Communication Optimization Research

Presentation Coordinators:

Prof. Anthony Skjellum and Prof. Amanda Bienz

Research Lightning Talks/Posters

- Riley Shipley, TN Tech: RAPIDS Channel API: Improved Persistent Communication
- Evan Suggs, TN Tech: Enabling Stream-Triggered MPI+X backends for Cabana Benchmarks
- Nicole Avans, TN Tech: Enabling Performant Inter-Node Communication for Kokkos Views
- Gerald Collom, UNM: Partitioned Communication in Sparse Matrix Operations
- Evelyn Namugwanya, TN Tech: Optimizing Collective Communication Using MPI RMA & Generalized Algorithms
- Mike Adams, UNM: Optimizing GPU-Aware Allreduce Operations
- Shannon Kinkead, UNM: Scaling All-to-all Operations Across Emerging Many-Core Supercomputer

Point-to-point and neighbor exchange communication abstractions

- Riley Shipley, TN Tech: RAPIDS Channel API: Improved Persistent Communication
- Evan Suggs, TN Tech: Enabling Stream-Triggered MPI+X backends for Cabana Benchmarks
- Nicole Avans, TN Tech: Enabling Performant Inter-Node Communication for Kokkos Views
- Gerald Collom, UNM: Partitioned Communication in Sparse Matrix Operations

RAPIDS Channel API: Improved Persistent Communication

Riley Shipley, Anthony Skjellum, Patrick Bridges, Purushotham Bangalore

State of the Art

- MPI has been integral to HPC from the start
- Vendor-locked alternatives (NCCL et al) are starting to take the lead
- MPI has been too slow to adapt and innovate due to standardization process (ex: GPU support)
- Optimization strategies have been explored extensively
- Generality limits performance potential (ex: MPI_Wait)

What is RAPIDS?

- Reduced API Data-transfer Specifications
- Goal: Define styles of communication used by applications as minimal (RISC-like) APIs that are composable
- Separating each kind of communication into its own library:
  - Reduces overhead
  - Promotes innovation
  - Simplifies adaptation to new architectures

Channel API

- Designed for stencil-based applications: *hypre*, AMG, PIC codes, etc.
- Eliminates matching and tag queues by creating dedicated one-way Channels between processes, implemented over RMA
- Tag semantics can still be used by making multiple Channels between the same ranks
- RMA buffer can be segmented to allow multiple put operations to occur without synchronizing, unlike persistent operations

Future APIs

GrabBag

- Irregular applications that know message destination, but not the source
- Ex: xRAGE and Cabana
- Removes the requirement for specifying a source on receive
- Source delivered with data

Concurrency

- Applications with dense data layouts (GPU-based)
- Ex: Regular stencil codes
- Allows for independent thread / GPU communication progress

Concurrent Channel

- Applications with fixed dense data layouts
- Applies the concepts of Channel and Concurrency libraries
- Result is multi-thread communication that avoids queues

Cabana Abstractions

Jason Stewart, **Patrick Bridges**

Using Cabana to enable performant, application-facing C++ communication interfaces

- Cabana includes a variety of communication abstractions
  - Originally drawn from Trilinos Teuchos - CommunicationPlan, Distributor, Particle/Grid Halo
  - Simpler place for student innovation/development than Trilinos
  - Range of interesting standalone benchmarks that use these abstractions – MD, PD, MPM
  - Goal is to integrate back to Trilinos later
- Filling out range of communication patterns, backends
- Benchmarking
  - Ported and expanded irregular halo benchmark from L7 to Cabana to look at broader range of abstractions
  - Implemented simple stream-triggered halo exchange and complex fluid interface benchmarks to drive research
- Future work:
  - Stream-triggered irregular exchanges,
  - Collective abstractions for grids and AoSoAs
  - Support Kokkos Comm and stream-triggered backends

Expanding Cabana Communication Abstractions

- Added *Collector* class to complement Cabana Distributor
  - Distributor: You know how you are sending to but not who you are receiving from.
  - Collector: You know who you are receiving from but not who you are sending to.
  - Collector needed in sparse matrix operations, new Beatnik unstructured finite element halo
  - PR submitted to Cabana main branch under review
- Added Cabana Infrastructure to support multiple communication backends
  - CommSpace::Mpi, CommSpace::MpiAdvance inspired by Kokkos Comm Communication Spaces
  - Cabana design in collaboration with Stuart Slattery and Sam Reeve (ORNL)
  - PR in development for submission next month
- Optimizing Irregular exchange abstractions in the MPI Advance Communication Space
  - Leverage MPI Advance neighbor discovery and neighbor exchange optimizations (e.g. MPI_Alltoall_crs)
  - PR in development for submission next month
- Co-designing Stream Triggering Abstractions for MPI Advance and Cabana

Enabling Stream-Triggered MPI+X Backends for Cabana

Evan Drake Suggs, Patrick Bridges,
Derek Schafer, Anthony Skjellum

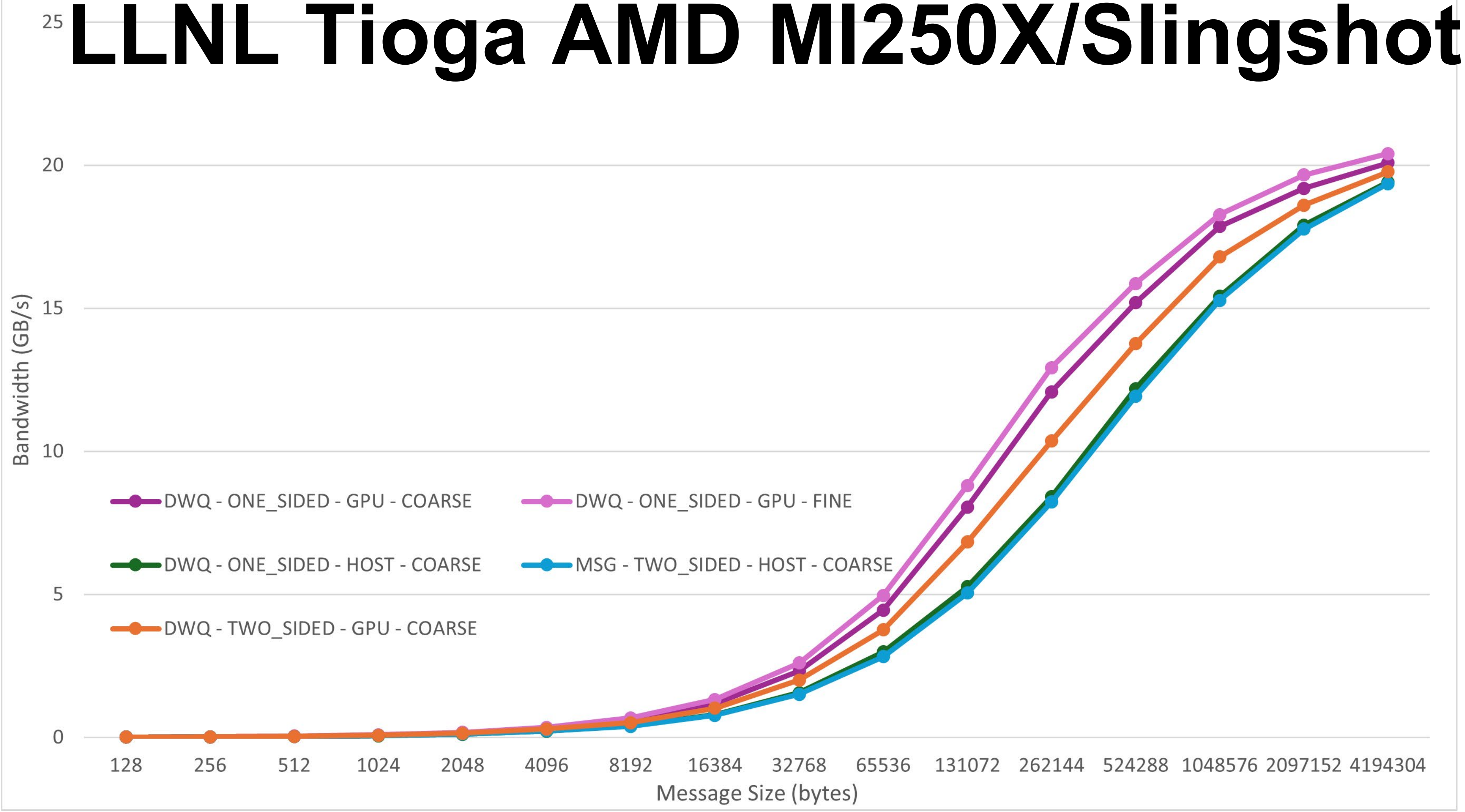
Stream Triggering for MPI

- Lots of different proposals for stream triggering for MPI
- None remotely close to standard, accepted, or portable
- Why is this hard?
  - Need to preserve existing semantics
  - Without adding lots of new operations (∀x: MPI_Enqueue_X)
  - Or relying on obscure, hard to use parts of the standard (e.g., PSCW)

	Area 1	Area 2: API Features				
Proposal	Control Path Used	Reuses Existing APIs	Changes API Semantics	Separate Initialize and Start	GPU Completion Support	Collective Support
MPI-GDS	Stream	Yes	Weaker	No	Full	Full[1]
MPI-ACX Enqueued	Stream	Yes[1]	No	Yes[1]	Full	No
MPICH Triggering	Stream	Yes	No[2]	No	Full	Partial
HPE Send-Recv	Stream	No	No	Yes	Full	No
Delorean	Stream	No[3]	No	Yes	Full	Full
HPE One-sided	Stream	Yes	Stronger[4]	Yes	Full	Group
Partitioned Comm.	Kernel	Yes	No	Yes	Partial[5]	Full[1]
HPE Persistent	Kernel	Yes	No	Yes	Full	No
Intel GPU-Init	Kernel	Yes	No	No	Full	No

Bridges, Skjellum, Suggs, Schafer, and Bangalore. **Understanding GPU Triggering APIs for MPI+X Communication**. In Proceedings of EuroMPI 2024.

