Spring 2024 TST Meeting

Prof. Patrick G. Bridges

Recommendations from 2023 Annual Review

Recommendations (1)

Summary of software use cases currently being explored

- Current Mini/Proxy Applications:
  - **Comb**: LLNL communication performance benchmarking tool
  - **MiniAero**: Manetevo mini-application that does Navier-Stokes equations for explicit unstructured finite volumes
  - **Hermes:** New global spatial sorting benchmark
  - **MiniGhost/CabanaGhost:** Existing and in-development regular halo exchange benchmark
  - **CLAMR**: LANL cell-based adaptive mesh refinement (AMR) mini-app, using the **L7** communication framework
  - **Beatnik:** New fluid interface benchmark to exercise FFT and mesh/particle remap communications
  - **CabanaMD/CabanaMPM:** Cabana particle/mesh library molecular dynamics and material point benchmarks
- Current Applications/Libraries
  - **xRAGE**: LANL Eulerian radiation/hydrodynamics code
    - **Cellar** = token library = communication framework in xRAGE (objects are called “tokens”)
  - ***hypre***: LLNL library of preconditioners and solvers featuring multigrid methods for the solution of large, sparse linear systems of equations
    - **AMG2023** – ATS5 driver benchmark
    - **SuiteSparse** – collection of wide range of sparse matricies
  - **Parthenon**: LANL performance portable block-structured adaptive mesh refinement framework
  - **HOSS:** LANL hybrid multi-physics software package using a range of element-based methods
- Upcoming Mini/Full Applications: **SPARC** (Siva R. to provide Ifpack2 SPARC matrix), **EMPIRE** via MiniEM, **SUNDIALS** (for Raja), **UME**

Recommendations (2)

Define MPI0, ExaMPI, MPI Advance, Beyond MPI, and any new MPI abstraction terms

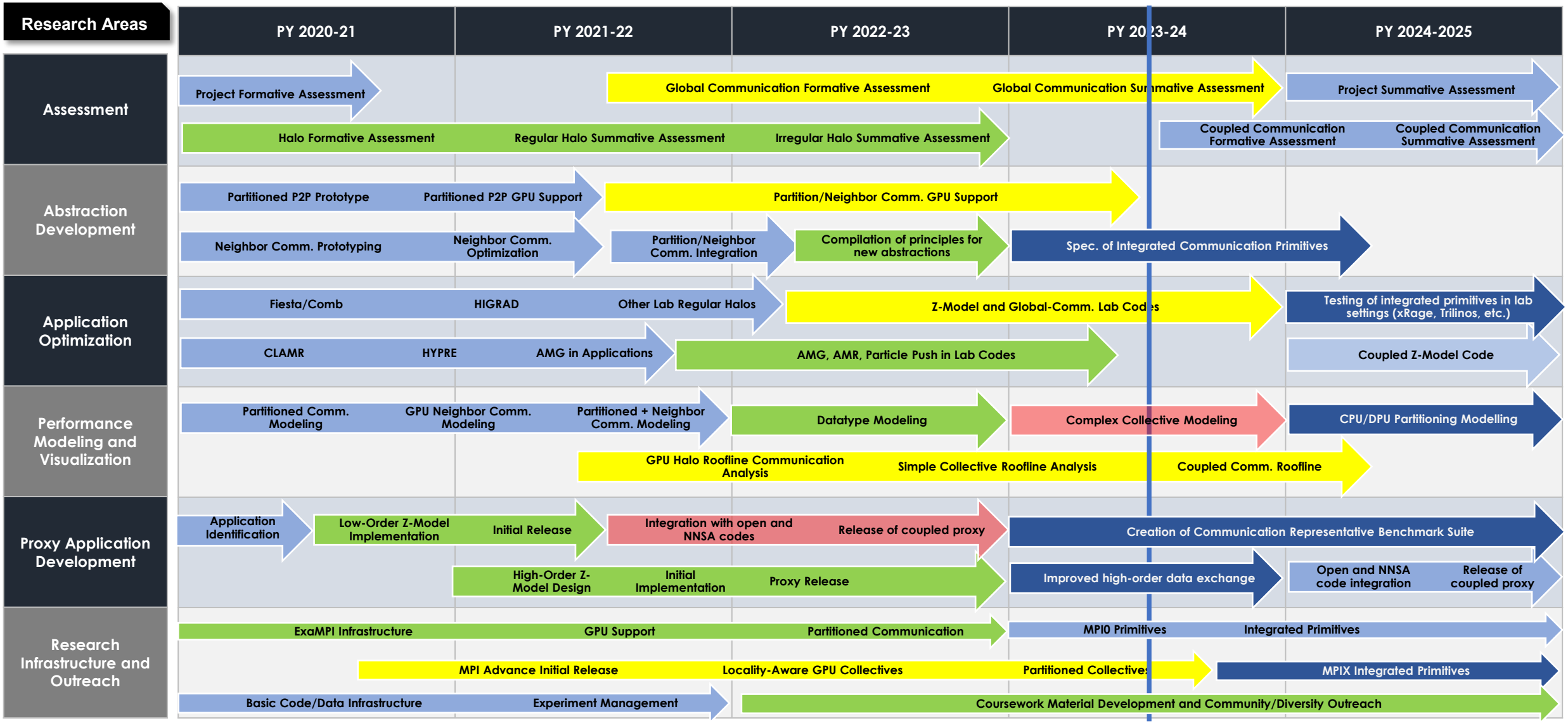
- **ExaMPI**: Our C++ research MPI implementation
- **Beyond MPI:** Application-oriented abstractions that leverage our findings, unconstrained by legacy MPI
- **MPI Advance:** Application-oriented abstractions that extend and push the frontier of existing abstractions
- **MPI0:** New low-level communication primitives for use by library writers for building new abstractions

Recommendations (3)

- *Please modify the software stack slide to more accurately convey your evolution in thinking about MPI abstraction layers, and what is being accomplished with each instantiation of each MPI abstraction – see later slides*
- *Also provide a table of proxy apps, e.g. Beatnik, being used to test various abstractions – in progress*
  - *Regular halos: Comb, MiniAero, CabanaMPM, MiniGHOST (making new Cabana version)*
  - *Irregular halos: CLAMR, AMG2023, HYPRE SuiteSparse, MiniEM, Trilinos Ifpack 2 solve, New Irregular exchange benchmark*
  - *Global exchanges: Beatnik (FFT, particle/mesh redistribution), Hermes (new global spatial sorting benchmark)*
- *Please think about developing MPI communication pattern extraction tools – in progress*
- *Look at other programming models and solver packages to generalize MPI abstractions - Raja/SUNDIALS upcoming*
- *When presenting data and results, please provide more context on the HPC configurations and experimental conditions – in progress*
- *Numerous data is being collected in a variety of configuration and conditions, yet it is difficult to draw conclusions at this point. We encourage you to take a data analytics and experimental design approach to determine the factors that most influence the HPC communications – examining use of Benchpark and Hatchet as tools to help enable this*

Current changes/challenges

Updated 5-year Project Roadmap



Changes/Challenges/Status

- Personnel
  - Thomas Hines left project
  - Multiple lab internships scheduled (Evelyn, Nicole, Grace, Nick, Gerald)
  - Ongoing challenge: recruiting students to join project
  - Rebalancing/rescheduling work due to personnel changes
- Vendor interaction
  - NDAs executed with AMD, HPE
  - Speed of interaction/meaningful response from some vendors is lacking
- Many papers being written/submitted
  - Paper accepted for CCGrid on irregular benchmarking
  - Multiple papers on irregular communication API and optimizations submitted to SC
  - ICPP and Cluster papers submissions planned

Overall Plan Discussion: Overview

Before break (30 minutes):

- Benchmarking and Performance Studies
- Communication Characterization
- Education and Outreach

After break: API/Abstraction Development and Evaluation

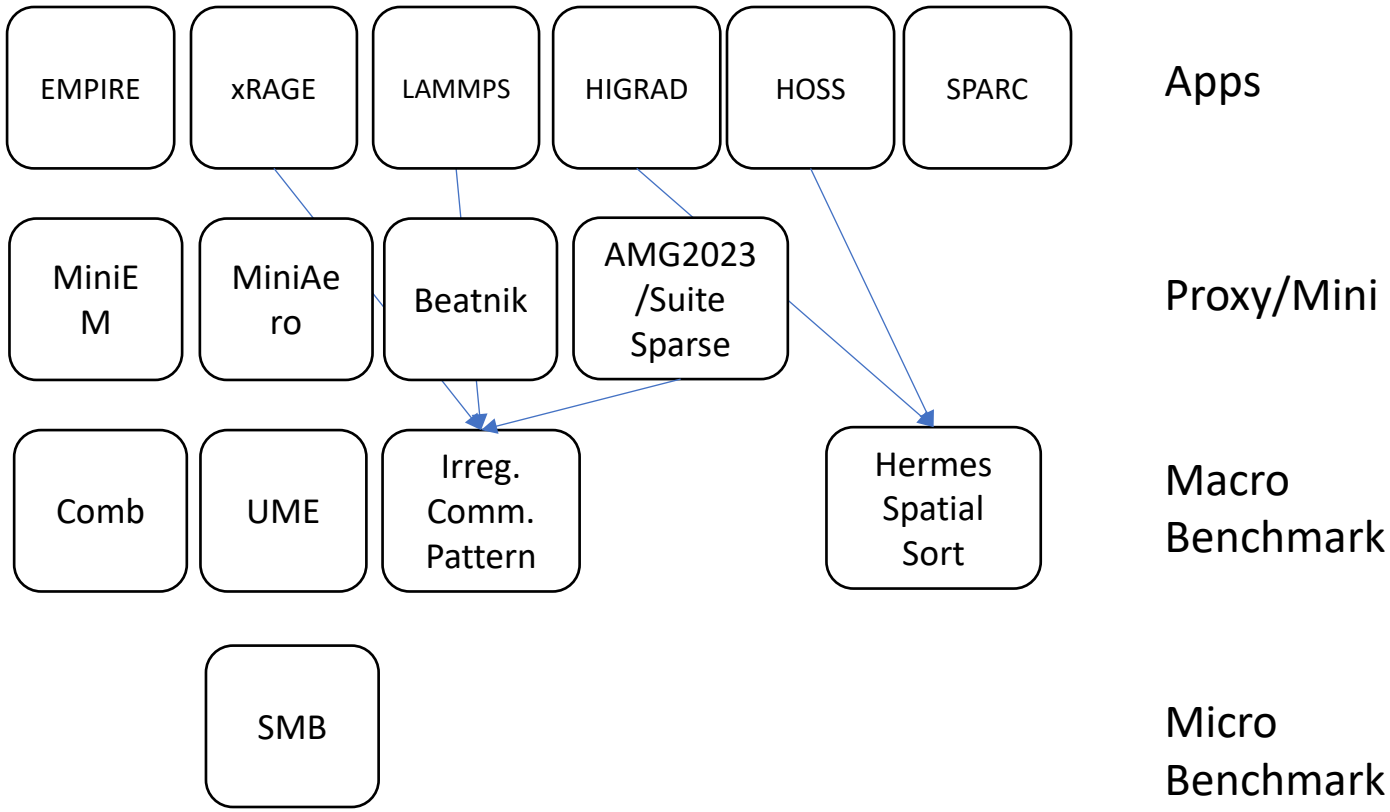
Overall Plan: Benchmarking and Performance Studies

Communication performance studies are painful

- Hard to use real applications and input decks for communication development
  - Require significant expertise to build, configure, run, and scale
  - Wide range of communication techniques and behaviors make it hard to isolate, understand, and optimize specific patterns
  - Communication tightly enmeshed with complex compute makes it difficult to try out novel communication abstractions
- Point efforts in this area
  - Communication-representative mini-applications work in this area: LAMMPS/miniMD, HACC/SWFFT (Aaziz et al., 2018), CTH/miniAMR (Aaziz et al., 2019), others
  - New miniapps with interesting input decks if you know where to look (ATS 5 acquisition benchmarks): MiniEM, AMG2023, Block AMR and Neutron Transport benchmarks, etc.
- Approach: Extract information from application runs to drive creation communication proxies and benchmarks across the **full space of HPC communication**

