Designing Next-Generation Communication Middlewares for Many-Core Architectures

Jahanzeb Maqbool Hashmi

OSU Booth Talk at SC'19

Department of Computer Science and Engineering

The Ohio State University

Overview

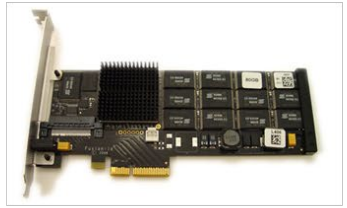
- **Introduction**
- Problem Statement
- Detailed Designs and Results
  - Shared Address Space Communication Substrate
  - Direct Zero-copy MPI Collectives
  - Efficient Zero-copy MPI Datatypes
  - Kernel-assisted Communication On-loading
- Broader Impact on HPC
- Expected Contributions

Trends in Modern HPC Architecture

Multi/ Many-core Processors

High Performance Interconnects – InfiniBand, Omni-Path
<1usec latency, 100Gbps Bandwidth>

Accelerators / Coprocessors high compute density, high performance/watt



SSD, NVMe-SSD, NVRAM

- Multi-core/many-core technologies
- High Performance Interconnects
- High Performance Storage and Compute devices
- Variety of programming models (MPI, PGAS, MPI+X)

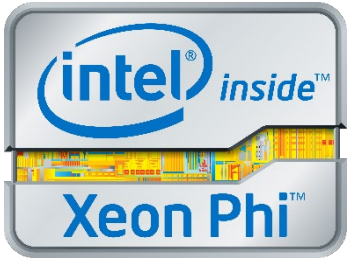
Summit

Sierra

Sunway TaihuLight

K - Computer

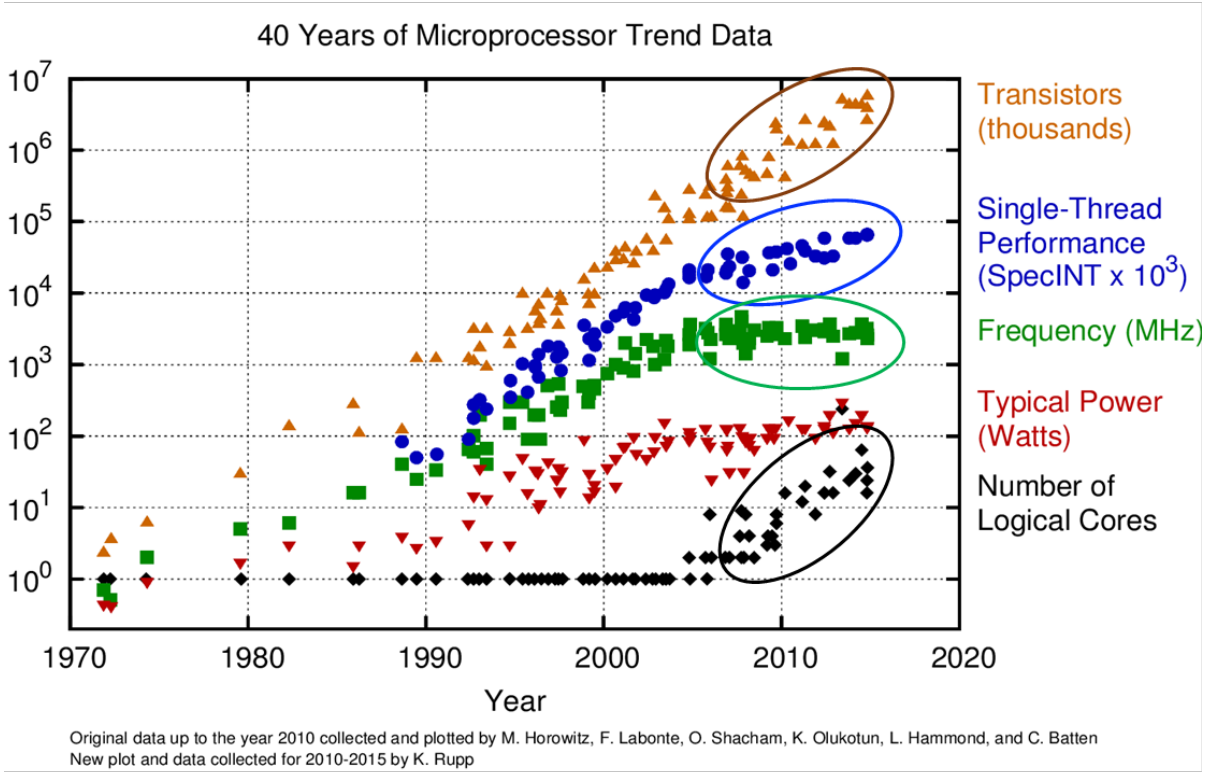
Diversity in HPC Architectures



	Knights Landing (KNL)	Xeon	OpenPower
Clock Speed	Low	High	Very High
Core count	High (64-72)	medium (8-24)	Low (8-20)
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

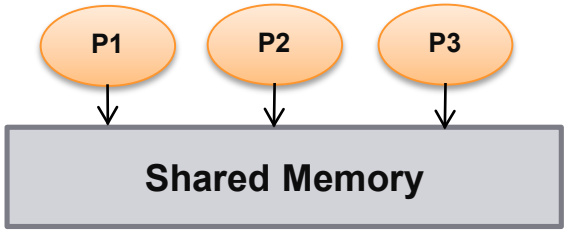
CPU Scaling Trends over Past Decades



https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/

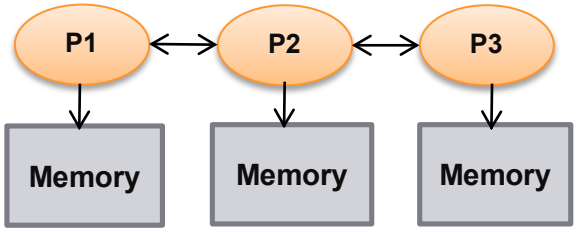
- Single thread performance increasing slowly
- Frequency increase has stopped
- Number of transistors continue to grow
- Number of cores rapidly increasing
- More compute power in small number of nodes

Parallel Programming Models Overview



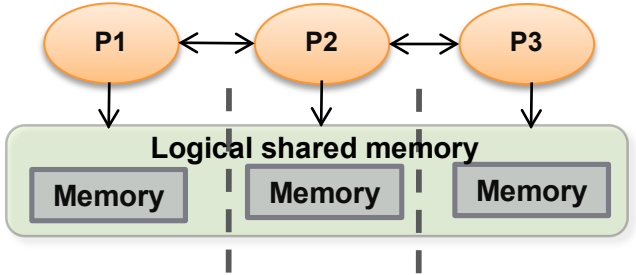
Shared Memory Model

SHMEM, DSM



Distributed Memory Model

MPI (Message Passing Interface)



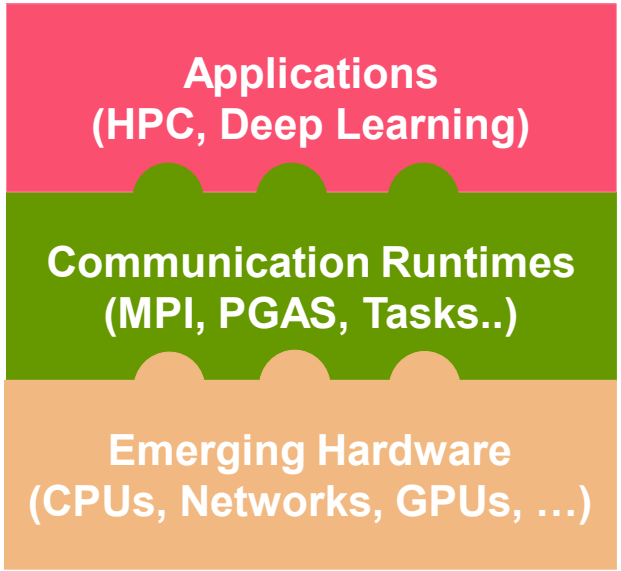
Partitioned Global Address Space (PGAS)

OpenSHMEM, UPC, UPC++, CAF …

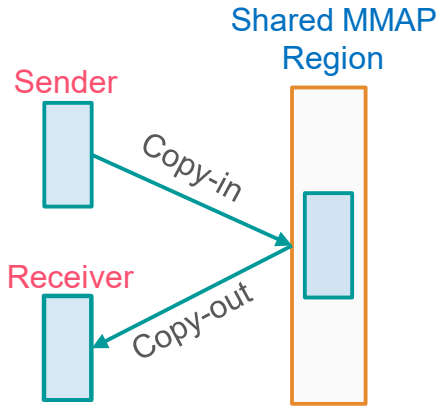
- Programming models provide abstract machine models
- Models can be mapped on different types of systems
  - e.g. Distributed Shared Memory (DSM), MPI within a node, etc.
- MPI is the de-facto programming model for writing parallel applications
- MPI offers various communication primitives and data layouts
  - **Point-to-point, Collectives**, Remote Memory Access
  - **Derived Datatypes**

Research Problems

- **Emerging multi-/many-cores e.g., Xeon Phi, OpenPOWER, etc. bringing new challenges**
  - High core-density
  - low memory available per core
- **Diverse application requirements**
  - Deep Learning (Allreduce)
  - Derived datatypes
- **Communication middlewares are under-optimized**
  - Lack of contention-free, truly zero-copy communication
  - Overheads of MPI derived datatypes
  - Lack of parallelism in communication

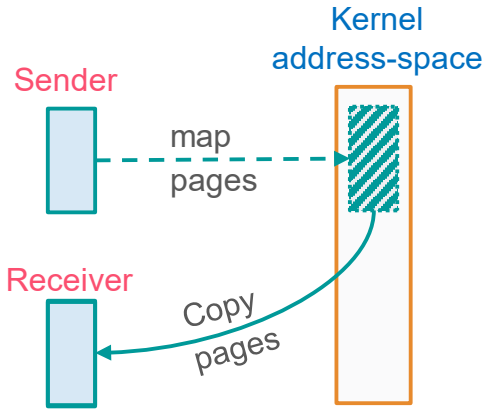


Motivating Example 1 – Intra-node Point-to-point MPI Communication



Shared Memory (POSIX)

Requires two copies
No system call overhead
Better for Small Messages

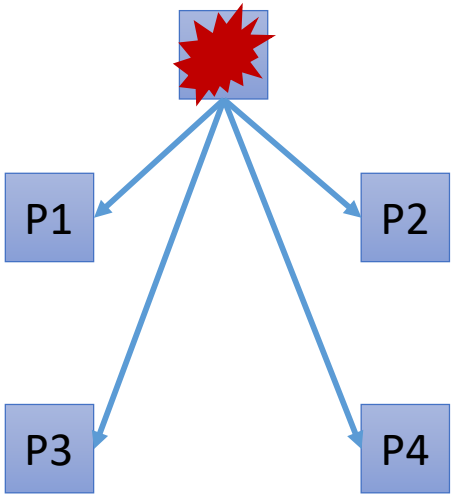


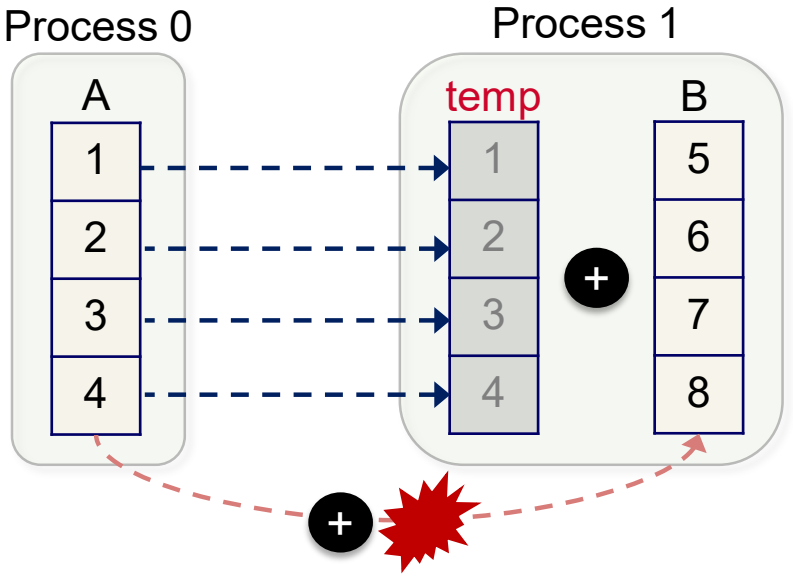
Kernel-mapping (CMA/LiMIC/KNEM)