Initial Benchmark Performance bears out on LLNL Tioga AMD MI250X/Slingshot 11



- Packing GPU ping-pong with MPI API on HPE CXI libfabric
- Lot of steps to fully exploit hardware
  - Packing to MI250X write-through memory
  - Readiness assertion avoids RTS/CTS
- 512KB bandwidth improved 33%, better for smaller messages
- Integrating into Cabana and Kokkos mini-applications

Integration of Stream-triggering

- We've identified an interface that works with MPI
- CUP-ECS's MPI Advance stream-triggering library works with several backends (CUDA, CXI, etc) to create a portable interface for stream triggering
- We created variants of Cabana that support stream triggering
  - Creates enqueue variants for scatter and gather
  - Uses stream-triggering's queue function to schedule and wait on the underlying communications
- We refactored our CabanaGhost benchmarks to utilize the new stream triggering interfaces added to Cabana

```
void Distributor::Distributor(ExecutionSpace &e)
{
    // Create non-blocking send and receive operations
    for ( int n = 0; n < num_n; ++n ) {
        ...
        auto recv_subview = Kokkos::subview(recv_buffer, recv_bounds);
        auto send_subview = Kokkos::subview(send_buffer, send_bounds);
        MPI_Recv_init( recv_subview.data(), recv_subview.size(), ...,);
        MPI_Send_init( send_subview.data(), send_subview.size(), ...,
    }
    // Pair up send/recv buffers, create a queue for starts and waits
    MPIX_Matchall(halo_ops.data(), halo_ops.size());
    MPIX_Queue_init(&queue, MPIX_QUEUE_TYPE_HIP, &e.hipStream());
}

void Distributor::distributeData(AoSoA_t& src, AoSoA_t& dst)
{
    // All operations are enqueued to the stream, which enqueue
    // them to the progress engine associated with the queue
    MPIX_Enqueue_startall(queue, halo_ops.data(), num_n);
    Kokkos::parallel_for(pack_buffer_func, src, send_buffer );
    MPIX_Enqueue_startall(queue, halo_ops.data() + num_n, num_n);
    MPIX_Enqueue_waitall(queue);
    Kokkos::parallel_for( unpack_recv_func, dst, rcev_buffer);
}
```

Idealized Stream-Triggering Interface

Diving Deeper into Cabana

- Designed Cabana Stream Halo for regular grids
  - Provides stream-triggered enqueueScatter and enqueueGather operations for halo exchanges
  - Added CommSpace::Mpich that uses existing (unoptimized) MPICH stream triggering primitives
  - Currently adding CommSpace::MpiAdvance backend to leverage new MPI Advance primitive
- Created CabanaGhost benchmark
  - Test performance of stream-triggered halo exchange primitives
  - Enable comparisons between different stream triggered backends
- Current work
  - Performing initial testing on UNM on Hopper system, targeting later testing on Tioga/Tuolumne
  - Developing support for stream-triggered collectives in MPI Advance and Cabana

Conclusions and Future Work

- **Conclusions**
  - This project successfully integrated MPI Advance stream-triggering with Cabana and CabanaGhost on Hopper
  - The results are promising but some issues remain
- **Future Work**
  - Stream-triggered CabanaGhost working with CXI on the Tioga system and CUDA on Hopper in the near-future
  - Creation of a Kokkos-level stream-triggered interface and further experiments with MPI stream-triggering
  - Port stream triggering into the KokkosComm library to enhance MPI+Kokkos integration and performance

Enabling Performant Inter-Node Communication for Kokkos Views

C. Nicole Avans, Carl Pearson[1], Jan Ciesko[1], Evan Drake Suggs, Stephen L. Olivier[1], Anthony Skjellum

[1]Sandia National Laboratories

Kokkos Comm

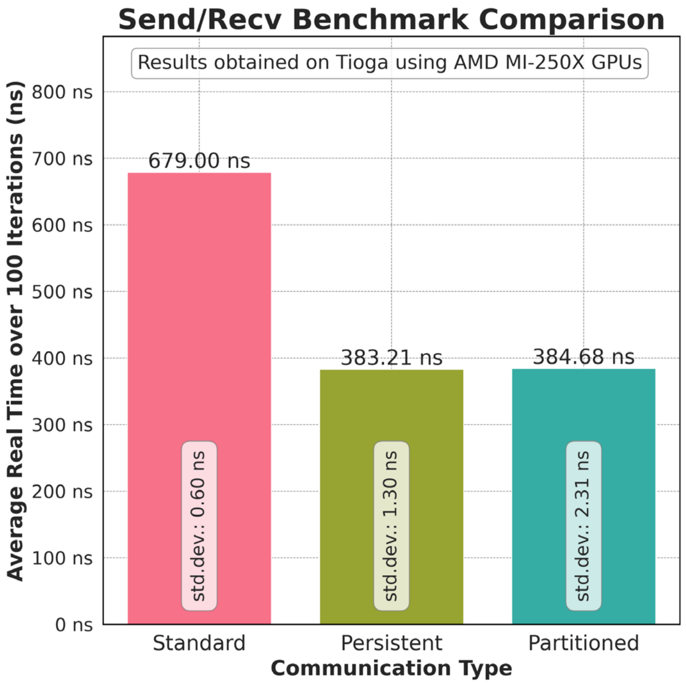
- The Kokkos Comm library introduces a new communication interface integrated with Kokkos Views
- A **unified performance-portable ecosystem** for on- and off-node parallel programming
- Kokkos Comm optimizes movement of data by abstracting implementation-specific details and marshaling and unmarshalling of data away from the end-user programmer

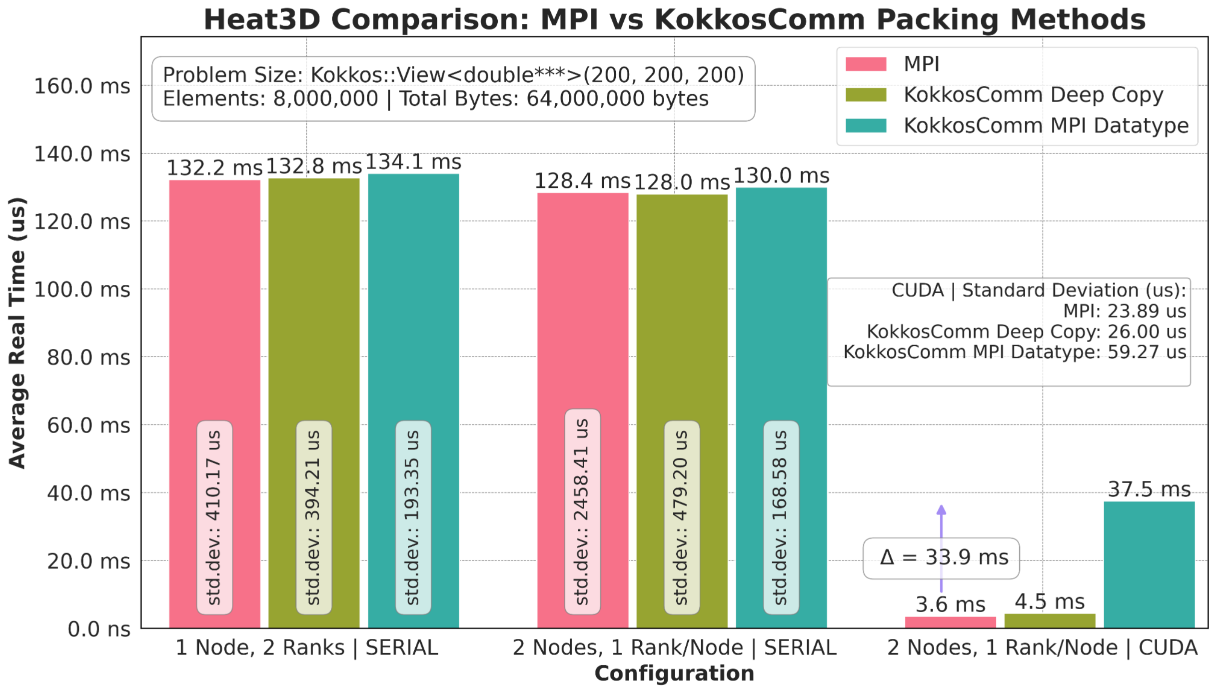
Design Communication for Performance, Portability, & Productivity

- Enable the fastest path for data to move without changing the program
- Native multi-transport communication support for performance and portability without reducing productivity
- GPU memory space communication is supported based on Kokkos enabled backends (e.g., CUDA, HIP)

Results

Experiments executed to optimize performance and identify bottlenecks on multiple nodes of SNL Weaver (NVIDIA Tesla V100) and LLNL Tioga (AMD MI250X)

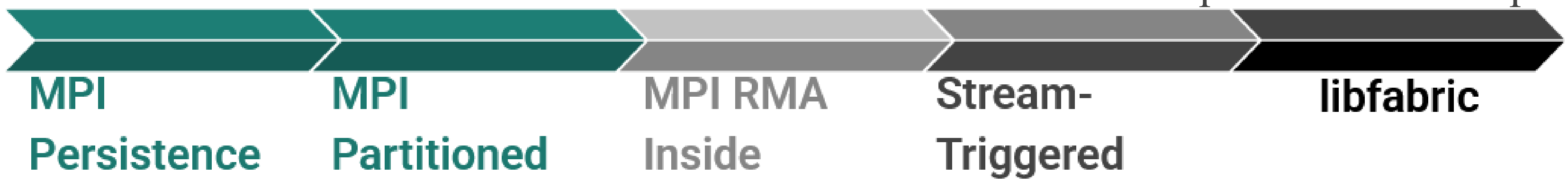




Future Work & Opportunities

- Enhance support to include arbitrary Kokkos View types and mdspan
- Add new communication space backends (e.g., NCCL) to extend Kokkos Comm beyond MPI-based communication for higher performance
- Add support for stream-triggered communication and libfabric to enable optimizations for system-specific performance