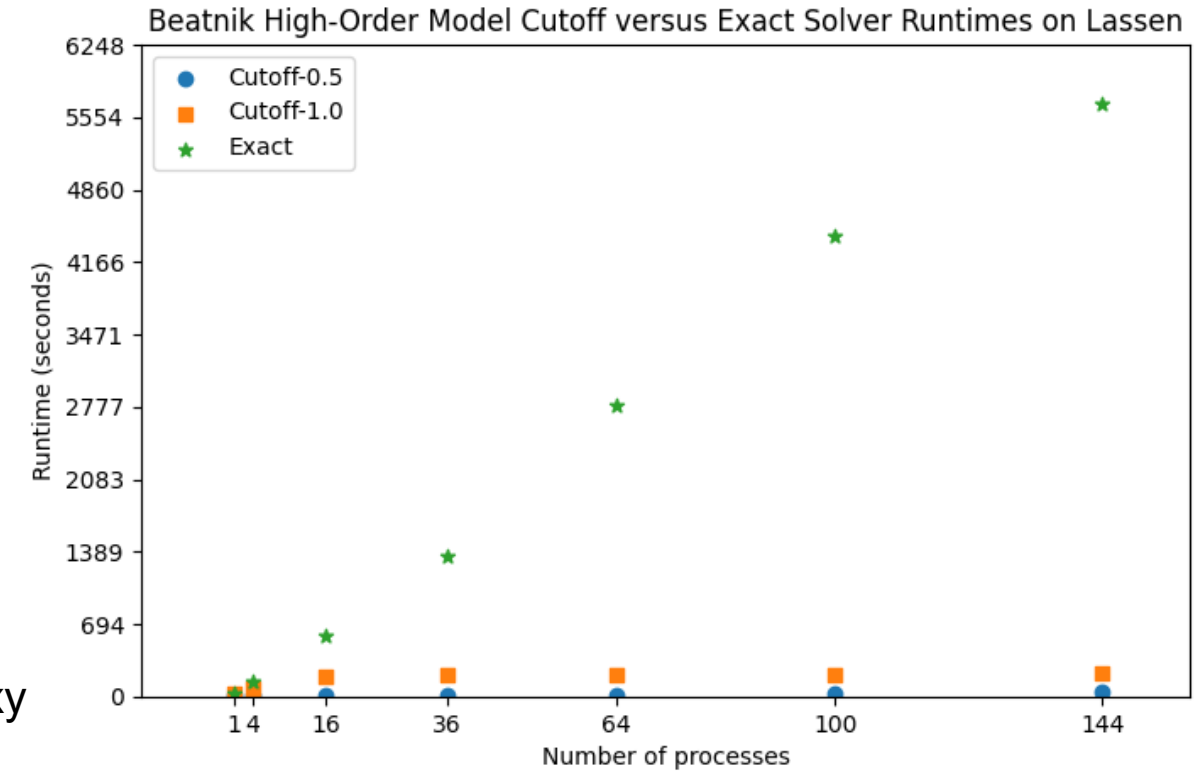
What's the full space, what are the gaps?

- Need full spectrum of communication types
  - Workload information from app through proxies to macro/micro benchmarks
  - Regular/irregular, tree communication, global communication, remapping
  - Static and dynamic variants of all of these
- Example gaps
  - Breadth of irregular exchanges in real applications (Derek later)
  - Sorting in real codes, e.g., spatial sorting of partially-sorted finite elements or particles
  - Remapping communications (Beatnik)
  - Tree communication
- Need help from the community
  - Flesh out the full of patterns to care about
  - Identify applications to use to find this data
  - Start fleshing out lines between apps/benchmarks



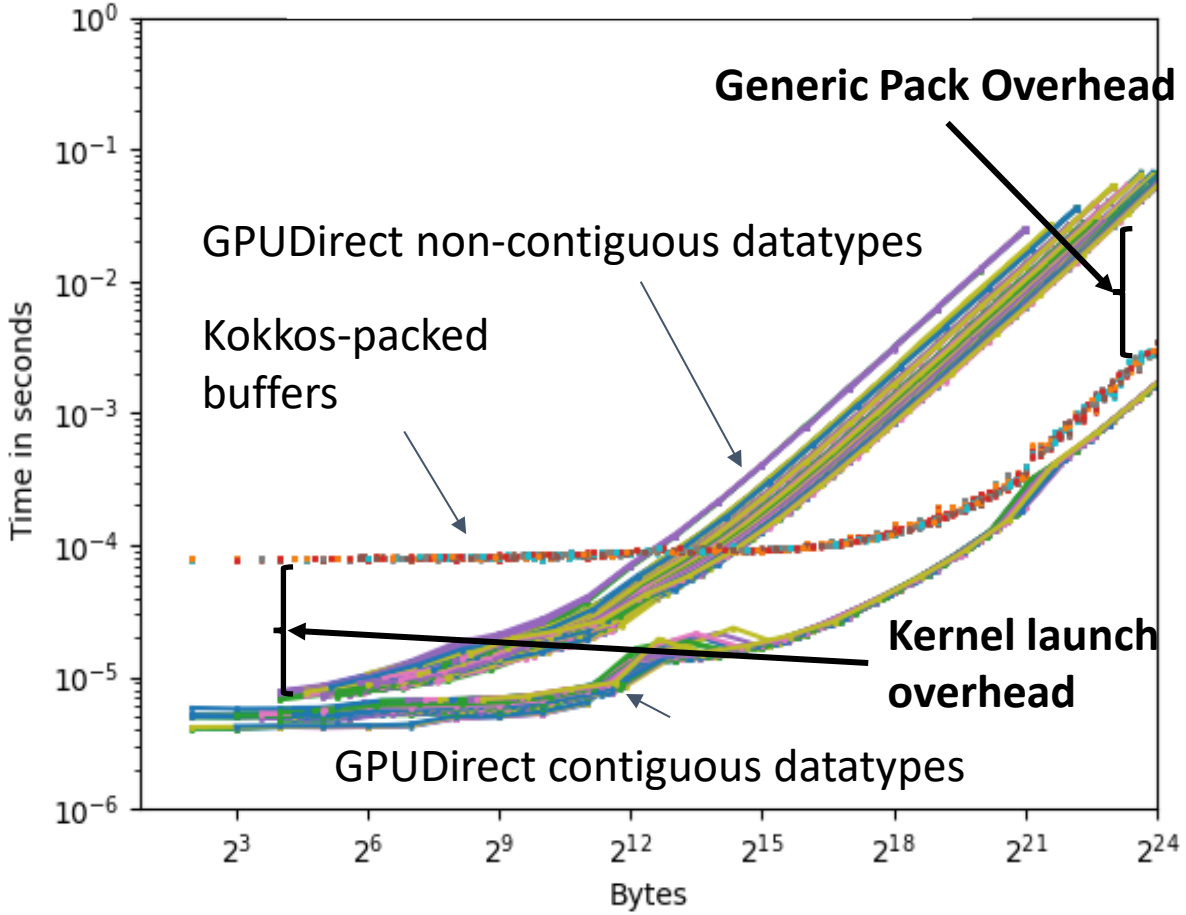
Beatnik: A Proxy for Remapping and Global Communication

- Release 1.0 available
  - https://github.com/CUP-ECS/beatnik
  - Scalable low-order solver (uses HeFFTe)
  - Non-scalable brute force high-order solver
  - Structure for implementing other solvers
- Current development – Jason's Poster
  - New cutoff-based solver that remaps between surface and spatial distributions
  - Adding support for irregular spatial meshes
  - Collecting communication profiles
  - Developing additional input decks
- Potential Future:
  - Addition of surface remeshing support
  - Integration with CFD miniapp for multi-scale proxy



Irregular Data Communication

- Struggled with arbitrary datatypes
  - Generic datatypes can be faster for small irregular data
  - Generic datatypes prohibitively slow for large irregular data
- The improvements to the right were not reproducible (mvapich bugs!). The slowdowns to the right were.
- Simple performance studies showed benefits of complex APIs for this not worth it.
  - Key remains getting rid of unnecessary synchronization for datatype handling
  - Stream and kernel triggering of communication is the right way to do this, not convoluted APIs



Any Questions?

Thank you!

Overall Plan: Communication Characterization

High-level Roadmap

- Goals:
  - To understand which communication characteristics affect performance
  - Obtain a deeper understanding of communication patterns outside of general timing metrics
    - Such an understanding would allow the creation of models or benchmarks that represent the true communication pattern(s) of applications
    - For example, understand the performance implications of an application using hand-crafted collectives instead of MPI equivalents
- How:
  - Instrumenting Applications
  - Instrumenting MPI implementations
  - Analyzing the data we get
- Nick (UNM) has a Sandia Internship scheduled for this summer, with the focus on trying to model how much time can overlapped by reducing synchronization costs

Example App: xRAGE (specifically the Cellar framework)

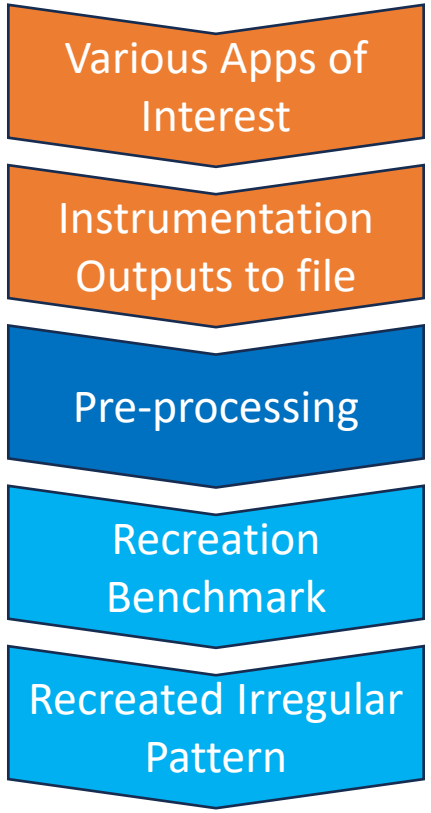
- Wanted to look into how the framework was performing at large scales
- Caliper timings can paint a picture of general framework effectiveness, but doesn't describe communication pattern
- Decided instrument two key areas:
  - Communication pattern creation:
    - Call site and ID (same across all ranks), ranks involved, and creation time
    - The base direction(s) and count(s) of data to be exchanged
    - Example:
      - Rank 0: 0 | B1:1-200, T2:53, F4:90, | 1.23 | T0
      - Rank 1: 0 | B0:200-1, B2:5-400,B3:1500-1600, | 0.97 | T0

B (both) (rank) : (To #, From #)
T (to) (rank) : (To #)
F (from) (rank) : (From #)

  - Communication usage:
    - ID, direction (scatter or gather) ,size of datatype involved (char, int, double, etc.), time
    - Example of three uses: 0:0:8|1.01, 1:0:8|2.45, 0:0:4|0.78

Selecting the Right Parameters

- Attempted to determine what parameters from irregular applications are needed to recreate communication patterns
  - Number of partners, size of message(s), stride, etc.
- General flow pictured on right
  - Chosen parameters are captured by manual logging
  - Generates distributions based on collected data
  - Distribution data is then used by re-creation benchmark
- Paper accepted to CCGrid24 (May 6th-9th, 2024)
  - *Quantifying and Modeling Irregular MPI Communication*
  - Carson Woods will present

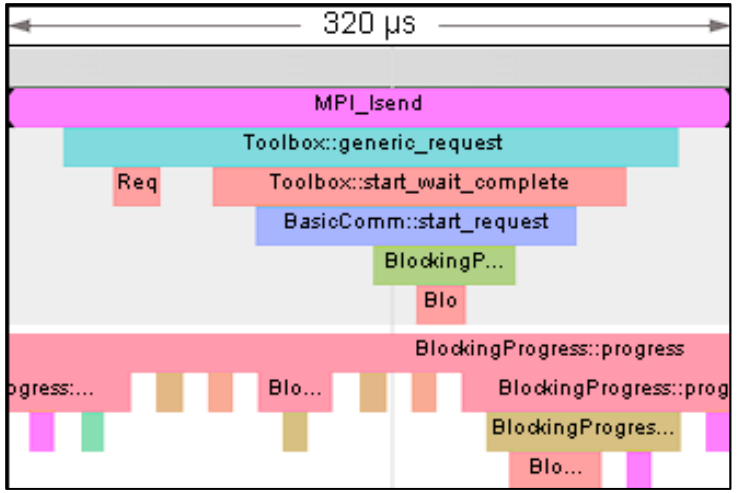


Instrumenting Applications

- Have explored several applications (xRAGE, Cabana, *hypre*, CLAMR, Parthenon), but each was manually instrumented
- Leverage learning to create a more portable approach
  - Likely to be an MPI eXtension Library
  - Also likely to exist within the MPI Advance collection of libraries
- New APIs to let users control which MPI calls are a part of a “pattern”
  - Mark a start/end of a region?
  - Record additional parameters?
  - Still considering what options to provide – happy to discuss ☺
- Exploring the MPI “Python-ization” tools to write boilerplate wrappers
  - Intercept all MPI calls, record requested parameters
  - Integration with Caliper (e.g. auto caliper regions if using this library), Kokkos profiling, NVTX

On the MPI Implementation Side

- Last year, we showed some profiling work using ExaMPI
- Used internal profiling to help optimize ExaMPI's general performance
  - Middle – Visualization of events in ExaMPI
  - Right – Comparison of ExaMPI and Spectrum MPI running Comb (visualized with Hatchet)
- Had also started exploring tracing MPI messages




```
1.665842 main
├─ 2.105263 MPI_Cart_coords
├─ 3.529202 MPI_Cart_create
├─ 1.857143 Mesh Host
│  └─ 1.341808 Comm mpi Mesh seq ... seq Ho
│     ├─ 0.678550 MPI_Barrier
│     ├─ 6.866003 MPI_Comm_dup
│     ├─ 132.375000 MPI_Comm_free
│     ├─ 3.200000 MPI_Comm_set_name
│     ├─ 0.617492 MPI_Reduce
│     ├─ 1.306083 add vars
│     │  └─ 4.464677 MPI_Cart_rank
│     ├─ 3.264014 bench comm
│     │  ├─ 2.108447 MPI_Barrier
│     │  ├─ 4.324561 cycle_0
│     │  │  ├─ 1.016166 post-comm
│     │  │  ├─ 3.075000 post-recv
│     │  │  │  └─ 2.084951 MPI_Irecv
│     │  │  ├─ 1.185704 post-send
│     │  │  │  └─ 2.371490 MPI_Isend
```

