |
| Cores Per Socket | 14 | 68 |
| Threads Per Core | 1 | 4 |
| RAM (DDR) | 128 GB | 96 GB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

- XPMEM based designs, implemented on MVAPICH2 is referred to as “MV2 (Proposed)”
- Comparison against various collectives design in MVAPICH2
- Comparison against other MPI libraries e.g., MVPAPICH2-2.3b, Intel MPI v2018.1.163, and OpenMPI v3.0.1

# Outline

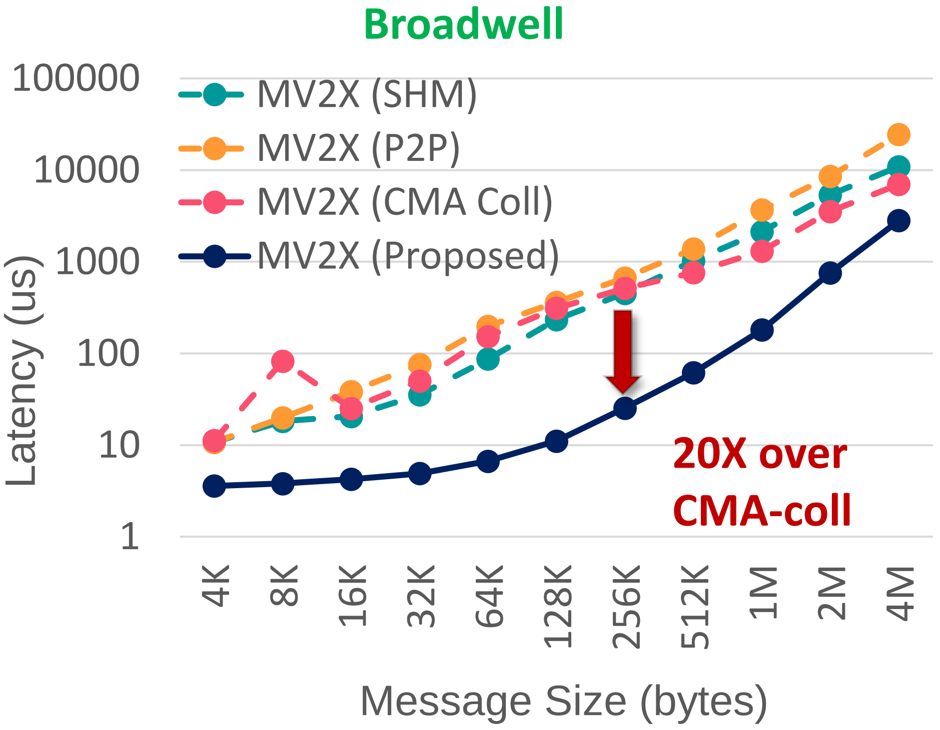
- Introduction and Motivation
- Background
  - Shared-memory vs. Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Collectives
- Performance Evaluation and Analysis
  - Contrasting different Collectives Designs
  - Comparison with other MPI libraries
  - Scaling Two-level designs via XPMEM
- Concluding Remarks

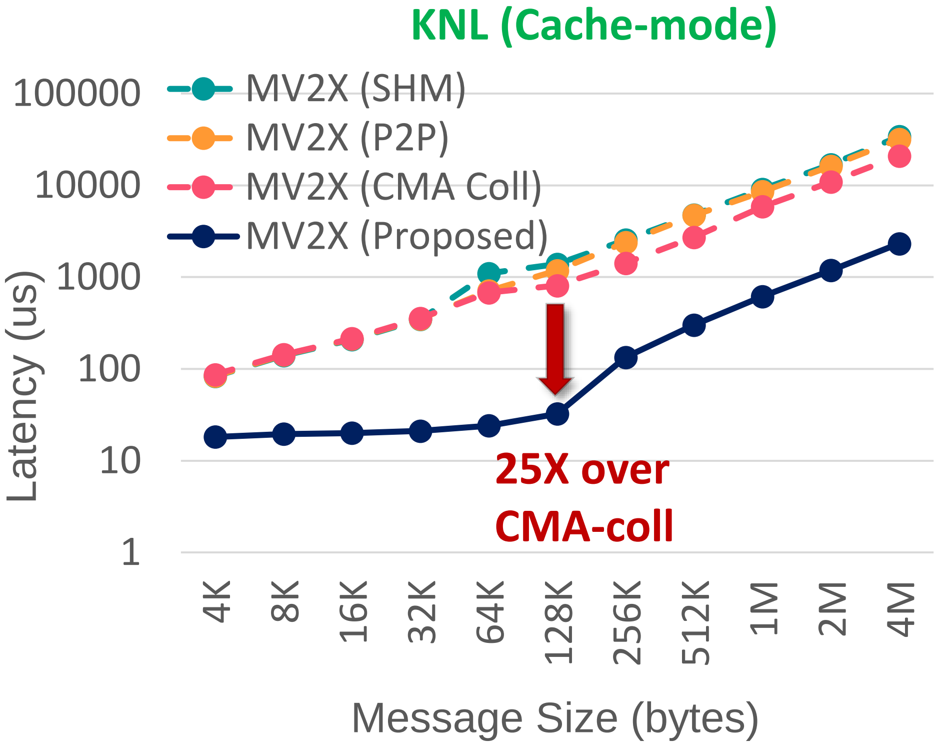
# Contrasting with Bcast Designs in MVAPICH2




- On Broadwell, up to 5X improvement over direct CMA collectives.
- Up to 13% improvement in Bcast latency over CMA collectives on KNL.

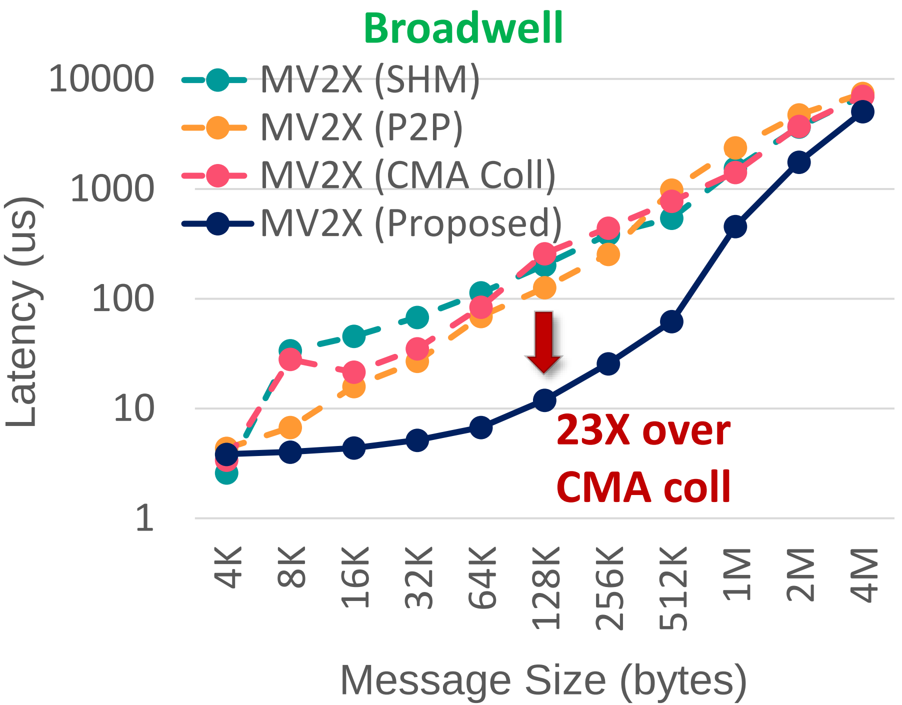
# Performance of Scatter on Broadwell and KNL