System call overhead
Lack of Load/store access
single (a.k.a “zero”) copy
Better for Large Messages

We require over-head free, user-space, load/store based inter-process communication mechanism, also called “**Shared Address Space**” communication

Motivating Example 2 – Intra-node Collective Communication in MPI



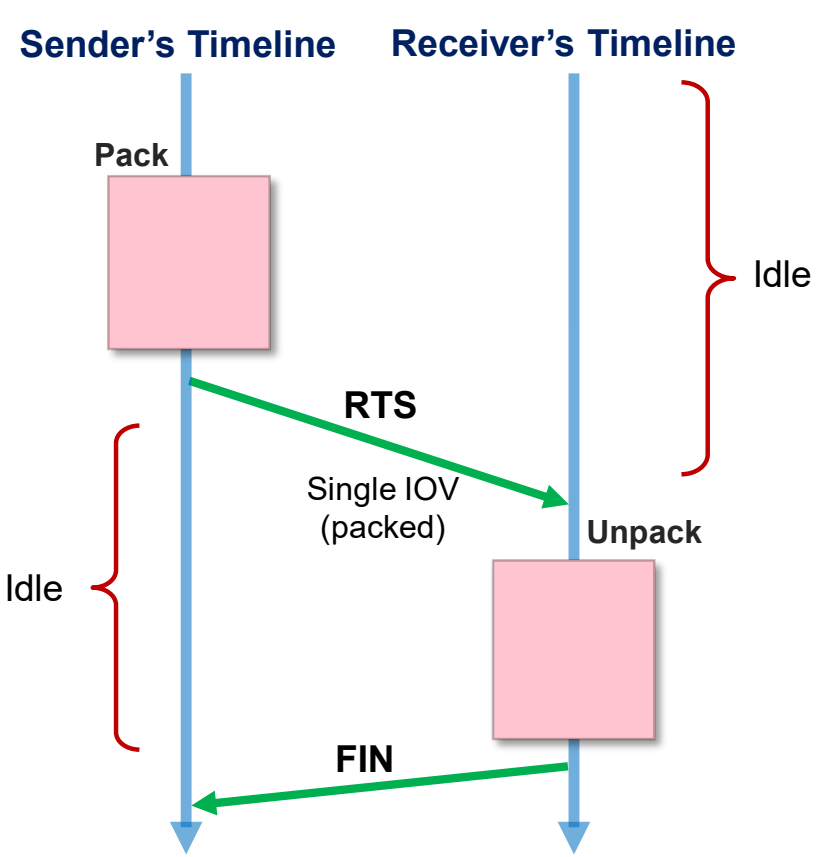


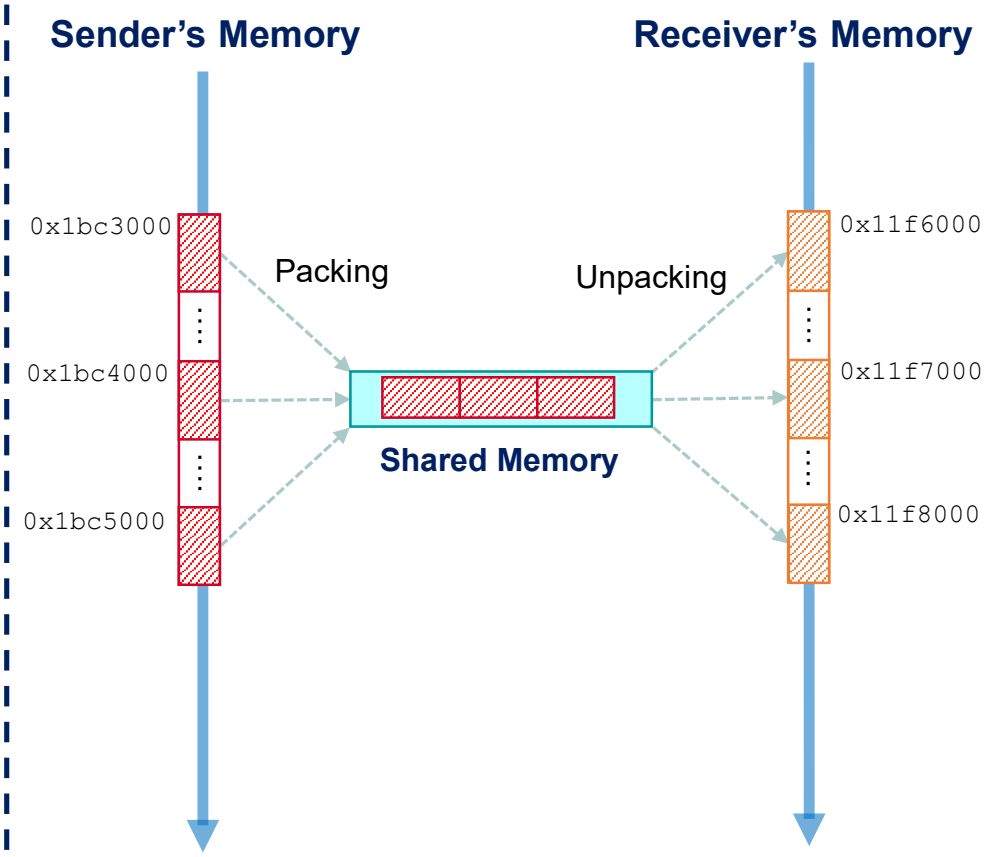
- One-to-all Communication (Broadcast / Scatter)
- Kernel-level contention
- Earlier designs[1] try to mitigate the contention
- Not completely removed

- Existing design lack zero-copy **Reductions**
- Remote data copied to temporary buffer before local process can perform operation

[1] 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*

Motivating Example 3 – Derived Datatypes Processing in MPI





Broad Challenge

Designing High-performance, Zero-copy, Contention-free, and packing-free MPI Designs by exploiting Shared-Address-Space Mechanisms on Modern High Core-density Architectures

Overview

- Introduction
- **Problem Statement**
- Detailed Designs and Results
  - Shared Address Space Communication Substrate
  - Direct Zero-copy MPI Collectives
  - Efficient Zero-copy MPI Datatypes
  - Kernel-assisted Communication On-loading
- Broader Impact on HPC
- Expected Contributions

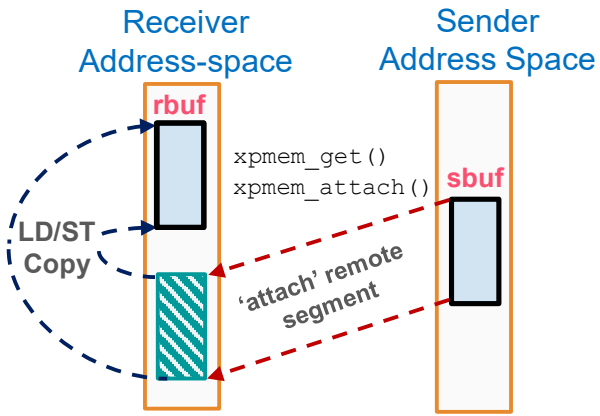
Problem Statement

- How can communication runtimes be re-designed to tackle the challenges posed by next-generation multi-/many-core architectures?
- Can we alleviate the bottlenecks of existing MPI intra-node designs and provide shared address space based efficient zero-copy MPI primitives?
- What are the overheads associated with MPI derived datatype processing designs and how can we alleviate these bottlenecks?
- How can we leverage the high core-density of modern architectures to assist MPI communication primitives via on-loading?
- What are the application level benefits that can be achieved through the proposed designs?

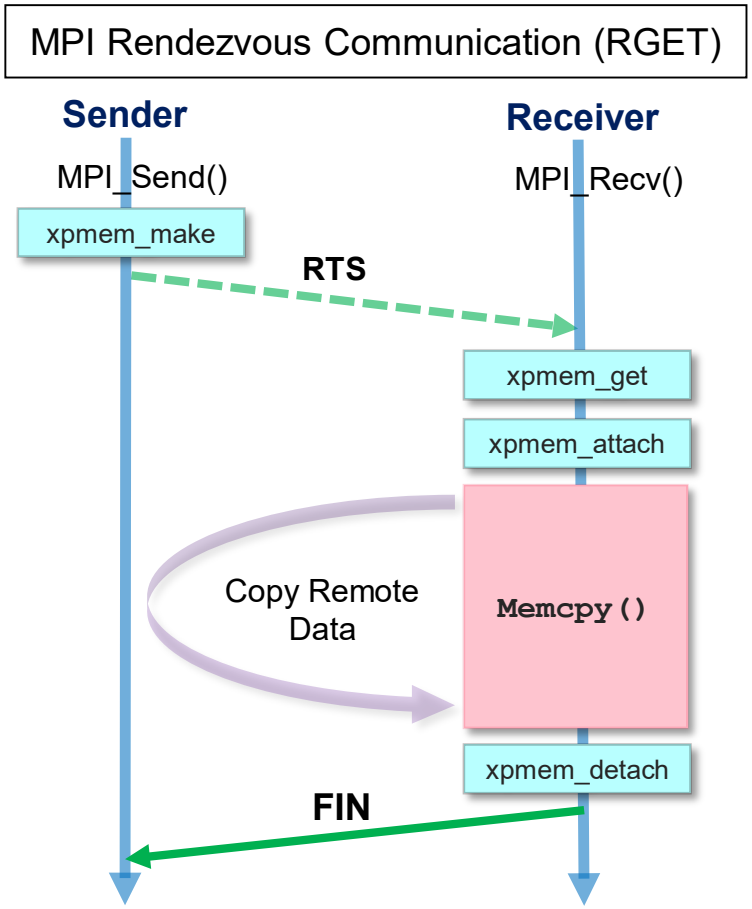
Overview

- Introduction
- Problem Statement
- **Detailed Designs and Results**
  - **Shared Address Space Communication Substrate**
  - Direct Zero-copy MPI Collectives
  - Efficient Zero-copy MPI Datatypes
  - Kernel-assisted Communication On-loading
- Future Research Directions
- Broader Impact on HPC
- Expected Contributions

Shared Address Space MPI Communication

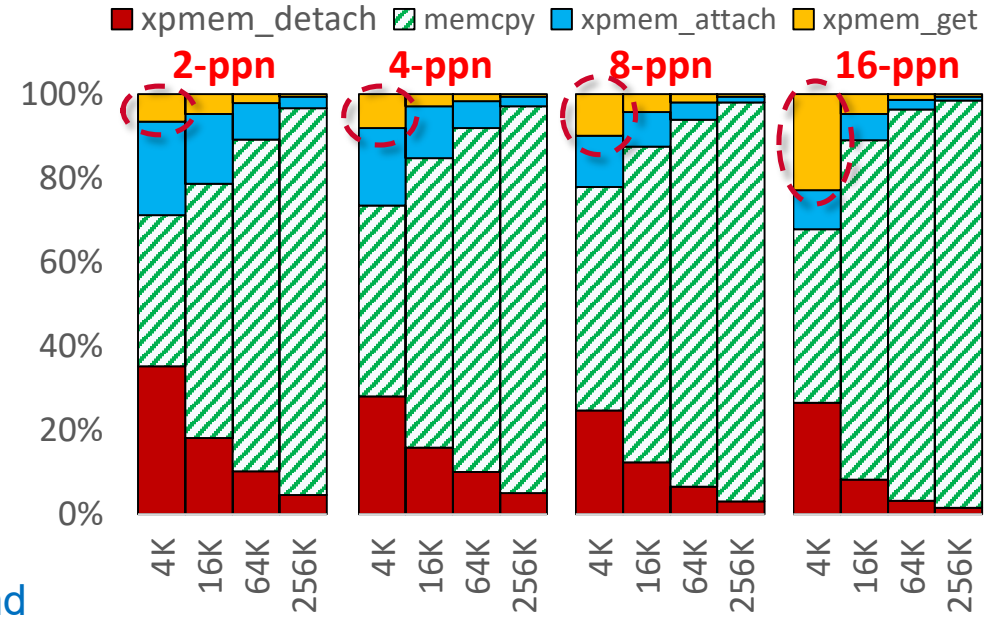


Cross-partition Memory (XPMEM) **– Kernel Module** with user-space API that allows a process to "*attach*" to the virtual memory segment of a remote process