Profiling MPI Performance Behaviors

- Goal: Understand the behaviors of many different kinds of messages as they travel through an MPI software stack.
  - First, isolate the stages of an MPI implementation to identify areas of poor performance
  - Later, determine factors that influence performance at each stage during the lifespan of a message
- Two TNTech students (Evelyn and Grace) have internships scheduled with LLNL this summer; Riley Shipley is overseeing this effort
- This will be made possible through two new tools:
  - A C++ logging library that will capture timing data for the entire message life cycle
  - A Python-based app that will calculate and visualize performance probability distributions based on the captured data

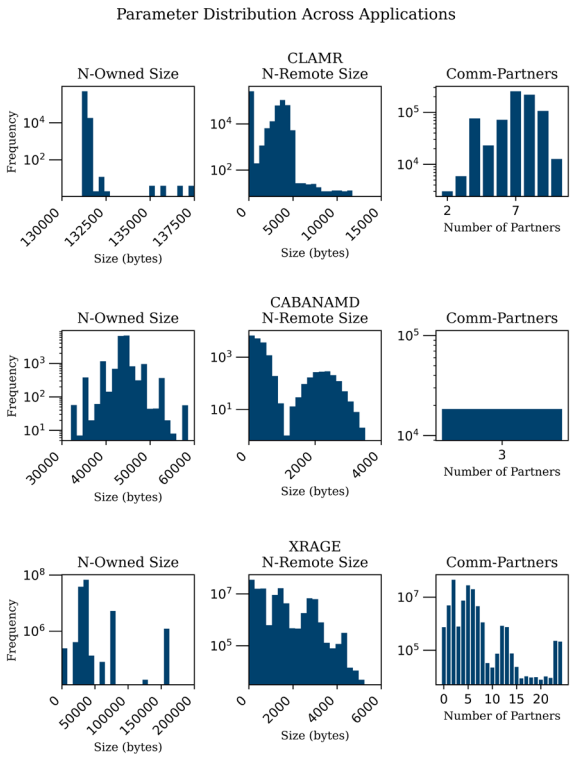
Capturing MPI Implementation Behaviors

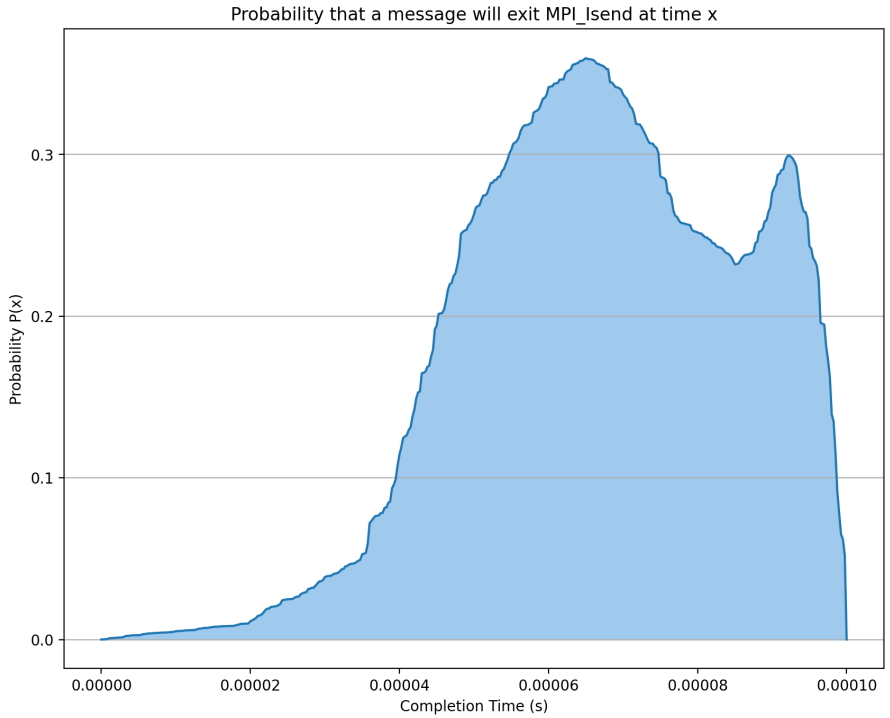
- Aim is to capture message performance data across the entire MPI implementation stack, including:
  - Setting up network resources
  - Setting up buffers/packing/unpacking
  - Time spent in networking libraries
  - Matching
- Amount and types of data available depends on stage in the MPI stack
- Need a library that is flexible to handle changing or missing data

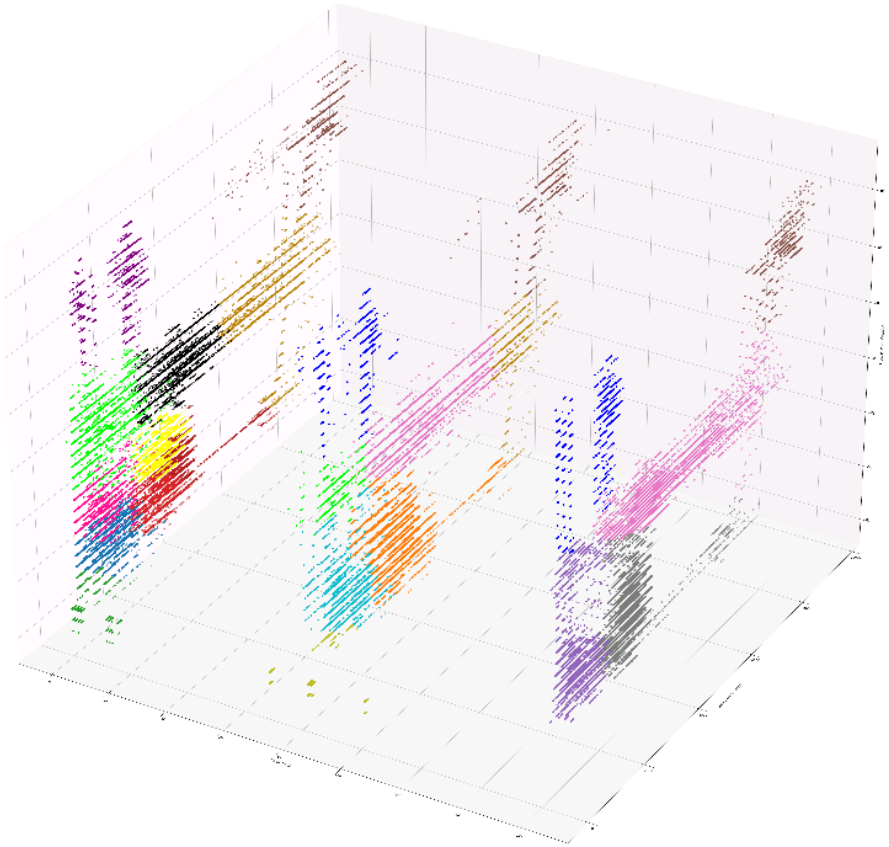
```
1  int main(int argc, char**argv) {
2  MPI_Init(&argc, &argv);
3  // ...
4
5  MPI_Send(…,dst=1);// Message to rank 1
6  MPI_Allgather(…); // Message to rank 1
7  }
```

Both MPI calls could produce similar looking messages on Rank 1

Analyzing the Data - Examples







Any Questions?

Thank you!

Overall Plan: Education and Outreach

Educational Activities

- Organized the 10th Workshop on Extreme Scale MPI (ExaMPI 2023) at SC 2023
- Tutorial on *MPI Advance - Optimizations and Extensions to MPI* at the NNSA-University Workshop on Exascale Simulation Technologies (NUWEST) workshop
- Organized SIAM PP24 mini-symposium on *Realistic Proxy Applications and Datasets for Heterogeneous Architecture Scalable Communication*
- Organizing the 4th Workshop on Compiler-assisted Correctness Checking and Performance Optimization for HPC (C3PO) at ISC 2024
- Hackathons
  - Weekly online mini-hackathons with students to help with their research
  - April 9-10, 2024 – in-person hackathon for students focused on specific research problems
- Course development – focusing on homework assignments

Courses Offered – Spring 2024

- UNM
  - CS 491/591 Special Topics: HPC
  - CS 591 Special Topics: Scalable Systems Seminar
- TNTech
  - CSC 4760/5760 Parallel Programming
  - CSC 7750 HPC Seminar
- UA
  - CS 691 Special Topics in HPC

Assignment #1

- Goal: Implement basic point-to-point send and receive primitives
- Explore different low-level networking primitives
  - Sockets
  - Light-weight Communication Interface (LCI)
  - Libfabric/UCX
- Using ExaMPI infrastructure to support process startup and teardown and integration with Slurm

API Specification

```
int     MY_MPI_Rank()       // Provided
int     MY_MPI_Size()       // Provided
void    MY_MPI_Barrier()    // Provided
void    MY_MPI_Init()       // Provided
void    MY_MPI_Finalize()   // Provided
double  MY_MPI_Wtime()      // Provided
void    MY_MPI_Ssend(void *buf, size_t len, int dest)          // TODO!
void    MY_MPI_Recv (void *buf, size_t len, int src)           // TODO!
void    MY_MPI_Bcast_linear(void* buf, size_t len, int root) // TODO!
void    MY_MPI_Bcast_ring  (void* buf, size_t len, int root) // TODO!
```

Next Assignments

Assignment #2

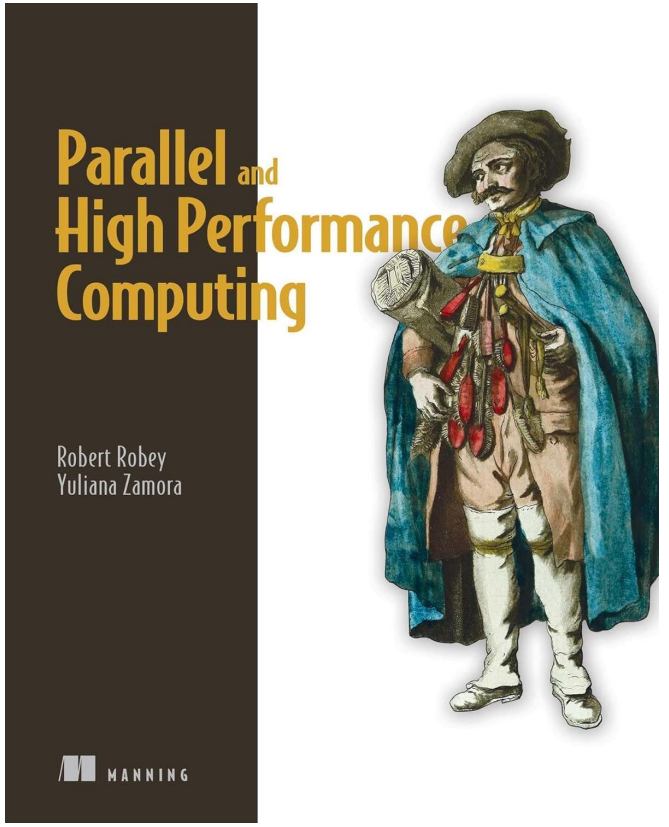
- Extend Assignment #1 to include
  - tags
  - eager and rendezvous protocols
  - non-blocking communication and progress

Assignment #3

- Implement collective primitives with different algorithms
  - reduce/allreduce
  - gather
  - scatter

TNTech – CSC 4760/5760 Parallel Programming

- Standard concepts of parallel programming (speedup, Flynn, etc)
- Focus on MPI and Kokkos Programming
- C++ as underlying programming language
- MPI+Kokkos Programming as advanced topic
- Specifically skip: OpenMP and CUDA in favor of Kokkos as the high-level, on-node programming model
- Rudiments of cluster usage and access issues
- Current book: Robey & Zamora
- Future: Our own book



TNTech – CSC 7750 HPC Seminar

- For PhD students and interested MS students
- Data reorganization
- Fault tolerance
- Polyalgorithms
- Many-task systems (WAMTA 2024 workshop highlights)
- Methods of HPC research
- Special topics include
  - Space-filling curves
  - Meta-programming
  - FFTs
  - Shared materials from PSAAP April Hackathon (RDMA)
- How to formulate research