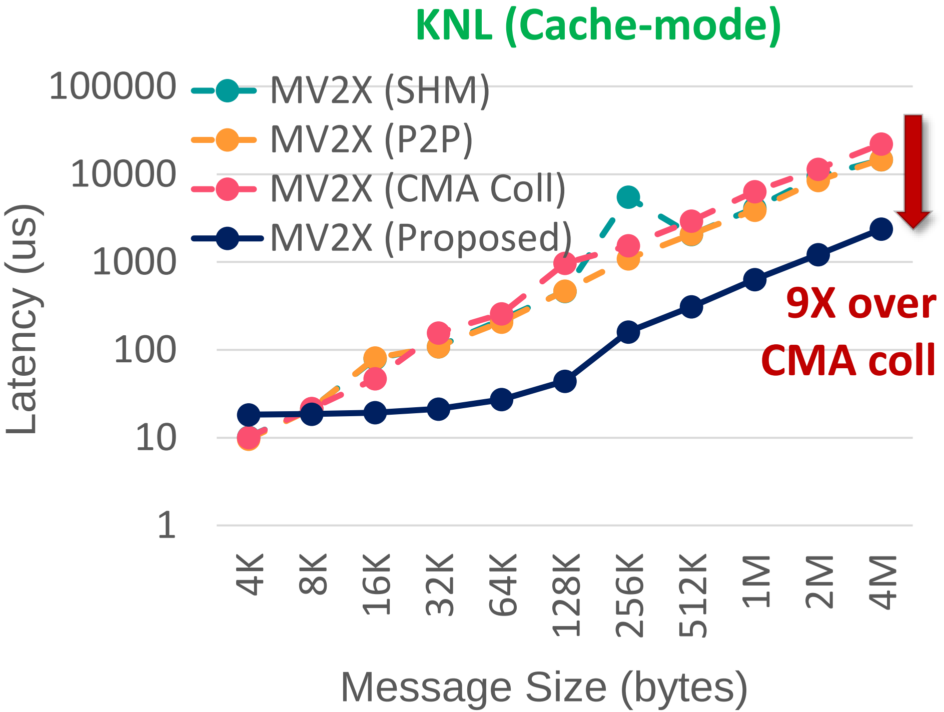



- XPMEM based direct Scatter achieve up to 20X and 25X improvement over direct CMA collectives on Broadwell and KNL, respectively.

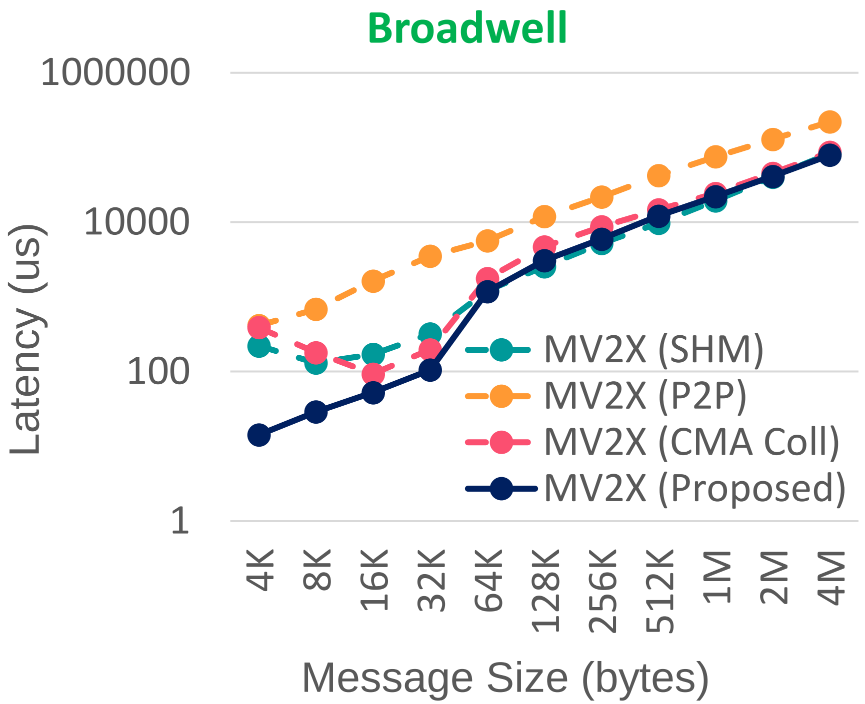
# Performance of Gather on Broadwell and KNL

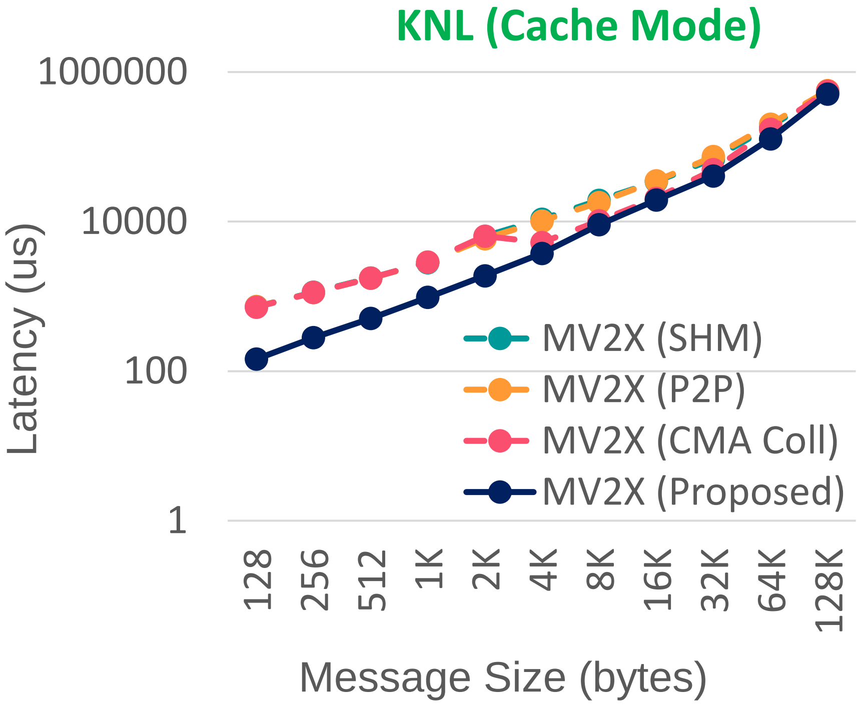



- XPMEM based direct Gather achieve up to 23X and 9X improvement over direct CMA collectives on Broadwell and KNL, respectively.

