MVAPICH

MPI, PGAS and Hybrid MPI+PGAS Library

FALCON: Efficient Designs for Zero-copy MPI Datatype Processing on Emerging Architectures

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

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

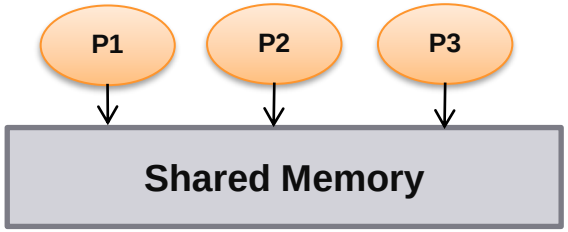
Network Based Computing Laboratory (NBCL)

The Ohio State University

Outline

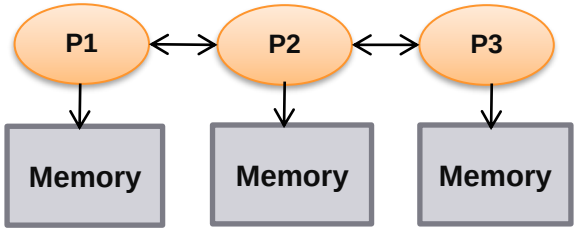
- **Introduction and Motivation**
  - Programming Models
  - Derived Datatypes
- **Existing Designs for Derived Datatypes**
  - Shared Memory based Packing/Unpacking
  - Challenges and Limitations
- **FALCON: Zero-copy based Derived Datatypes**
  - Basic Design and limitations
  - Pipelined Zero-copy
  - Memoization based Design
  - Design Optimizations
- **Results and Discussion**
  - Micro-benchmark Evaluation
  - Application Kernels: WRF, MILC, NAS_MG
  - Scalability Results: 3D-Stencil
- **Conclusion and Future Impact**

Parallel Programming Models Overview



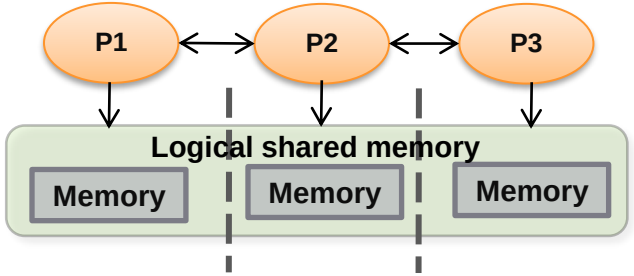
Shared Memory Model

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.
- 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)

Communication and Data Layout in MPI

- Communication primitives
  - Point-to-point
  - Collectives
  - Remote Memory Access (RMA)
- Communication Protocols
  - Eager
  - Rendezvous

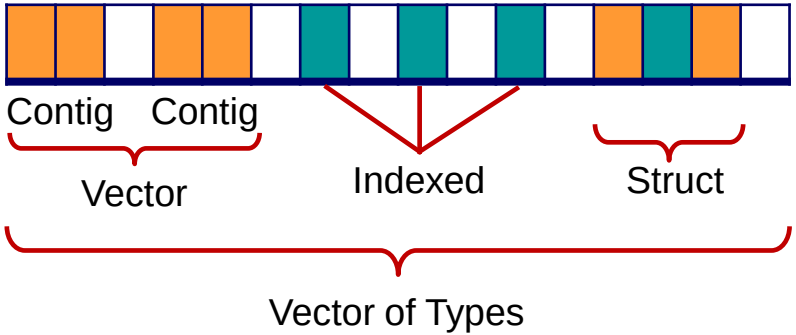
- MPI Datatypes
  - Serialize arbitrary layouts
- Intrinsic types
  - MPI_INT, MPI_DOUBLE, etc.
- Derived Datatypes
  - Contiguous
  - Non-contiguous (vector, structs, indexed, etc.)
- Used by Scientific Applications
  - MILC, WRF
  - NAS Parallel Benchmarks (MG)

MPI Derived Datatypes

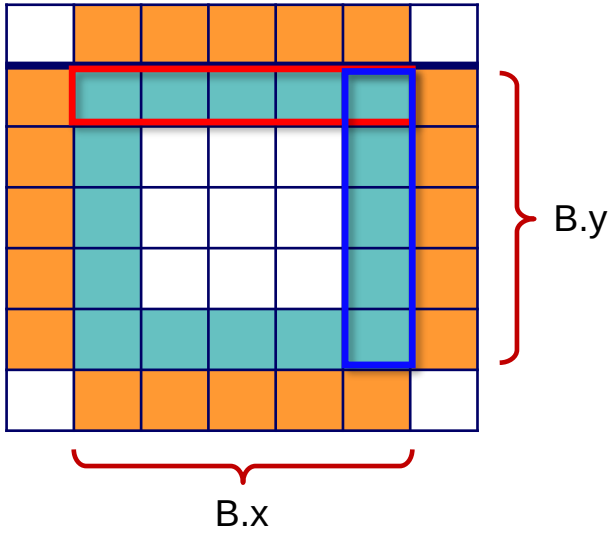
MPI_Type_contiguous

MPI_Type_vector

Nested Type Example



Halo Exchange Example

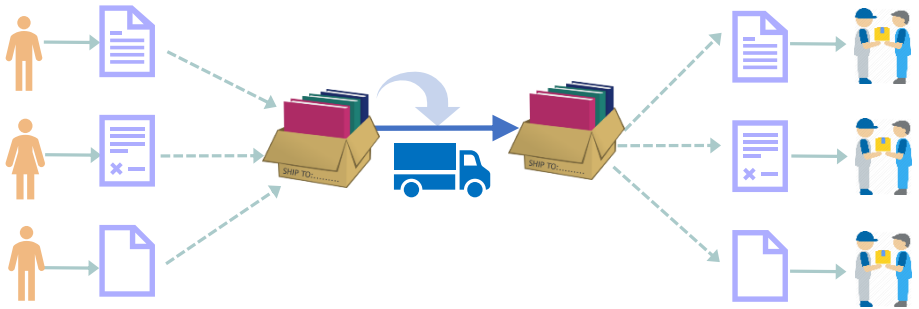


MPI_Type_contiguous (count=B.x, MPI_DOUBLE, …)

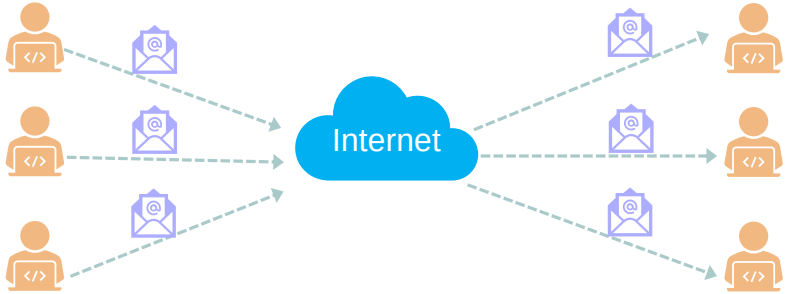
MPI_Type_vector (count=B.y, blocklen=1, stride=B.x+2, …)

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

Understanding Implementation Choices – An Example



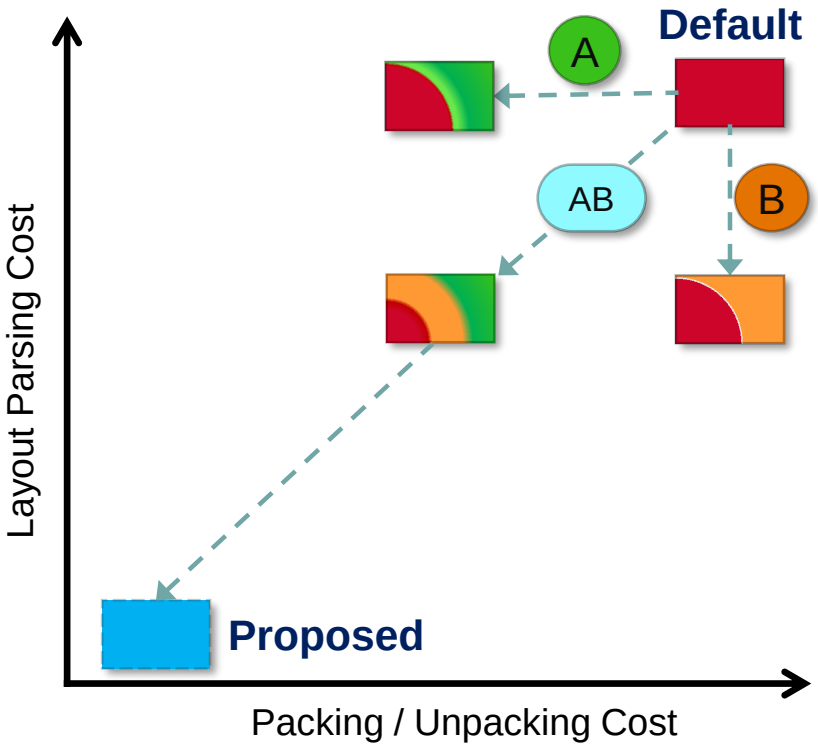
Packing/Unpacking



Zero-Copy

Standing on the shoulders of giants

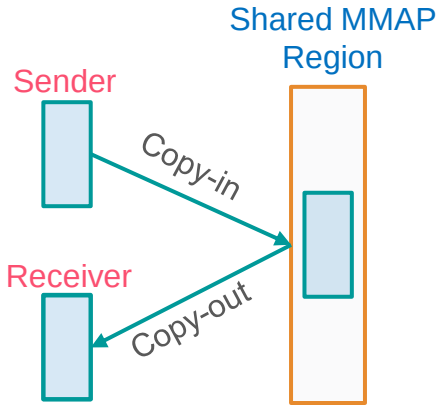
- **Two broad categories of optimizations**
  - A. Faster Pack/Unpack
    - Efficient packing [Gropp et al., Thakur et al.]
    - GPU Accelerated pack/unpack [Chu et al.]
  - B. Layout parsing optimizations
    - Flattening-on-the-fly algorithm [Träff et al.]
    - Automatic type generation [Kjolstad 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