Outreach Activities

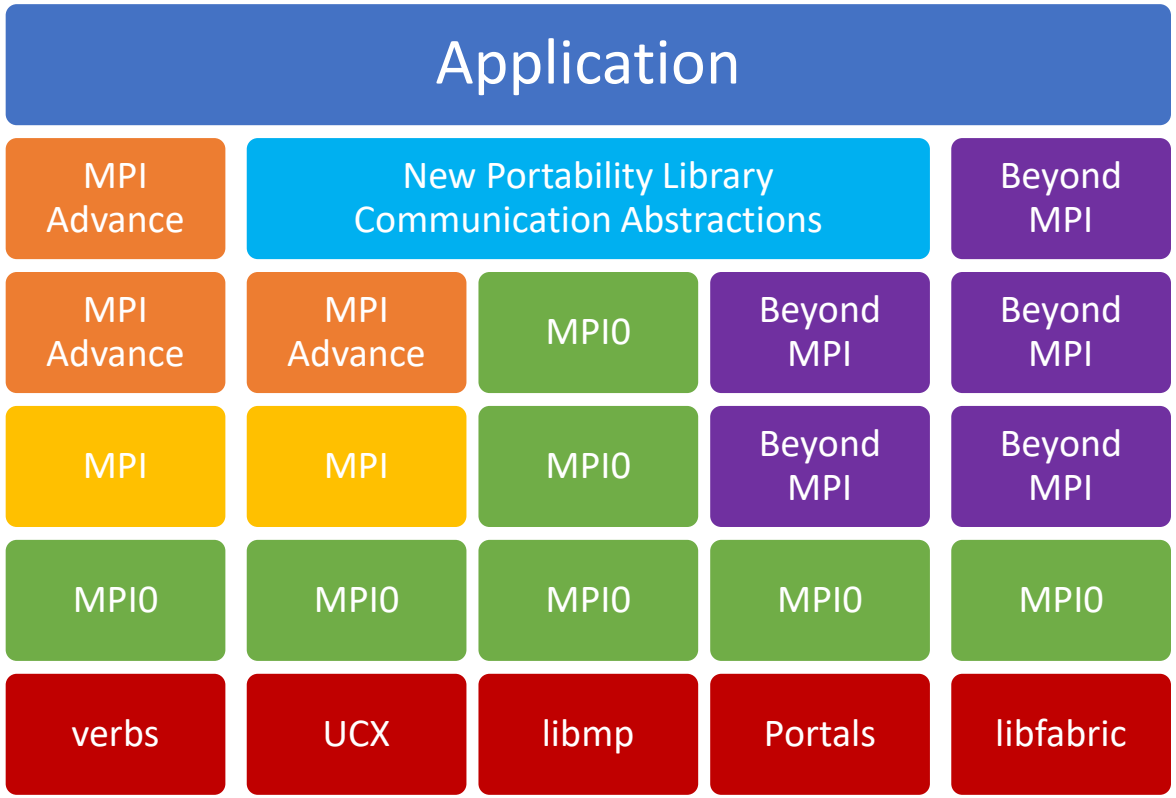
- UNM
  - PSAAP staff member and student leading and mentoring team for SC Student Cluster Competition
  - Dr. Bienz served as a panelist on *Breaking Barriers: The Career Odysseys of Diverse Women in Computing* organized by SIAM Activity Group on Equity, Diversity, and Inclusion (SIAG-EDI) at SIAM PP24
- TNTech
  - Seminar on HPC research to the graduate student club
  - Meet with and encourage diverse high school students (CS 38-30 students) to pursue CS degrees
- UA
  - Seminar on HPC to the ACM UA Student Chapter
  - Guest lecture on HPC in *CS 121 The Discipline of Computing* course

Questions?

Break

Overall Plan: API Development

Communication Abstraction Stack View



- Goal is to develop new abstractions for applications and libraries
- Development at each level is driven by careful assessment, benchmarking, and modeling
- Through the first two years, focus has mostly been on Advance-level primitives
- Shifting focus to MPIO and Beyond abstractions

Major API Development Thrusts

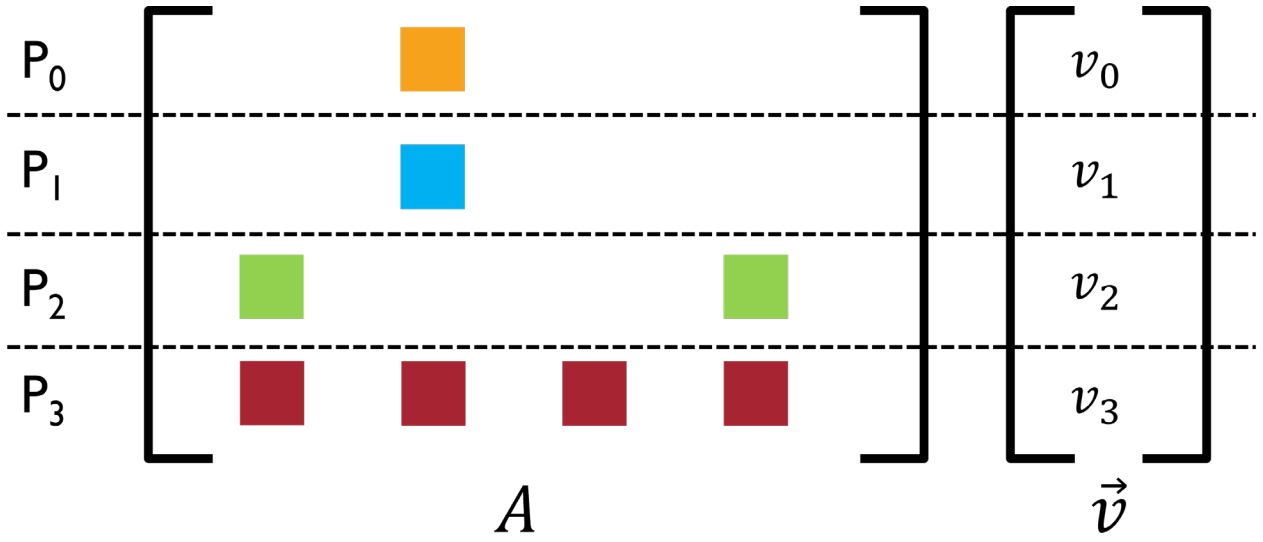
- MPI Advance
  - Optimized Irregular Collective API Abstractions (next) - Amanda
  - Profiling of communication patterns (previously discussed) – Derek
- Portability Library Primitives:
  - New Cabana/Kokkos fine-grain compute/communicate primitives – Patrick
  - Communication interfaces for Kokkos – Evan
- Low-level Communication Primitives - Derek

Irregular Communication

Amanda Bienz, UNM

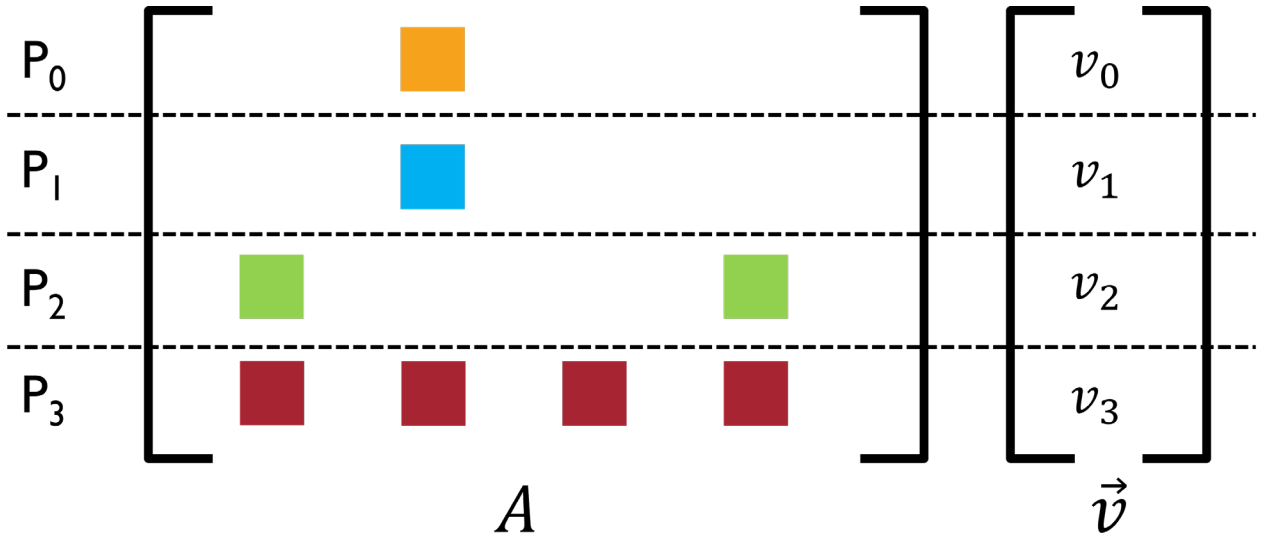
April 11, 2024

Sparse Matrix Operations



1. Form communication pattern
   No existing MPI API
2. Create graph communicator
   MPI_Dist_graph_create_adjacent
3. Initialize communication
   MPI_Neighbor_alltoallv_init
4. Per-iteration communication
   MPI_Start
   MPI_Wait

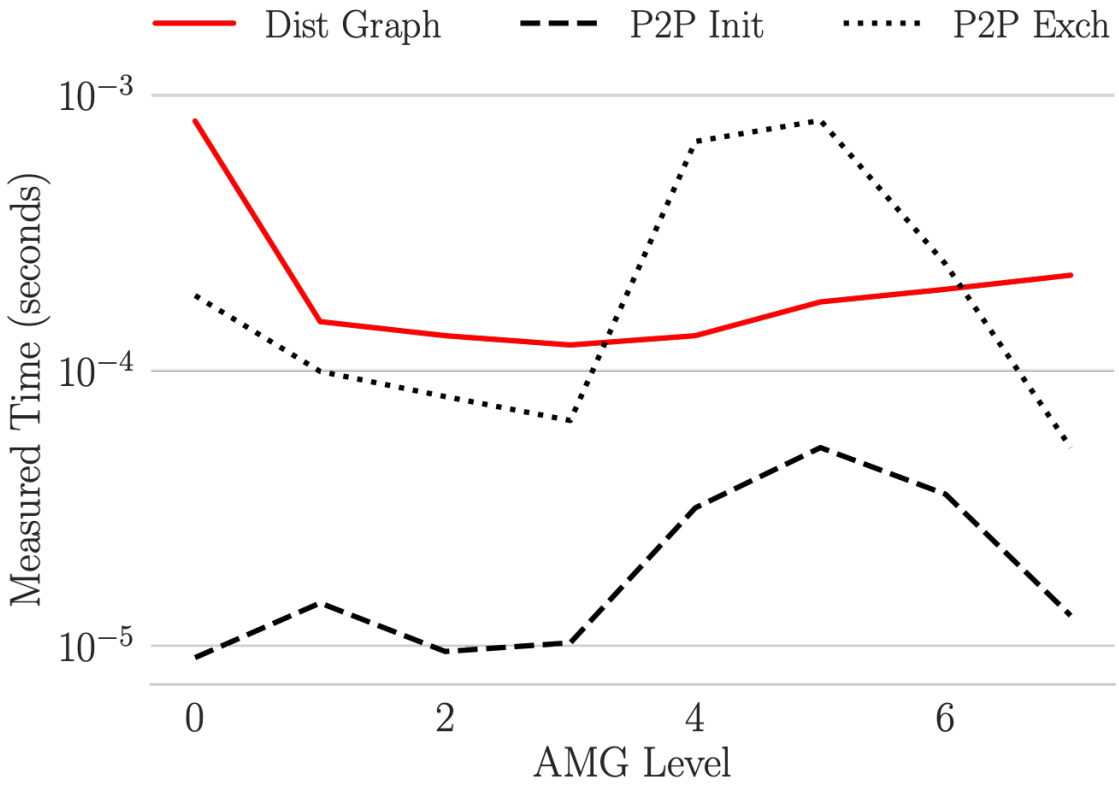
Problems with Current Approach



1. Form communication pattern
   No existing MPI API
2. Create graph communicator
   MPI_Dist_graph_create_adjacent
3. Initialize communication
   MPI_Neighbor_alltoallv_init
4. Per-iteration communication
   MPI_Start
   MPI_Wait