# Performance of Alltoall on Broadwell and KNL



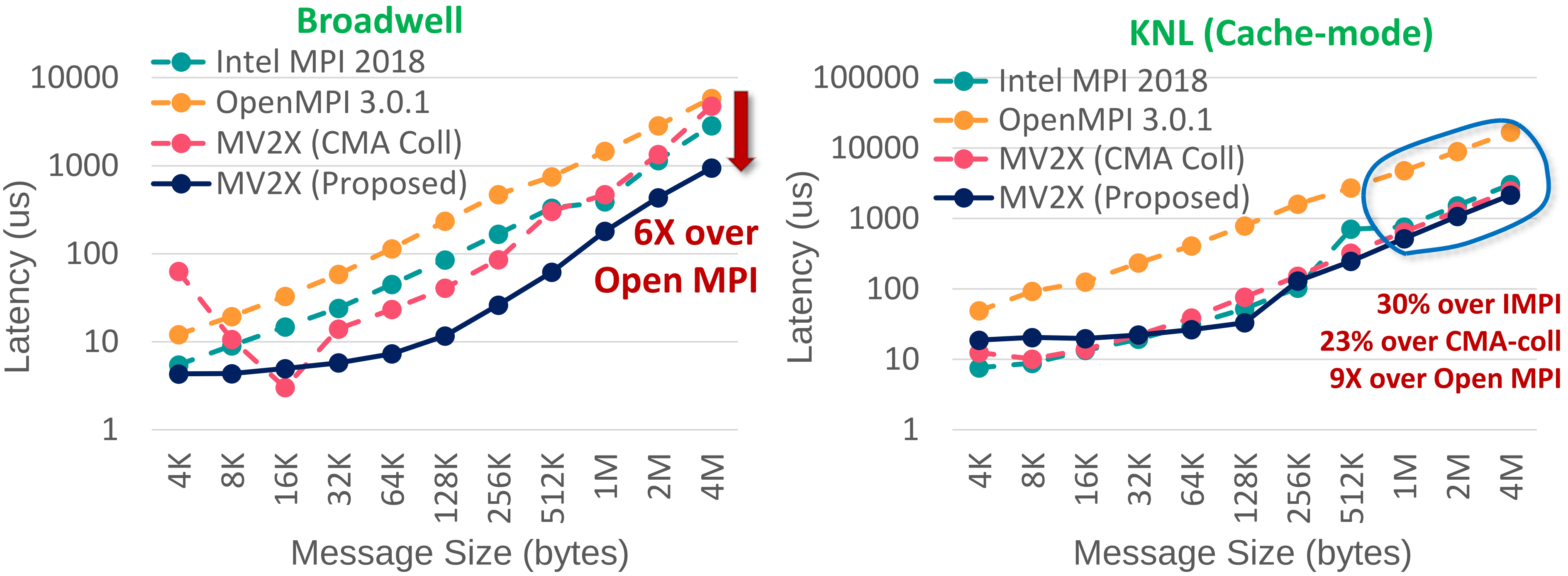




- XPMEM based direct Gather achieve up to 20X and 25X improvement over direct CMA collectives on Broadwell and KNL, respectively.

# Outline

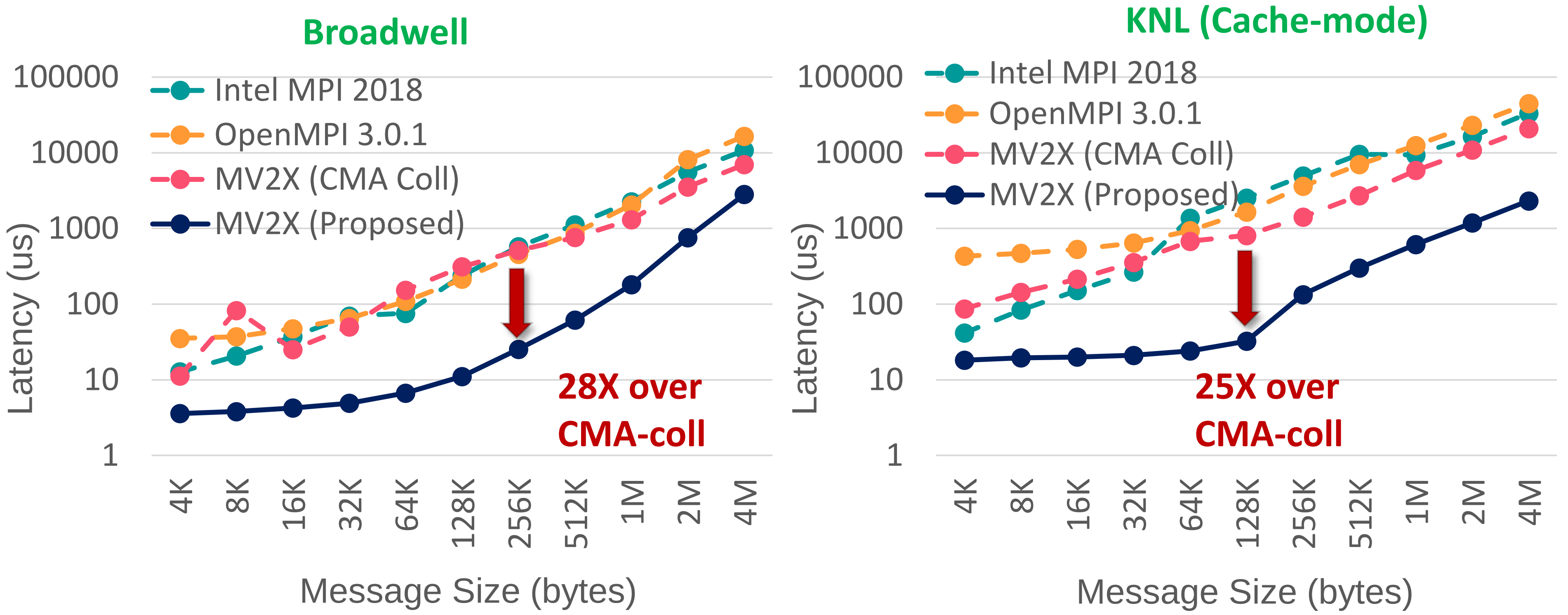
- Introduction and Motivation
- Background
  - Shared-memory vs. Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Collectives
- Performance Evaluation and Analysis
  - Contrasting different Collectives Designs
  - Comparison with other MPI libraries
  - Scaling Two-level designs via XPMEM
- Concluding Remarks

# Performance of Bcast on Broadwell and KNL



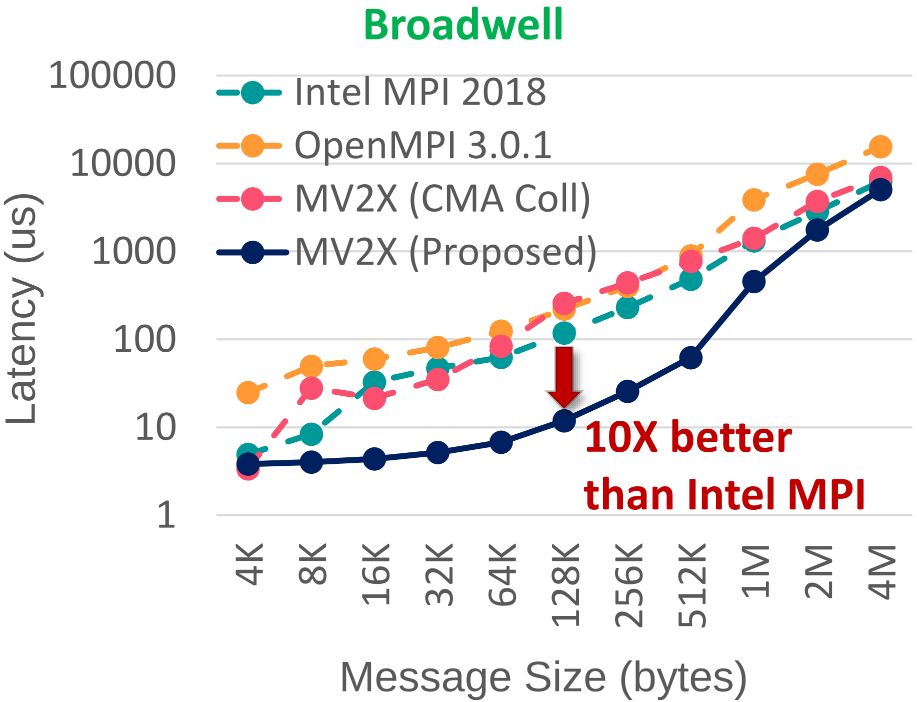

- Up to 6X improvement over Open MPI on Broadwell.
- Up to 30%, 23%, and 9X improvement over IMPI, direct CMA collectives, and Open MPI, respectively, on KNL