Quantifying the Overheads of XPMEM-based Communication

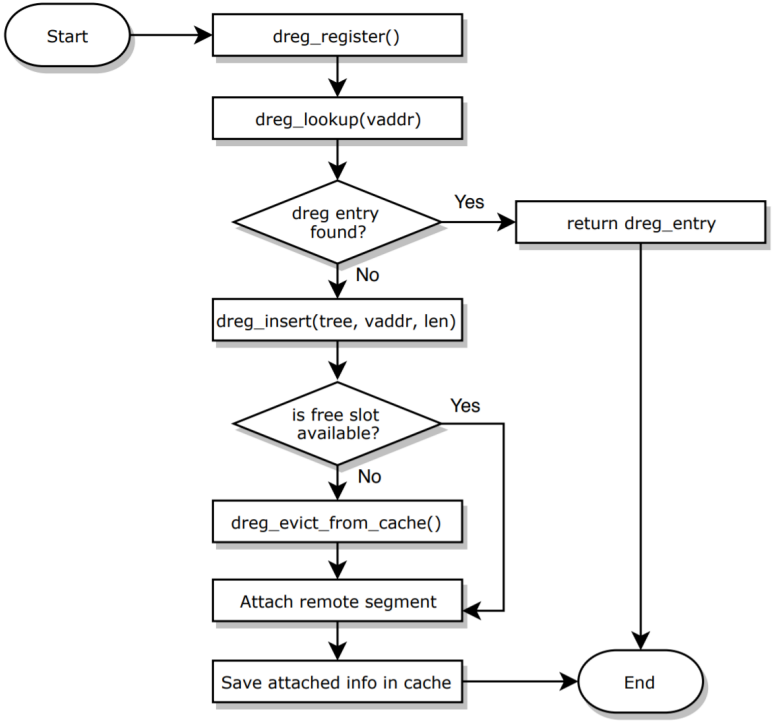
- XPMEM based one-to-all latency benchmark
- Up to **65%** time spent in XPMEM registration for short message (4K)
- Increasing PPN increases the cost of xpmem_get() operation
  - Lock contention
  - Pronounced at small messages
- How can we alleviate these overheads and improve the performance of shared address-space based MPI communication?



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.

Proposed Registration Cache for XPMEM based Communication

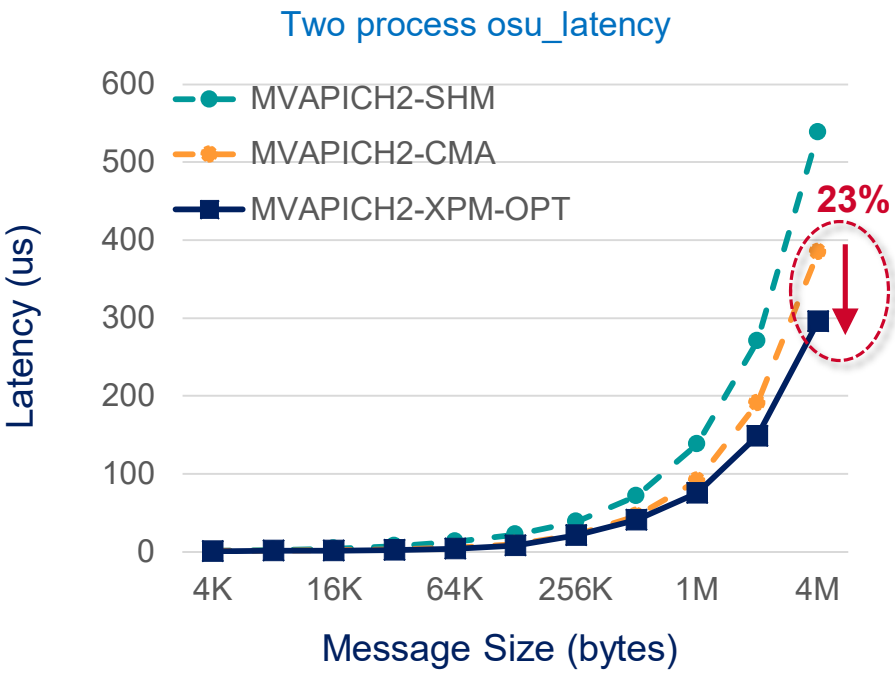
- AVL tree maintains remote attached pages
- Memory de-registration is delayed
  - Detach pages only in *MPI_Finalize()* or when capacity-miss occurs (FIFO)
- MPI calls on same buffers cause cache-hit
- Multiple calls to malloc/free on the remote buffers lead to invalid mappings
  - Access to attached buffer which has been freed on remote rank, is considered invalid
  - Interception of malloc/free calls to invalidate remote mappings

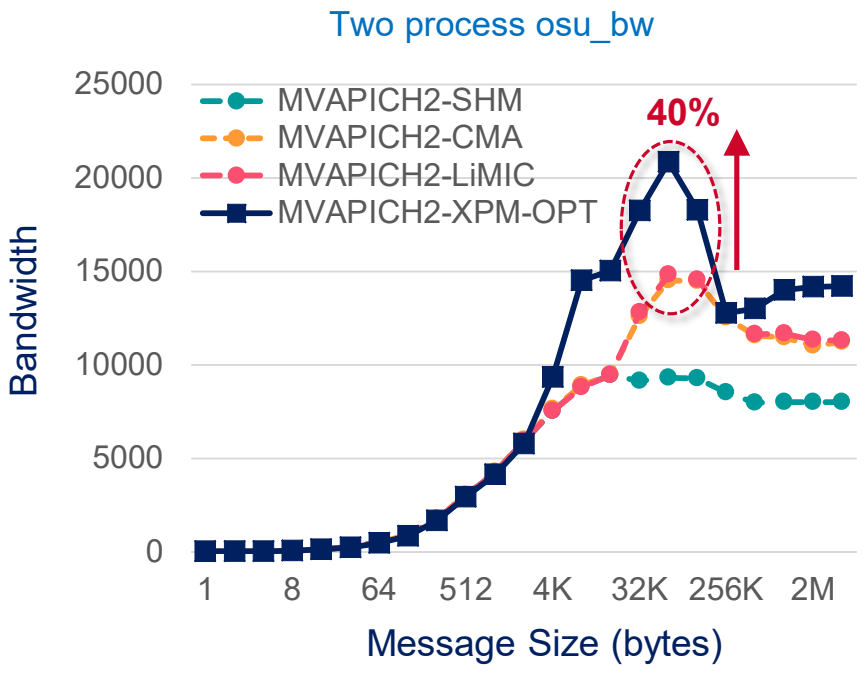


A high-level flow of the proposed Dynamic Registration Cache

J. Hashmi, S. Chakraborty, M. Bayatpour, H. Subramoni, and D. K. Panda, Designing Efficient Shared Address Space Reduction Collectives for Multi-/Many-cores, in Proceedings of the 32nd IEEE Intl' Parallel and Distributed Processing Symposium (IPDPS '18)

Impact of Optimized XPMEM design on MPI Performance





- 23% improved latency and 40% improved bandwidth over CMA even for two processes
- Takes advantage of user-space `memcpy()` optimizations e.g., AVX2, AVX512 etc.

Overview

- Introduction
- Problem Statement
- **Detailed Designs and Results**
  - Shared Address Space Communication Substrate
  - **Direct Zero-copy MPI Collectives**
  - Efficient Zero-copy MPI Datatypes
  - Kernel-assisted Communication On-loading
- Broader Impact on HPC
- Expected Contributions

Design Space of Existing 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 with zero-copy
  - Performance relies on the communication pattern of the collective
- **Problems with existing approaches**
  - No zero-copy reductions
  - Lock contention is mitigated but not removed[1]

[1] 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*

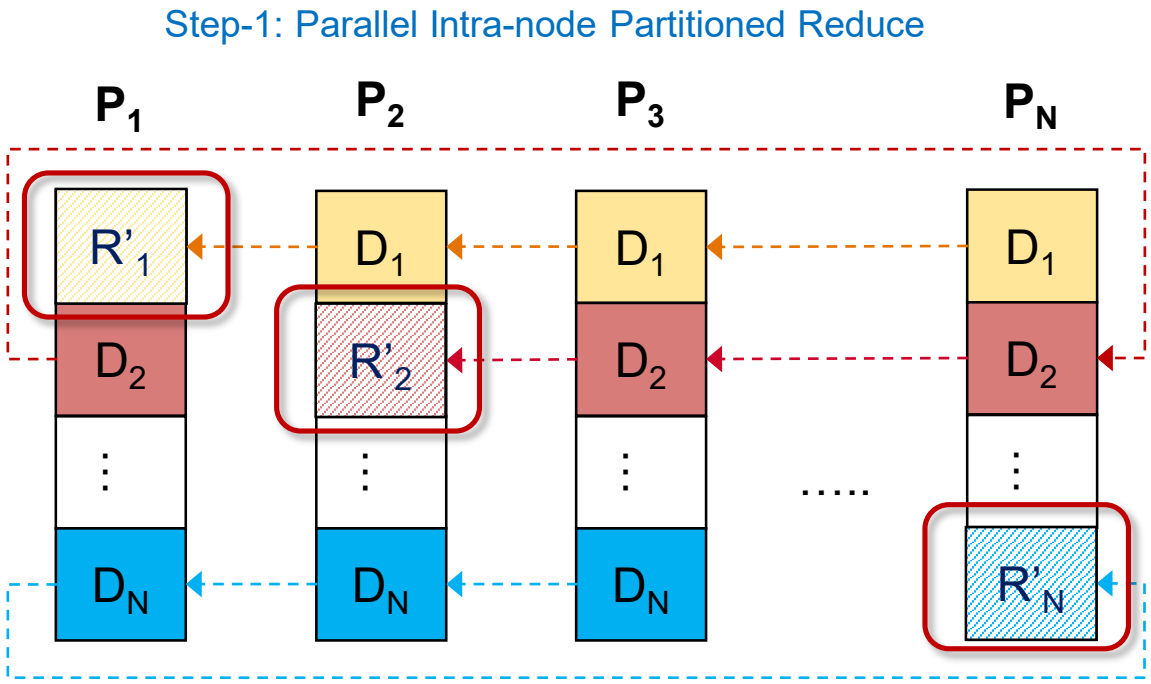
Design Overview of XPMEM based Direct MPI Collectives