All applications perform own optimizations

Problems with Current Approach



1. Form communication pattern
   No existing MPI API

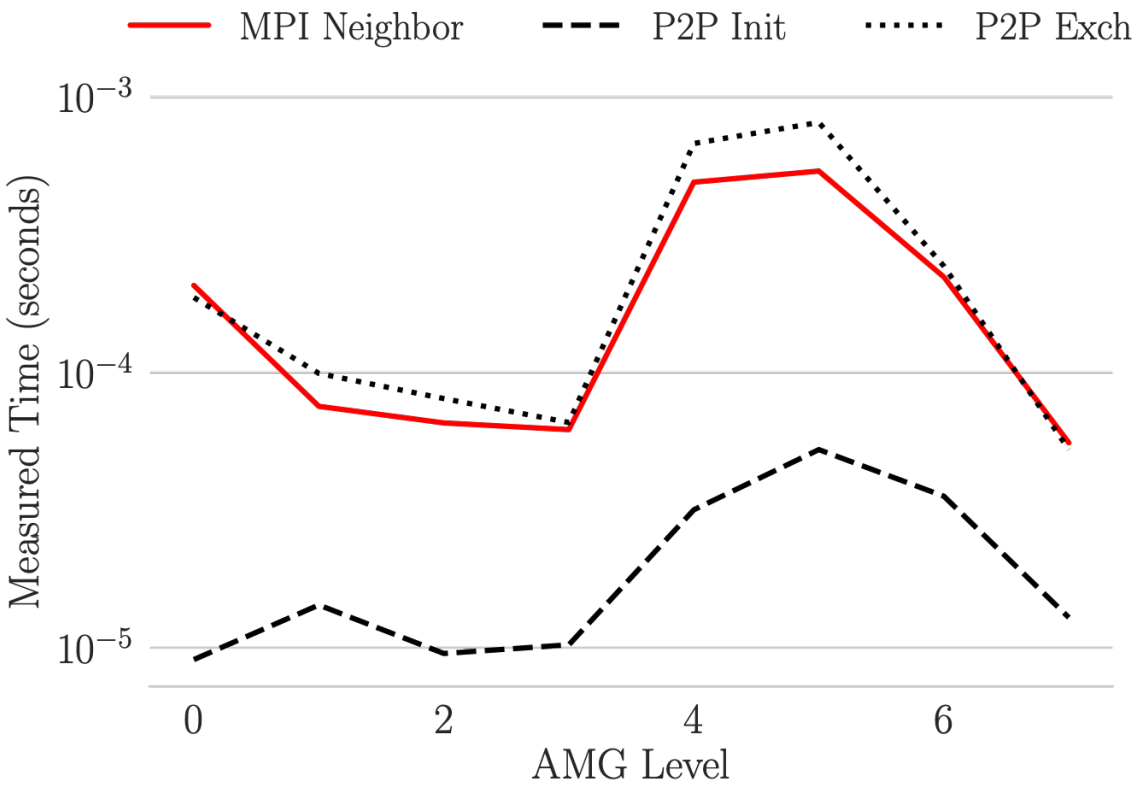
2. Create graph communicator
   MPI_Dist_graph_create_adjacent

3. Initialize communication
   MPI_Neighbor_alltoallv_init

4. Per-iteration communication
   MPI_Start
   MPI_Wait

Large overhead from graph communicator

Problems with Current Approach



1. Form communication pattern
   No existing MPI API

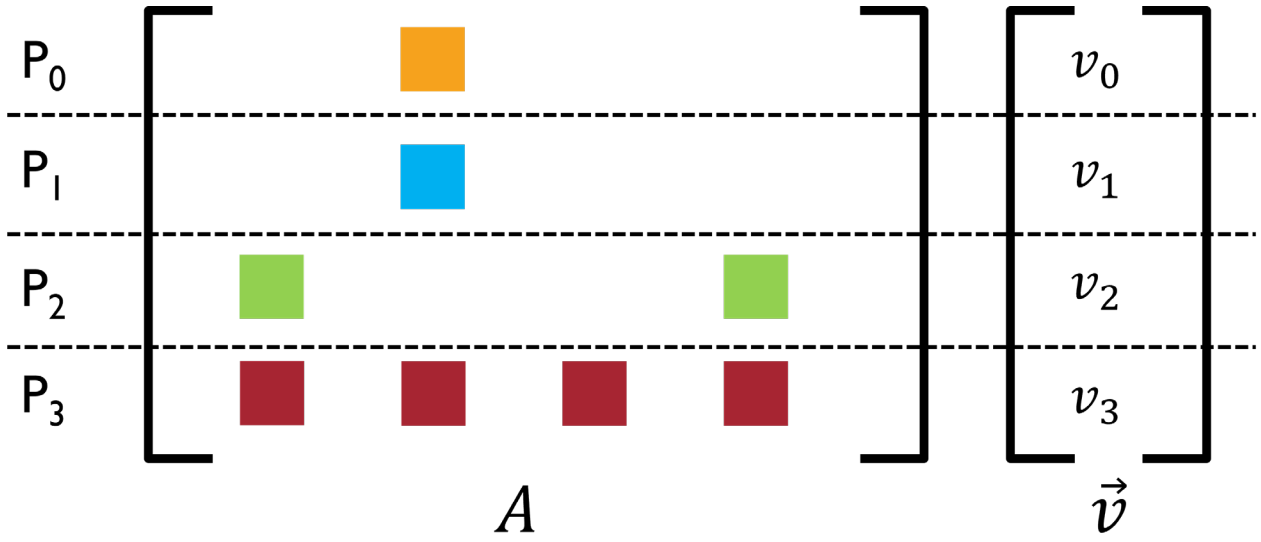
2. Create graph communicator
   MPI_Dist_graph_create_adjacent

3. Initialize communication
   MPI_Neighbor_alltoallv_init

4. Per-iteration communication
   MPI_Start
   MPI_Wait

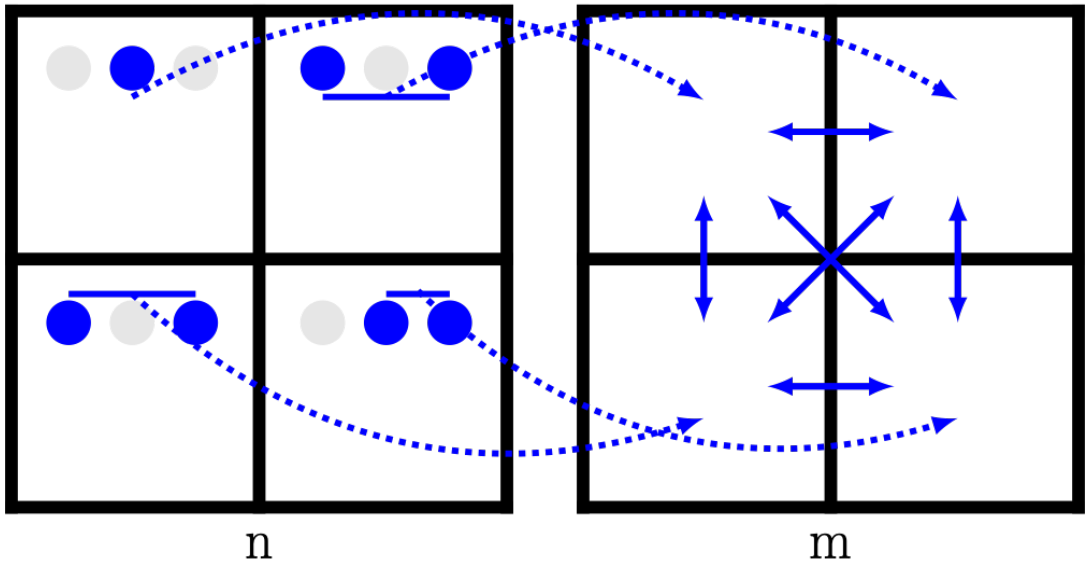
Neighbor collectives are unoptimized

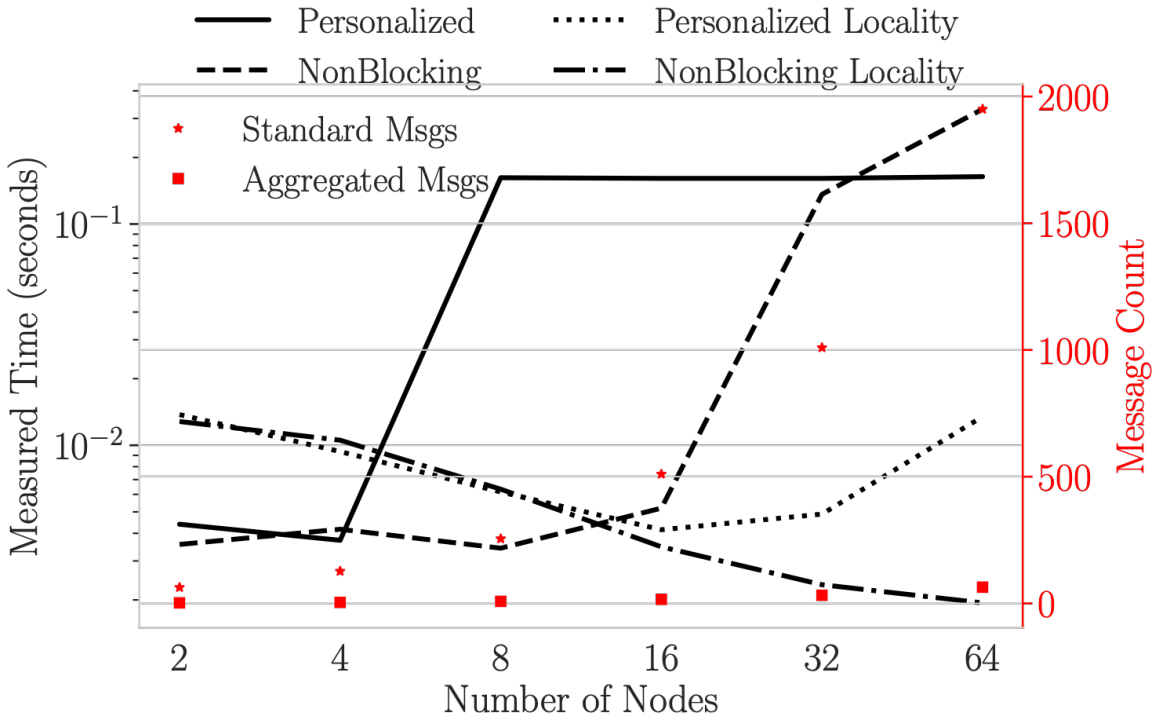
Topology Discovery



- MPIX_Alltoallv_crs performs sparse dynamic data exchange under the hood
- Standard implementations wrapped in this API, available in MPI Advance
- Also, includes MPIX_Alltoall_crs

Locality-Aware Topology Discovery

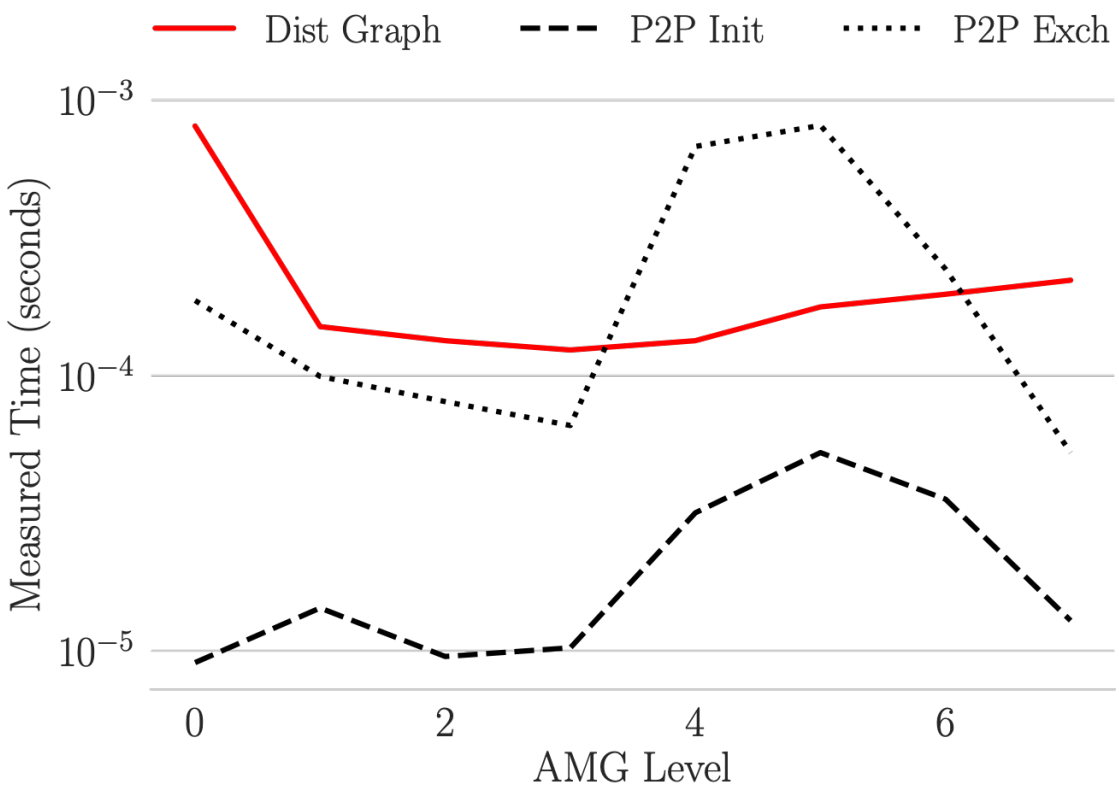




More on Topology Discovery

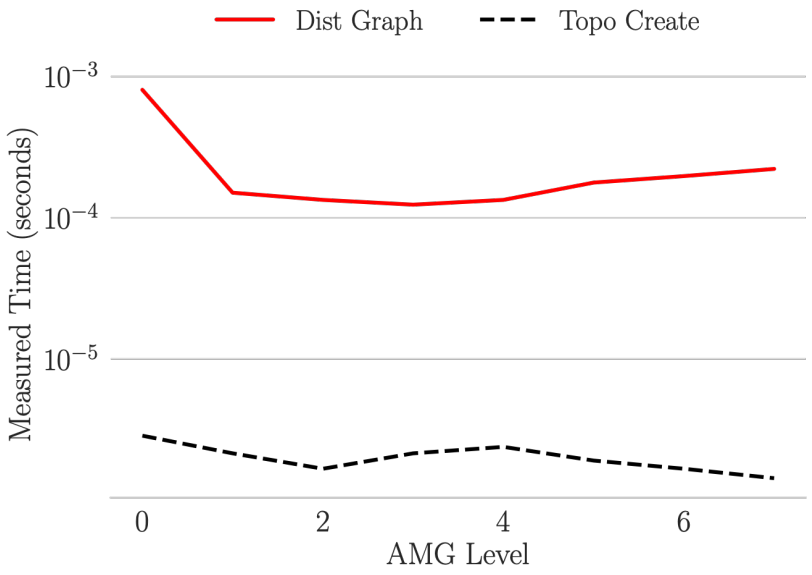
- Andrew Geyko's poster
- *A More Scalable Sparse Dynamic Data Exchange* paper in submission
  available on arxiv *https://arxiv.org/abs/2308.13869*