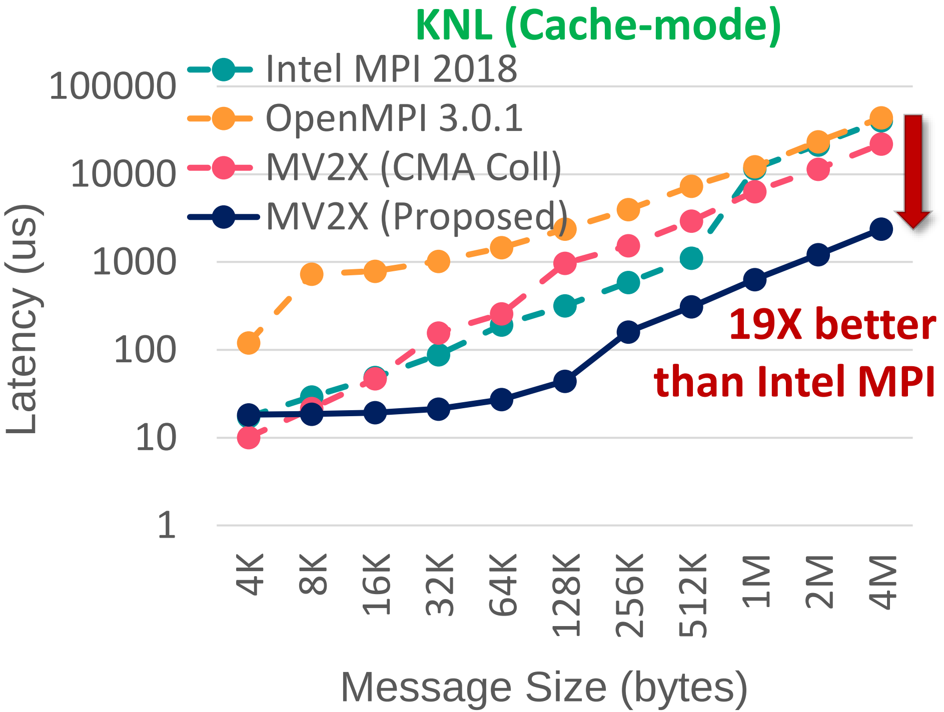
# Performance of Scatter on Broadwell and KNL




- Propose XPMEM designs achieved up to 28X and 25X improvement over direct CMA collectives on Broadwell and KNL, respectively.

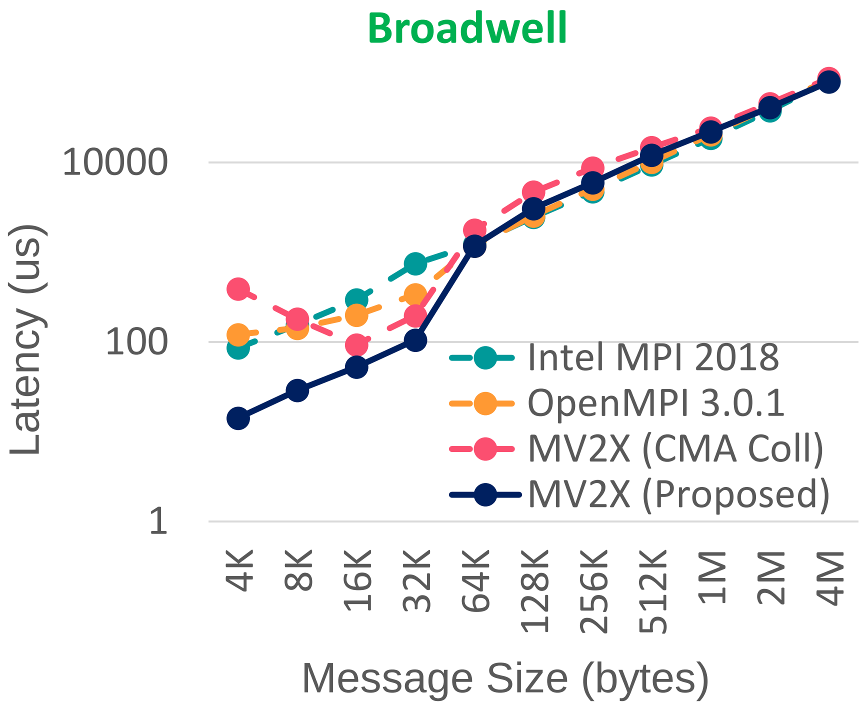
# Performance of Gather on Broadwell and KNL

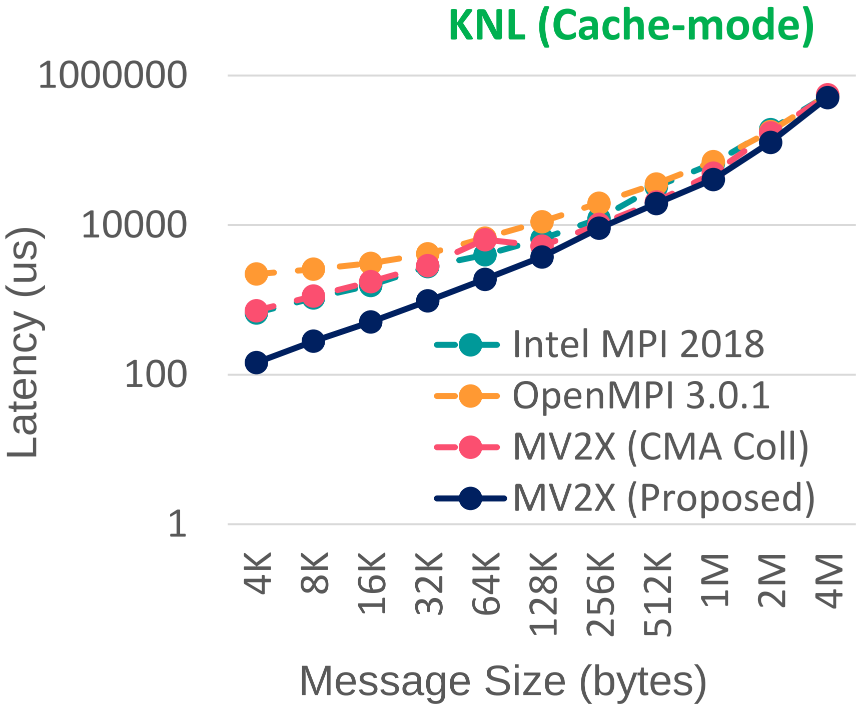



- Up to 10X improvement over Intel MPI on Broadwell.
- Up to 19X better Gather latency over Intel MPI is observed.