- MPI collectives over shared address space
- Ranks exchange buffer information
  - Tuple of <vaddr, len, segid>
- Ranks map remote peer's memory segments
  - Load/store access is permitted
- An intra-node barrier is enforced to ensure correctness and ordering
- Call our proposed XPMEM based collective implementation routines e.g.,
  - XPMEM_bcast, XPMEM_reduce, etc.

```
/* 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

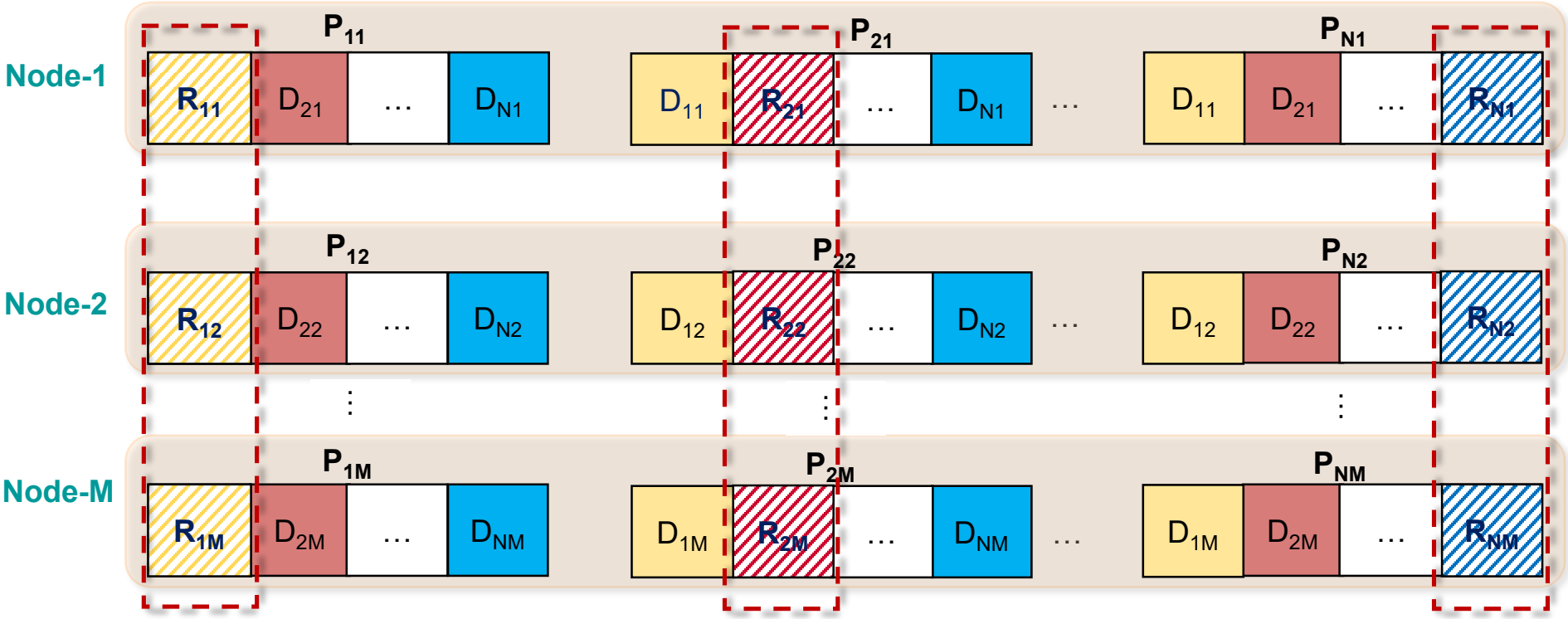
Proposed Zero-copy MPI_Allreduce



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

Proposed Zero-copy MPI_Allreduce (Cont'd)

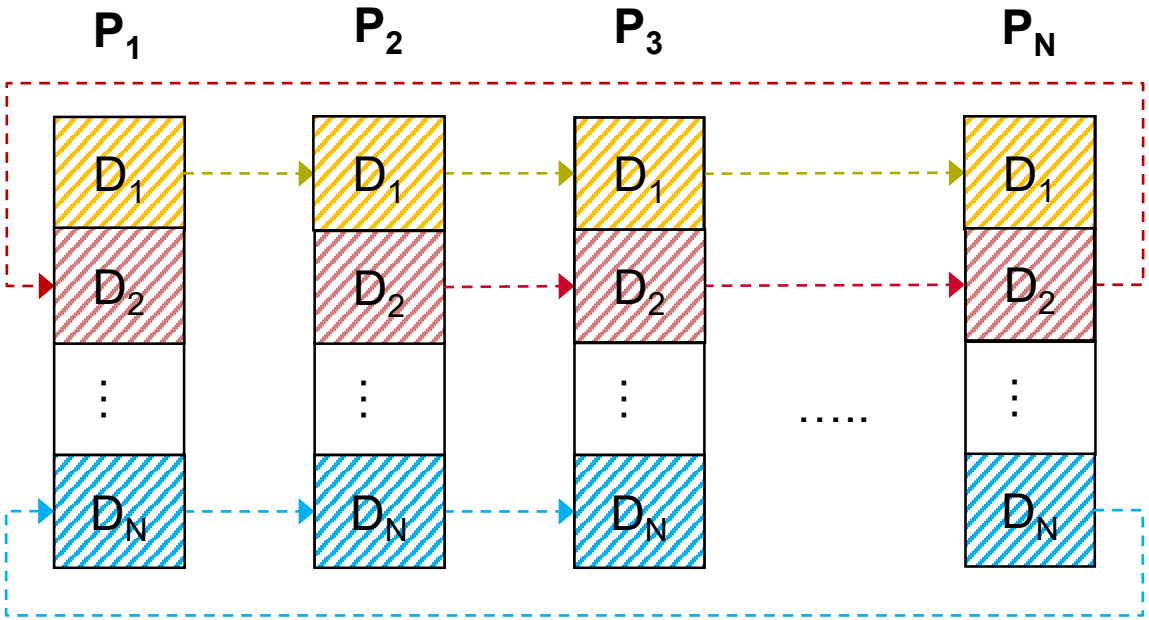
Step-2: Multi-root Inter-node Allreduce



Node-leaders concurrently perform Allreduce on respective partition of the data

Proposed Zero-copy MPI_Allreduce (Cont'd)

Step-3: Parallel Intra-node Partitioned Broadcast



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

J. Hashmi, S. Chakraborty, M. Bayatpour, H. Subramoni, and D. K. Panda, Designing Efficient Shared Address Space Reduction Collectives for Multi-/Many-cores, in Proceedings of the 32nd IEEE Intl' Parallel and Distributed Processing Symposium (IPDPS '18)

Evaluation Methodology and Cluster Testbeds

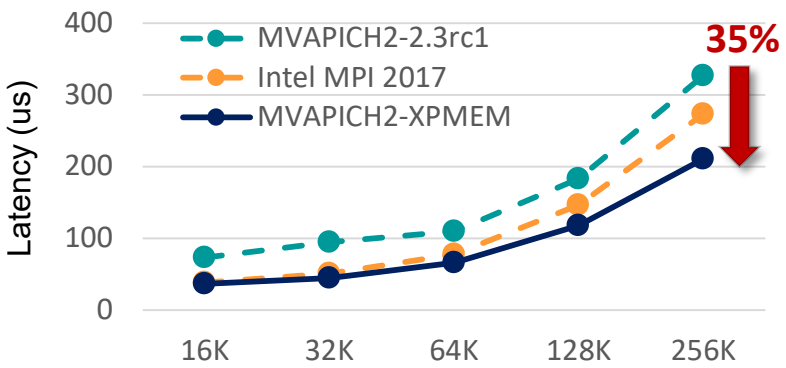
Hardware Specification of Cluster Testbeds

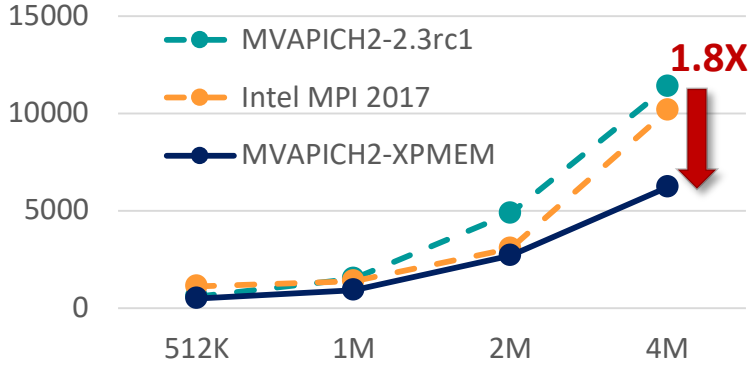
Specification	Xeon	Xeon Phi	OpenPOWER
Processor Family	Intel Broadwell	Knights Landing	IBM POWER-8
Processor Model	E5 2680v4	KNL 7250	PPC64LE
Clock Speed	2.4 GHz	1.4 GHz	3.4 GHz
No. of Sockets	2	1	2
Cores Per Socket	14	68	10
Threads Per Core	1	4	8
RAM (DDR)	128 GB	96 GB	256 GB
Interconnect	IB-EDR (100G)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

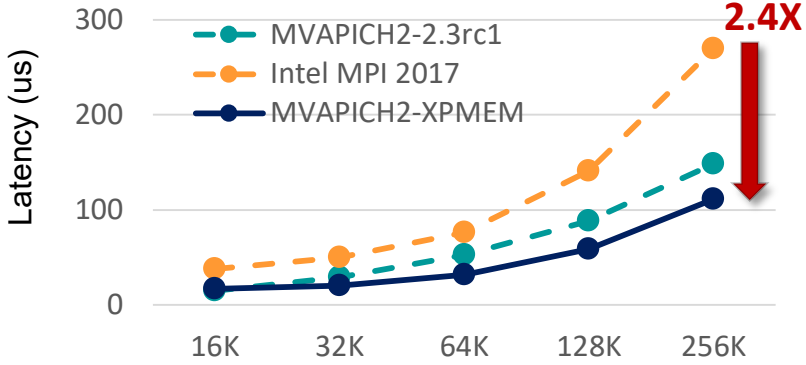
Micro-benchmark Evaluation on Broadwell Cluster

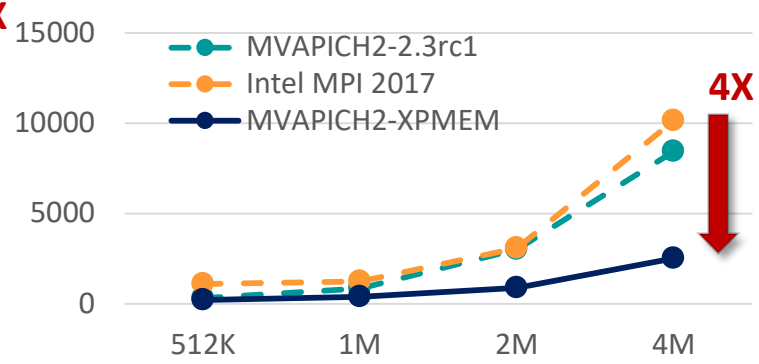
OSU_Allreduce





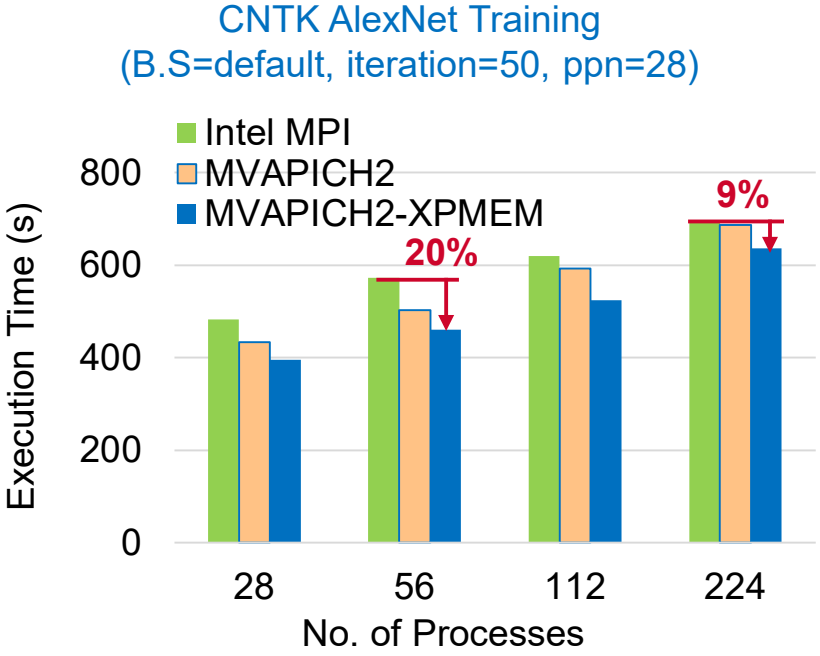
OSU_Reduce

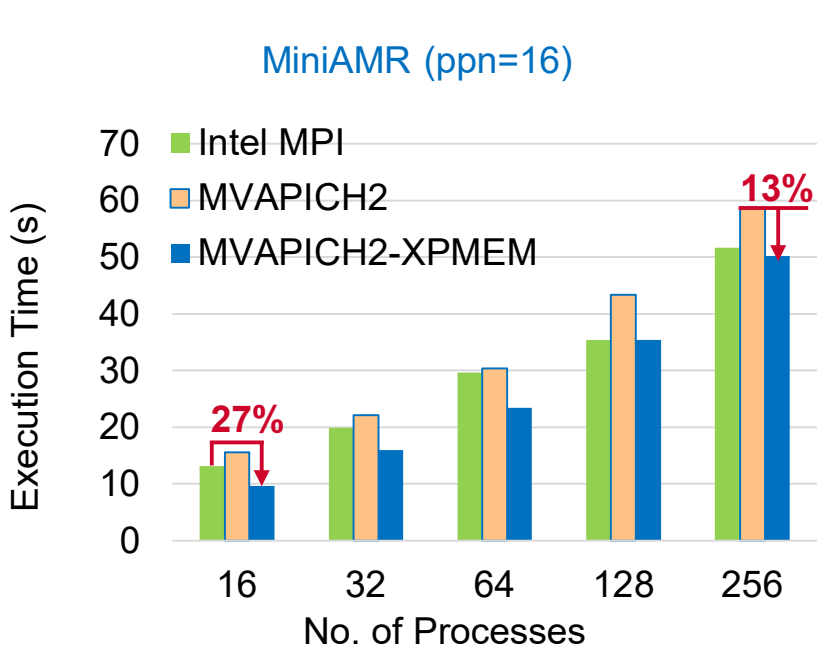




- 16 nodes, **256 processes** of dual-socket Broadwell system
- Up to **1.8X** improvement for 4MB AllReduce and **4X** improvement for 4MB Reduce, over Intel MPI

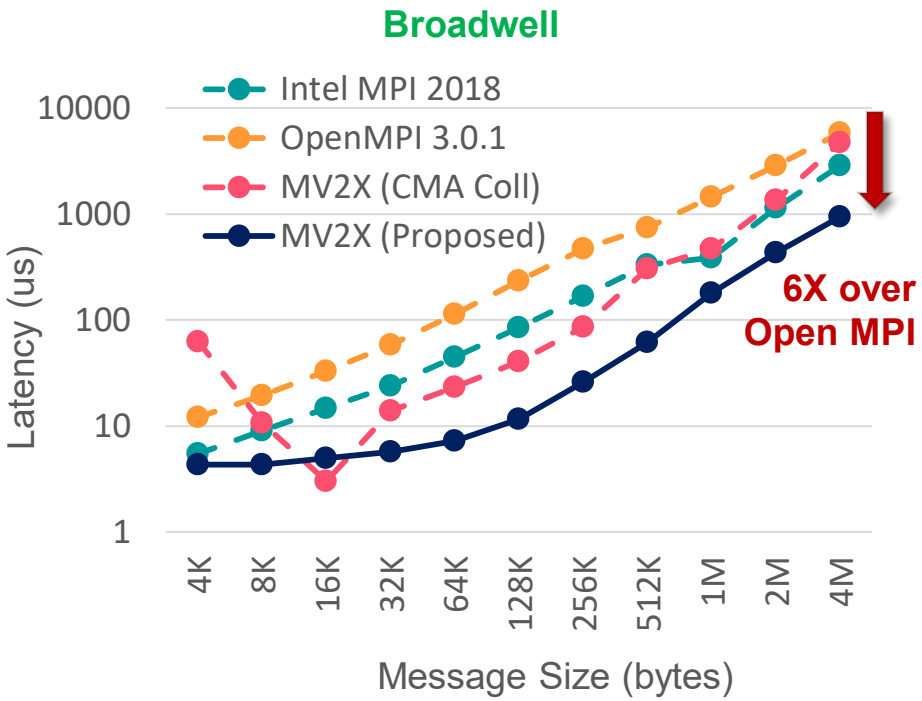
Applications Evaluation on Broadwell

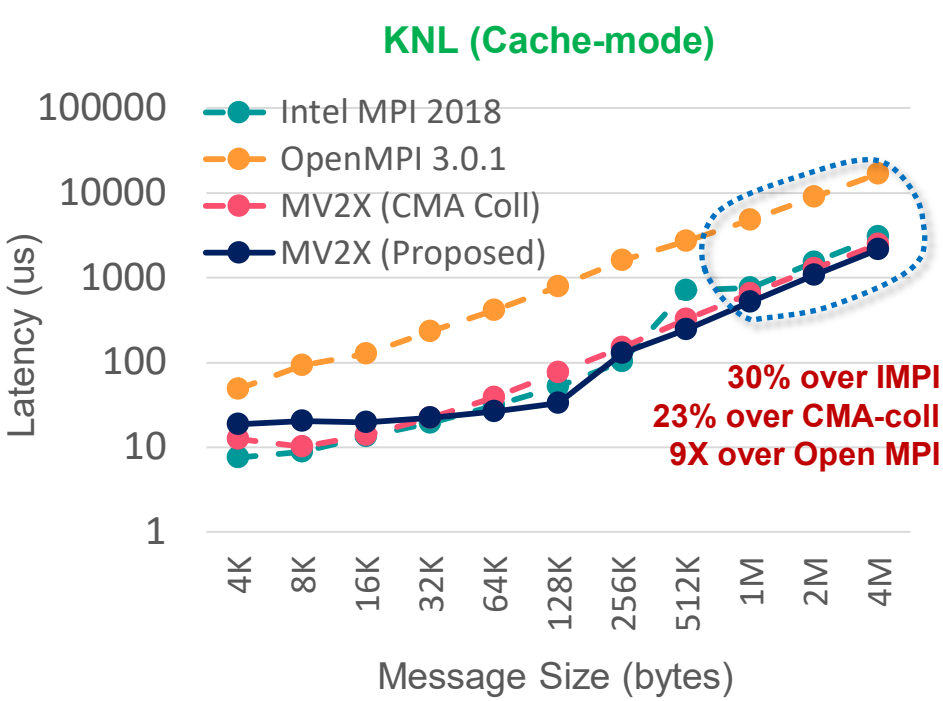




- Up to **20%** benefits over IMPI for CNTK MLP model and MNIST dataset using AllReduce
- Up to **27%** benefits over IMPI and up to **13%** improvement over MVAPICH2 for MiniAMR application kernel

Contention-free Designs for Non-reduction Collectives (MPI_Bcast)





- Shown for **Bcast** only but designs are also available for **Scatter**, **Gather**, **Allgather**, **Alltoall**
- Up to 30%, 23%, and 9X benefits over IMPI, direct CMA collectives, and Open MPI, respectively, on KNL

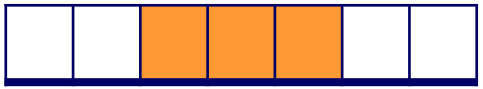
J. Hashmi, S. Chakraborty, M. Bayatpour, H. Subramoni, and D. K. Panda, Design and Characterization of Shared Address Space MPI Collectives on Modern Architectures, accepted to the 19th IEEE/ACM Intl' Symposium on Cluster, Cloud, and Grid Computing (CCGrid '19)