Planned Development Roadmap



Partitioned Communication in Iterative Sparse Matrix Operations

Gerald Collom

Amanda Bienz

SpMBV

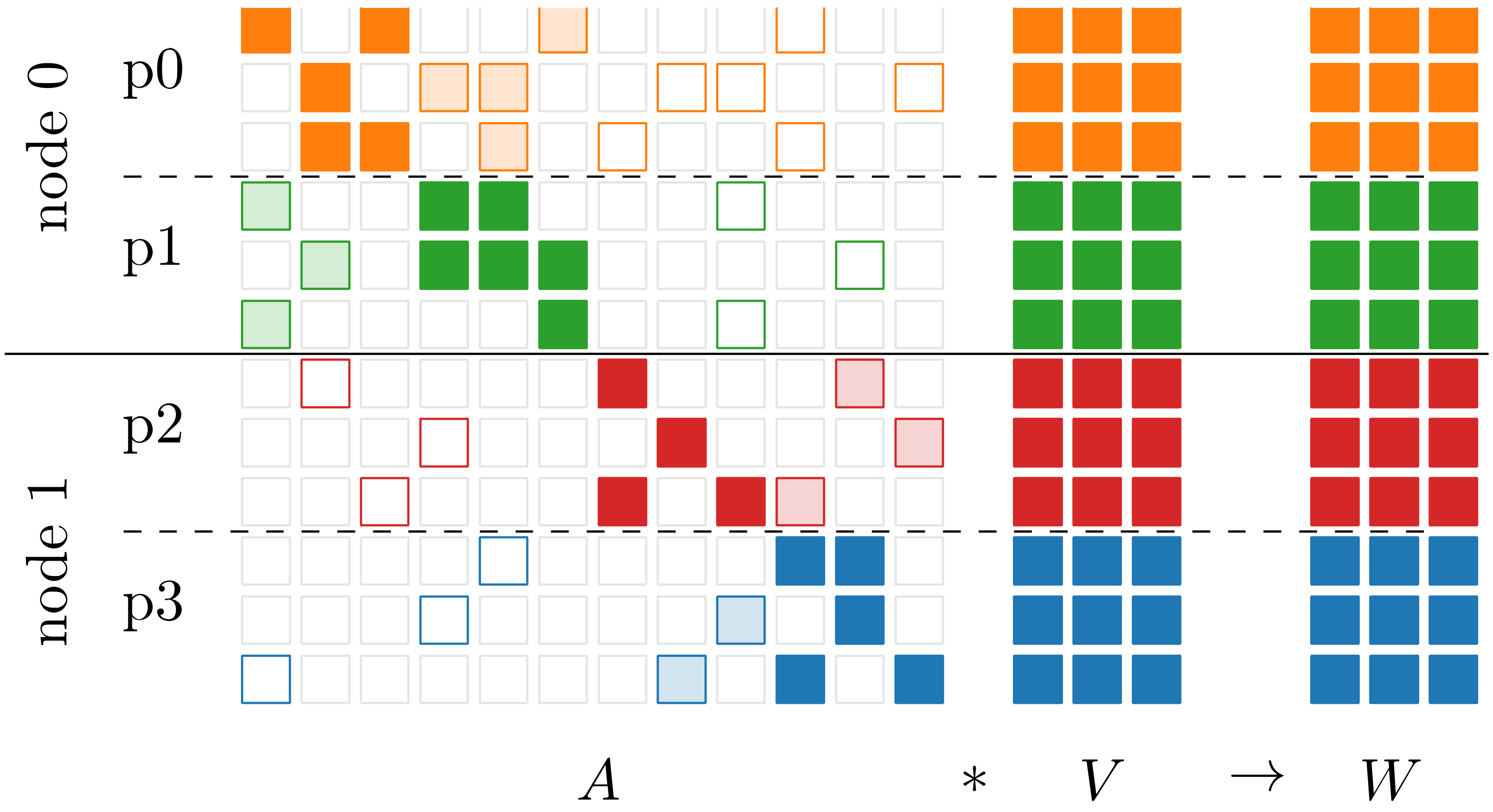


Image source:
Performance Analysis and Optimal Communication for ECG by Lockhart et al, 2023

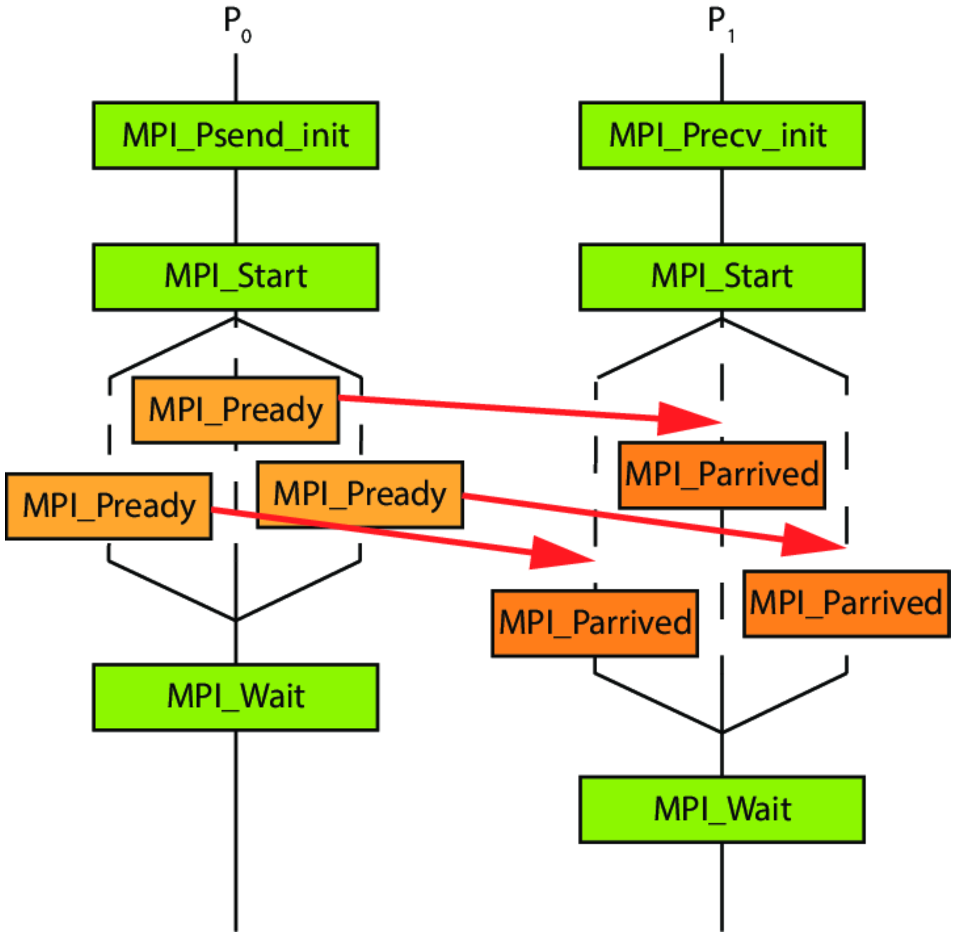
Motivations

- Deep learning
- Block Krylov Methods
- Graph Algorithms
- Quantum State Propagation
- Sparse PCA

SpMBV Benchmark

- Initialize communication
- Iterations:
  - Pack message buffer
  - Communicate
  - Multiply
- Cleanup communication requests

Partitioned MPI



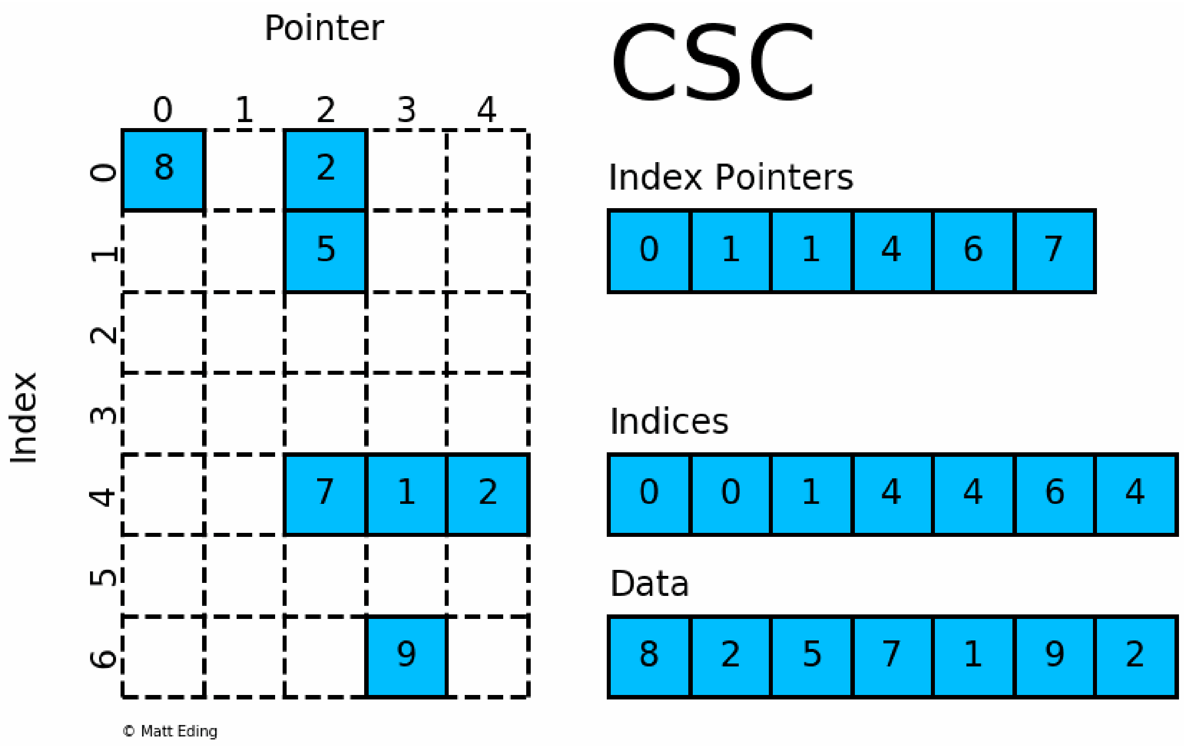
In Benchmark:

- Add threaded region for packing
- Threads asynchronously advance to call Pready