# Performance of Alltoall on Broadwell and KNL



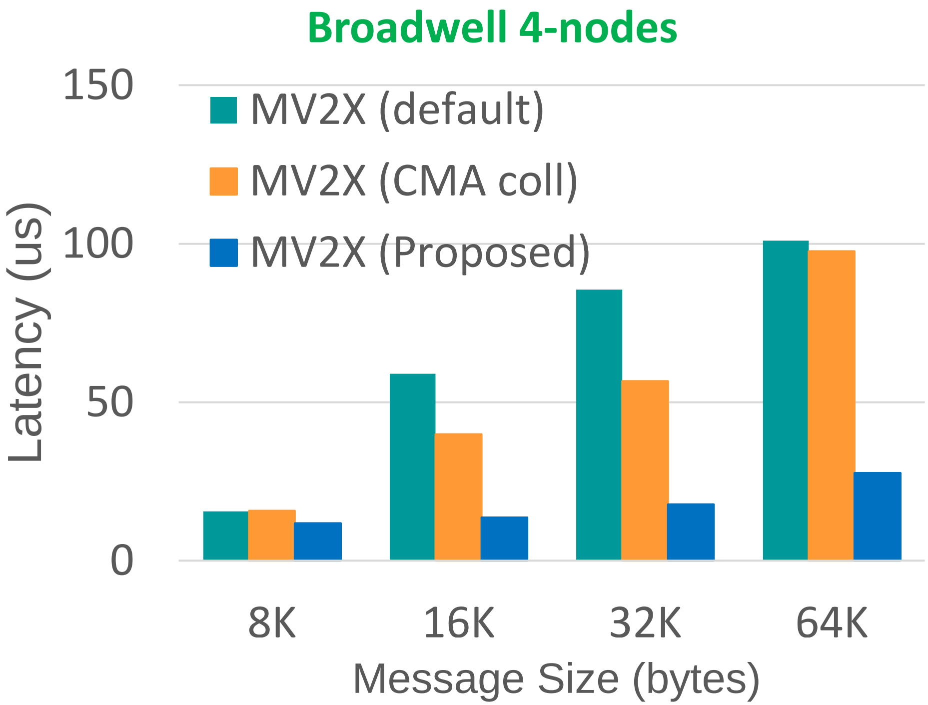



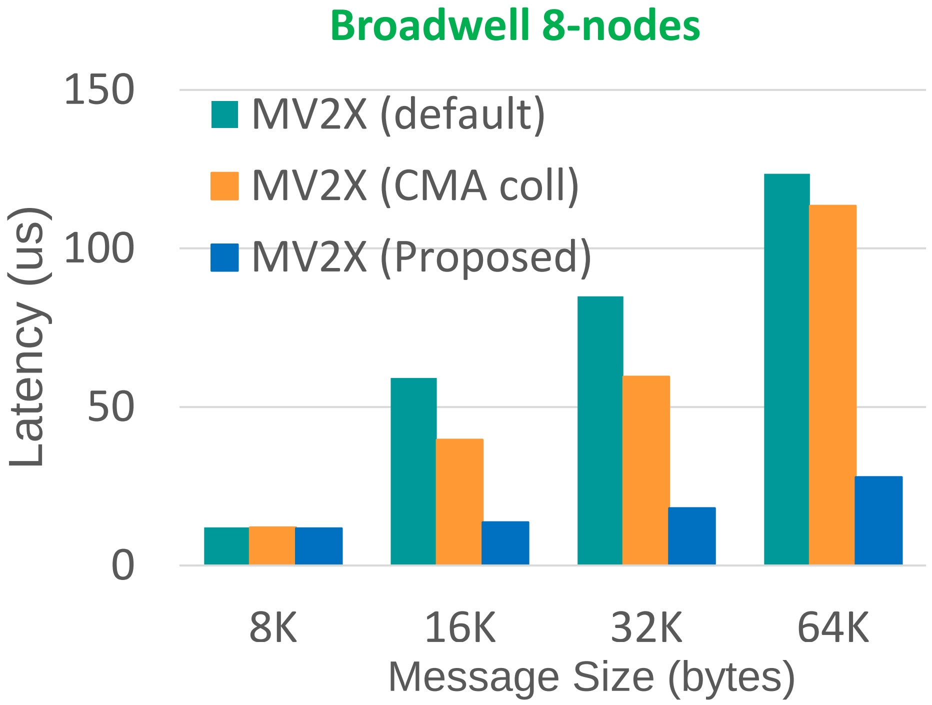

- Alltoall performance of direct algorithms depend on cache size
- For small to medium message, good improvement is observed over other libraries

# Outline

- Introduction and Motivation
- Background
  - Shared-memory vs. Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Collectives
- Performance Evaluation and Analysis
  - Contrasting different Collectives Designs
  - Comparison with other MPI libraries
  - Scaling Two-level designs via XPMEM
- Concluding Remarks

# Impact on Inter-node Scaling via two-level Collective Designs: MPI_Gather




- Hierarchical collectives use XPMEM based direct algorithms for intra-node phases.
- Proposed XPMEM collectives achieve scalable performance with multiple nodes.

# Outline

- Introduction and Motivation
- Background
  - Shared-memory vs. Kernel-assisted Communication
- Shared Address-space (XPMEM) based Communication
  - Quantifying Performance Bottlenecks
  - Mitigating the Overheads with Proposed Designs
- Designing XPMEM based Collectives
- Performance Evaluation and Analysis
  - Contrasting different Collectives Designs
  - Comparison with other MPI libraries
  - Scaling Two-level designs via XPMEM
- Concluding Remarks

# Concluding Remarks

- Characterized the performance trade-offs involved in designing Shared address-space based collectives communication in MPI
  - Registration cache based schemes to overcome performance bottlenecks
  - Alleviate the overheads posed by the memory allocator interactions with reg-cache
- Design and Implementation of MPI collectives using Shared Address-spaces
  - Demonstrated the performance benefits of new MPI_Bcast, MPI_Scatter, MPI_Gather, MPI_Allgather, and MPI_Alltoall multi- and many-core architectures
- Demonstrated the efficacy of the proposed solutions for various microbenchmarks
  - Improved performance over state-of-the-art collectives design in MVAPICH2
  - Significant improvement over prevalent MPI libraries
- We plan to expand our designs to other architectures e.g., ARM etc.