Overview

- Introduction
- Problem Statement
- **Detailed Designs and Results**
  - Shared Address Space Communication Substrate
  - Direct Zero-copy MPI Collectives
  - **Efficient Zero-copy MPI Datatypes**
  - Kernel-assisted Communication On-loading
- Broader Impact on HPC
- Expected Contributions

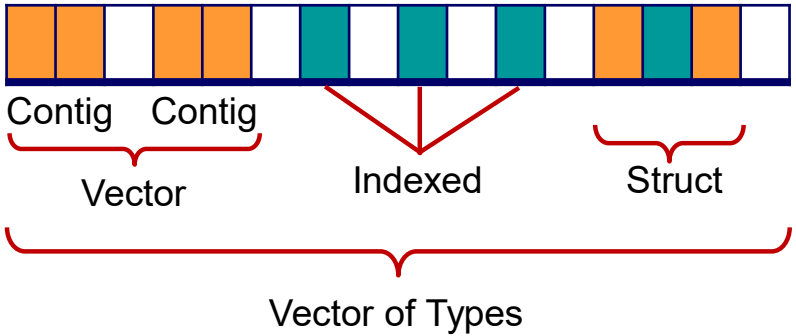
MPI Derived Datatypes Overview

MPI_Type_contiguous

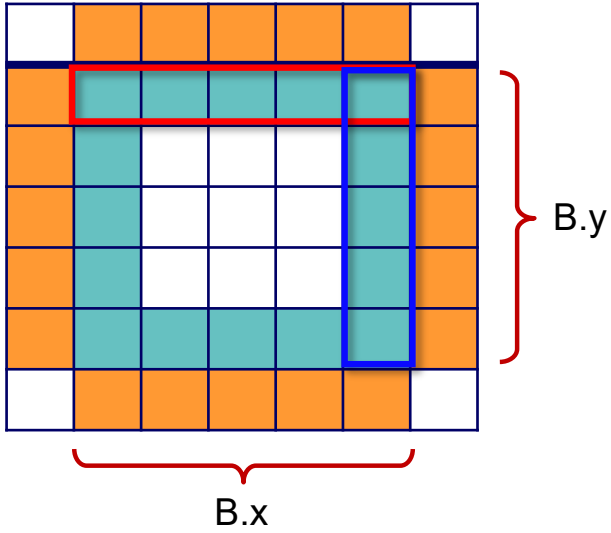


MPI_Type_vector

Nested Type Example



Halo Exchange Example



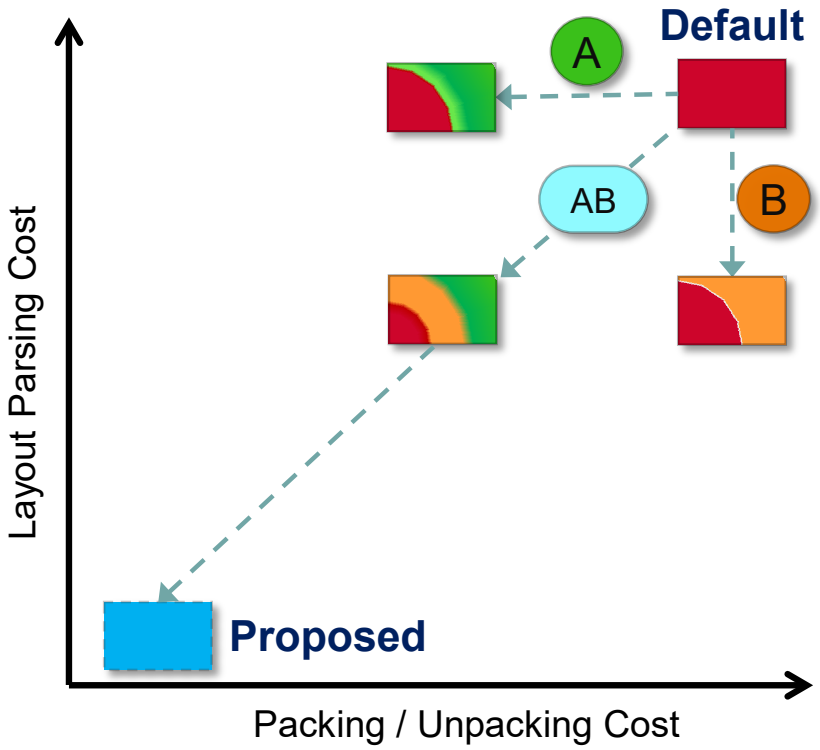
MPI_Type_contiguous (count=B.x, MPI_DOUBLE, …)

MPI_Type_vector (count=B.y, blocklen=1, stride=B.x+2, …)

Courtesy: https://www.mcs.anl.gov/~thakur/sc16-mpi-tutorial/slides.pdf

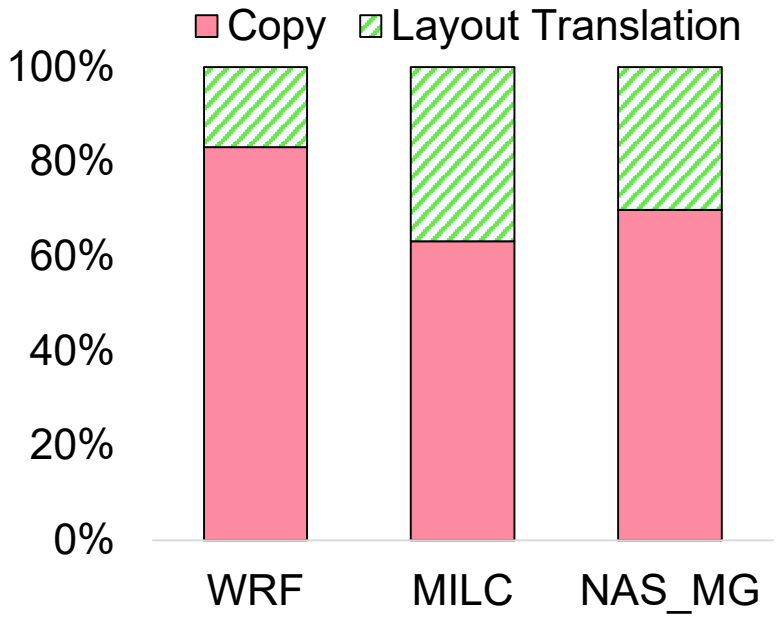
Standing on the shoulders of giants

- **Two broad categories of optimizations**
  - A. Faster Pack/Unpack
    - Efficient packing [Gropp et al., Thakur et al.]
    - GPU Accelerated pack/unpack [Chu et al.]
  - B. Layout parsing optimizations
    - Flattening-on-the-fly algorithm [Träff et al.]
    - Automatic type generation [Kjolstad et al.]
- **This work asks fundamental question:**
  - **"What if (A) and (B) are not required?"**
- **Fundamentally re-think design space**
  - No layout parsing overheads
  - No packing/unpacking required



Limitations of Existing Pack/Unpack based Designs

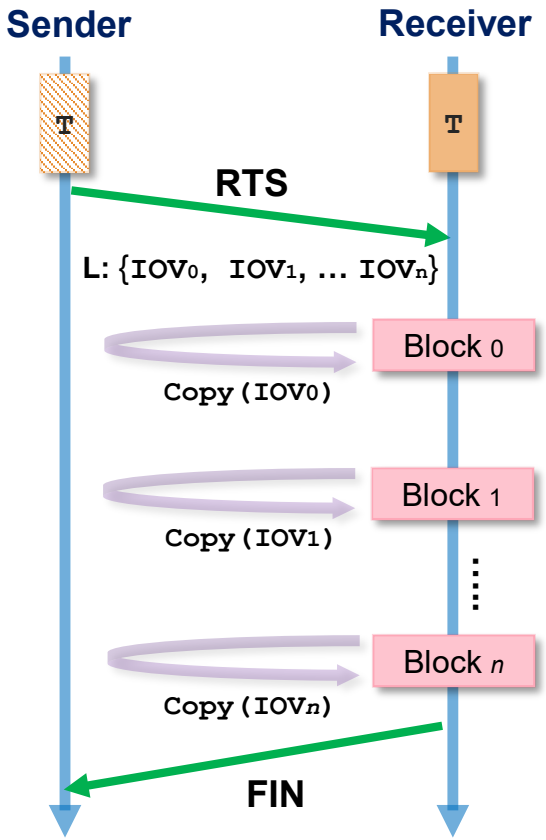
- **Layout Translation**
  - Flattening the layout into list of I/O vectors elements
  - Significant overhead for nested (hierarchical) datatypes
  - Applications can use any layout
- **Pack/Unpack requires two copies**
  - 2X overhead for large messages!!
- **Proposed Design**
  - **FALCON — FA**st and **L**ow-overhead **Zero-copy** MPI datatype processing **CO**mmunication e**N**gine



Cost breakdown of existing Pack/Unpack designs on Broadwell

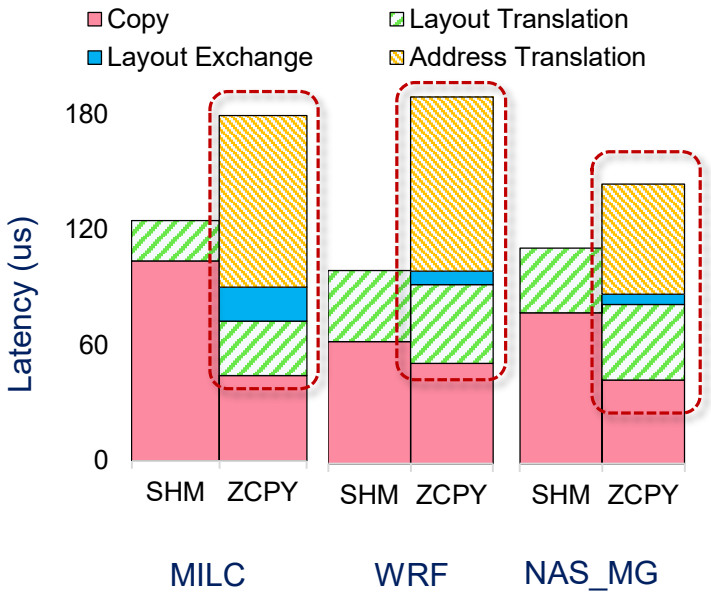
FALCON Designs

- Proposed four different designs
  - Basic, Pipelined, Memoization, and Optimized Memoization
- Basic Design
  - Sender and receiver translate local layouts
  - Sender appends IOV list to RTS
  - Receiver extracts the sender's IOVs
  - Directly copy each IOV from sender's virtual address space (CMA/XPMEM)
  - XPMEM offers user-space transfers
  - Receiver sends a FIN packet



Limitations of Basic Zero-copy design

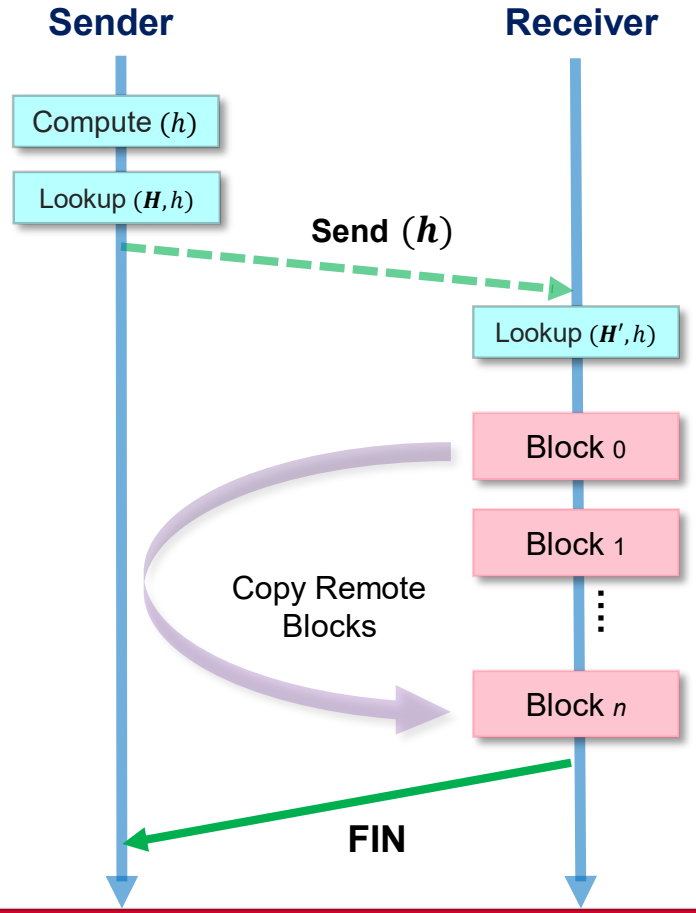
- **Layout-translation** is still required
- **Layout-exchange**
  - Layouts are local to rank in MPI
  - Sender has to send its layout to receiver
- **Remote address translation** for XPMEM
  - XPMEM attach
- **High fragmentation** means large IOV lists
  - RTS can exceed actual payload
- Combined overheads take significant time
  - **Up to 70%** of total communication
  - Overheads outweigh the benefits



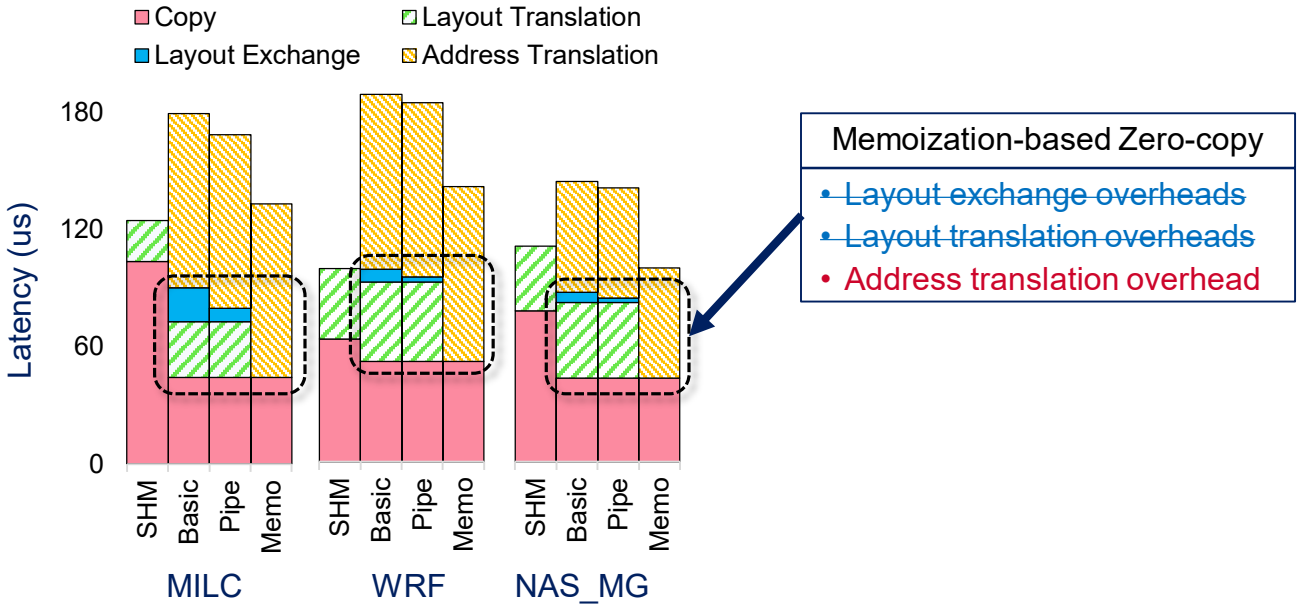
The time for data copy has been reduced but at additional cost are added.

FALCON: Memoization based Zero-copy Design

- Exploit application's layout re-usability
  - Avoids unnecessary layout exchange