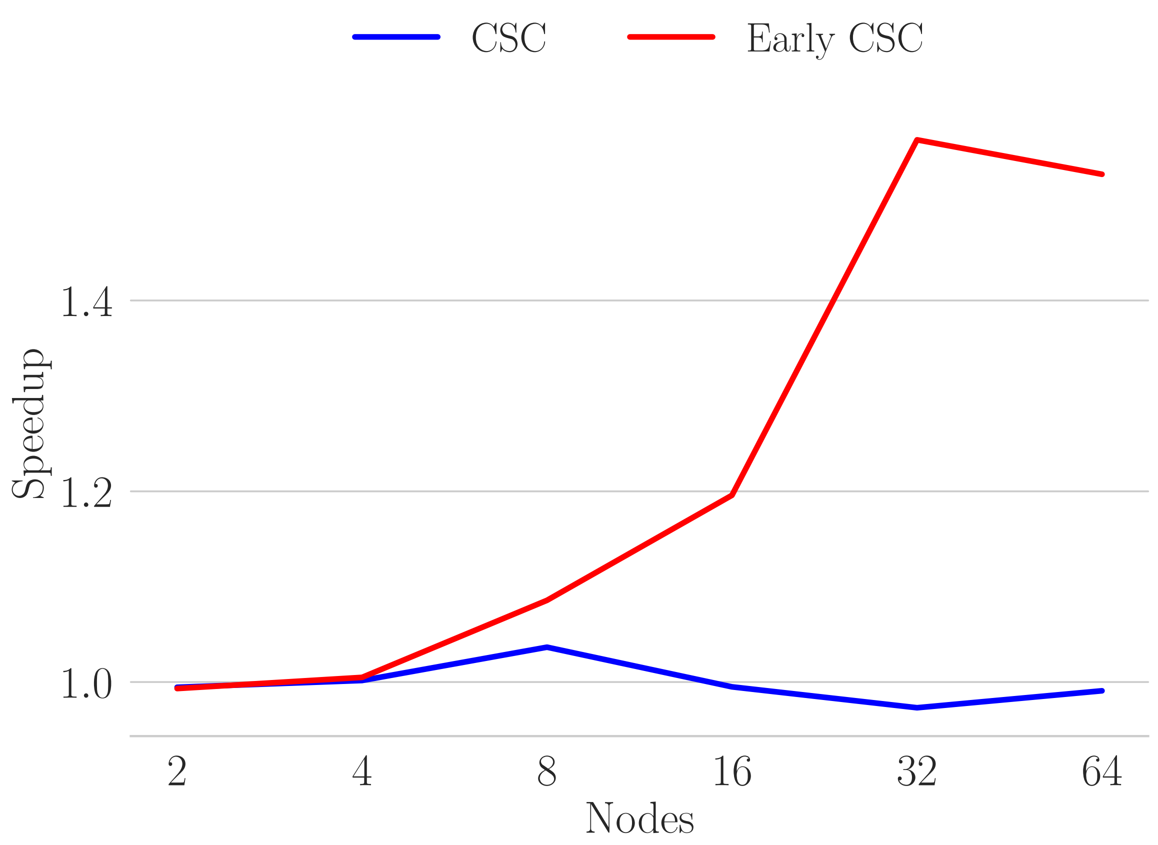
Early Communication and Computation

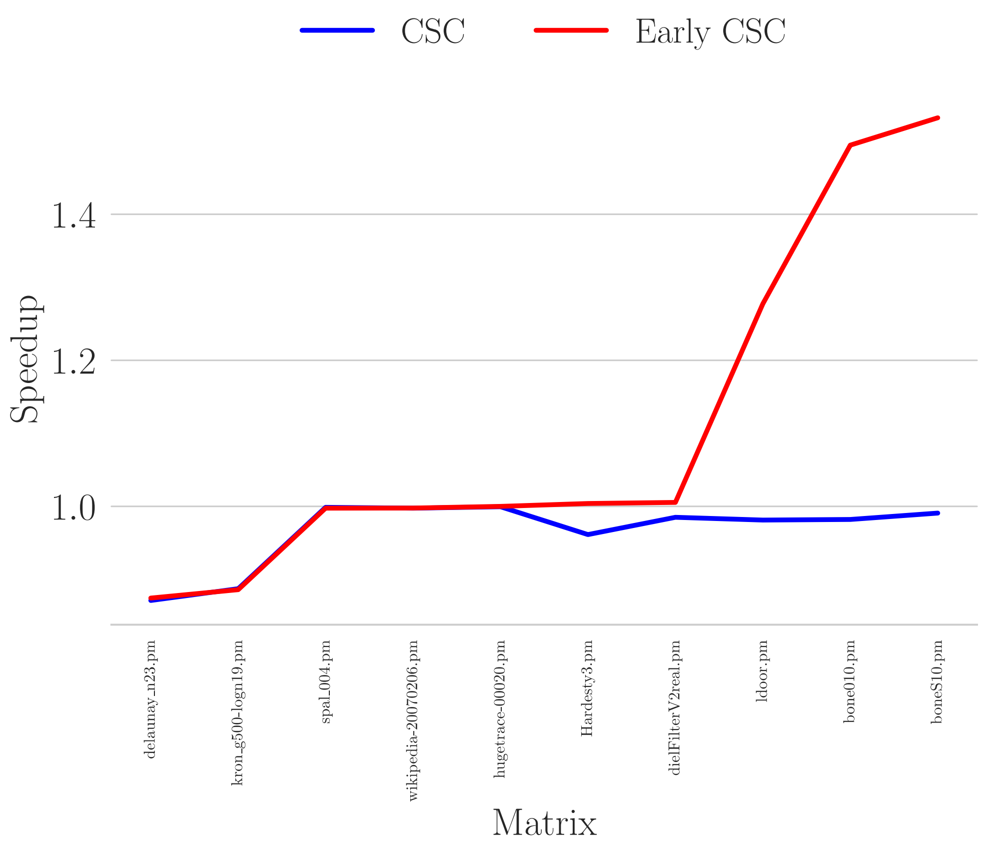
Benchmark Modifications:
- Extend threaded region to include computation

Allow threads to independently advance between computation and communication



Preliminary Results





Collective Communication Abstractions and Optimizations

- Evelyn Namugwanya, TN Tech: Optimizing Collective Communication Using MPI RMA & Generalized Algorithms
- Mike Adams, UNM: Optimizing GPU-Aware Allreduce Operations
- Shannon Kinkead, UNM: Scaling All-to-all Operations Across Emerging Many-Core Supercomputer

RMA-Based Alltoallv: Performance Analysis

Evelyn Namugwanya, Amanda Bienz,
Matthew Dosanjh[1], Anthony Skjellum
[1]Sandia National Laboratories

CUP ECS Center for Understandable, Performant Exascale Communication Systems

Tennessee TECH

Introduction

- Alltoallv is critical in high-performance computing (HPC).
- MPI_Alltoallv is widely used for variable-size data exchange.
- RMA (Remote Memory Access) enables one-sided communication.
- Goal: Evaluate performance of RMA-based Alltoallv variants.

Methodology

- Implemented several versions of alltoallv_rma.
- Evaluated on LLNL Lassen and Dane clusters.
- APIs: MPI_Win_fence, MPI_Win_lock, MPI_Win_flush
- Caliper profiling
- Focused on reducing sync overhead and improving scalability

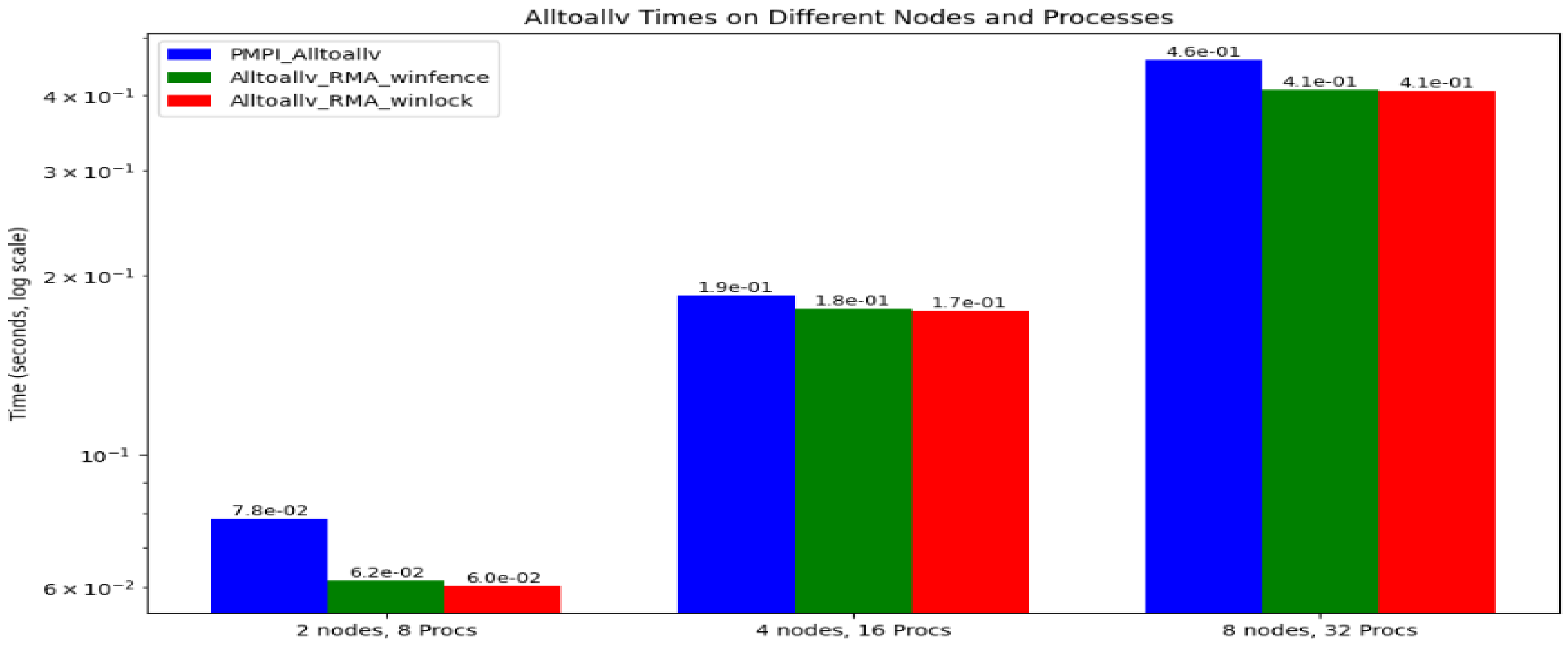
Fence-based Alltoallv RMA

- Uses MPI_Win_fence to begin/end RMA epoch.
- MPI_Put used to transfer data into target memory.
- Synchronizes with two MPI_Win_fence calls and MPI_Barrier.
- Simpler but incurs global synchronization overhead.

Lock-based Alltoallv RMA

- Employs MPI_Win_lock_all and MPI_Rput.
- Asynchronous data transfer with MPI_Rput.
- Uses MPI_Win_flush_all to ensure completion.
- Finer-grain control compared to fence.
- Ends with MPI_Win_unlock_all and MPI_Barrier.