Graph Communicator Creation

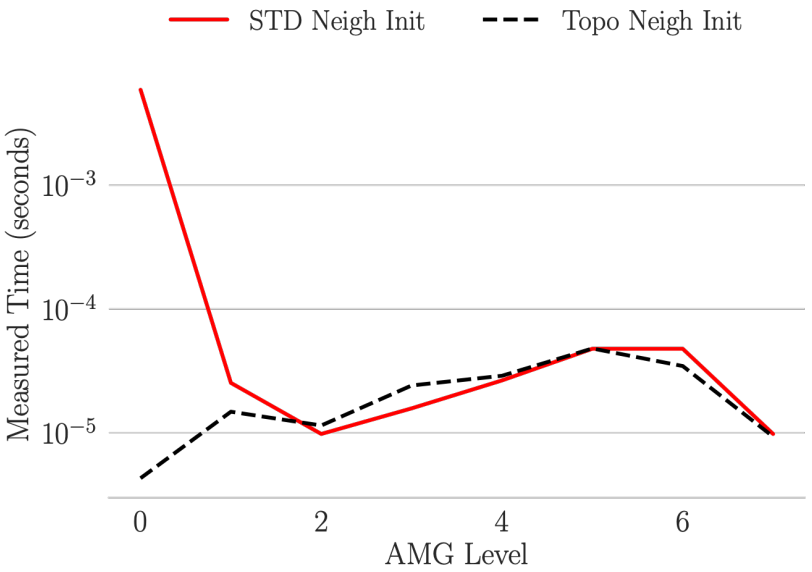


- Already know topology information (discovered as a part of the previous work)
- This information is passed to graph communicator creation
- August 2023 Hackathon : students created a topology object to replace graph communicators

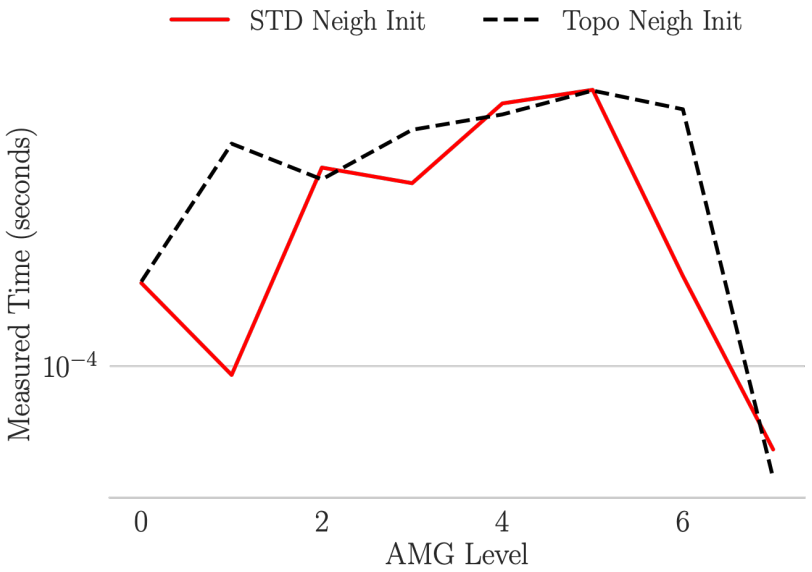
Graph Communicator Creation



Graph Communicator Creation



Persistent Neighborhood Collective Initialization

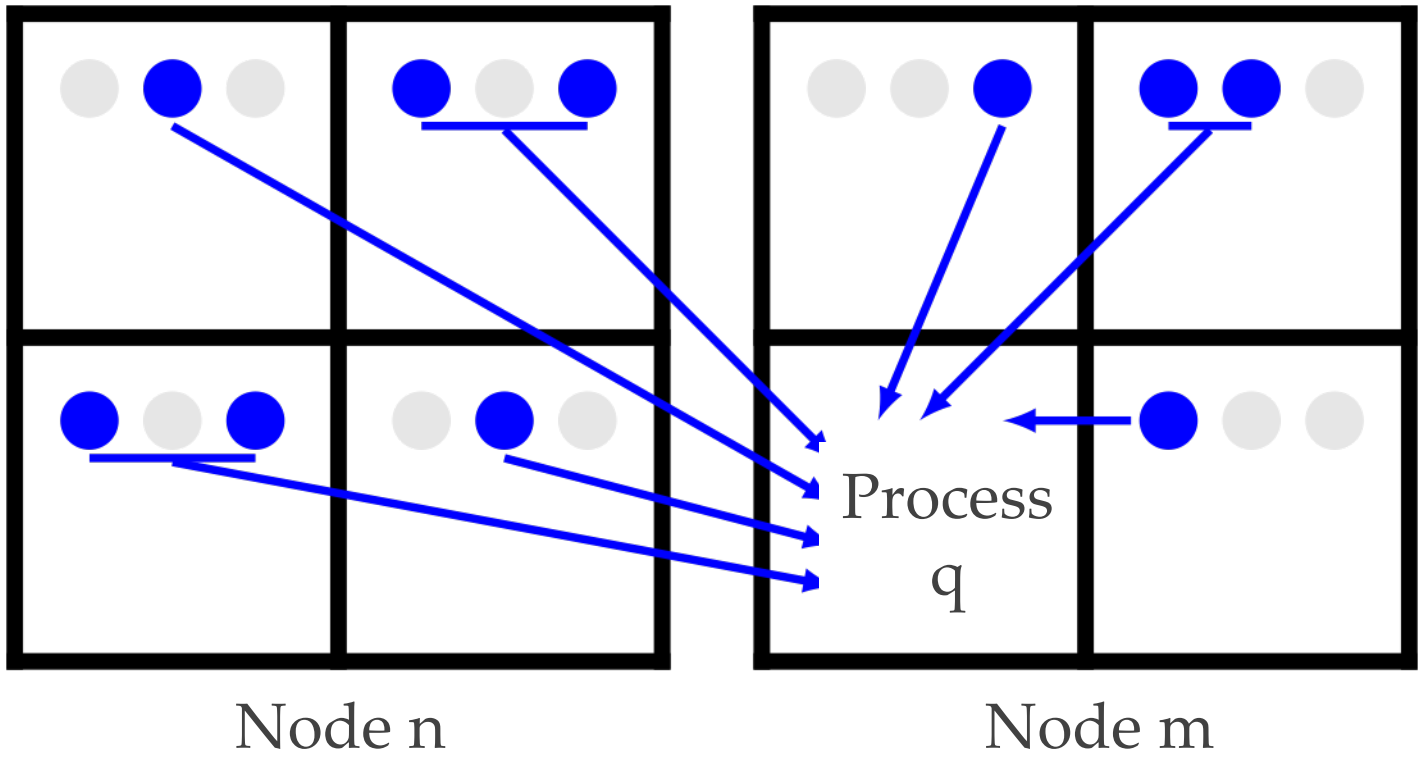


Per-Iteration Start/Wait

More on Topology Objects

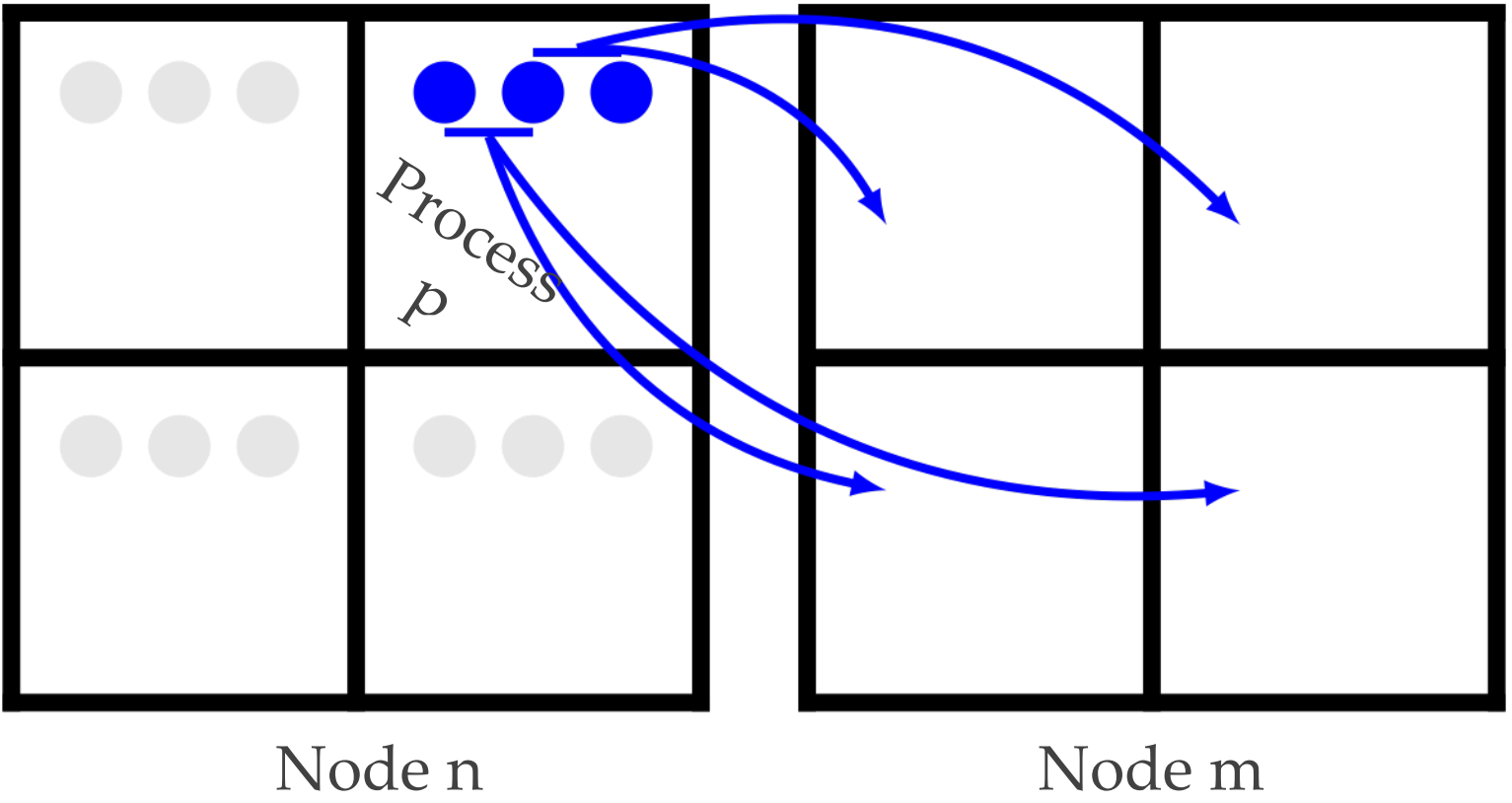
- Gerald Collom's Poster
- *Optimizing Neighbor Collectives with Topology Objects* paper in submission