# Thank You!

hashmi.29@osu.edu

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

MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

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

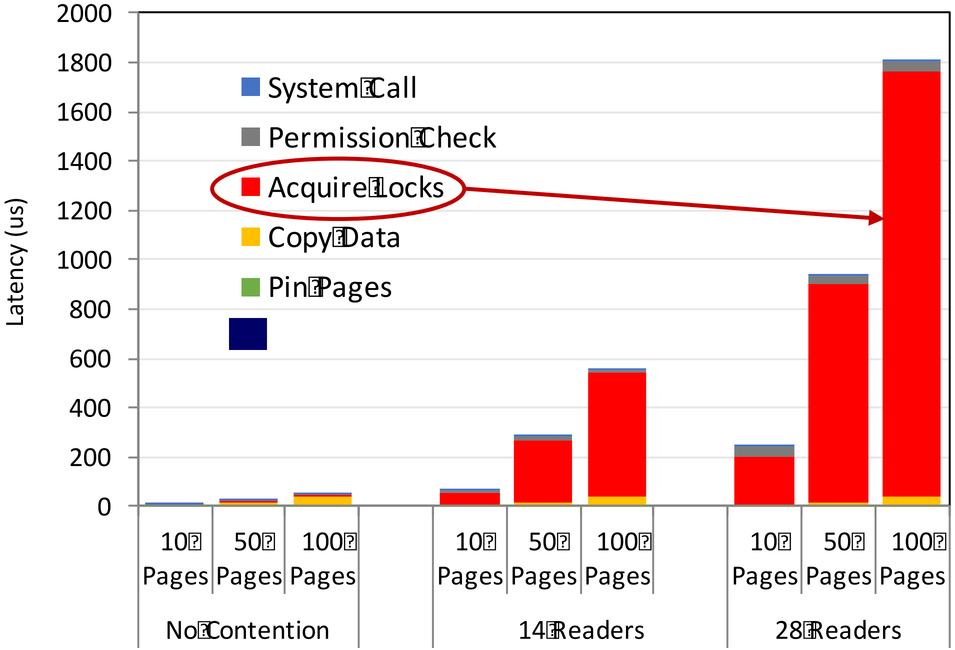
HiBD
High-Performance Big Data

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

HiDL
High-Performance Deep Learning

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

# Breakdown of a CMA Read operation



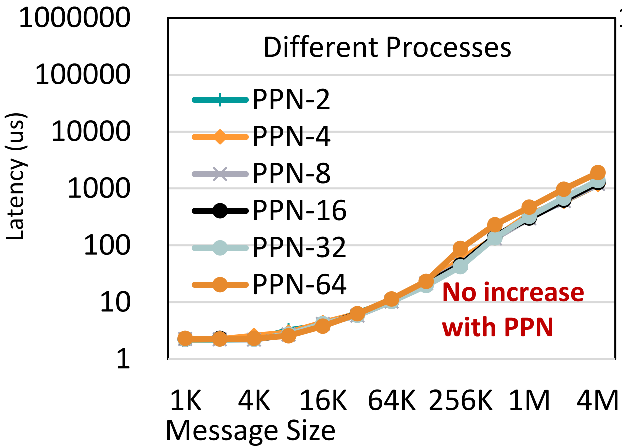

- CMA relies on `get_user_pages()` function
- Takes a page table lock on the target process
- Lock contention increases with number of concurrent readers
- Over 90% of total time spent in lock contention

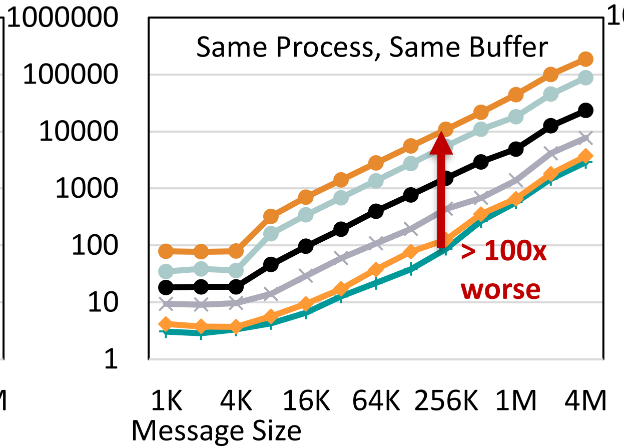
- One-to-all communication on Broadwell, profiled using `ftrace`

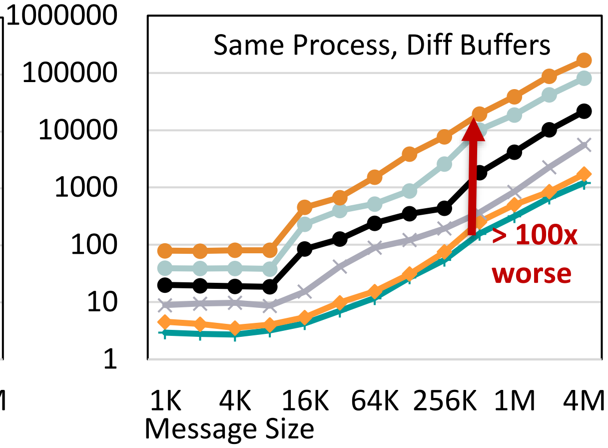
- Lock contention is the root cause of performance degradation
- Present in other kernel-assisted schemes such as KNEM, LiMiC as well

*S. Chakraborty, H. Subramoni, and D. K. Panda,* Contention Aware Kernel-Assisted MPI Collectives for Multi/Many-core Systems, *IEEE Cluster '17, BEST Paper Finalist*

# Impact of Collective Communication Pattern on CMA Collectives



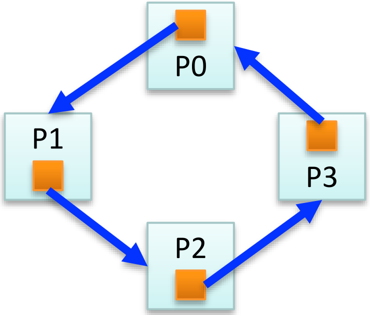



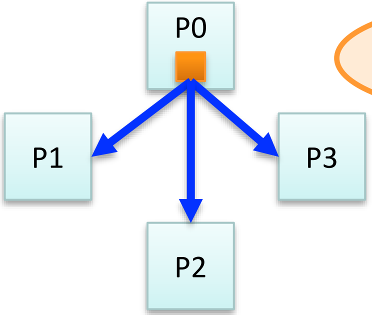




**All-to-All – Good Scalability**

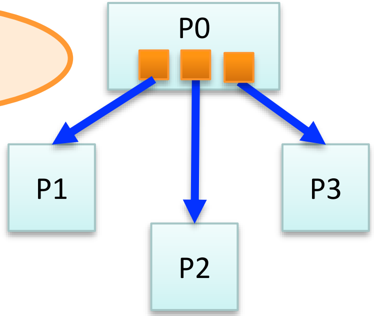
**One-to-All - Poor Scalability**

**One-to-All – Poor Scalability**



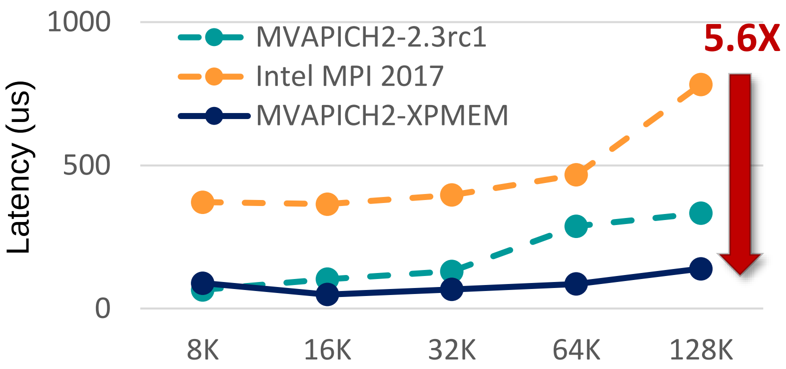

**Contention is at Process level**

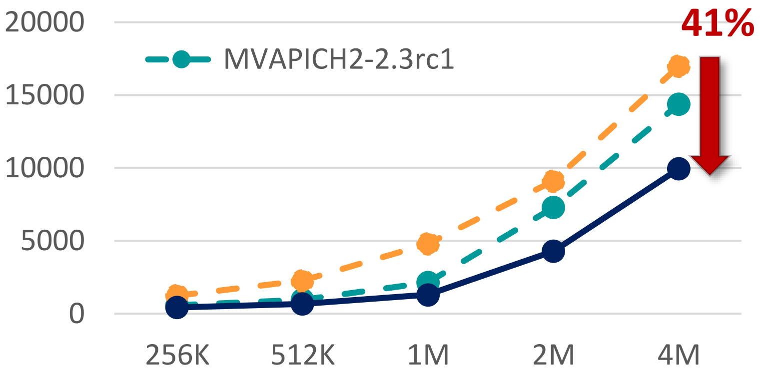



*S. Chakraborty, H. Subramoni, and D. K. Panda,* Contention Aware Kernel-Assisted MPI Collectives for Multi/Many-core Systems, *IEEE Cluster '17, BEST Paper Finalist*