Alltoallv Comparison



Lower is better

Message size = 33,554,432 bytes

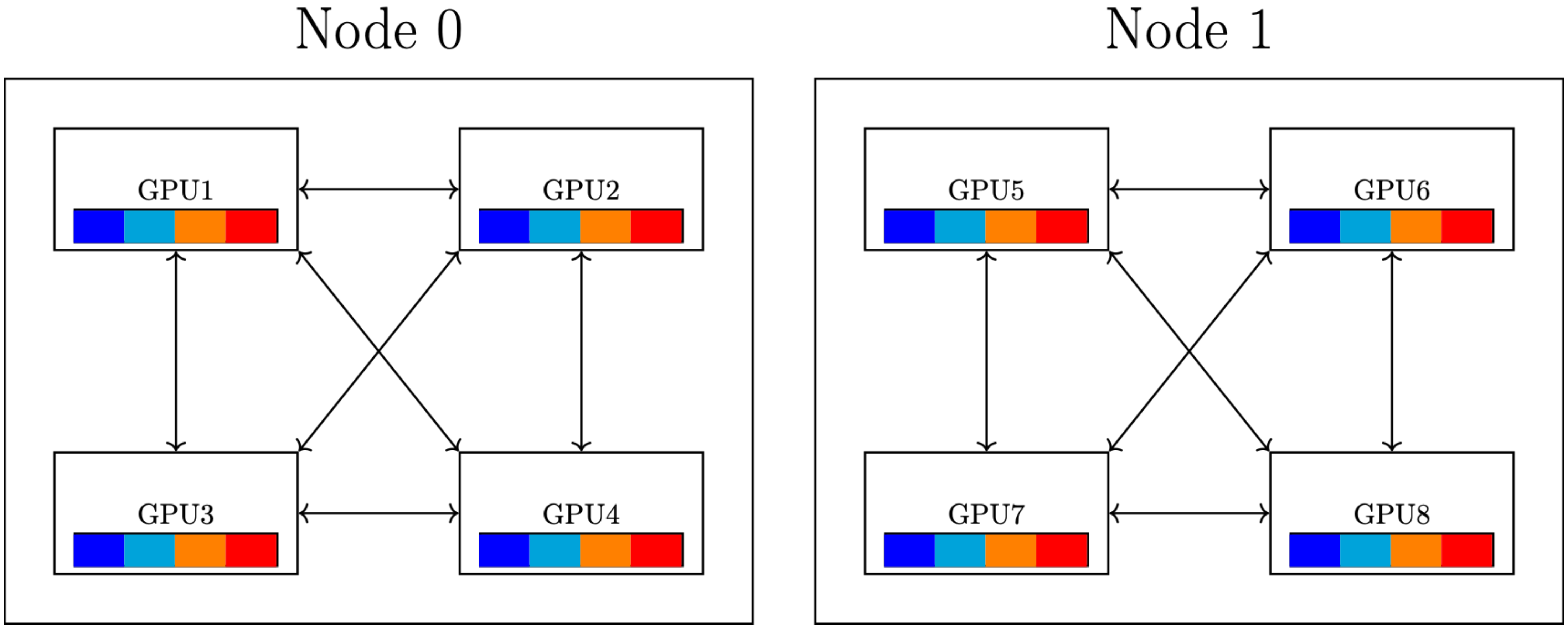
Results and Future work

- Experiments with our Alltoallv Benchmark on LLNL Lassen and Dane supercomputers.
- The default MPI_Alltoallv offers strong performance on small process counts.
- RMA variants, especially with fine-grain locking, scale better with more nodes and process count.
- This can be attributed to its flexibility as compared to MPI_Win_fence.
- Synchronization overhead is a key bottleneck.
- Future work includes using OpenSHMEM and comparing persistent RMA vs. OpenSHMEM Alltoallv.

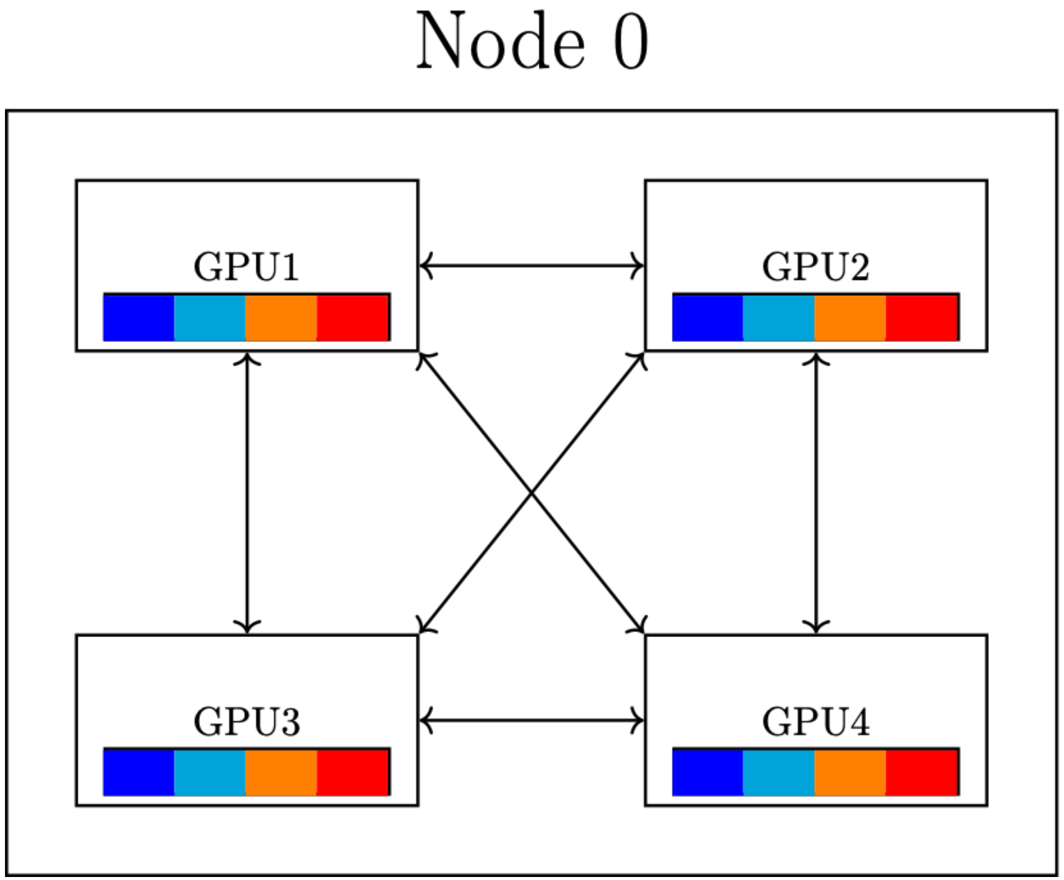
Optimizing GPU-Aware Allreduce Operations

Mike Adams, **Amanda Bienz**

GPU-Aware Allreduce



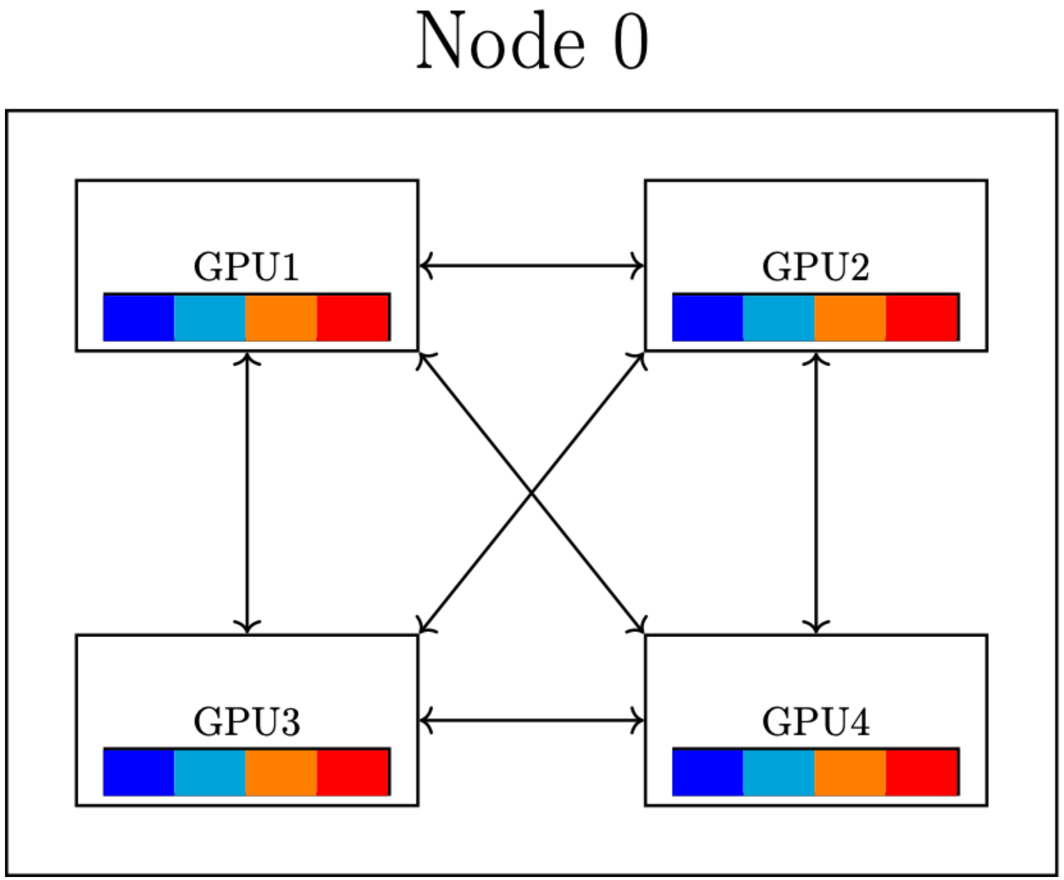
Multi-Lane Allreduce



1. MPI_Reduce_scatter on node
2. MPI_Allreduce with only corresponding GPUs on each other node
3. MPI_Allgather on node

J. L. Träff and S. Hunold, "Decomposing MPI Collectives for Exploiting Multi-lane Communication," *2020 IEEE International Conference on Cluster Computing (CLUSTER)*