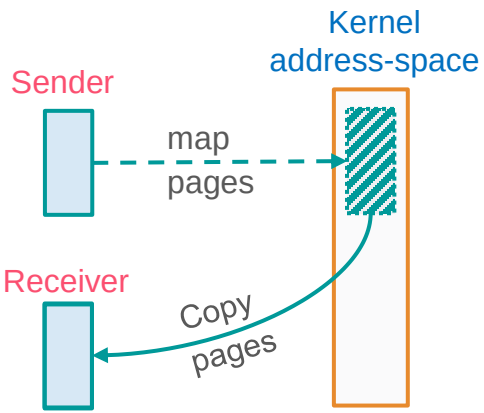


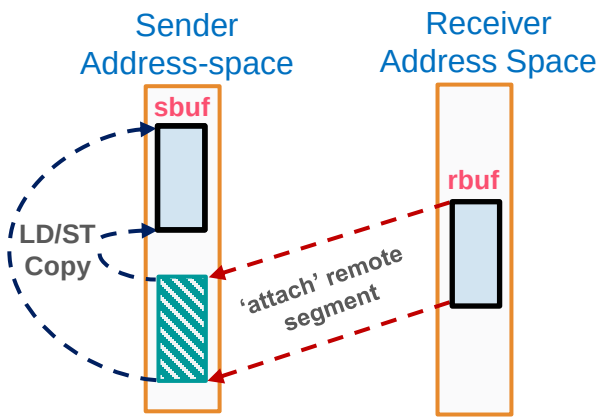
Outline

- Introduction and Motivation
  - Programming Models
  - Derived Datatypes
- **Existing Designs for Derived Datatypes**
  - Shared Memory based Packing/Unpacking
  - Challenges and Limitations
- **FALCON: Zero-copy based Derived Datatypes**
  - Basic Design and limitations
  - Pipelined Zero-copy
  - Memoization based Design
  - Design Optimizations
- **Results and Discussion**
  - Micro-benchmark Evaluation
  - Application Kernels: WRF, MILC, NAS_MG
  - Scalability Results: 3D-Stencil
- **Conclusion and Future Impact**

Realizing Intra-node Communication in MPI







Shared Memory

Requires two copies
No system call overhead
Better for Small Messages

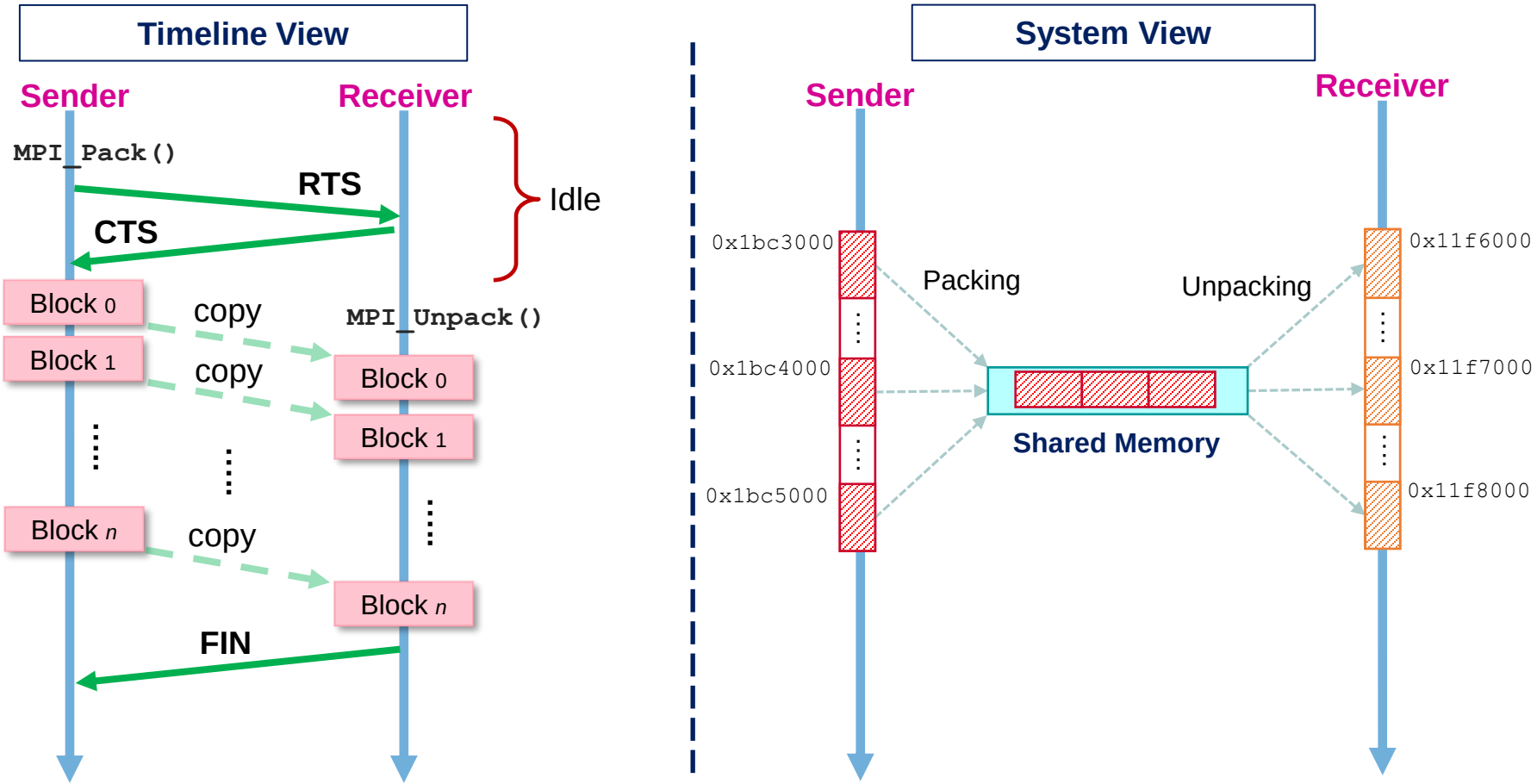
Kernel-assisted Copy (CMA)

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

Shared Address Space (XPMEM)

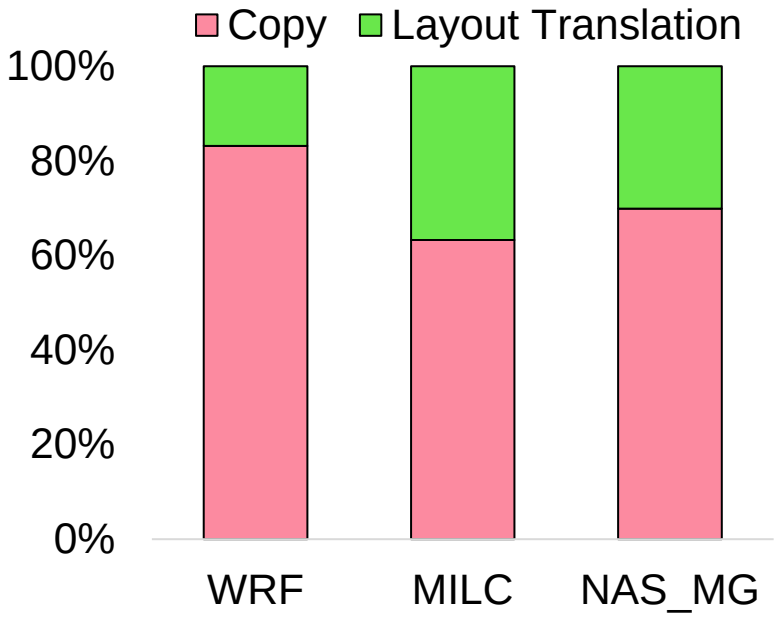
No system call overhead
User-space Zero-copy
Good for Small/Large Messages

Existing Pipelined Packing/Unpacking based Design



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!!
- **Zero-copy datatype processing**
  - Can it be better than state-of-the-art?



Cost breakdown of existing Pack/Unpack designs on Broadwell

Broad Questions

1. Can zero-copy based Datatype processing be a viable solution?

2. Can we quantify the associated benefits and overheads of zero-copy based design?

3. Can novel designs be proposed to overcome the limitations of zero-copy designs to achieve higher performance benefits?

4. What are the expected performance benefits for real world scientific applications?

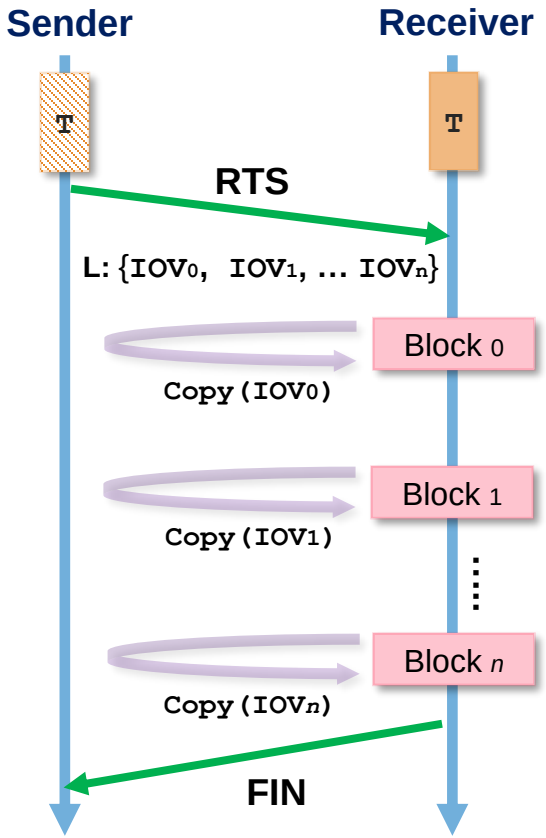
FALCON — FAst and **L**ow-overhead **Zero-copy** MPI datatype processing **CO**mmunication e**N**gine

Outline

- Introduction and Motivation
  - Programming Models
  - Derived Datatypes
- Existing Designs for Derived Datatypes
  - Shared Memory based Packing/Unpacking
  - Challenges and Limitations
- **FALCON: Zero-copy based Derived Datatypes**
  - Basic Design and limitations
  - Pipelined Zero-copy
  - Memoization based Design
  - Design Optimizations
- **Results and Discussion**
  - Micro-benchmark Evaluation
  - Application Kernels: WRF, MILC, NAS_MG
  - Scalability Results: 3D-Stencil
- **Conclusion and Future Impact**