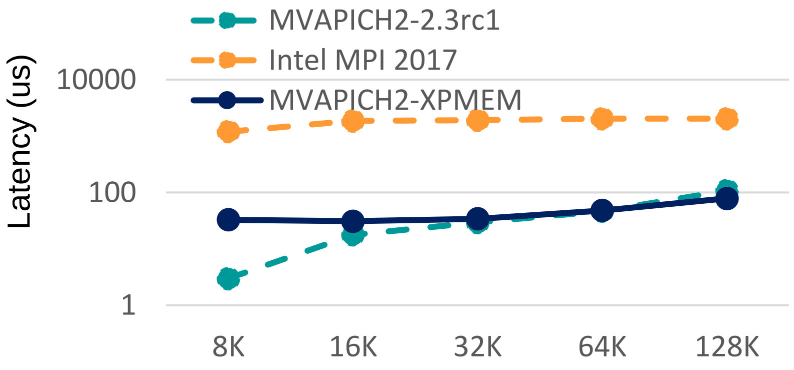
# Scalability Evaluation on Broadwell Cluster

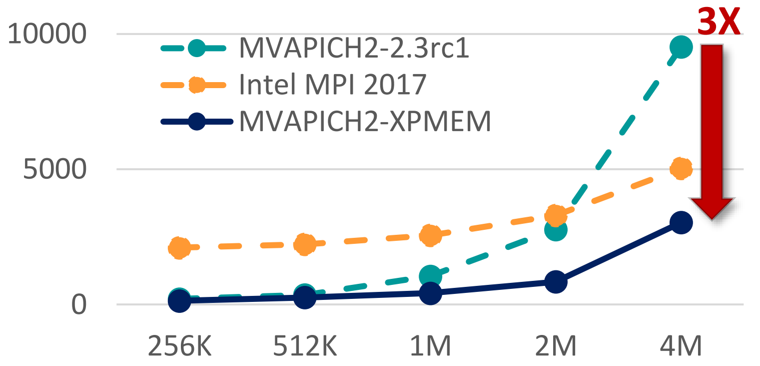
OSU_Allreduce




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

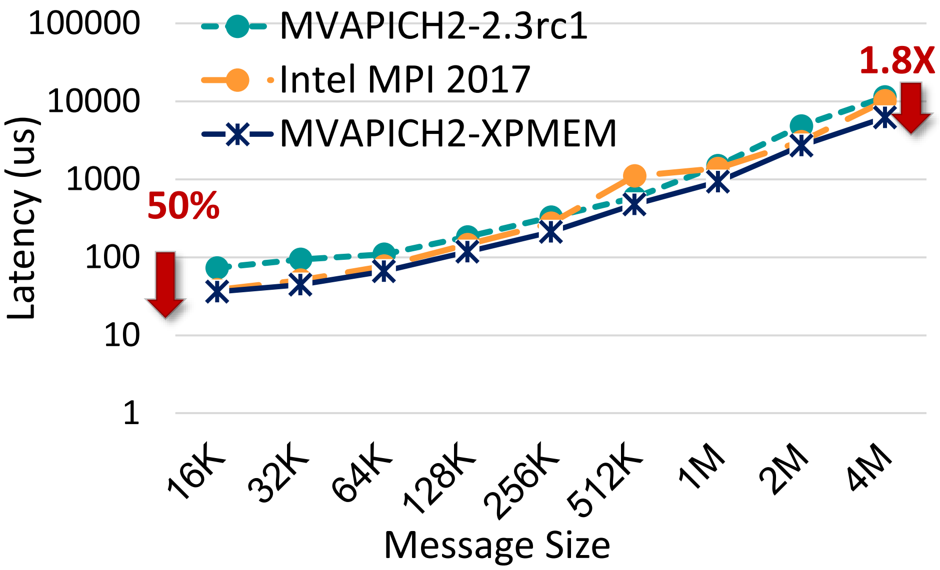
# Registration Cache Miss-rate Analysis on Various Benchmarks

| 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 |

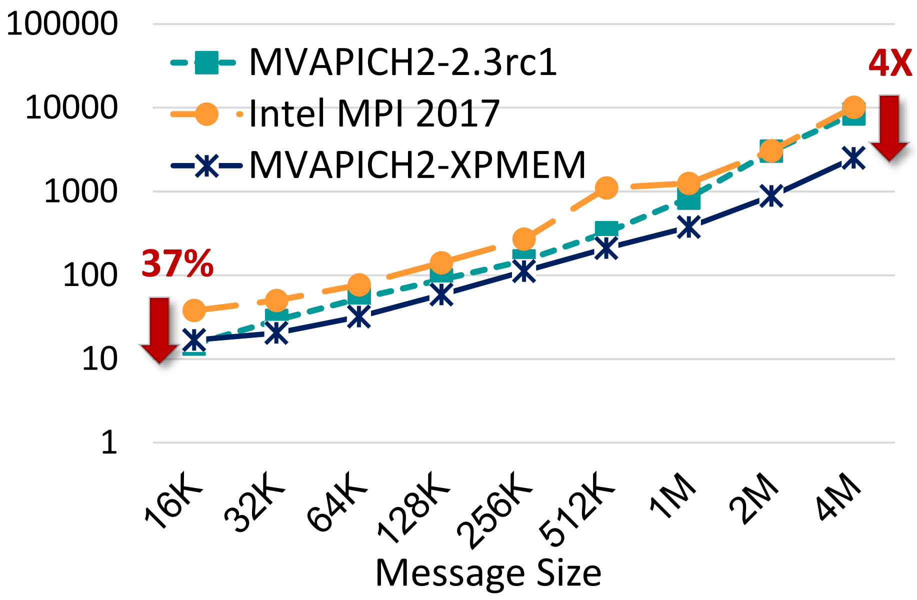
Registration cache Hit/miss (per-process) analysis on Broadwell System

- 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

# Reduction Collectives on Broadwell Cluster



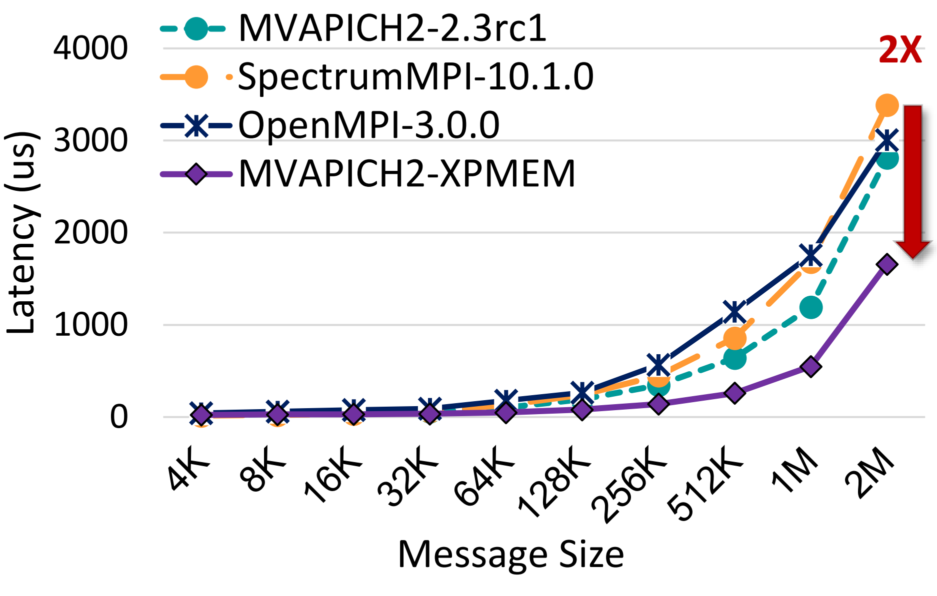

OSU_Allreduce (Broadwell 256 procs)