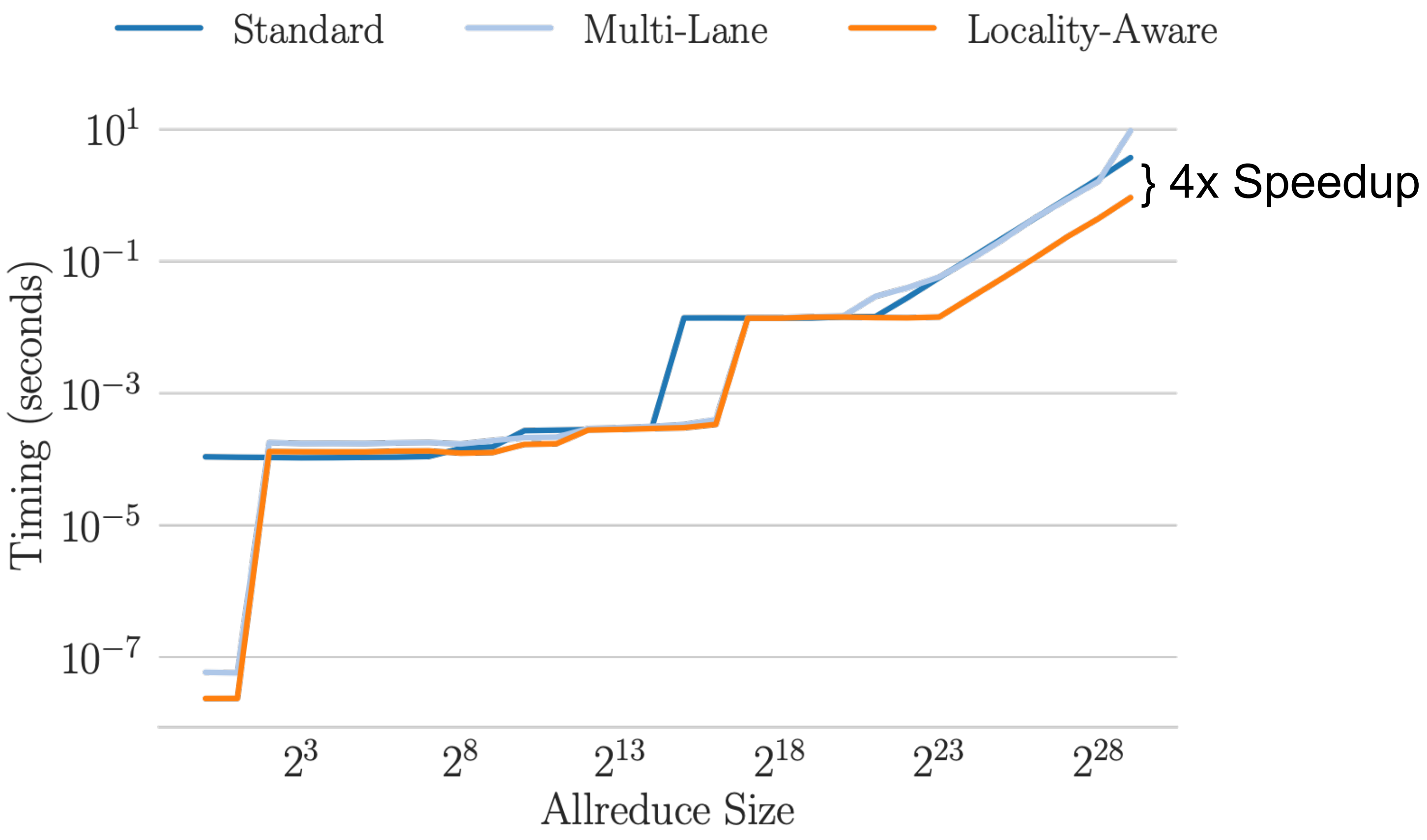
Locality-Aware Allreduce



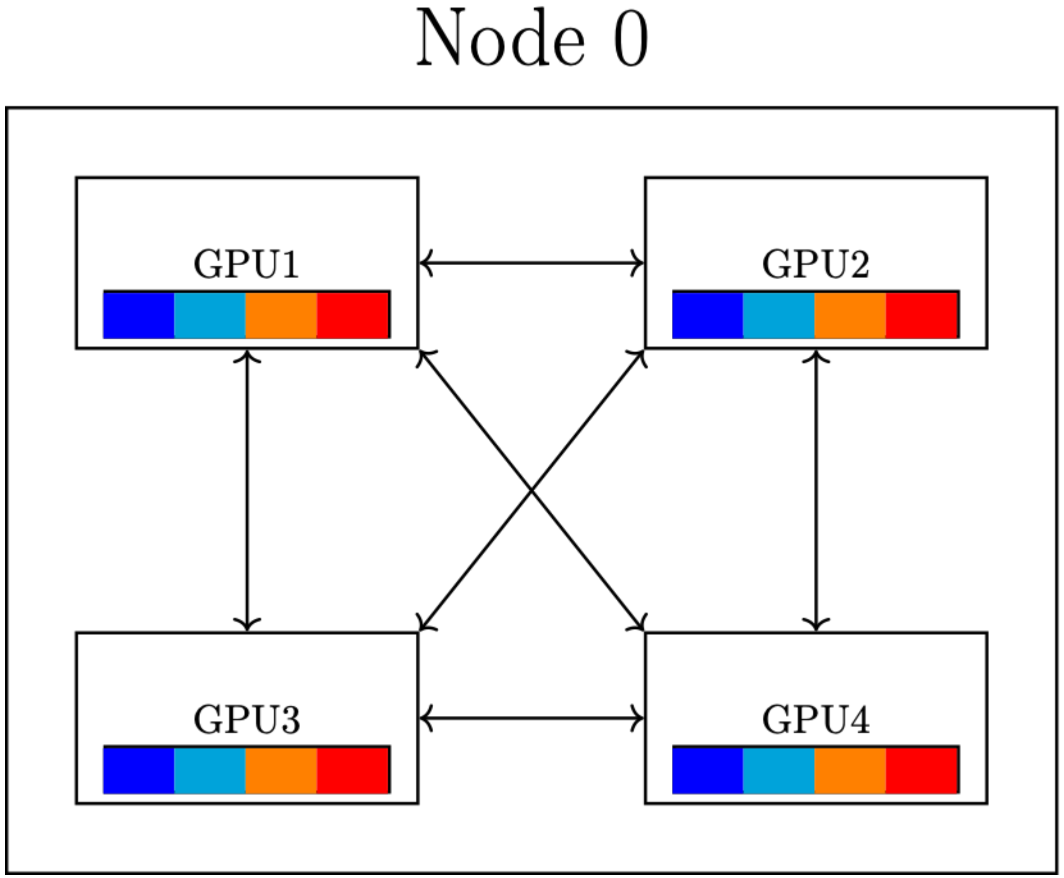
1. MPI_Allreduce on node
2. MPI_Allreduce with only corresponding GPUs on each other node

Locality-Aware Allreduce

- DeltaAI (Grace Hopper)
- 4 GPUs per node
- 16 Nodes



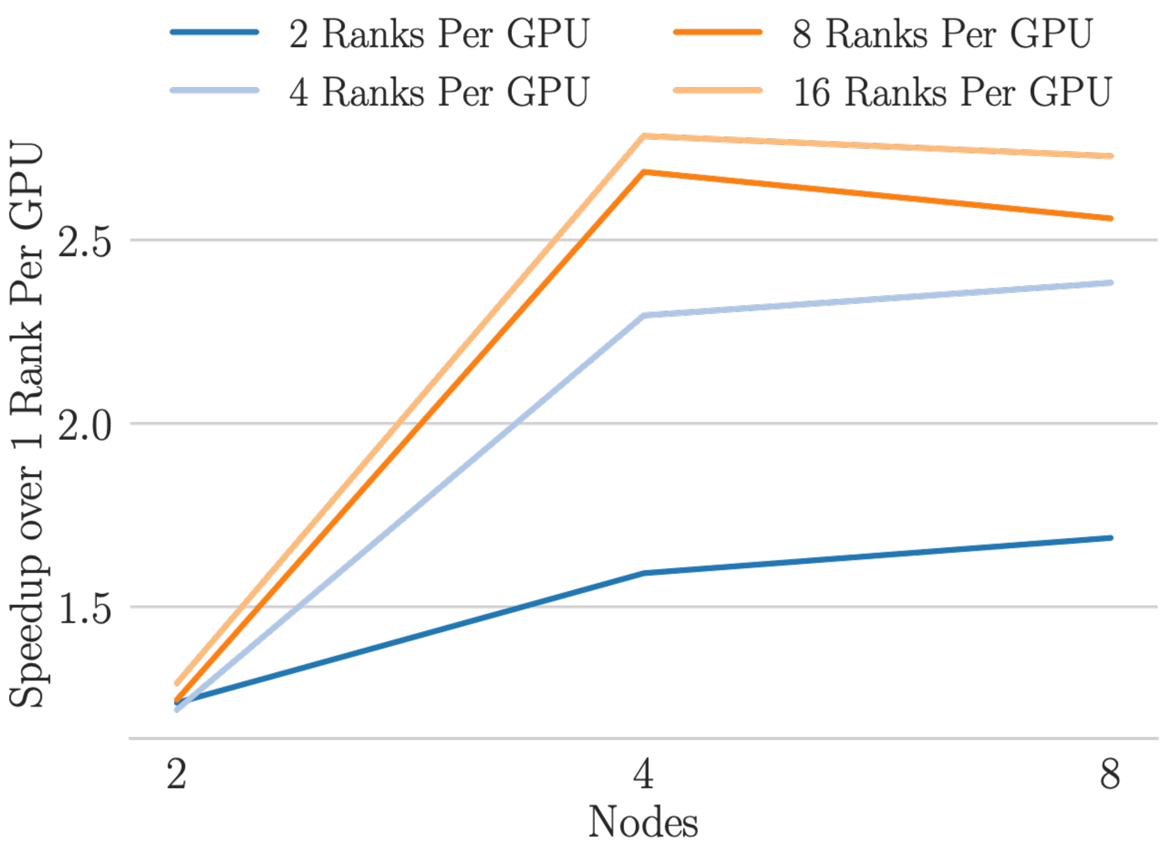
Multiple MPI Ranks per GPU



- Each GPU has 1 leader MPI rank and many other non-leader ranks
  - Leader creates buffer
  - Shares IPC handle with other ranks
  - Each rank reduces their local portion of buffer