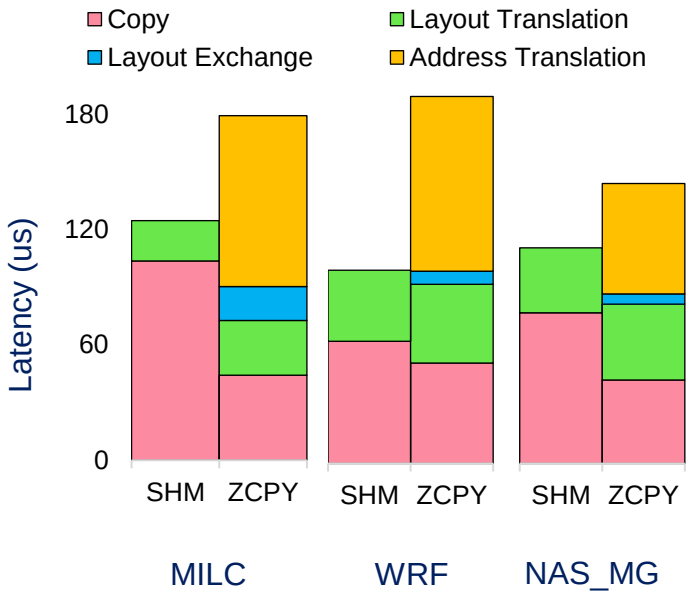
FALCON: Basic Zero-copy Design

- 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 using CMA or XPMEM
  - XPMEM offers user-space transfers
- Receiver sends a FIN packet
  - Transfer complete



Limitations of Basic Zero-copy design

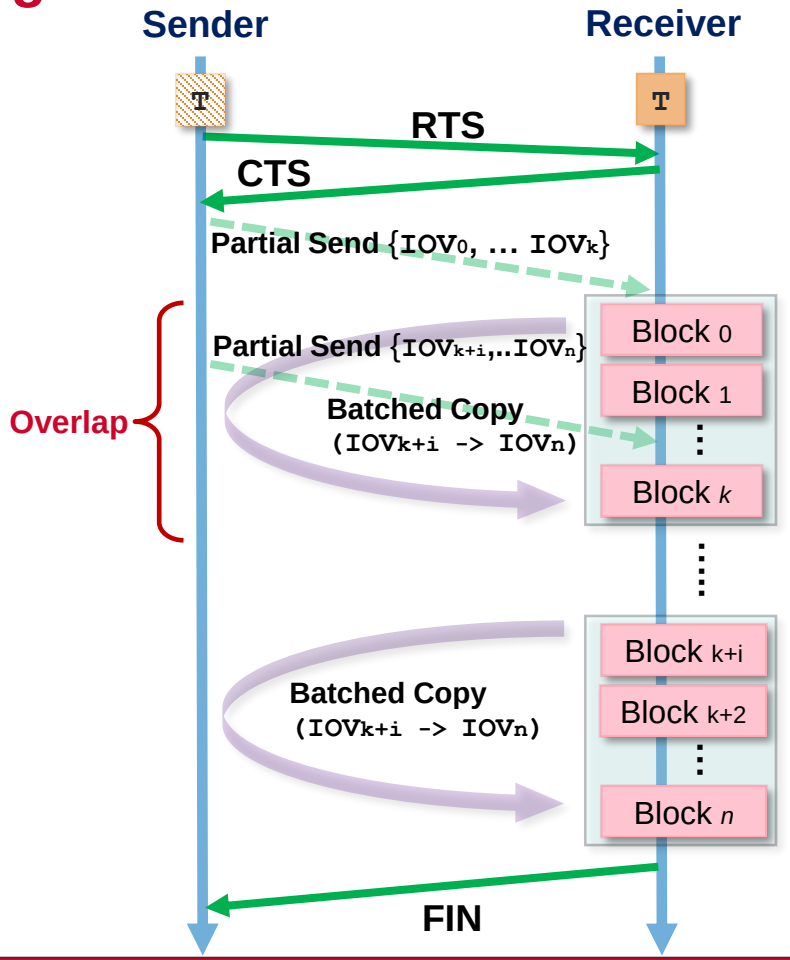
- **Layout-translation** is still required
- **Layout-exchange**
  - Layouts are local to rank in MPI
  - Sender has to send its layout to receiver
- **Remote address translation** for XPMEM
  - XPMEM attach
- **High fragmentation** means large IOV lists
  - RTS can exceed actual payload
- Combined overheads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



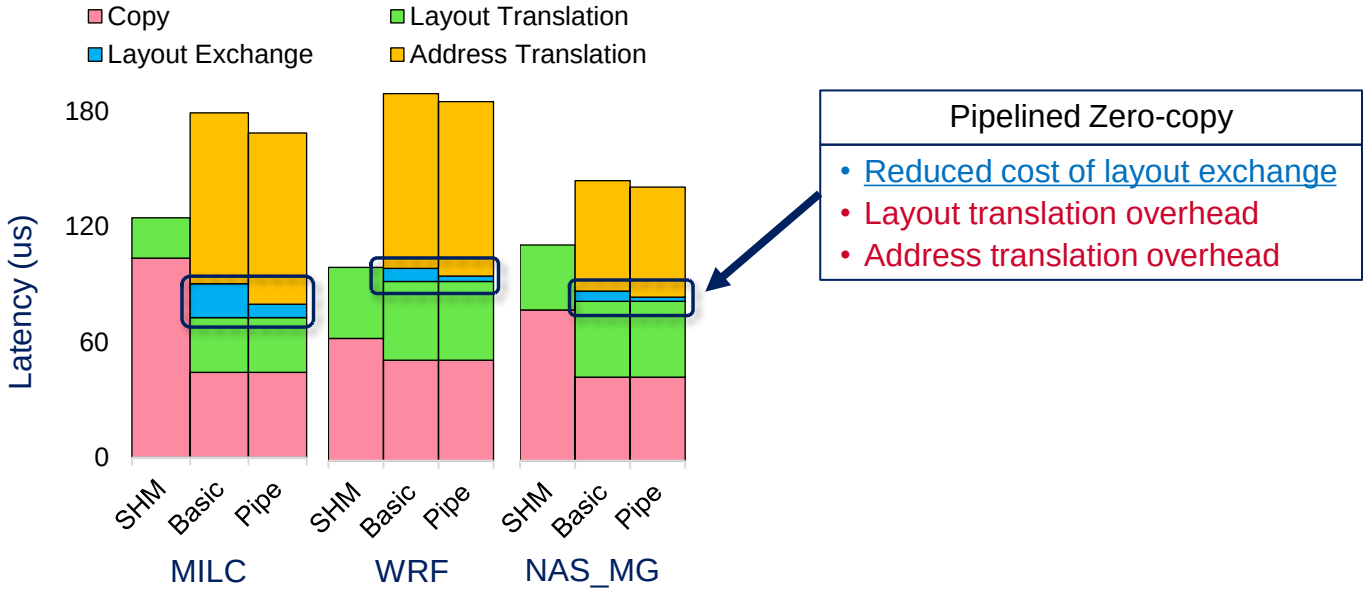
The time for data copy has been reduced but at additional cost are added.

FALCON: Pipelined Zero-copy Design

- Sender and receiver translate local layouts
- Sender and Receiver perform handshake
- Sender sends first *k*-IOVs
- Receiver copies *k*-blocks of data
- Sender posts next chunk of IOVs
  - Overlapped with receiver's copy
- **Layout-exchange** overhead mitigated
  - Other overheads still present