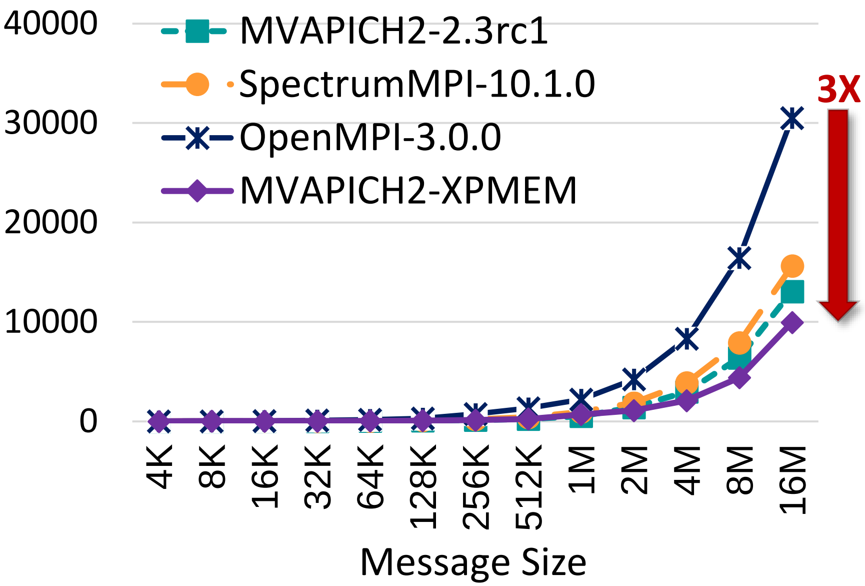

OSU_Reduce (Broadwell 256 procs)

- "*Shared Address Space*"-based true *zero-copy* Reduction collective designs in MVAPICH2
- Offloaded computation/communication to peers ranks in reduction collective operation
- Up to **4X** improvement for 4MB Reduce and up to **1.8X** improvement for 4M AllReduce

# Reduction Collectives on OpenPOWER



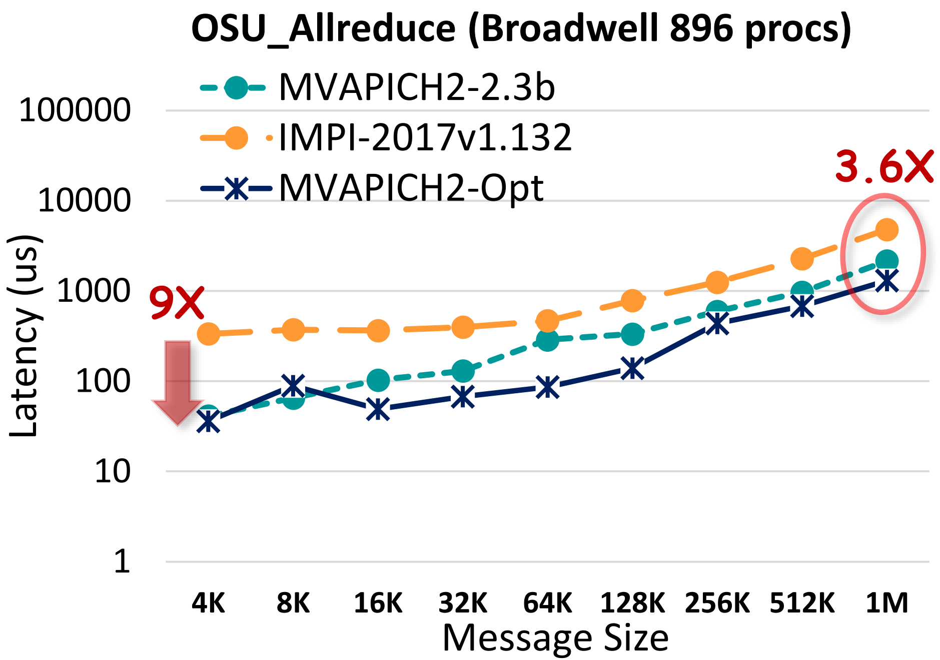

OSU_Allreduce (POWER8 nodes=2, ppn=20)

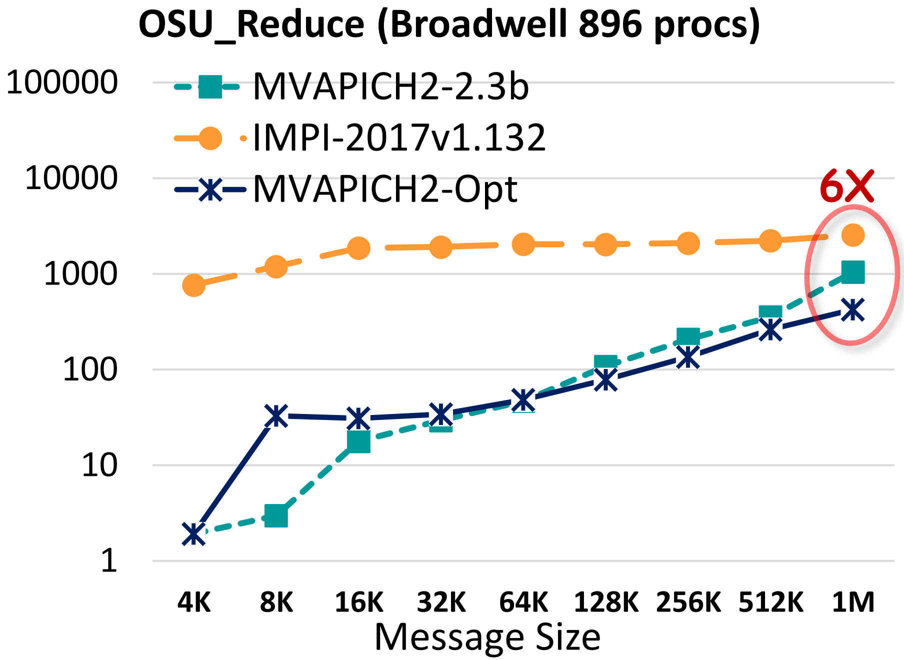



OSU_Reduce (POWER8 nodes=2, ppn=20)

- OpenPOWER system with 2xPOWER8 nodes
- Significant performance gains over OpenMPI and Spectrum MPI
  - Up to **2X** improvement for 4MB Allreduce and up to **3X** improvement for 4M Reduce

# Reduction Collectives at Scale




- Shared Address Space based true zero-copy Reduce/AllReduce designs in MVAPICH2
- Significant performance improvement over existing designs by avoiding memory copies and sharing computation/communication to peers ranks in collective operation