Locality-Aware Neighbor Collectives



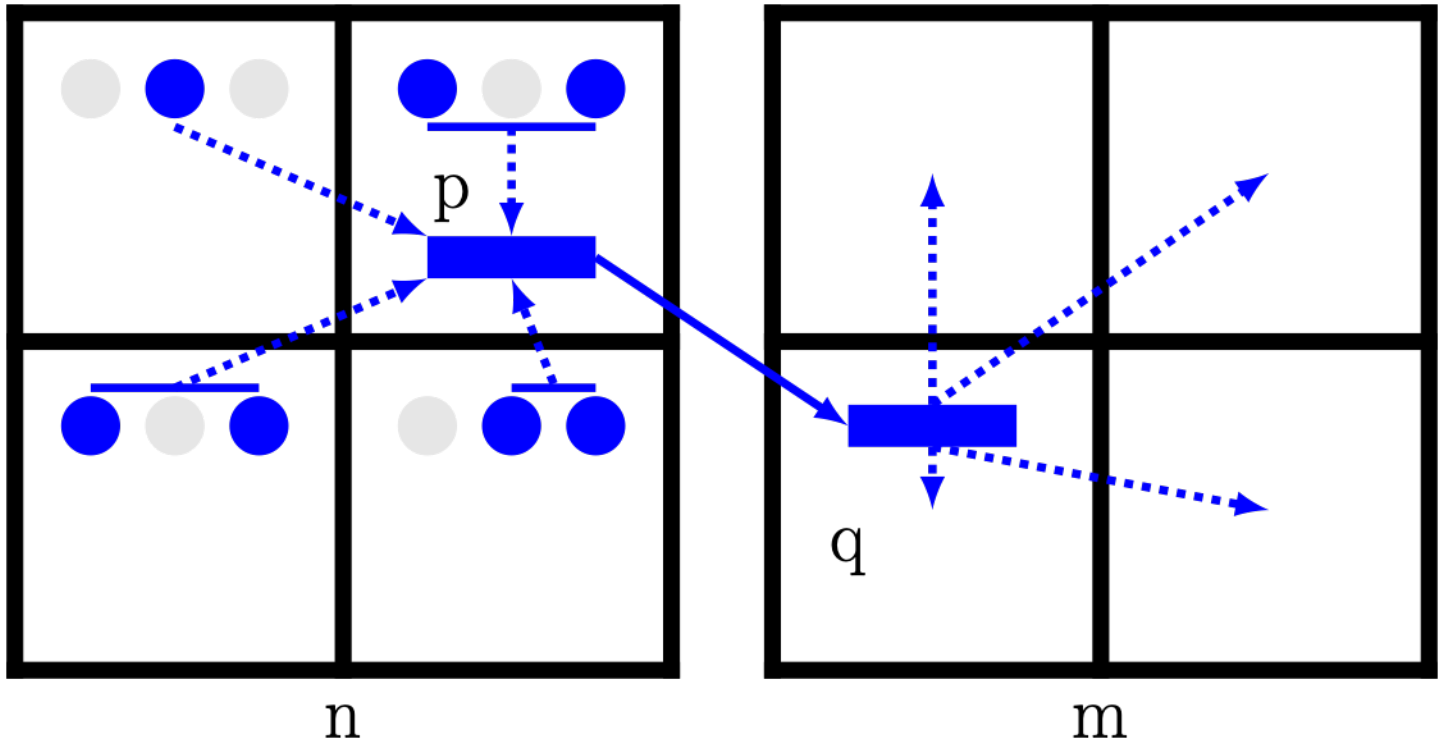
- Neighbor collectives typically wrap standard P2P communication
- Multiple messages between sets of regions (e.g. nodes)

Locality-Aware Neighbor Collectives



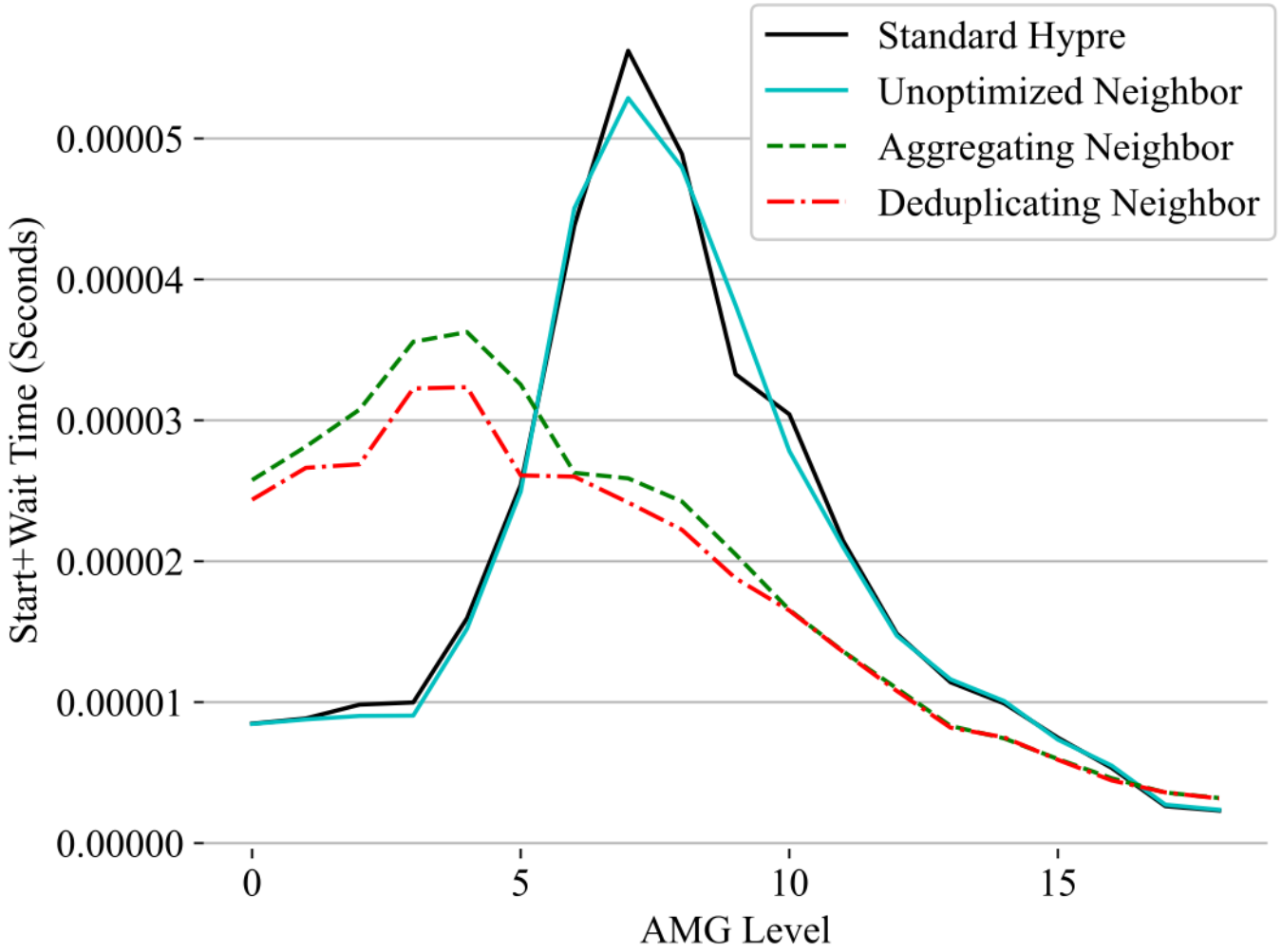
- Neighbor collectives typically wrap standard P2P communication
- Multiple messages between sets of regions (e.g. nodes)
- Data sent multiple times between single set of regions

Locality-Aware Communication



All processes per node are active in communication

Neighbor Collectives in HYPRE



- Per-iteration costs greatly reduced on coarse levels
- Gerald's poster
- Optimizing Irregular Communication with Neighborhood Collectives and Locality-Aware Parallelism (https://arxiv.org/pdf/2306.01876.pdf)

More on Neighbor Collectives

- Gerald Collom's Poster
- *Optimizing Irregular Communication with Neighborhood Collectives and Locality-Aware Parallelism* published at ExaMPI workshop at SC23

Questions?

Fine-grain Communication APIs

Prof. Patrick Bridges

Communication limits on scaling

- GPU kernel launches and synchronizations add latency to communication in GPU systems
  - Launches and syncs for data packing and unpacking
  - Launches and syncs for sends and receives
  - Syncs on compute when strong scaling
  - Difficult to hide communication behind application or pack/unpack compute
- Currently examining how this limits strong scaling in GPU codes
  - Starting with miniAero, UME or a Cabana mini-application are possible next steps
  - Goal: Communication limits on strong scaling of Ifpack2 on a SPARC matrix
  - Nick Bacon will be examining this in his internship at Sandia this summer
- API Design Approach:
  - Top down pass starting from high-level application/library communication primitives and how they can be be modified to enable hiding or eliminating these costs
  - Bottom up pass starting from NIC triggering primitives to understand what primitives could efficiently do

Where are the costs hiding in the application communication primitives?

```
// Declare the basic tiled array layouts, arrays, and halos similar
// to current Cabana abstractions
Cabana::ArrayLayout array_layout(...);
Cabana::Array src_array(array_layout), dst_array(array_layout);
Cabana::Halo stencil_halo(array_layout, Cabana::Halo::NodePattern<2>, 2);

// Declare the portion of the array each process will be iterating over
// and create a stencile functor for the iteration
Cabana::IndexSpace index_space = array_layout.indexSpace(Own(), Cell());

for (int i = 0; i < maxiter; i++) {
    stencil_halo.gather(src_array); // Host performs a synchronous gather
    // Then launch a parallel for loop to compute the values from haloed data
    Kokkos::View<double ***> src_view = src_array->view();
    Kokkos::View<double ***> dst_view = dst_array->view();
    StencilFunctor functor(src_view, dst_view);
    Cabana::grid_parallel_for("Stencil Loop", DefaultExecutionSpace(),
        index_space, functor);
    Kokkos::fence(); // Fence the stream so that we know calculation is done before
                     // doing the gather for hte next iteration
    src_array.swap(dst_array);  // switch the source and destination arrays
}
```

- Halo gather includes kernel launches and syncs for data packing and unpacking
- Explicit synchronization between compute kernel and next communication invocations

Stream triggering to eliminate some costs

```
for (int i = 0; i < maxiter; i++) {
    stencil_halo.gatherEnqueueStart(src_array); // Start a BSP halo exchange
    stencil_halo.gatherEnqueueWait(); // Wait for it to finish
    auto src_view = src_array->view();
    auto dst_view = dst_array->view();
    StencilFunctor functor(src_view, dst_view);
    // Enqueue the parallel for on the haloed data
    Cabana::grid_parallel_for("Stencil Loop", DefaultExecutionSpace(), index_space,
        functor);
    src_array.swap(dst_array);
}
Kokkos::fence(); // Wait for all enqueued communication and computation
```

- Fence between computation and gather removed
- Fences between packing, sending, receiving, and unpacking removed inside implementation of gather primitive
- Same type of primitives should also work for stream triggering Cabana AoSoA, xRAGE Token, and other irregular halo primitives
- Currently implementing with Cray stream triggering primitives on Tioga, Chicoma
- Our previous studies show modest weak-scaling benefits, will be working to quantify strong scaling benefits
- Still significant kernel launch overheads, especially with non-fused irregular buffer packing

Early Send to Remove Packing Launch

- Leverage nested parallelism to fuse packing and compute kernel
  - Define *Tiled* versions of Cabana Arrays, IndexSpaces, and Halos
  - Tiled halo can use thread team to parallel pack outgoing data
  - Tiled halos can use either GPU partitioned send or RMA put/get
  - Also provides access to block shared memory for compute and packing
  - Single kernel iterates over ***owned*** tiles/cells to both compute and pack data to send
- Can also just fuse packing by iterating only over **boundary** tiles/cells
- Doesn't require cooperative launch, fine-grain tasking, or other specialized scheduler support to avoid deadlocks
- But receives and unpacking still unfused and stream triggered

```
tiled_halo.gather(src_array); // Start with a single host synchronous gather
for (int i = 0; i < maxiter; i++) {
    // Enqueue the start of the halo for the next iteration. It's safe to start sending
    // our destination array to their source, because they've already completed and sent
    // their source to us.
    tiled_halo.gatherEnqueueStart(dst_array);
    // Now do a parallel for over tiles with owned data that does early
    // sends of the halo for the next iteration as tiles finish
    auto src_view = src_array->view();
    auto dst_view = dst_array->view();
    StencilFunctor functor(src_view, dst_view);
    Cabana::grid::grid_parallel_for("Tiled Early Send Stencil", ExecutionSpace(),
        tiled_index_space,
    KOKKOS_LAMBDA(member_type team_member) {
        // Now the threads on this team iterate over the cells
        // in the tile the team owns.
        nested_parallel_for(tiled_index_space, team_member, functor);
        team_member.barrier(); // then the team barriers
        // Finally, the threads on the team pack and send any data
        // they have computed!
        tiled_halo.gatherReady(index_space, team_member, dst_array);
    });
    // Send is kernel triggered, but receives and their unpacking are
    // are still stream triggered to avoid polling from the kernel.
    tiled_halo.gatherEnqueueWait(dst_array);
    dst_array.swap(src_array);
}
Kokkos::fence(); // Wait for all enqueued communication and computation
```

Early Work to Fuse Receive Unpacking

```
for (int i = 0; i < maxiter; i++) {}
    tiled_halo.gatherEnqueueStart(src_array);
    // Now do a parallel for over tiles that can start work on data at is received
    auto src_view = src_array->view();
    auto dst_view = dst_array->view();
    StencilFunctor functor(src_view, dst_view);
    Cabana::grid_parallel_for("Tiled Early Work Stencil", ExecutionSpace(), index_space,
    KOKKOS_LAMBDA(member_type team_member) {
        // All team threads pack halo data to send and then
        // after a team barrier, one thread marks data as ready
        tiled_halo.gatherReady(index_space, team_member, src_array);
        // One team thread polls for data availability nd then
        // after a team barrier all unpack needed halo data
        tiled_halo.gatherAvailable(index_space, team_member, src_array);
        team_member.barrier(); // wait for all unpacking to finish
        // Compute using available halo data
        Cabana::nested_parallel_for(index_space, team_member, functor);
    });
    tiled_halo.gatherEnqueueWait(dst_array); // Waits for all receives to finish
    dst_array.swap(src_array);
}
Kokkos::fence(); // Wait for all enqueued communication and computation
```