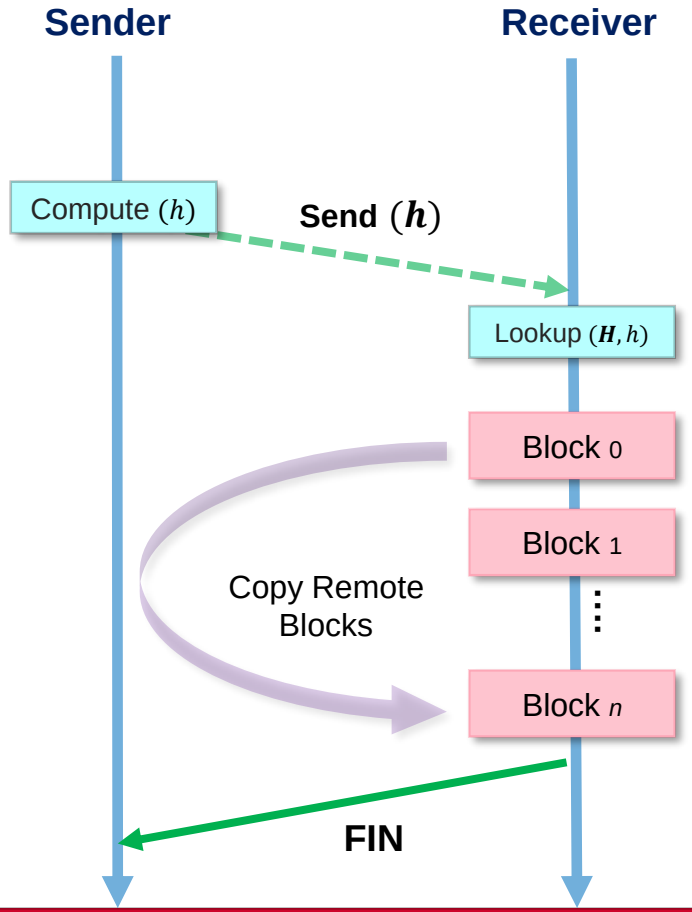
Benefits of Pipelined Zero-copy Design



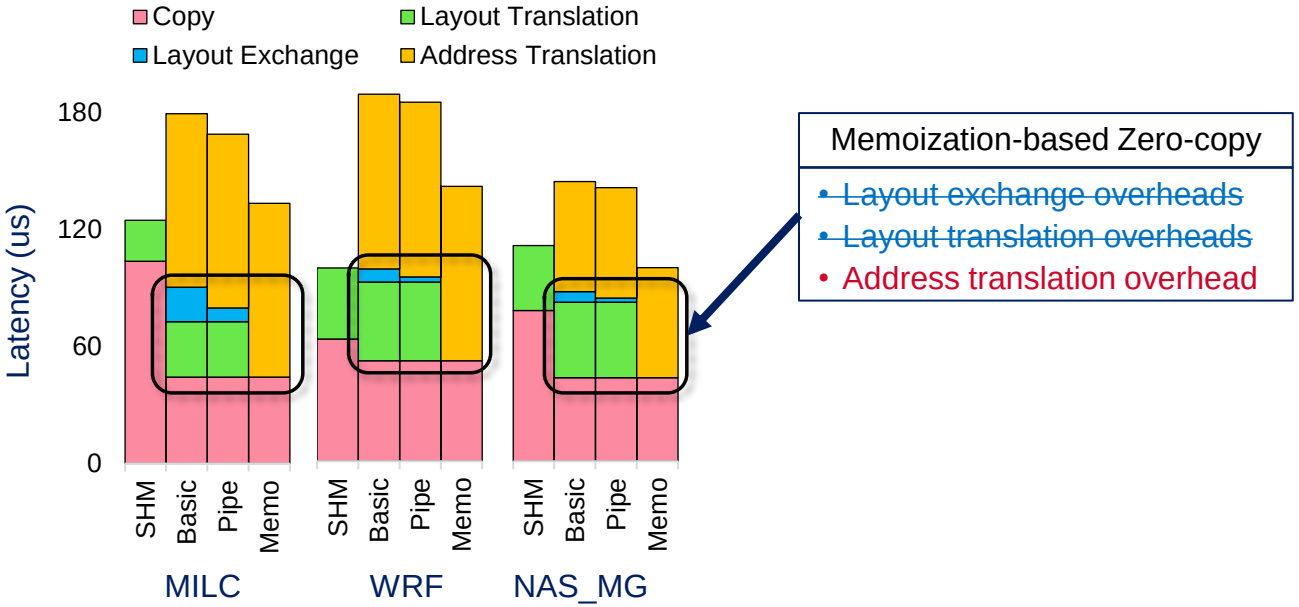
Pipelined Zero-copy reduced the **layout exchange** overhead by overlap and pipelining

FALCON: Memoization based Zero-copy Design

- Exploit application's layout re-usability
  - Sender memoizes **translated** layouts (L)
  - Receiver memoizes **exchanged** layouts (L')
  - Hash Table (H) stores $< k, v > = < h, L >$
- Avoids unnecessary layout exchange
  - Sender sends computed hash value (h) in RTS
  - Receiver looks-up (h) in Table (H')
- If Hash (h) is found in (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) and (L)
  - Receiver add (h) and (L) to (H')
  - Receiver copies sender's data using (L)



Benefits of Memoization based Zero-copy Design

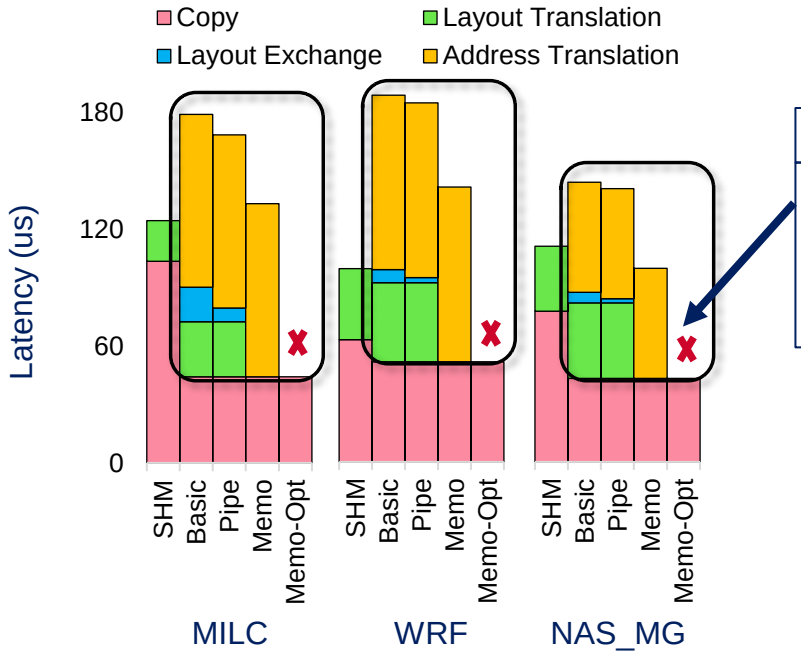


Memoization based design additionally reduced the **layout translation** overhead

FALCON: Design Optimizations

- **Avoiding Remote Virtual Address Translation**
  - XPMEM attached segments are cached
  - Future accesses re-use attached IOVs
  - No costly registration/de-registration required
- **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>*
- **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**

Outline

- Introduction and Motivation
  - Programming Models
  - Derived Datatypes
- Existing Designs for Derived Datatypes
  - Shared Memory based Packing/Unpacking
  - Challenges and Limitations
- FALCON: Zero-copy based Derived Datatypes
  - Basic Design and limitations
  - Pipelined Zero-copy
  - Memoization based Design
  - Design Optimizations
- **Results and Discussion**
  - Micro-benchmark Evaluation
  - Application Kernels: WRF, MILC, NAS_MG
  - Scalability Results: 3D-Stencil
- **Conclusion and Future Impact**

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

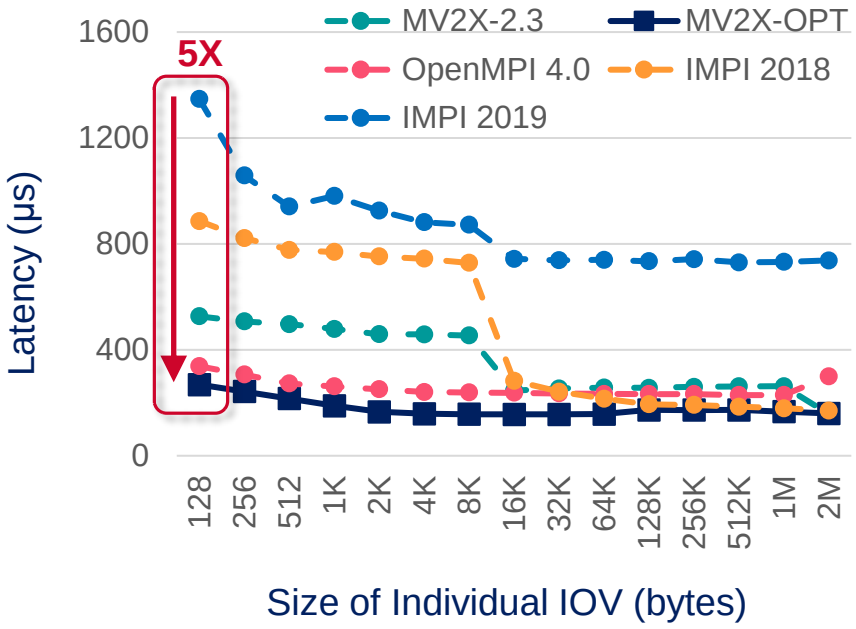
Performance Evaluation: Cluster Testbeds

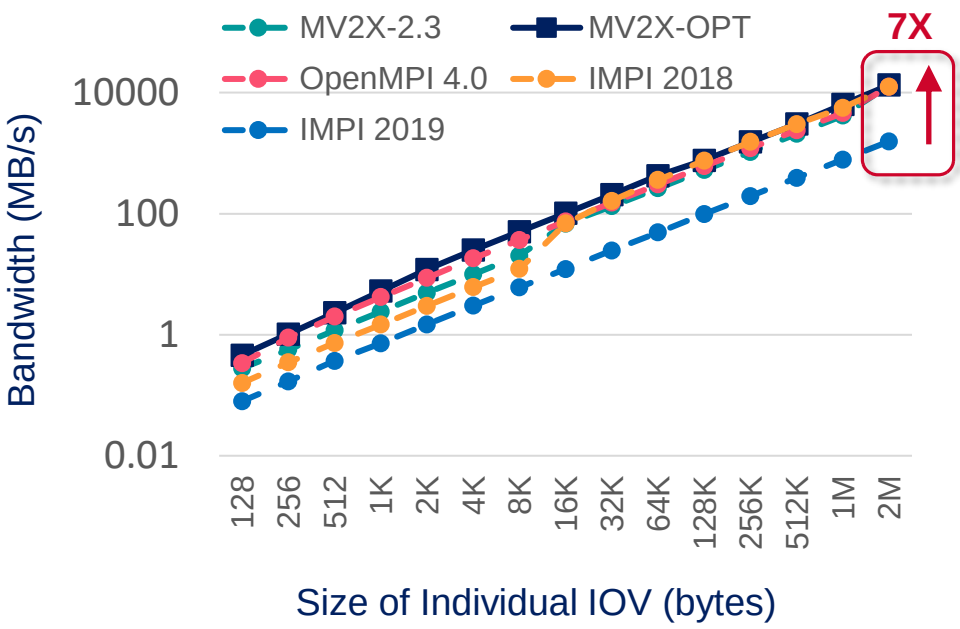
Hardware Specification of Cluster Testbeds

Specification	Xeon	Xeon Phi	OpenPOWER
Processor Family	Intel Broadwell	Knights Landing	IBM POWER8
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)

- Due to time limitations, here we only show Broadwell and OpenPOWER results
  - Please refer to the paper for more details on KNL results