Initial Results

- Delta Supercomputer
- 4 GPUs per node
- Each timing: best time for multiple underlying algorithms



Scaling All-to-all Operations Across Emerging Many-Core Supercomputers

Shannon Kinkead[1], **Amanda Bienz**

[1]Sandia National Laboratories

All-to-all Operations

Algorithm 1: Pairwise Exchange

Input: p {process rank}
n {process count}
s_{size}, s_{type}, s_{buf} {send size, type, and buffer}
r_{size}, r_{type}, r_{buf} {recv size, type, and buffer}

for $i \leftarrow 0$ **to** n **do**
  $s_{\text{proc}} = p + i \mod n$
  $r_{\text{proc}} = p + n - i \mod n$
  `MPI_Sendrecv`($s_{\text{buf}}, s_{\text{size}}, s_{\text{type}}, s_{\text{proc}}, \ldots$) $r_{\text{buf}}, r_{\text{size}}, r_{\text{type}}, r_{\text{proc}}, \ldots$)

Algorithm 2: Non-blocking

Input: p {process rank}
n {process count}
s_{size}, s_{type}, s_{buf} {send size, type, and buffer}
r_{size}, r_{type}, r_{buf} {recv size, type, and buffer}

for $i \leftarrow 0$ **to** n **do**
  $s_{\text{proc}} = p + i \mod n$
  $r_{\text{proc}} = p + n - i \mod n$
  `MPI_Isend`($s_{\text{buf}}, s_{\text{size}}, s_{\text{type}}, s_{\text{proc}}, \ldots$)
  `MPI_Irecv`($r_{\text{buf}}, r_{\text{size}}, r_{\text{type}}, r_{\text{proc}}, \ldots$)

`MPI_Waitall`($2 \times (n-1), \ldots$)

- Each process exchanges an equal sized amount of data with every other process
- **Pairwise:** scheduled exchange, one send/recv at a time
- **Nonblocking:** initialize all communication, wait
- *Emerging systems: many cores per node, intra- and inter-node communication performance differs greatly*

Hierarchical All-to-all Operations

- **Hierarchical:** one process per node performs all inter-node communication
- **Multi-leader:** a number of leaders per node each perform a subset of inter-node communication

Algorithm 3: Hierarchical

Input: p {process rank}

ount} n {process c

uffer} $s_{\texttt{size}}$, $s_{\texttt{type}}$, $s_{\texttt{buf}}$ {send size, type, and bu

uffer} $r_{\texttt{size}}$, $r_{\texttt{type}}$, $r_{\texttt{buf}}$ {recv size, type, and bu

gion} `local_comm` {All processes local to re

of local_comm} ppn, l {Size and rank

qual local rank} `group_comm` {All processes with e

$s_{\texttt{buf}_{leader}} \leftarrow$ `buffer of size` $s_{\texttt{size}} \cdot n \cdot ppn$

$r_{\texttt{buf}_{leader}} \leftarrow$ `buffer of size` $r_{\texttt{size}} \cdot n \cdot ppn$

// Gather to leader

omm) MPI_Gather($s_{\texttt{buf}}$, $s_{\texttt{size}} \cdot n$, …, $s_{\texttt{buf}_{leader}}$, …, local_c

`Repack Data`

// Alltoall exchange between leaders

ader, $r_{\texttt{size}} \cdot ppn^2$…, group_comm) MPI_Alltoall($s_{\texttt{buf}_{leader}}$, $s_{\texttt{size}} \cdot ppn^2$, …, $r_{\texttt{buf}_{le}}$

`Repack Data`

// Scatter from leader

$r_{\texttt{buf}}$, …, local_comm) MPI_Scatter($r_{\texttt{buf}_{leader}}$, $r_{\texttt{size}} \cdot n$, …

Locality-Aware All-to-all Operations

- **Node-Aware:** all-to-all between all processes with equal local_rank, then all-to-all on node
- **Locality-Aware:** Same as node-aware, but multiple groups per node

Algorithm 4: Locality-Aware

Input: p {process rank}

ount} n {process c

uffer} $s_{\texttt{size}}$, $s_{\texttt{type}}$, $s_{\texttt{buf}}$ {send size, type, and b

uffer} $r_{\texttt{size}}$, $r_{\texttt{type}}$, $r_{\texttt{buf}}$ {recv size, type, and b

gion} `local_comm` {All processes local to re

of local_comm} ppn, l {Size and rank

qual local rank} `group_comm` {All processes with e

$tmp_{\texttt{buf}} \leftarrow$ `buffer of size` $s_{\texttt{size}}$

// Inter-region Alltoall

MPI_Alltoall($s_{\texttt{buf}}$, $s_{\texttt{size}} \cdot ppn$, …, $tmp_{\texttt{buf}}$, $r_{\texttt{size}} \cdot ppn$… , `group_comm`)

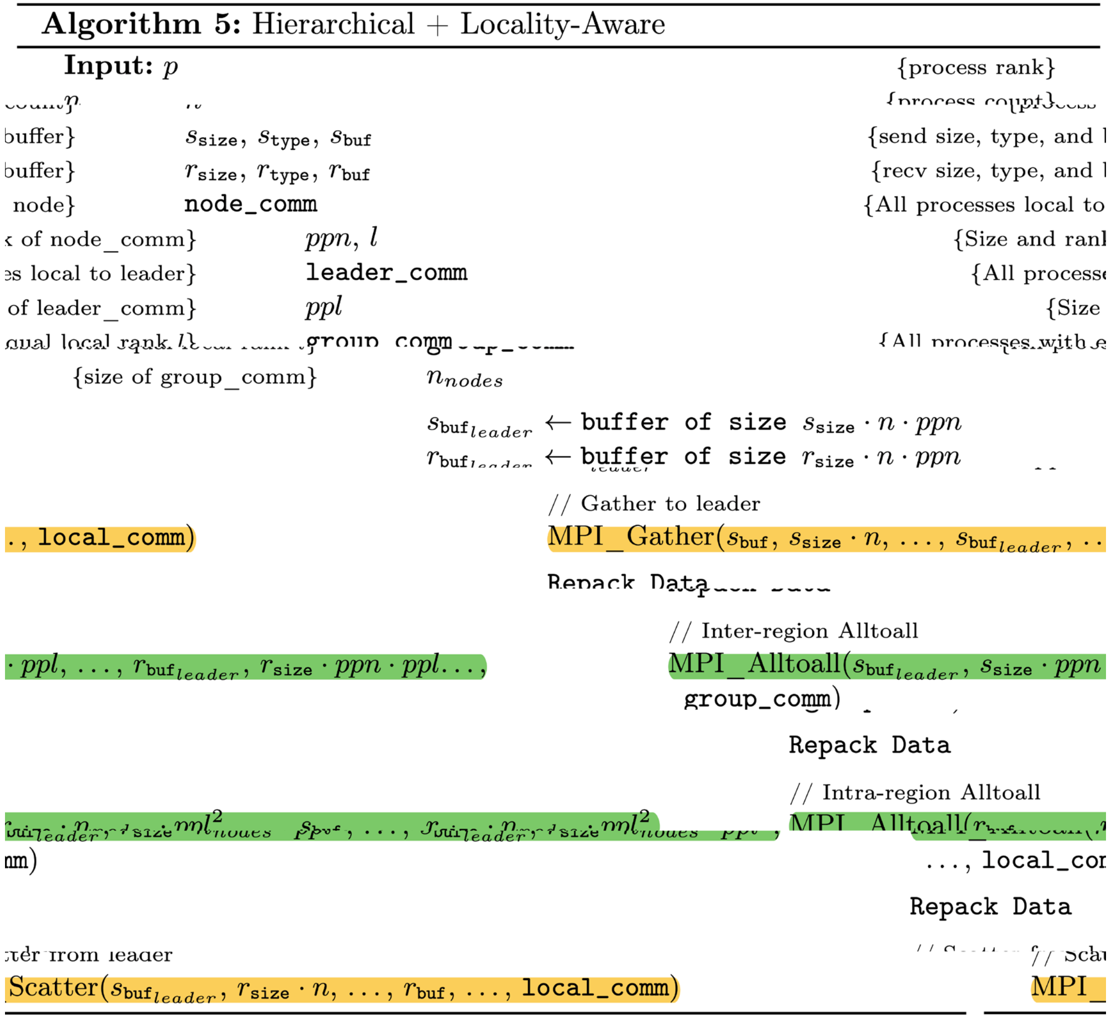
`Repack Data`

// Intra-region Alltoall

MPI_Alltoall($tmp_{\texttt{buf}}$, $r_{\texttt{size}} \cdot ppn$, …, $r_{\texttt{buf}}$, $r_{\texttt{si}}$ ze $\cdot$ ppn…, `local_comm`)