- Thread team needs to poll for get/recv completion
- Fuses unpacking with compute and packing
- Requires support to avoid deadlock
  - Runtime system support: fine-grain tasks or cooperative launch
  - Hardware scheduler: preemption or new features
- Could reduce compute efficiency
- Lots of tradeoffs and parameters to tune and model in these approachs
- Extension to unstructured more complex, plan to examine in Cabana, UME

Need new communication primitives to make building these abstractions easy

- Explicit MPI Progress abstraction that integrates community stream triggering ideas
- Enumerating potential design principles for new channel abstraction
  - Support both one-sides and two-sided operations with convenient, easy-to-understand semantics
  - Separate expensive bulk operations (setup, matching, remote completion) from inexpensive data movement operations
    - Match related buffers independently of the set of operations (send, receive, put, get) that will be performed on them
    - Match on *collective* channel operations as well to support asynchronous collective matching
    - Indicate buffers ready for use (send or receive) independent of specifying operations being performed
  - Use host and stream synchronization for heavyweight bulk operations
  - Use partitioning, trigger increments, and polling for lightweight fine-grain operations
  - Focus on semantics and usability in modern languages (C++, etc.) over C/Fortran interface
  - Pay close attention to needs of performance portability frameworks – need to contact Raja team

```
// Operations for creating channels. Currently only point-to-point
// Extend to neighbor/collective communication later
enum MPIX_Channel_Completion_t = {MPIX_CHANNEL_COMPLETE_IMPLICIT,
    MPIX_CHANNEL_COMPLETE_EXPLICIT};

MPIX_Channel_init(buffer, int len, int npart, peer, tag, comm, completiontype, info, &
    channelhandle, &requesthandle);

// One-sided channel operations similar to active target.
// Put/Get implicit complete occurs on on all local partition put/gets completing.
// Ready implicit completes occurs on all *remote* partitions completing.
MPIX_Channel_put_init(channelhandle, &operationhandle, &inithandle);
MPIX_Channel_get_init(channelhandle, &operationhandle, &inithandle);
MPIX_Channel_ready_init(channelhandle, &operationhandle, &inithandle);

// Two-sided channel operations similar to partitioned send/recv. Implicit complete
// occurs on all send/recv completing locally.
MPIX_Channel_send_init(channelhandle, &operationhandle, &inithandle);
MPIX_Channel_recv_init(channelhandle, &operationhandle, &inithandle);

// Per-partition operations - kernel-callable. Will need to add a peer
// argument for a later neighbor collective version
MPIX_Pready(operationhandle, int part);
MPIX_Pcompleted(operationhandle, int part);
MPIX_Preset(operationhandle, int part);

// Interface for explicitly closing a channel operation - need to specify
// the local/remote semantics of this.
MPIX_Channel_complete_init(operationhandle, &completehandle);
```

Questions?

On-Node and Off-Node Parallel Programming Integration: Kokkos and MPI

Evan Drake Suggs
Anthony Skjellum

Collaborators at Sandia:
Jan Ciesko
Stephen L. Olivier

Center for Understandable, Performant Exascale Communication Systems

Tennessee TECH

Introduction

- The Kokkos programming system provides in-memory advanced data structures, concurrency, and algorithms to support advanced C++ parallel programming
- MPI provides a widely used message passing model for inter-node communication.
- This work improves integration of MPI and Kokkos with performance and productivity benefits for exascale applications and libraries.
- Present strategy: Kokkos objects can be passed directly to new MPI API functions.
- Already could be used to enhance Trilinos and Cabana-based applications, etc.
- Work has motivated greater attention for Kokkos+MPI integration.

MPI_Kokkos_Send & Recv Implementation

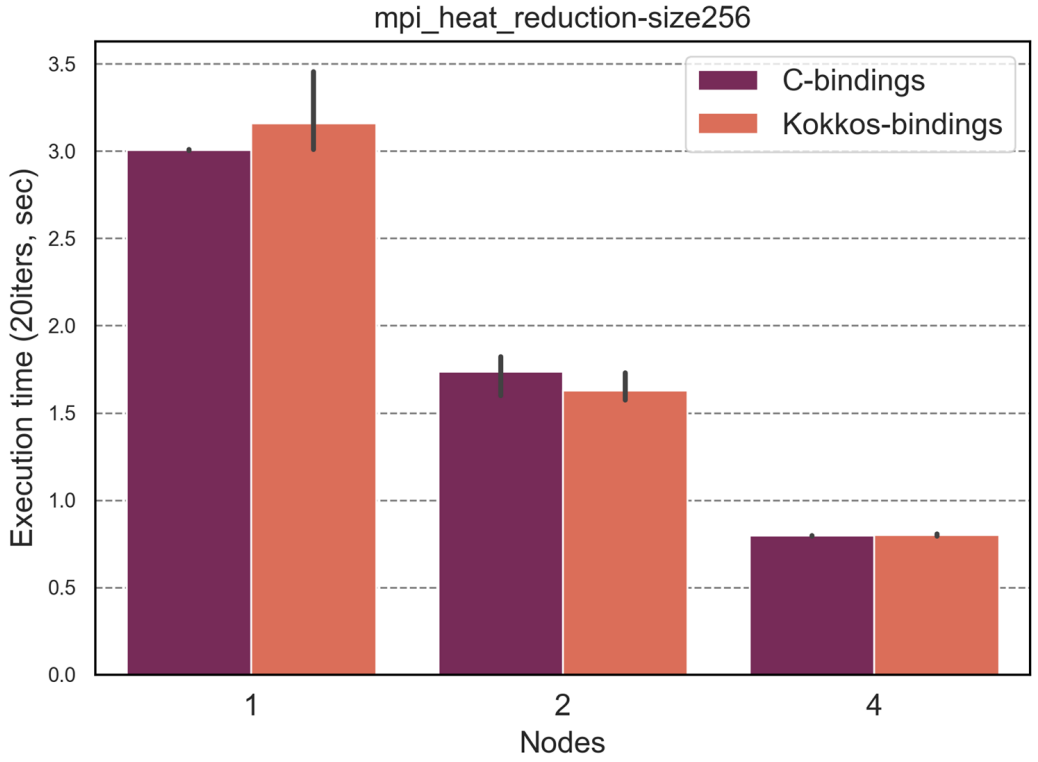
MPI_Send(View_t * buf, int dest, int tag, MPI Comm comm)

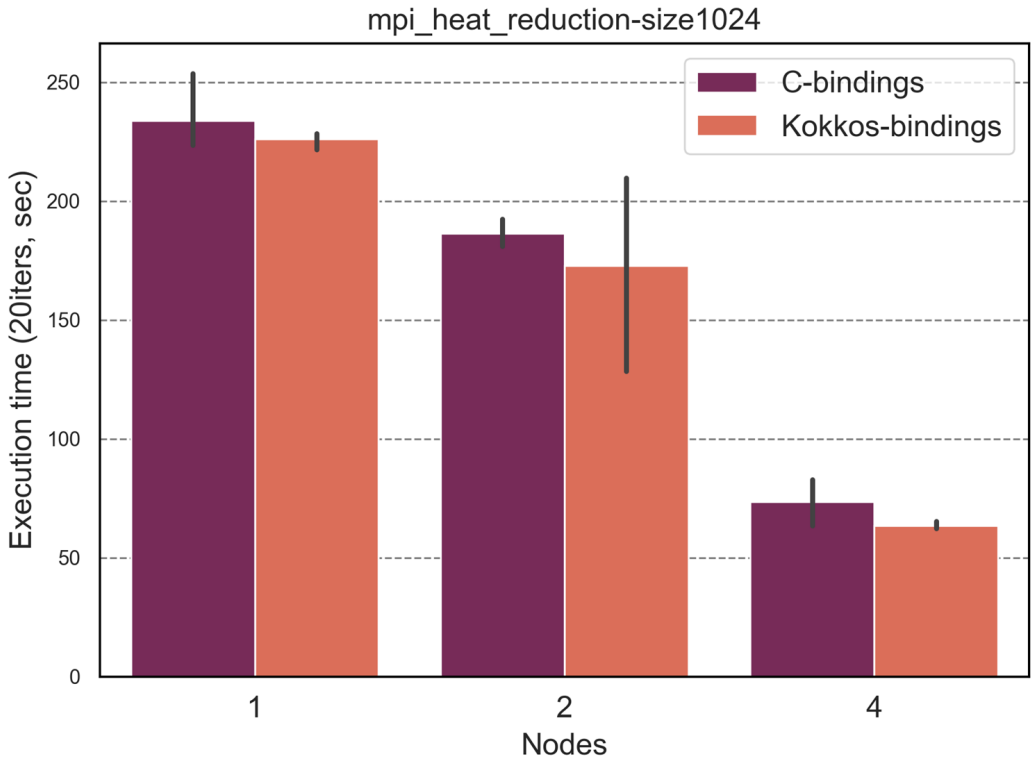
MPI_Recv(View_t * buf, int source, int tag, MPI Comm comm)

```
// How people used to program MPI+Kokkos
        Kokkos::View<int**> recv_check(recv_buf, n, n);
        MPI_Recv(recv_check.data(), n*n, MPI_INT, 1, tag, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
// new method - object aware and polymorphic (C++ with C look)
        Kokkos::View<double**> A("New Method View", n, n);
        MPI_Recv(A, 1, tag, MPI_COMM_WORLD);
// next method - subviews
            MPI_Send(subview(A, make_pair(2, 5), make_pair(4, 6)), 1, tag,
            MPI_COMM_WORLD);
            MPI_Recv(subview(A, make_pair(6, 9), make_pair(1, 3)), 0, tag,
            MPI_COMM_WORLD);
```

Heat Reduction Tests

- This test uses the heat reduction code in Kokkos tutorials comparing new bindings with traditional methods
- The Kokkos-bindings have a slightly better execution time, especially for the increased 1,024 size





Summary

- This work integrates two programming models, MPI and Kokkos.
  - Immediate Productivity Goals
  - Working toward performance enhancements next
- Implemented several MPI bindings with Kokkos View objects as their primary buffers without sacrificing the C++ nature of the Kokkos View.
- Using ExaMPI benefitted the project, since it enables the use of templates to better interact with Kokkos. This direction can lead to greater performance.
- New APIs performed similarly to the original APIs.
- New APIs open avenues for greater performance integration within an MPI implementation.

Next Steps

- Near term
  - Co-developing with Sandia an MPI-Kokkos interop for standard MPI (wrappers)
  - New NCCL-based GPU performance work and device-specific support (MPI_Send<View, class, Device>). [Nicole Avans, internship]
- Medium term
  - Transparently use partitioned and persistent communication where possible, pipelined communication, as well as (de)serialization concepts
  - Testbed for new functions, such as byte-mapping-based transports.
- Longer term
  - A creation of new backends to increase speed for specific types of Views (Views on GPUs, non-contiguous, etc.).
  - This work has impact on C++23 and a new C++ MPI language interface for MPI-5+.

Questions?

Low-level API Design
(bottom-up approach)

Derek Schafer - UNM

April 11th, 2024

Low-level API Development

- Wide range low-level APIs for programming high speed fabrics
  - All mutually incompatible, many vendor-specific
  - Many don't support all features of modern NICs (triggering, network collectives)
  - Hard to use for developers of performance portability libraries
- MPI0 (appropriately renamed) should provide an easier-to-use interface that provides access to all modern NIC features
- Current work:
  - Understand NIC features and their interfaces, particularly for GPUs
  - Enumerate principles and needed features of API (Thomas Hines's MPI0 work)
- Following this: Create an API that makes using these functions easier for the end user

NVIDIA – LibMP and MLX4 libraries

- Lightweight messaging library built on top of LibGDSync (and IB Verbs)
- APIs to support GPUDirect asynchronous communication
  - MPI used to setup IB connections
  - No MPI calls are used for actual communications
  - Uses only point-to-point and one-sided communications (no collectives)
  - No tags, no wildcards, no data types
- LibMP seems to be brittle research-ware
  - Tends to lockup in weird ways
  - Not great at handling unexpected messages
- Talking to Ryan Grant (Queen's University), who has done similar work (but with mlx4 driver)

Libfabric Triggering

- Designed to enable the delaying of an operation until a condition has been met
  - Uses completion counters (that track other operations) or manual counter adjustments from application itself
  - Has mechanisms for triggering from GPUs; vendors still iterating on best approach
- HPE CXI provider support for triggering through **D**eferred **W**ork **Q**ueue libfabric abstraction
  - Extended version of triggering APIs designed for application-level collectives
  - Support a variety of operations (messages, tagged, rma)
- Creating DWQ objects is fairly straightforward, though providers may have limitations
- If anyone wants to talk code, we have code to show :)

Where we are now

- Working with HPE
  - Trying to get running on test systems (Chicoma, Tioga)
  - Frontier access would be helpful!
- CXI libfabric provider source is public on GitHub (with CXI extensions)
  - HPE GPU Transport Layer and *MPIX_* libraries source code is not available
  - But still helps to understand their papers and models
  - E.g. what kind of host progress is needed for *MPIX_Enqueue_recv()*
- Current: Use these operations to implement the higher-level stream and kernel triggering APIs described earlier
- Follow-up: Collaborate with Ryan Grant to design performance-portable interfaces for GPU triggering

Any Questions?

Thank you!