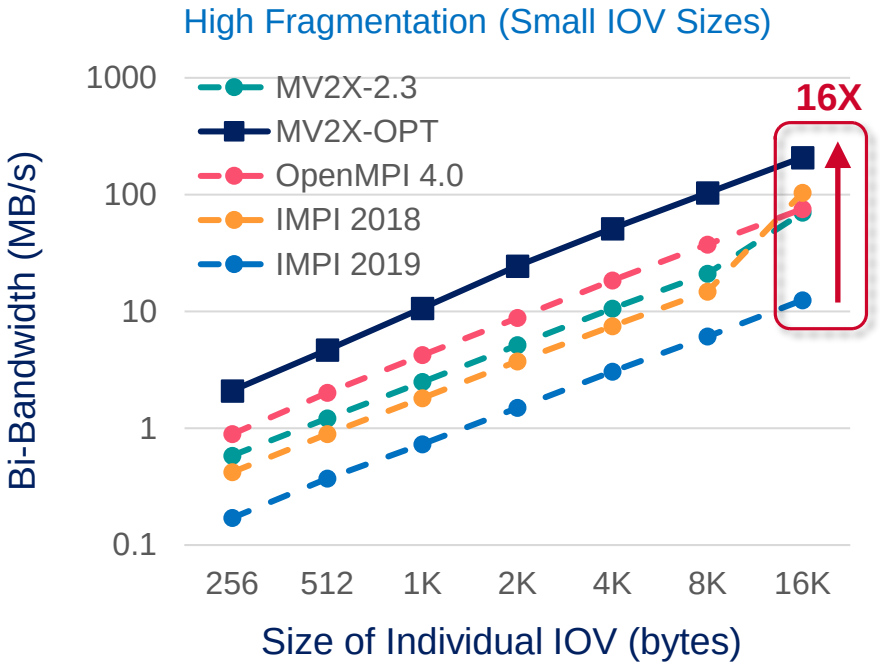
MicroBenchmark Level Evaluation - Broadwell

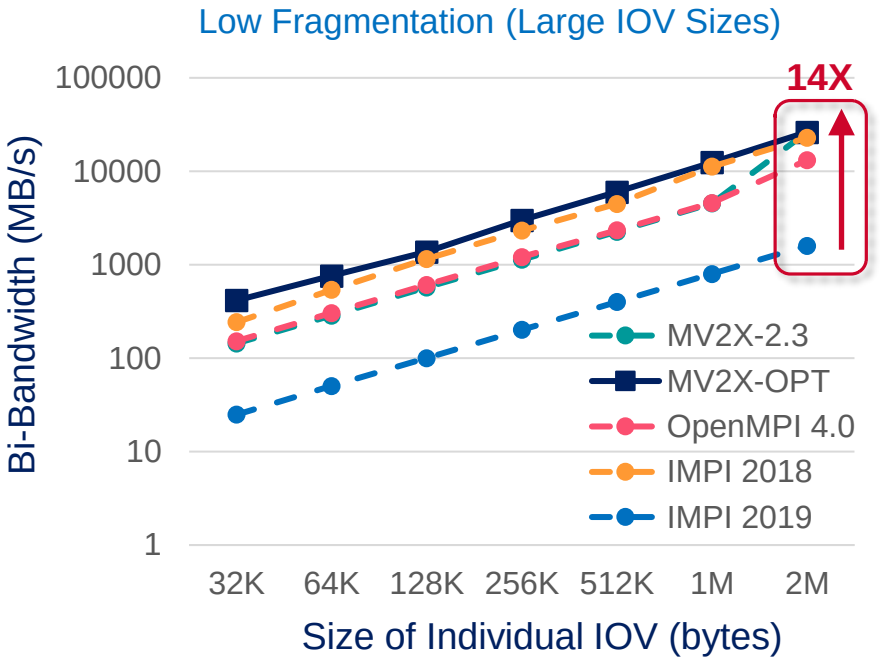




- Optimized memoization-based design in MVAPICH2X-OPT shows significant benefits
- Up to **4.8X** improved latency over Intel MPI 2019 and **19%** improvement over Open MPI for small IOVs.

MicroBenchmark Level Evaluation - Broadwell (Contd.)





- Significant improvement in bi-bandwidth as compared to other MPI libraries
- Up to **2.8X** and **16X** improved bandwidth over Open MPI and Intel MPI 2019 for 16KB IOVs

Performance Evaluation: Application Kernels

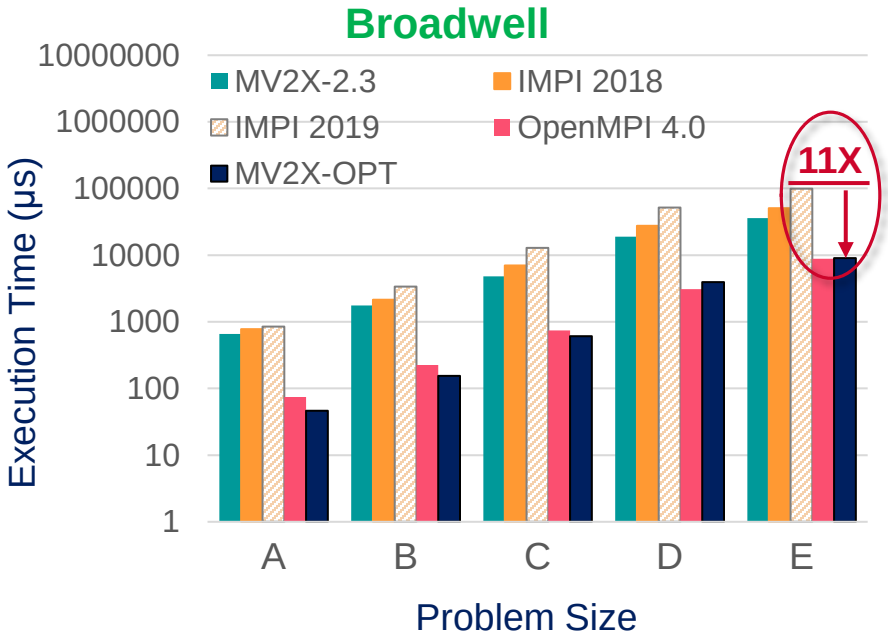
Communication Kernels

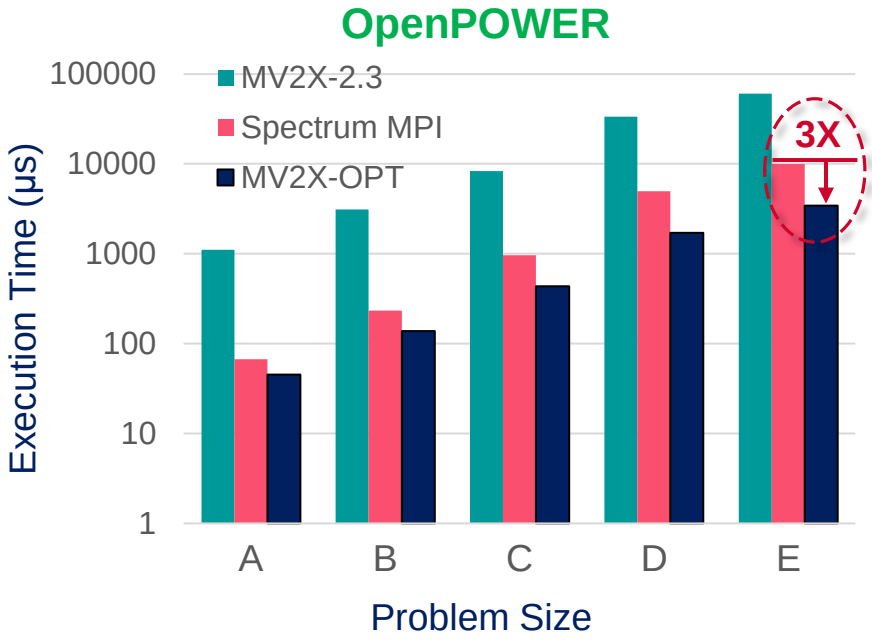
Application Kernel	Application Domain	Datatype Layout
MILC_su3_zd	Quantum Chromodynamics	Nested Vectors for 4D face exchanges
WRF_y_vec	Atmospheric Science	Nested Vectors and Subarrays
NAS_MG_z	Fluid Dynamics	Vectors and Nested vectors for 3D face exchanges
3D-Stencil	Stencil Communication	7-point stencil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

[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

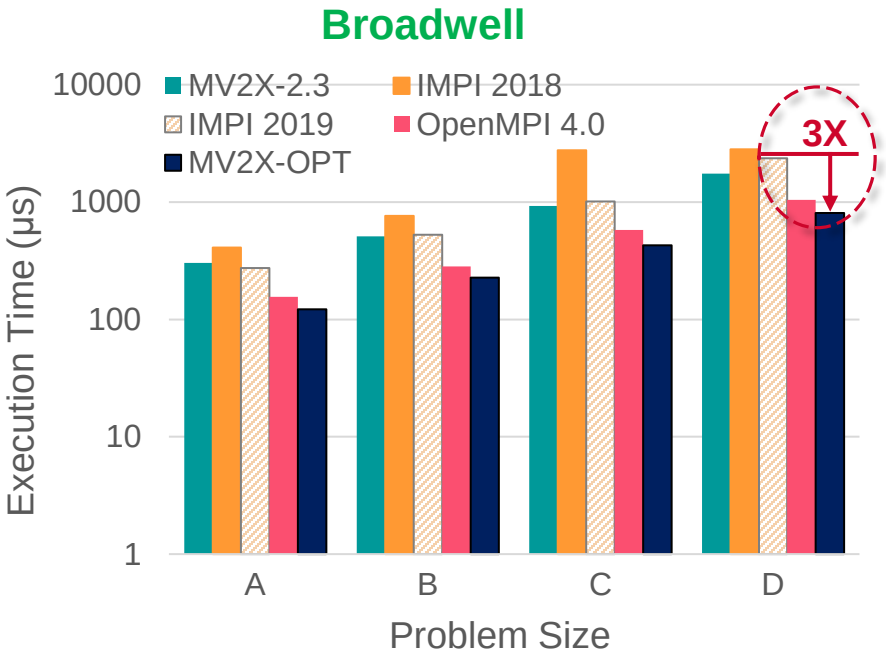


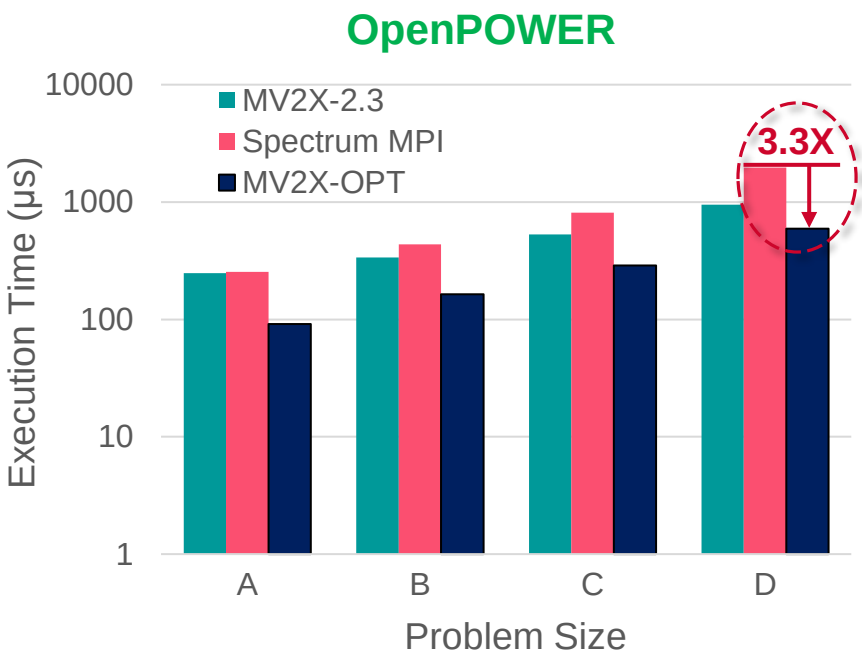


Grid Dimensions – A = (16, 16, 32, 32); B = (32, 32, 32, 32); C = (64, 64, 32, 32); D = (128, 128, 32, 32); E = (128,128,64,64)