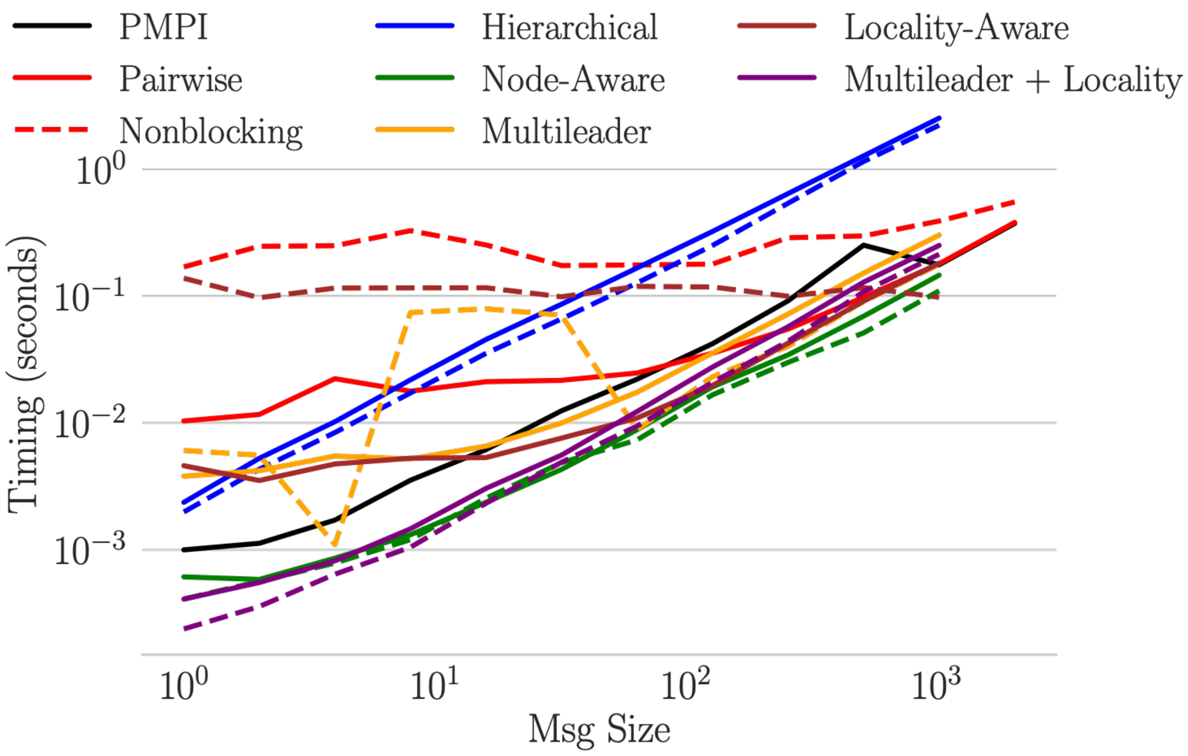
Multileader Locality

- Multiple leaders per node, each performing a node-aware all-to-all



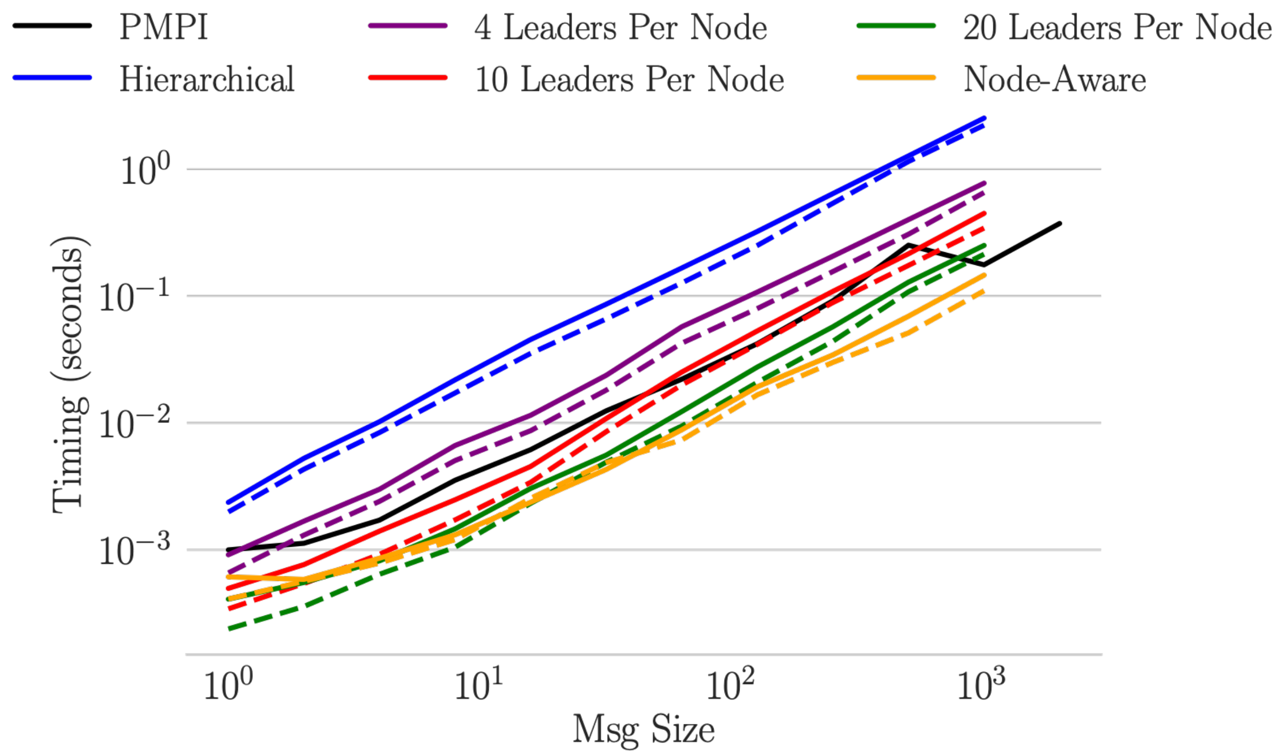
Scaling Results (Dane, 32 Nodes)

Solid: Pairwise
Dotted: Nonblocking

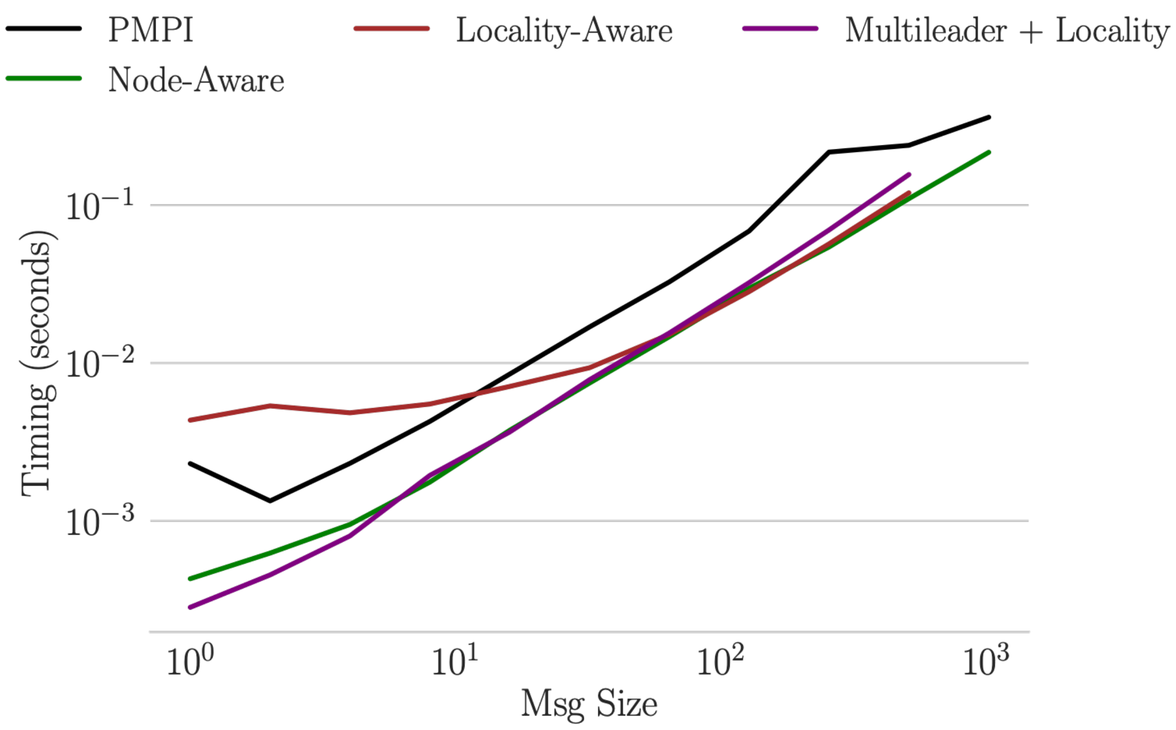


Multileader + Locality

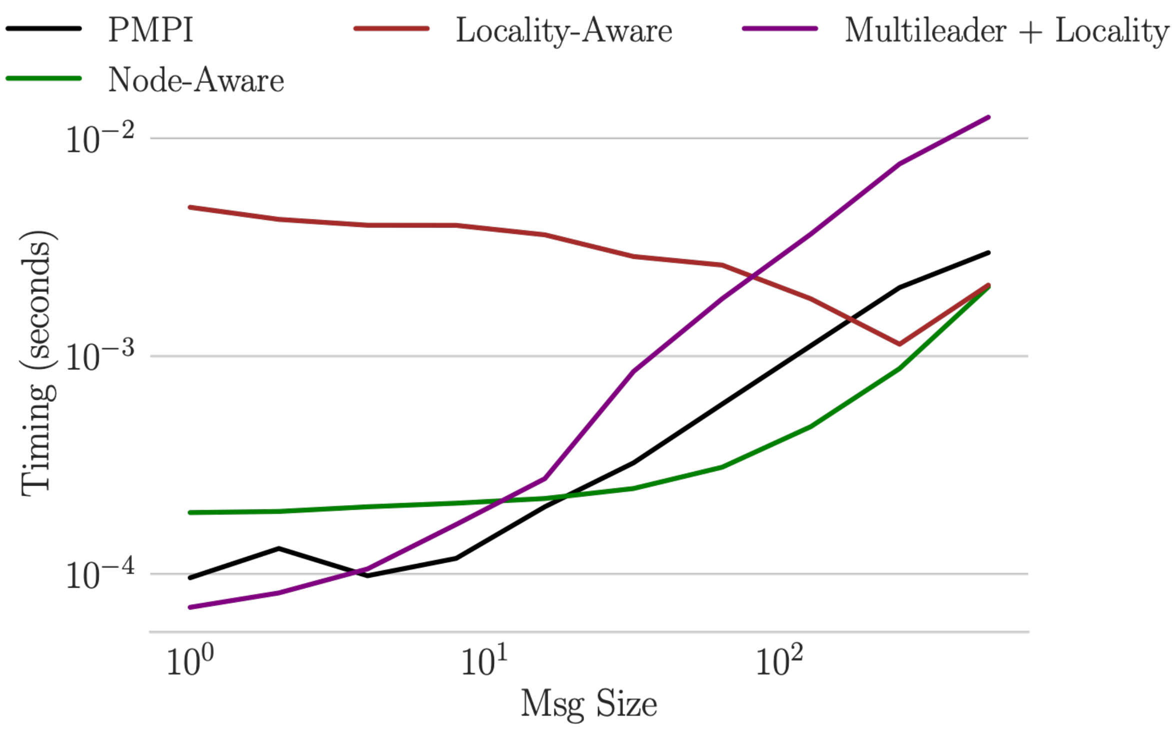
Solid: Pairwise
Dotted: Nonblocking



More Results



Amber



Tachi

Poster-based Discussion Time

Lunch provided at Noon