  - Sender memoizes **translated** layouts (L)
  - Receiver memoizes **exchanged** layouts (L')
  - Sender's Hash Table (H) stores $< k, v > = < h, L >$
- If Hash (h) is found in sender's (H)
  - Sender only sends computed hash (h)
  - Receiver copies sender's data using found (L')
- If Hash (h) is not found
  - Sender sends the hash (h) + (L)
  - Receiver adds received (h) + (L) to (H')
  - Receiver copies sender's data using (L)



Benefits of Memoization based Zero-copy Design

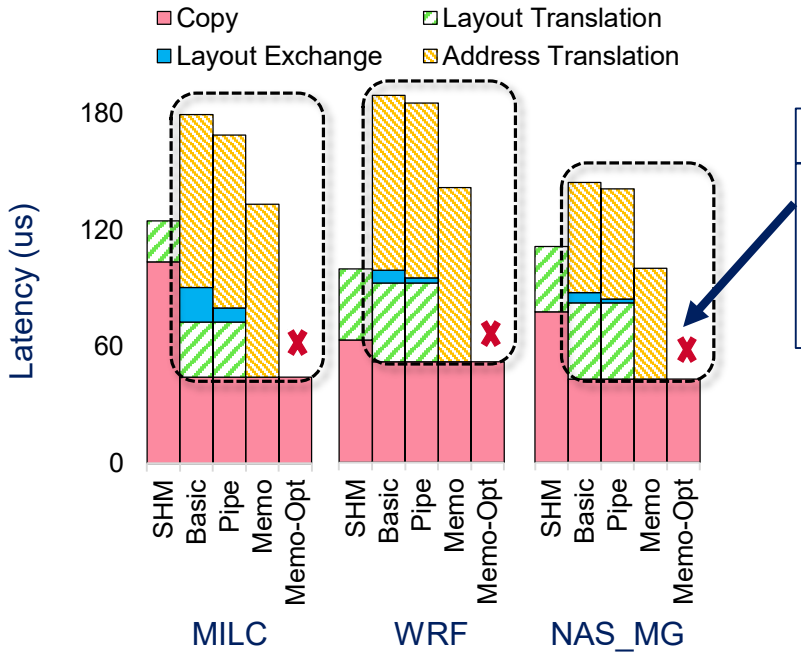


Memoization based design additionally reduced the **layout translation** overhead

FALCON: Design Optimizations

1. **Avoiding Remote Virtual Address Translation**
   - XPMEM attached segments are cached
   - Future accesses re-use attached IOVs
   - No costly registration/de-registration required
2. **Communication pattern as input to the Hash function**
   - Computing Hash on IOV list can be costly
   - Request object has enough information to uniquely identify the layout
   - *<Datatype, Count, Destination Rank, Tag, Communicator>*
3. **Re-using receiver side layouts**
   - Layout re-use is common at applications
   - Cache translated IOVs at receiver as well
   - Avoid local layout translation by receiver's IOV

Impact of Optimized Memoization based Zero-copy Design



Optimized Memoization Design (Final)

- ~~Layout exchange overheads~~
- ~~Layout translation overheads~~
- ~~Address translation overheads~~
- No-overhead remaining

Optimized Memoization design **removed all the overheads** with **zero-copy benefits**

__J. Hashmi__, S. Chakraborty, M. Bayatpour, H. Subramoni, and D. K. Panda, FALCON: Efficient Designs for Zero-copy MPI Datatype Processing on Emerging Architectures, accepted to the 32nd IEEE Intl' Parallel and Distributed Processing Symposium (__IPDPS '19__), __Best Paper Nominee__

Performance Evaluation: Application Kernels

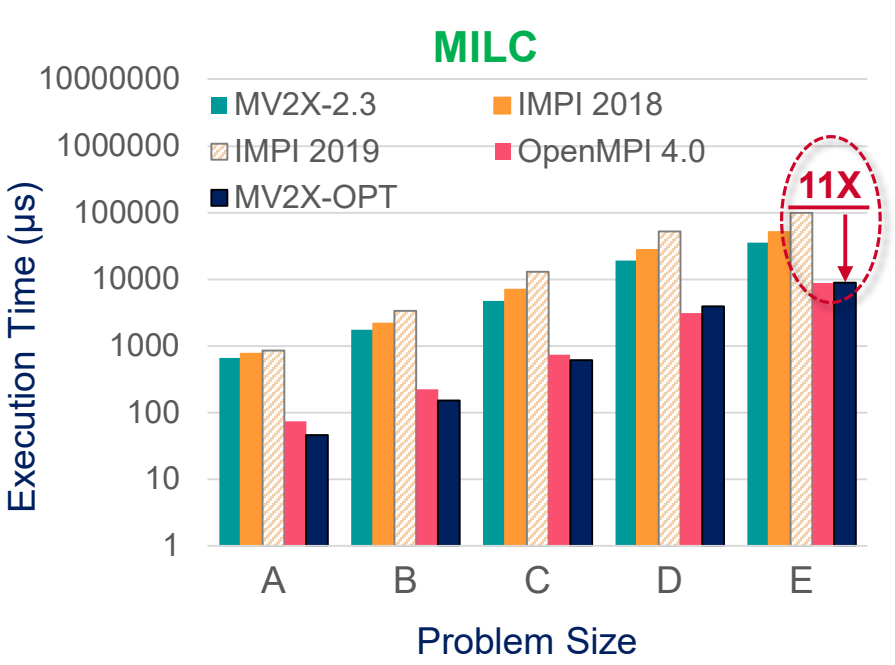
Communication Kernels

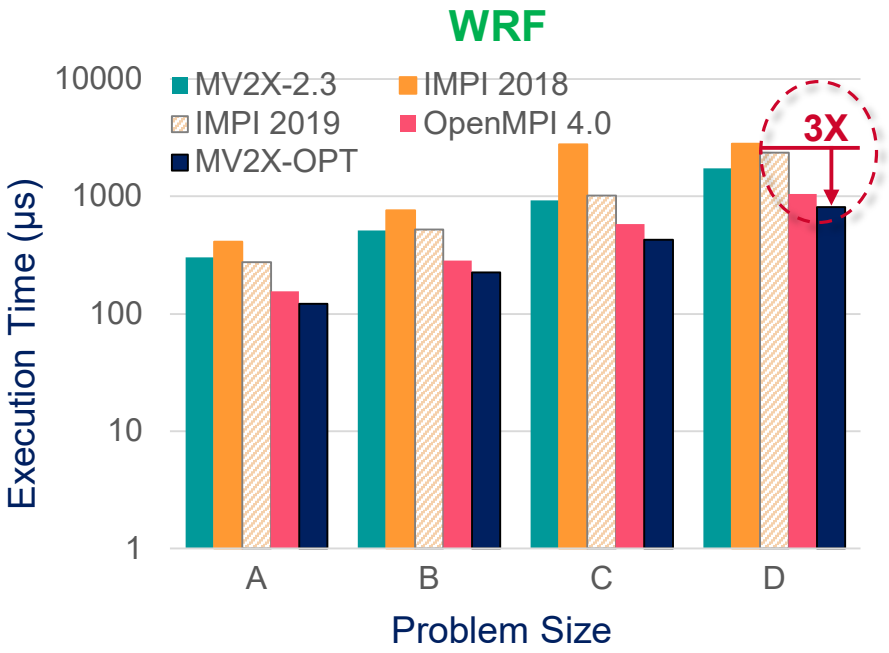
Application Kernel	Application Domain	Datatype Layout
MILC_su3_zd	Quantum Chromodynamics	Nested Vectors for 4D face exchanges
WRF_y_vec	Atmospheric Science	Nested Vectors and Subarrays
NAS_MG_z	Fluid Dynamics	Vectors and Nested vectors for 3D face exchanges
3D-Stencil	Stencil Communication	7-point stencil using Subarray datatypes

- We used various application kernels e.g., MILC, WRF, and NAS from DDTBench[1].
- Derived datatype based communication kernels of these applications are used to measure the communication latencies.
- Evaluated on Broadwell, OpenPOWER, and KNL but only showing Broadwell here

[1] T. Schneider, R. Gerstenberger, and T. Hoefler. Micro-applications for Communication Data access Patterns and MPI datatypes. In European MPI Users' Group Meeting, pages 121–131. Springer, 2012.

Application Kernels – MILC and WRF (Broadwell)



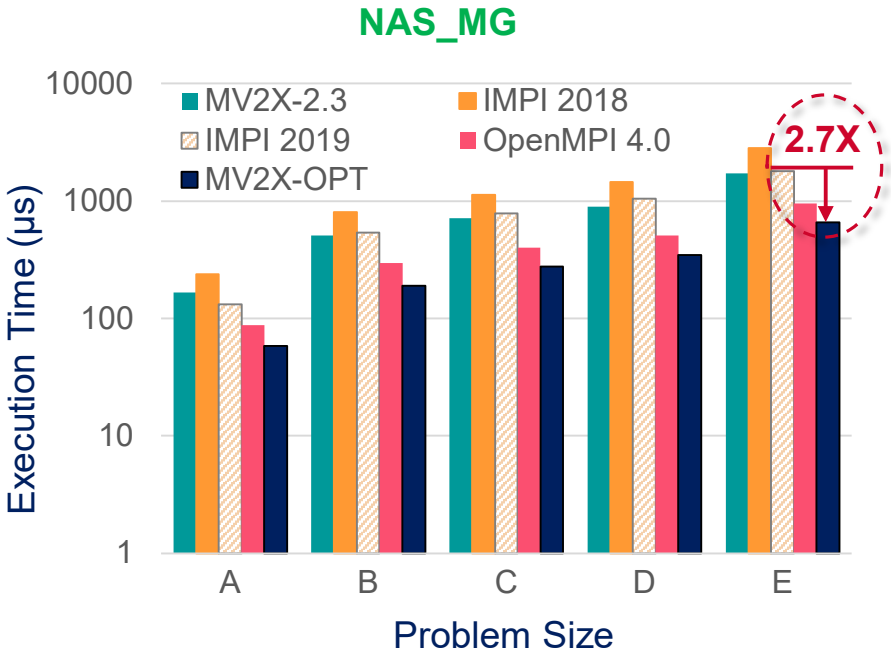


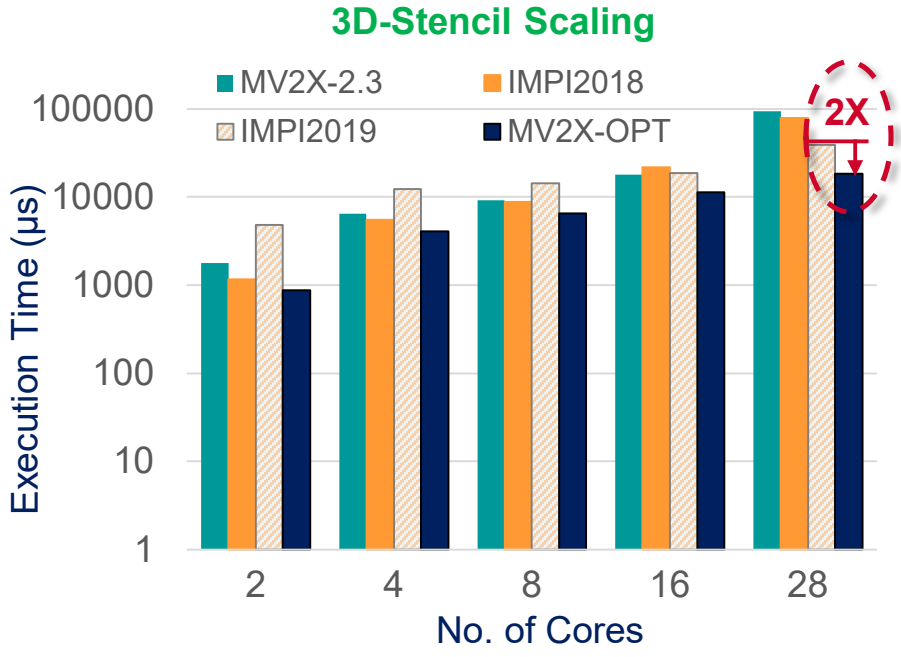
MILC Params – A = (16, 16, 32, 32); B = (32, 32, 32, 32); C = (64, 64, 32, 32); D = (128, 128, 32, 32); E = (128,128,64,64)