- On Broadwell, for Problem-B (768-KB), up to **31%** improvement over Open MPI, up to **11X** over MVAPICH2-X, and up to **3X** over Spectrum MPI is observed

Application Kernels – WRF



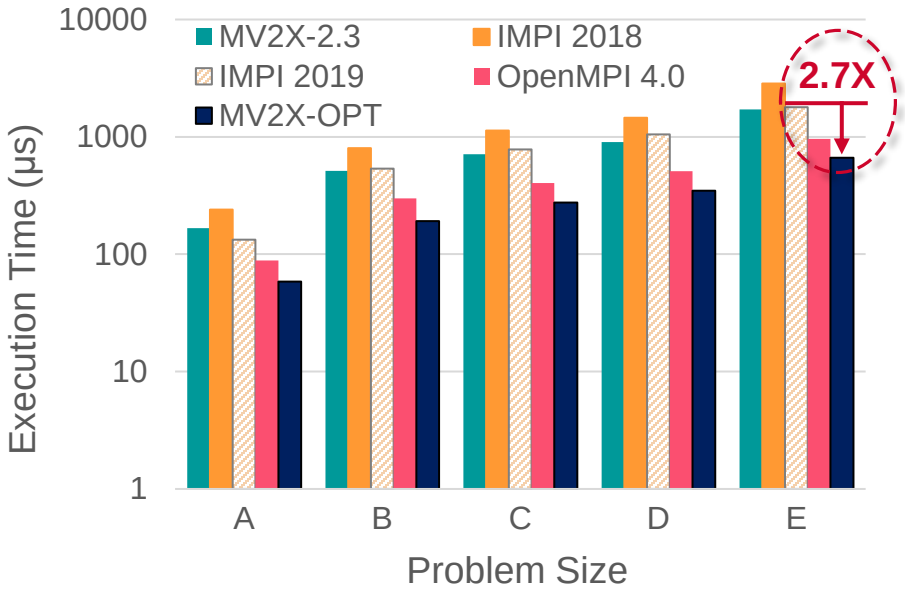


Params (ims, ime, is, ie) – A = (4, 140, 8, 136); B = (4, 268, 264, 8); C = (4, 524, 8, 520); D = (4, 1036, 8, 1032)

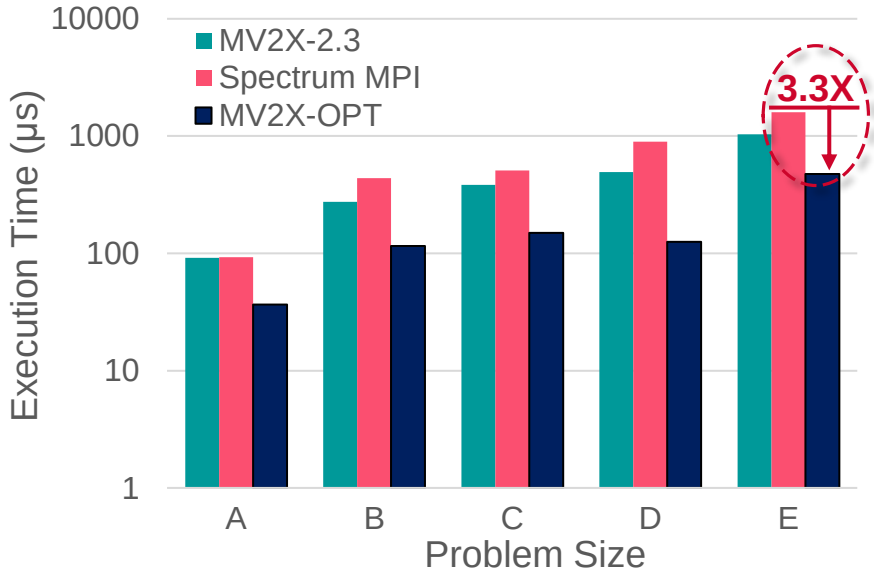
- On OpenPOWER using Problem-D, up to **3.3X** improvement over Spectrum MPI
- On Broadwell, up to **2.1X** and **3X** improved latency over MVAPICH2-X and Intel MPI 2019

Application Kernels – NAS_MG_z

Broadwell



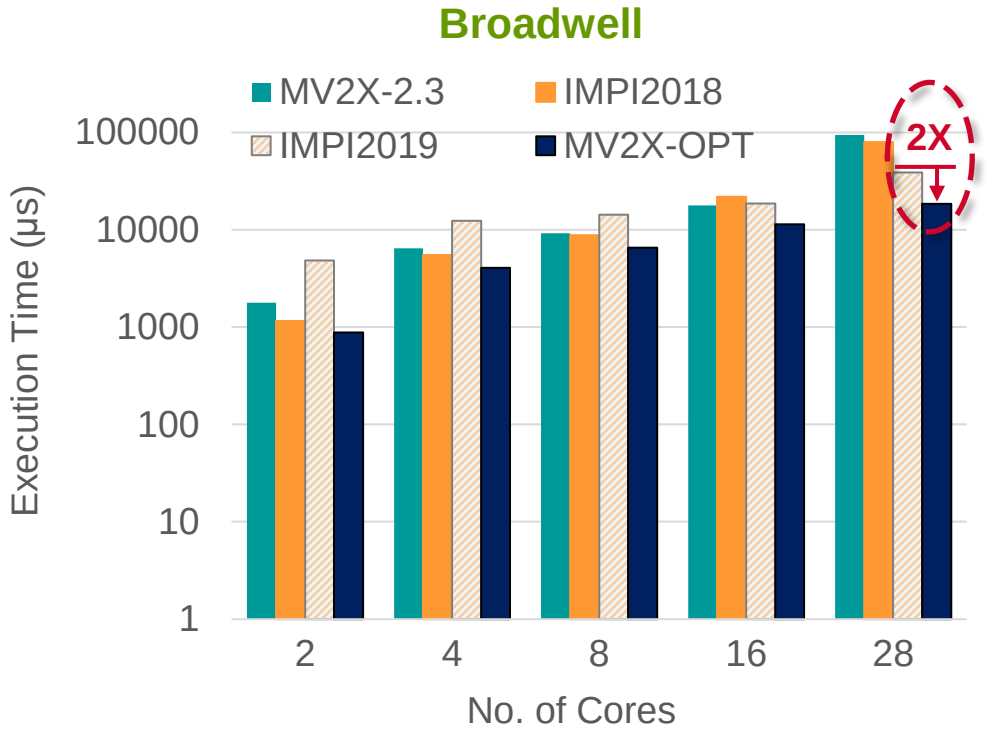
OpenPOWER



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

- On Broadwell, up to **2.7X** and **4.2X** improvement over MVAPICH2-X and Intel MPI 2018
- Using Problem-E, On OpenPOWER up to **3.3X** improvement over Spectrum MPI

Application Kernels – 3D-Stencil



- 7 point stencil with subarray datatypes. Grid dimension = 512^3
- On 28-cores, up to **5X** and **2X** improvement over MVAPICH2-X and Intel MPI, respectively

Outline

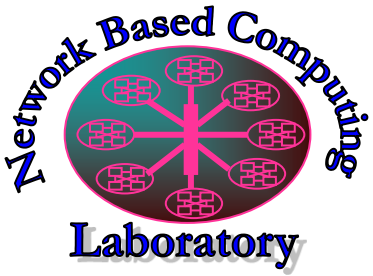
- Introduction and Motivation
  - Programming Models
  - Derived Datatypes
- Existing Designs for Derived Datatypes
  - Shared Memory based Packing/Unpacking
  - Challenges and Limitations
- FALCON: Zero-copy based Derived Datatypes
  - Basic Design and limitations
  - Pipelined Zero-copy
  - Memoization based Design
  - Design Optimizations
- Results and Discussion
  - Micro-benchmark Evaluation
  - Application Kernels: WRF, MILC, NAS_MG
  - Scalability Results: 3D-Stencil
- **Conclusion and Future Impact**

Conclusion and Future Impact

- Multi-/many-core architectures bring new challenges
  - Fundamentally re-think MPI derived datatype designs
  - Provided optimized designs to address the limitations of MPI derived datatypes
- Achieved significant improvements over state-of-the-art
  - Micro-benchmarks and Scientific applications: MILC, WRF, NAS_MG, 3D-Stencil
- **Future Impact on the HPC eco-system**
  - Designs geared towards next-generation many-cores in Exa-scale era
  - Transparent De-coupling of application optimizations and data-movement
  - Designs applicable to any MPI implementations
  - Application developers need not worry about Packing/Unpacking of data

Thank You!