WRF Params (ims, ime, is, ie) – A = (4, 140, 8, 136); B = (4, 268, 264, 8); C = (4, 524, 8, 520); D = (4, 1036, 8, 1032)

- On MILC, for Problem-B (768-KB), up to **11X** over IMPI 2019
- On Broadwell, up to **2.1X** and **3X** improved latency over MVAPICH2-X and Intel MPI 2019

Application Kernels – NAS_MG and 3D-Stencil (Broadwell)





NAS Grid Dimensions – A = (258, 130, 130); B = (512, 258, 258); C = (768, 258, 258); D = (1024, 258, 258); E = (2048, 258, 258)

7-point 3D-Stencil Grid Dimensions – 512^3

- On NAS_MG_z, up to **2.7X** and **2.5X** improvement over Intel MPI 2019 and MVAPICH2-X
- On 3D-Stencil using On 28-cores, up to **5X** and **2X** improvement over MVAPICH2-X and Intel MPI 2019

Overview

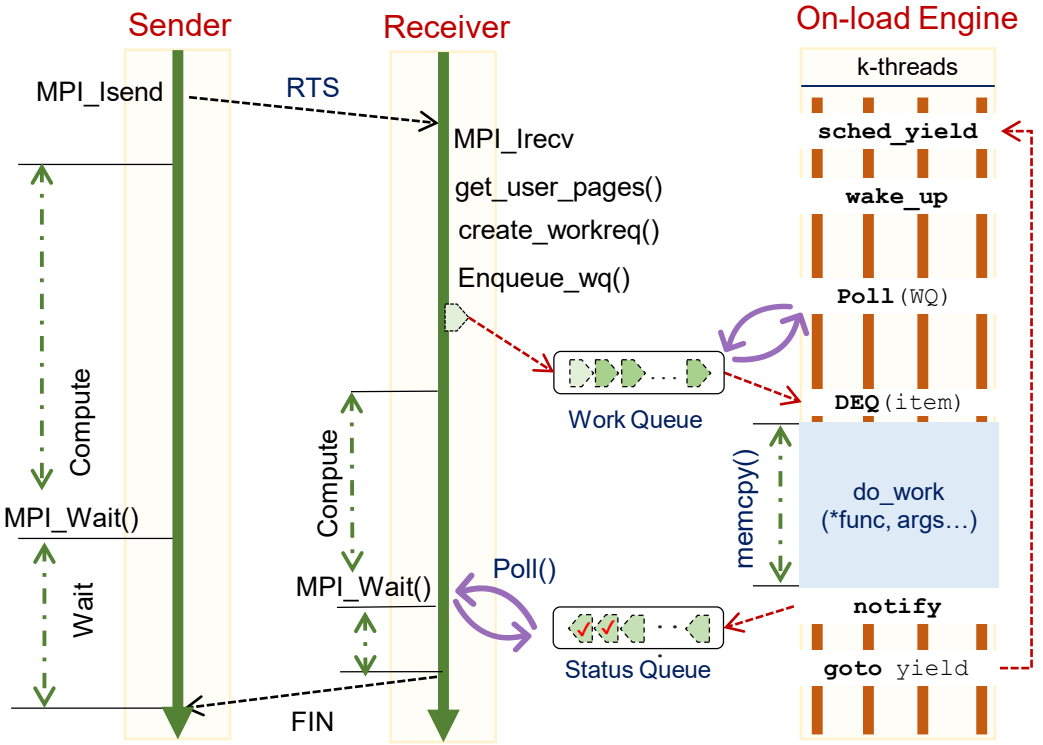
- Introduction
- Problem Statement
- **Detailed Designs and Results**
  - Shared Address Space Communication Substrate
  - Direct Zero-copy MPI Collectives
  - Efficient Zero-copy MPI Datatypes
  - **Kernel-assisted Communication On-loading**
- Broader Impact on HPC
- Expected Contributions

Assisted Communication on Many-cores

- Many-cores such as KNL, OpenPOWER have lots of SMT cores
  - Can we dedicate some cores to derive communication?
- Intra-node MPI transfers mainly use blocking memory copies
  - Can we partition large-message communication
- **Broad Questions**
  - *Can we design an efficient mechanism to effectively utilize KNL resources and bring concurrency to the communication phases in MPI?*
  - *Can we design a communication engine that can asynchronously derive the communication in MPI?*
- **Kernel-assisted Communication on-loading Engine**
  - New designs to bring concurrency to communication
  - Performance, portability, and programming abstraction

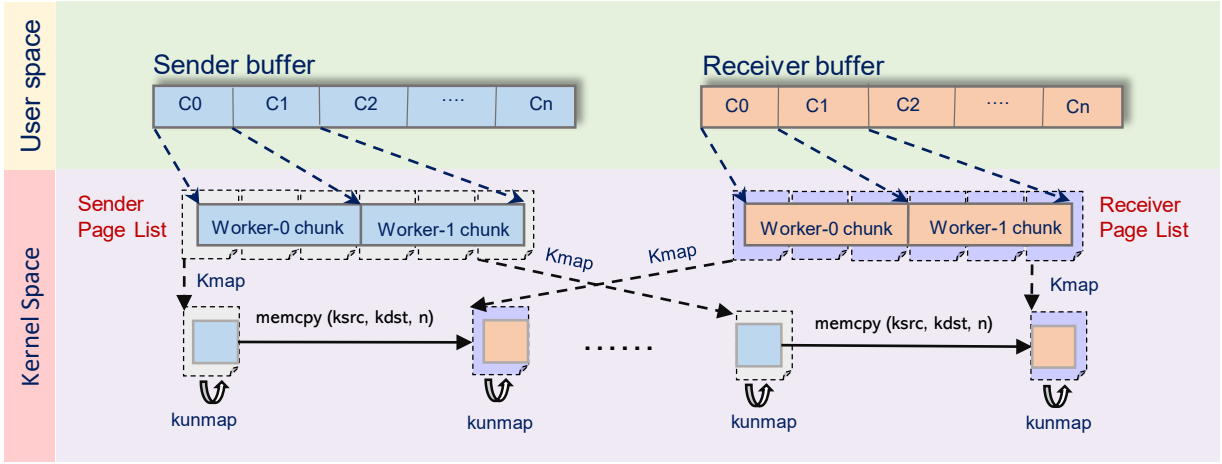
Designing Kernel-assisted Communication On-loading Engine

- Two design components
  - Programming abstraction
  - Kernel-module implementation
- High-level API used by MPI runtime to delegate tasks
  - Integration with MVAPICH2
- Kernel module handles scheduling, threading, task and signaling queues



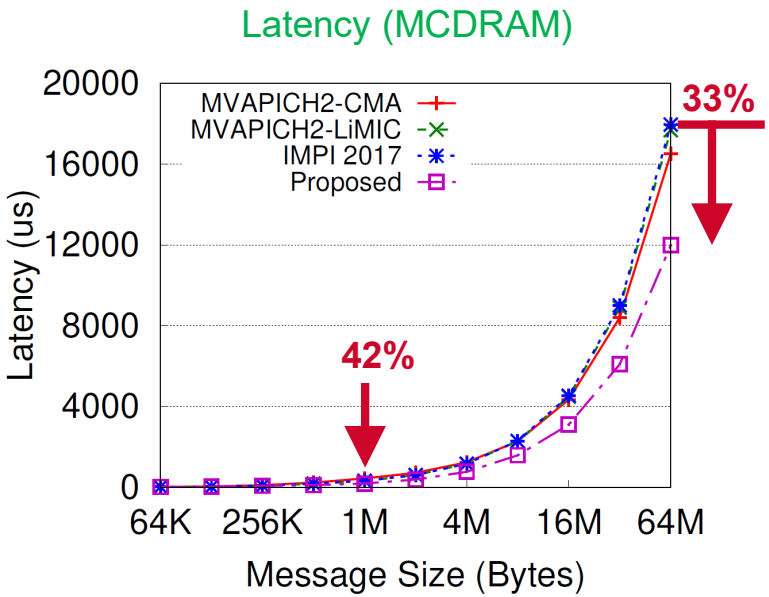
*J. Hashmi, K. Hamidouche, H. Subramoni, and D. K. Panda, Kernel-assisted Communication Engine for MPI on Emerging Manycore Processors, in Proceedings of the 24th IEEE Intl' Conference on High Performance Computing, Data, and Analytics (**HiPC '17**)*

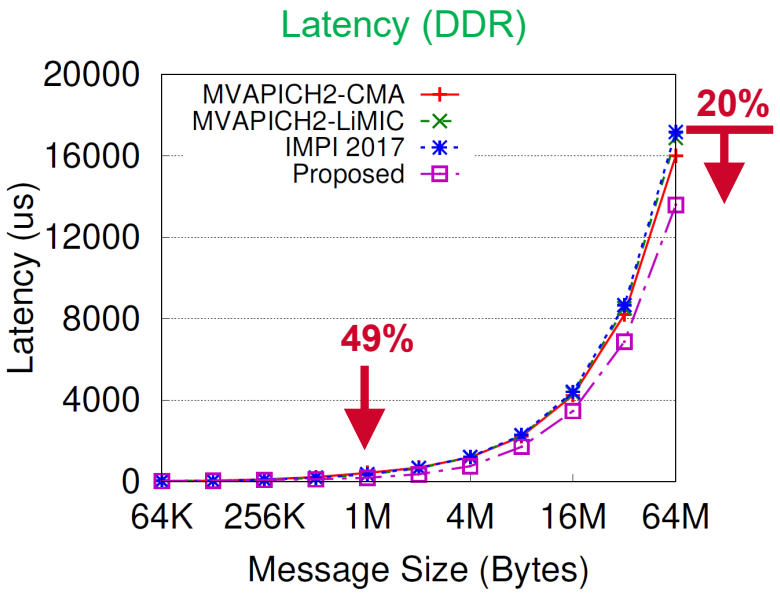
Realizing Zero-copy Communication using On-load Engine



- Example of on-loading function
  - e.g., Parallel kernel-mapped zero-copy communication
  - Multiple threads perform memory-mapped zero-copy
  - Map sender and receiver pages and copy the data

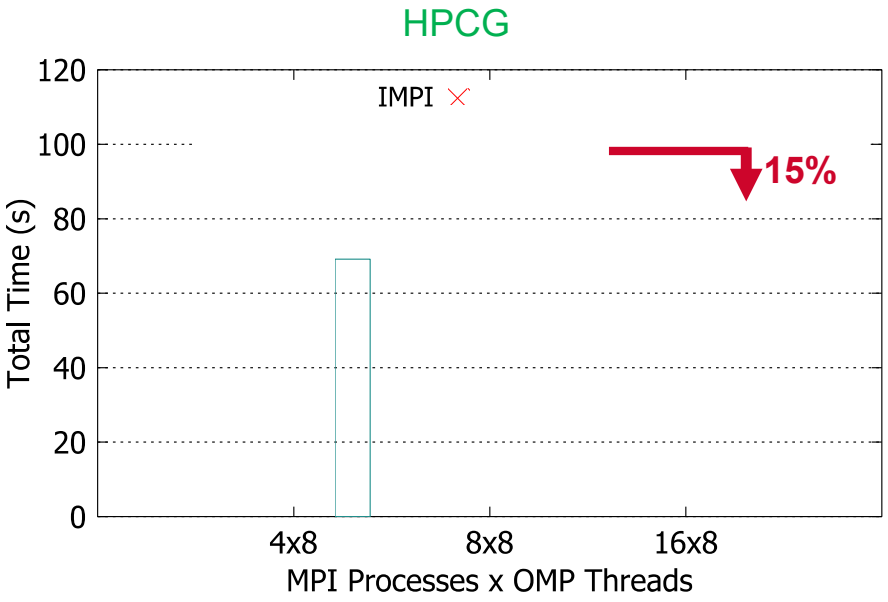
Microbenchmark Evaluation on KNL (Latency)

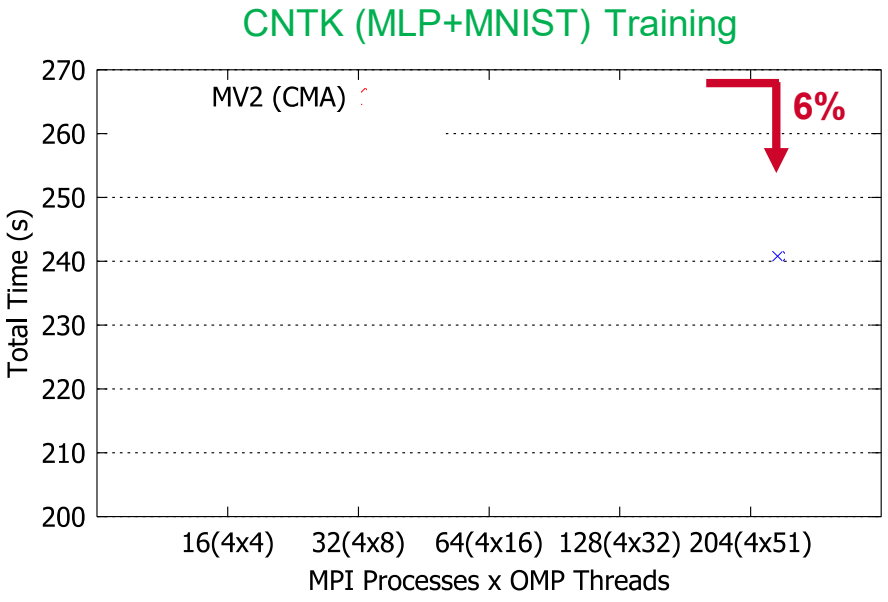




- OMB `osu_latency` benchmark by allocating memory on MCDRAM and DDR using two processes
- Four kernel threads running in assisted mode.