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/

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

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

Proposed Enhancements to MPI Semantics

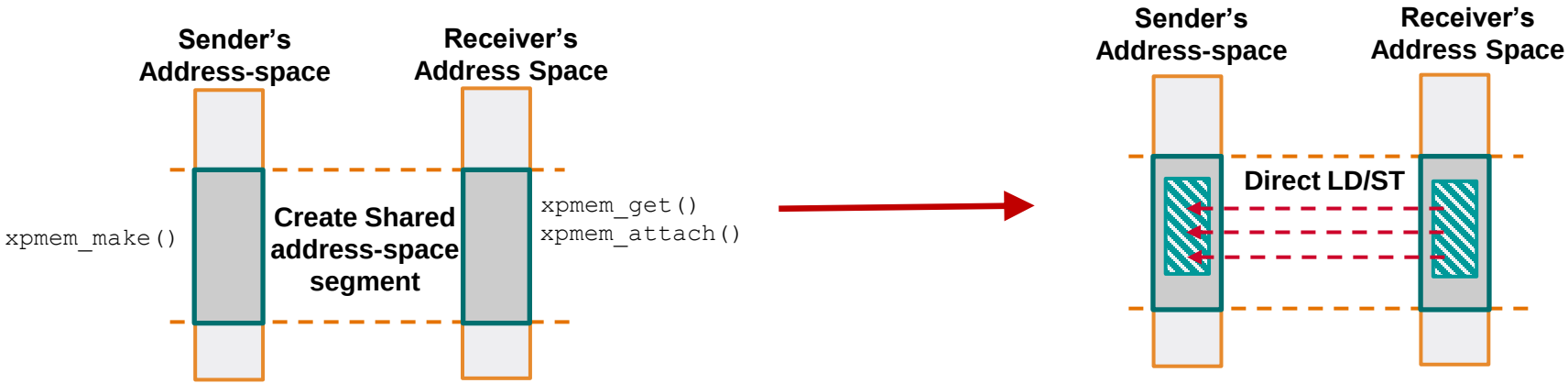
- Datatype semantics are process local
  - No knowledge of each other's layout
- **Collective datatype creation routines**
  - Global monotonically increasing handles
  - Unique values across all ranks
  - No need to exchange layouts

```
/* Create a new Datatype */
int MPIX_Type_commit_comm(
        MPI_datatype *datatype,
        MPI_Comm comm);

/* Free an existing Datatype */
int MPIX_Type_free_comm(
        MPI_datatype *datatype,
        MPI_Comm comm);
```

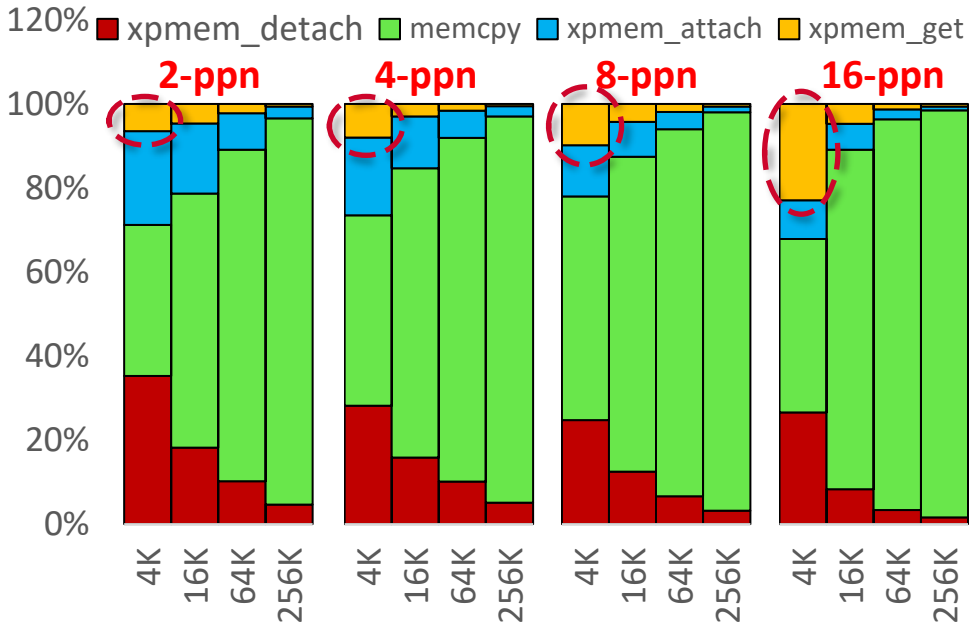
Shared Address-space based Communication

- 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

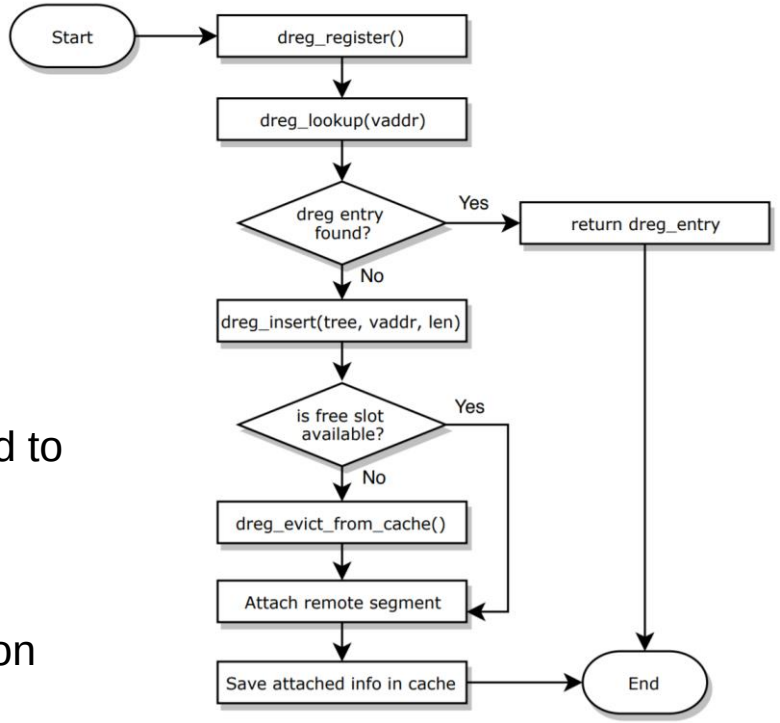
- XPMEM based one-to-all latency benchmark
  - Mimics rooted collectives
- A process needs to attach to remote process before memcpy
- Up to 65% time spent in XPMEM registration for short message (4K)
- Increasing PPN increases the cost of `xpmem_get()` operation
  - Lock contention
  - Pronounced at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

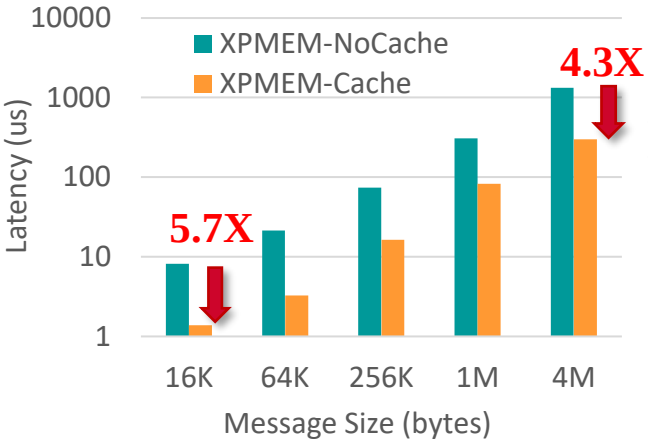
Registration Cache for XPMEM based Communication

- 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

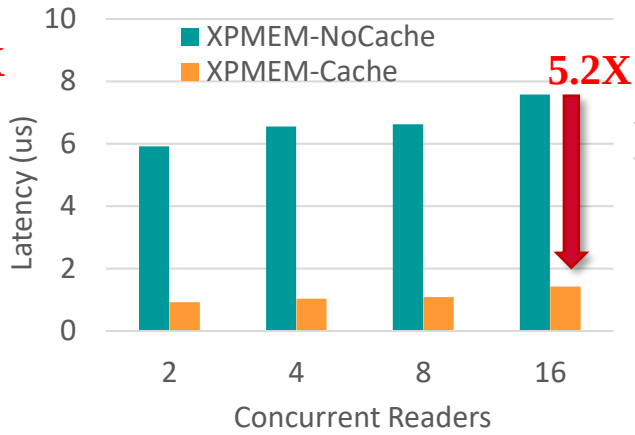


A high-level flow of the proposed Dynamic Registration Cache

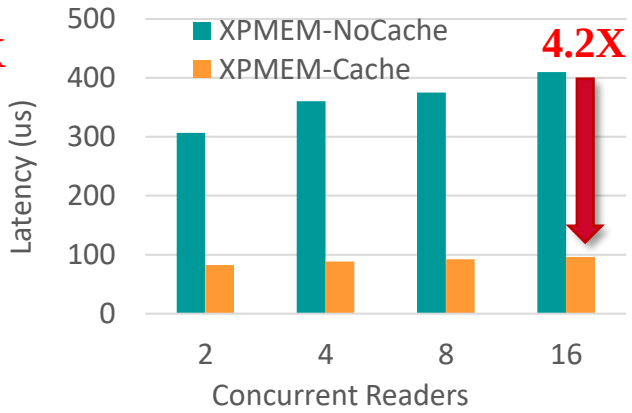
Impact of Registration Cache on the Performance of XPMEM based Point-to-point Communication