- Up to **42% and 49% improved latency** on a single KNL when using MCDRAM and DDR, respectively

Application Evaluations on KNL





- HPCG with MPI+OpenMP running 8 OpenMP threads per MPI process.
  - Main benefits come from **DDOT, MG, and DDOT Allreduce** phases of HPCG
  - Overall execution time is reduced by **15%** over Intel MPI
- CNTK Multi-level Perceptron (MLP) feed-forward neural network using MNIST dataset

Overview

- Introduction
- Problem Statement
- Detailed Designs and Results
  - Shared Address Space Communication Substrate
  - Direct Zero-copy MPI Collectives
  - Efficient Zero-copy MPI Datatypes
  - Kernel-assisted Communication On-loading
- **Broader Impact on HPC**
- Expected Contributions

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 540,000 (> 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

MVAPICH2 Releases

- **MVAPICH2-X 2.3rc2 (04/02/2019)**
  - XPMEM-based contention-free MPI collectives (Broadcast, Gather, Scatter, Allgather)
  - Extended support for XPMEM based reduction collectives on PSM channel
- **MVAPICH2-X 2.3rc1 (09/21/2018)**
  - Support for XPMEM-based point-to-point operations
  - Efficient registration cache for XPMEM communication
  - Efficient truly zero-copy reductions (MPI_Reduce and MPI_Allreduce)
- **Upcoming***
  - FALCON: Efficient Zero-copy MPI Derived Datatypes
  - Assisted communication runtimes

Impact on HPC Community

- Fundamental designs geared towards next-generation HPC systems
  - High core-density architectures
- Designs made available to the HPC community via MVAPICH2-X releases
  - Used by wider HPC and DL community
- Other communication runtimes beginning to adopt the core ideas
  - Shared address space communication
  - Efficient Datatype processing
- The designs as part of the MVAPICH2-X software stack
- Proposed designs are empowering several of Top500 supercomputers
  - TACC Stampede2, Frontera
  - OSC Owens, Pitzer

Overview

- Introduction
- Problem Statement
- Detailed Designs and Results
  - Shared Address Space Communication Substrate
  - Direct Zero-copy MPI Collectives
  - Efficient Zero-copy MPI Datatypes
  - Kernel-assisted Communication On-loading
- Broader Impact on HPC
- **Expected Contributions**

Conclusion

- High core-density architectures are building next-generation ultra-scale systems
  - Communication middlewares need to be re-designed to meet the diversity
- This thesis optimizes MPI communication for emerging multi-/many-cores
  - Point-to-point, Collectives, Datatype processing
- Presents novel Shared Address Space communication mechanism
  - Direct load/store communication
  - Efficient caching mechanisms
- Fundamentally re-think and propose new MPI datatype designs
  - Efficient zero-copy datatype processing
- Proposed and designed an on-load engine abstraction with concurrency, portability, and programmability for modern many-core architectures
- Significant impact on the community in transition to next-generation multi-/many-cores
- Broader outreach through MVAPICH2/MVAPICH2-X public releases

Thank You!

Questions?