Two-process latency at varying messages



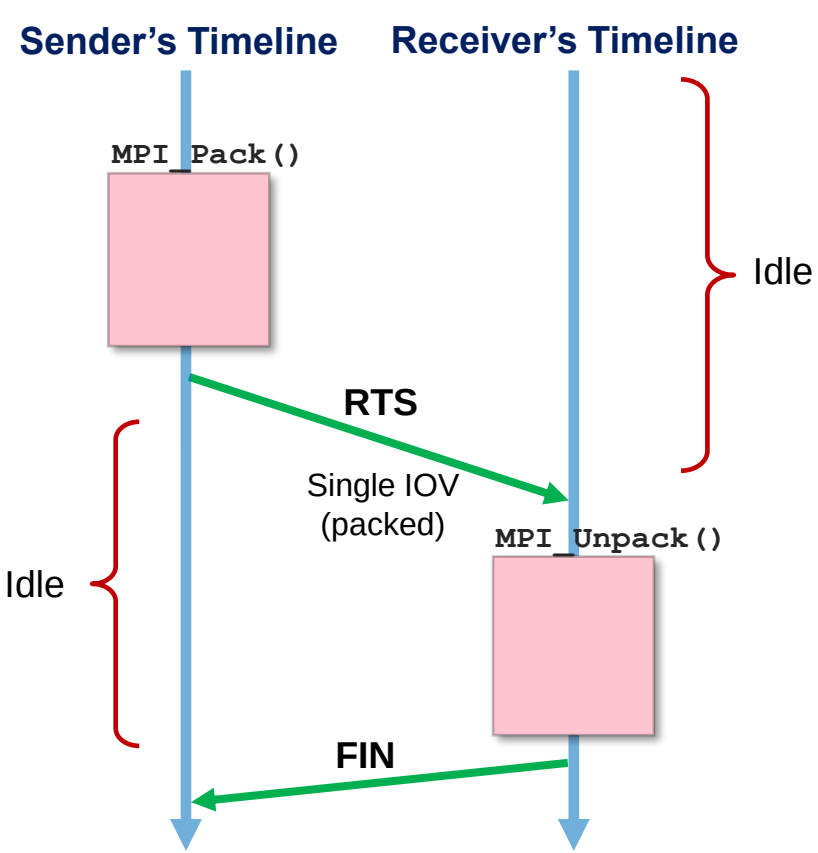
Multi-process latency at 16KB message

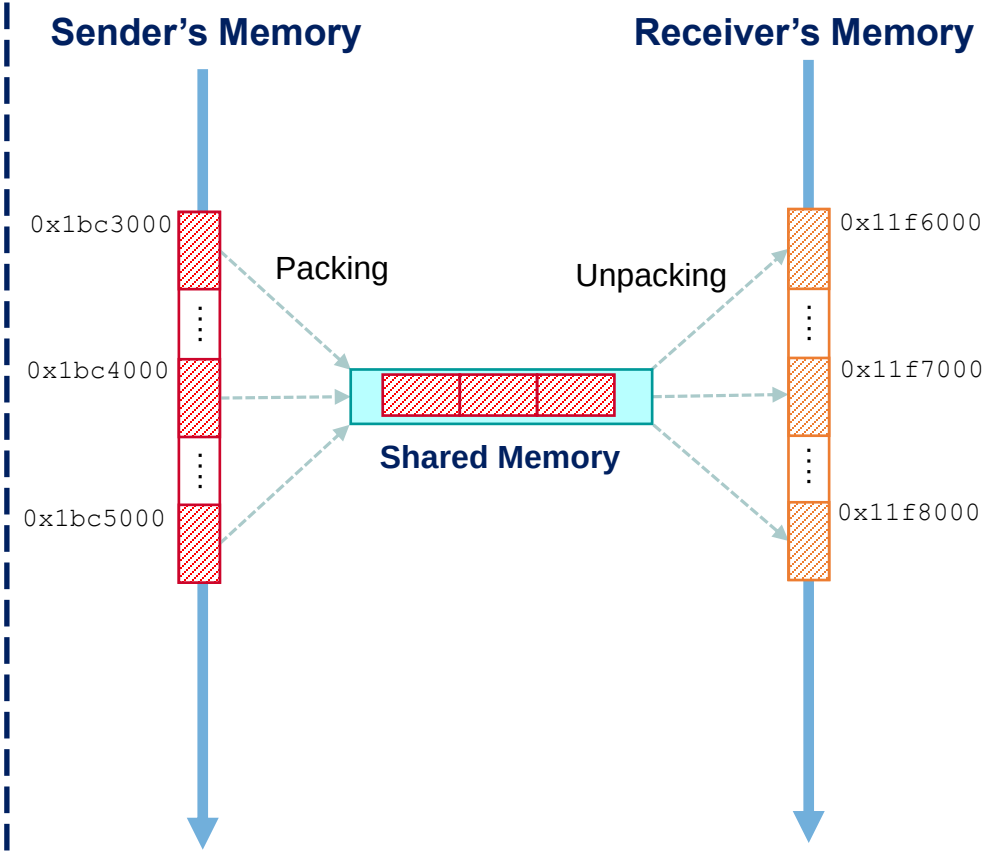


Multi-process latency at 1MB message

- 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

Naïve Pack/Unpack Design using Shared Memory





Memoization-based Zero-copy Design – I

Algorithm 1: Memoization Protocol: Sender Side

$h \leftarrow$ Generate_Hash (req)
if $h_s \in Table_s$.keys **then**
  $l_s \leftarrow$ Value_At ($Table_s$, h_s)
  Append_RTS (pkt, h_s)
  Send_RTS (pkt)
else
  $h_s \leftarrow$ Generate_Hash (req)
  $l_s \leftarrow$ Translate_Layout (req)
  Insert_Hash ($Table_s$, h_s, l_s)
  Append_RTS (pkt, h_s, l_s)
  Send_RTS (pkt)
end if

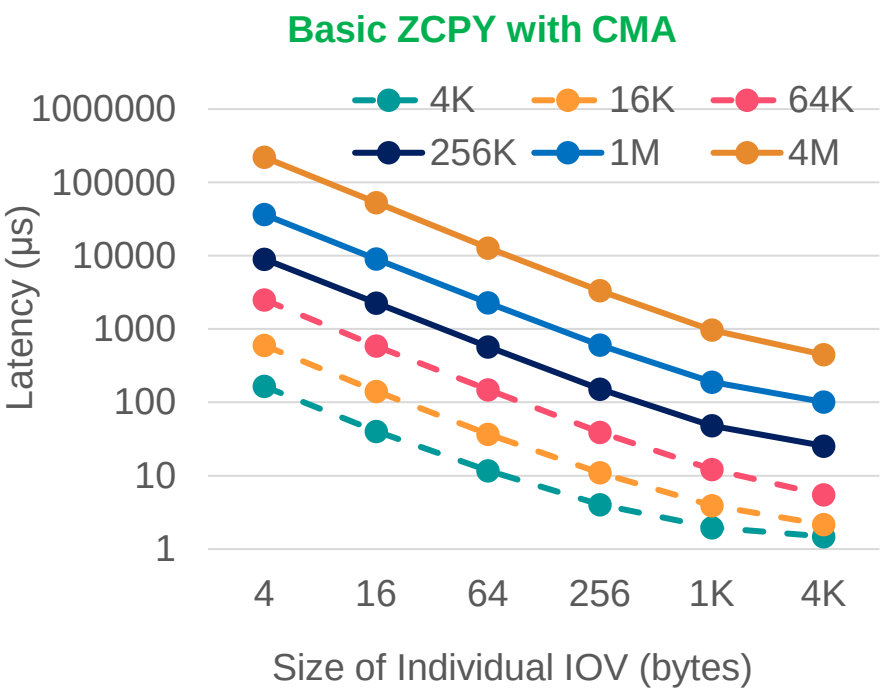
Memoization-based Zero-copy Design – II

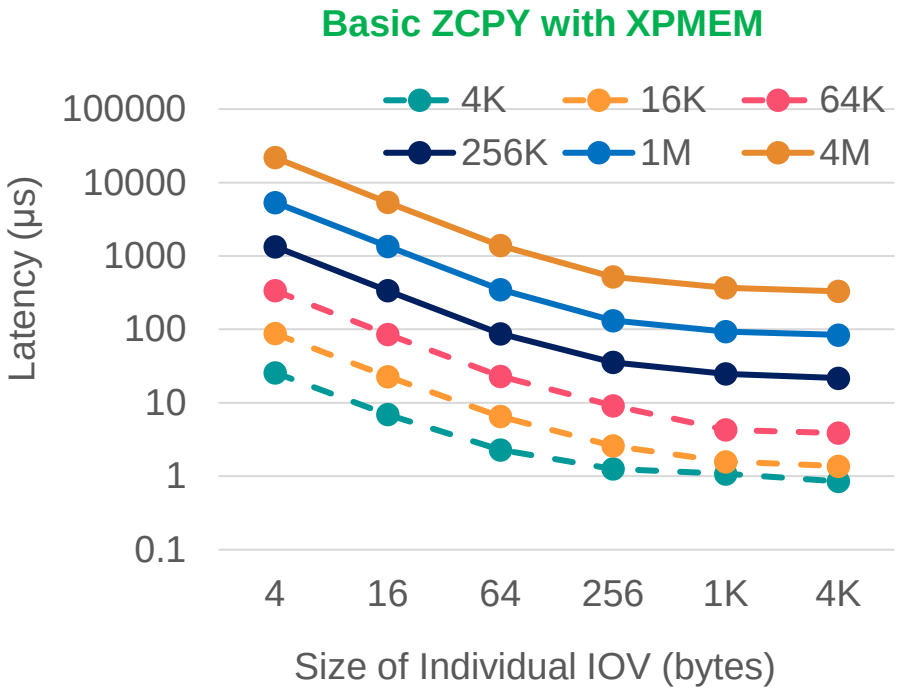
Algorithm 2: Memoization Protocol: Receiver Side

Ensure: Receiver has received the RTS packet
Ensure: Receiver's local layout was previously cached and found as IOV_{local}

$Table_r \leftarrow Tables[\text{srank}]$
$h_s \leftarrow$ Extract_Hash (pkt)
if $h_s \in Table_r$.keys **then**
  $l_s \leftarrow$ Value_At ($Table_r$, h_s)
  Copy_Remote_Segments(l_s, IOV_{local})
  Send_FIN (pkt)
else
  $h_s \leftarrow$ Extract_Hash (pkt)
  $l_s \leftarrow$ Extract_Layout (pkt)
  Insert_Hash ($Table_r$, h_s, l_s)
  Copy_Remote_Segments(l_s, IOV_{local})
  Send_FIN (pkt)
end if

Impact of Datatype Fragmentation on Basic Zero-copy





- The lines represent total size of the communication buffer ranging from 4-KB to 4-MB.
- Higher fragmentation leads to significant degradation while increasing IOV size helps