Regular Halo Applications

Tony Skjellum, UTC

August 23-24, 2021

Overview

- Halo applications current/traditional use of MPI
- What's changed?
- Overlap of Communication and Computation
- Persistent and Partitioned Comms
- Assessments/Achievements
- Next Steps
- Lifecycle of innovation to production
- Q&A

Halo applications current/traditional use of MPI

- Point-to-point communication
  - Irecvs , Isends, Waitall
  - Earlier: Sendrecv, cart topologies, etc.
  - Various orderings of how these operations are coded
  - Coded for deadlock-free operation
- Data-moving collective communication
  - Alltoall*
  - [Neighbor versions]
  - Nonblocking forms (since MPI-3)
- Reductions
  - Global Allreduce / Allgather
  - Neighborhood Reductions / Gathers
  - Nonblocking forms (since MPI-3)

- In regular codes, communication neighbors are static or only slowly changing
- Derived datatypes or pack/unpack
- Originally for multicore nodes + MPI – usually “MPI everywhere”
- Overlap of communication and computation not emphasized

What's changed?

- Accelerator-based architectures
  - more time spent in MPI - upwards of 50% of some application time
  - Complex interactions of accelerator, kernels, CPU, with MPI
  - Low performance

Overlap of communication/computation

- Lab codes have not achieved comm/comp overlap
- Providing a means for overlap is now valuable
- Wasn't valuable enough before:
  - pre-GPU comm percentage – 5% in MPI
  - now ~ 50% of time in MPI
  - now it matters
- Why didn't they overlap?
  - Weak progress MPI? → Didn't matter
- Our goals
  - Take out of hands of application developers,
  - Achieve "what" vs. "how" tradeoff for halo codes via new abstractions
  - Leverage/show value of strong progress in MPI

Rationale/Explanation – Persistent and Partitioned Comms

- Persistent send/receive in MPI-1 – but weak semantics
  - Can optimize derived datatypes/local resources, not typically done
- Full planned transfer modes of persistence were added in MPI-4
  - Persistent Collective Operations
  - Partitioned Point-to-point Operations
- Work well with static, bulk-synchronous codes
- Use cases for applications such as machine learning have been identified too

Persistent Collectives

- Stages

```
MPI_Allreduce_init(…,request) – all parameters fixed
MPI_Start(request)
[Opportunity for overlap]
MPI_Wait(request)
```

- Relaxed ordering rules vs. normal collective operations
- They take static arguments
- Applies to communication operations only in MPI-4

Persistent Collectives – Why optimizable

- Select best algorithm (can be expensive, do just once)
- Lock-down static resources (do this once)
- Adapt over time (optionally update strategy internally)
- Elimination of point-to-point receive queues – can be RDMA based
- Semantically aligned with
  - NIC and switch offload of collectives
  - Can choose to offload operations in reductions to accelerators too

Partitioned Point-to-point

- What's a partition?
- Sender: Decides how many partitions it will break its data into

```
MPI_Psend_init(…,request) – all parameters fixed
MPI_Start(request) – moves no data
[Sync step]
MPI_Pready()… each partition
[Opportunity for overlap]
MPI_Wait(request)
```

- Receiver: Decides how many partitions it will break its received data into

```
MPI_Precv_init(…,request) – all parameters fixed
MPI_Start(request)
[Sync step]
MPI_Parrived()… each partition
[Opportunity for overlap]
MPI_Wait(request)
```

Partitioned pt2pt – Why optimizable

- Select best partitioning strategy (can be expensive, do once)
- Match endpoints (do this once)
- Lock-down static resources (do this once)
- Allows multi-threaded/kernel interaction with single large buffers
- Finer-grain overlap revealed by application
- Hides tail-latency of laggard threads; supports “early-bird communication”
- Reduces need for `MPI_THREAD_MULTIPLE` in entire MPI Code
- Elimination of point-to-point receive queues – can be RDMA based
- Semantically aligned with NIC and switch offload of collectives
- Deciding optimal offload semantics for accelerators still being worked on

Persistent/Partitioned Comms Big Picture

- It is straightforward to convert bulk-synchronous codes to use these primitives
- Better interfaces when MPI processes have concurrent producers and consumers of their buffers
- Revealing APIs that eliminate receive queues when optimized
- Enable algorithm selection that's optimal
- Providing accelerator, NIC, and switch-friendly APIs
- Support overlap of communication, communication, and computation
- Begin process of MPI being concurrency-aware inside a process, beyond the MPI_THREAD_* settings
- Help with performance and performance-portability

More work remains for specification, standardization, and demonstration…

- Areas
  - Partitioned Collectives
  - Updateable persistent operations
  - Synchronization/other interactions with accelerator state
- More exploration/exploitation of features in halo codes

Assessments/Achievements

Fiesta Assessment and Performance Improvements

FIESTA

- UNM Fiesta - Kokkos implementation of open capabilities of LANL HIGRAD shock hydrodynamics solver
  - By Romero et al., open source release planned soon
  - Simple 3D relatively large (3 ghost layers, 5 vars) halo exchange - 500^3 per GPU mesh exchanges 30MB messages
- Initial code was clear but not well-optimized - 3D bubble expansion test spent 50% of its time in halo exchange (mainly data packing and copying)
- Conceptually simply optimizations yielded 15-20% performance improvements
  - Overlapping Y/Z direction packing with the communication in the previous direction – 8%
  - CUDA-aware MPI when available – 8%
  - Doing this was not as trivial as it would seem – challenges with MPI, Kokkos, and C++ interactions
- Neighbor collectives **should** be able to do this
  - Not feasible today due to poor implementation of MPI datatypes on GPUs
  - Trying to use MPI datatypes resulted in a >1000% slowdown!
- Performance study details later on the poster by Ryan Goodner

Lots of possible new and modified abstractions now possible, I

- Persistent/partitioned point-to-point communication (MPI-4)
- Persistent collective operations (MPI-4)
- Partitioned collective operations (MPI-5)
  - Persistent
  - Neighbor
  - Initiation and Completion semantic options
- Newer possible abstractions that address data transfers at a higher level
- All persistent operations have the potential for reducing derived datatype “costs”
- Modifiable persistent and partitioned operations with reduced overhead—MPI-5

Lots of possible new and modified abstractions now possible, II

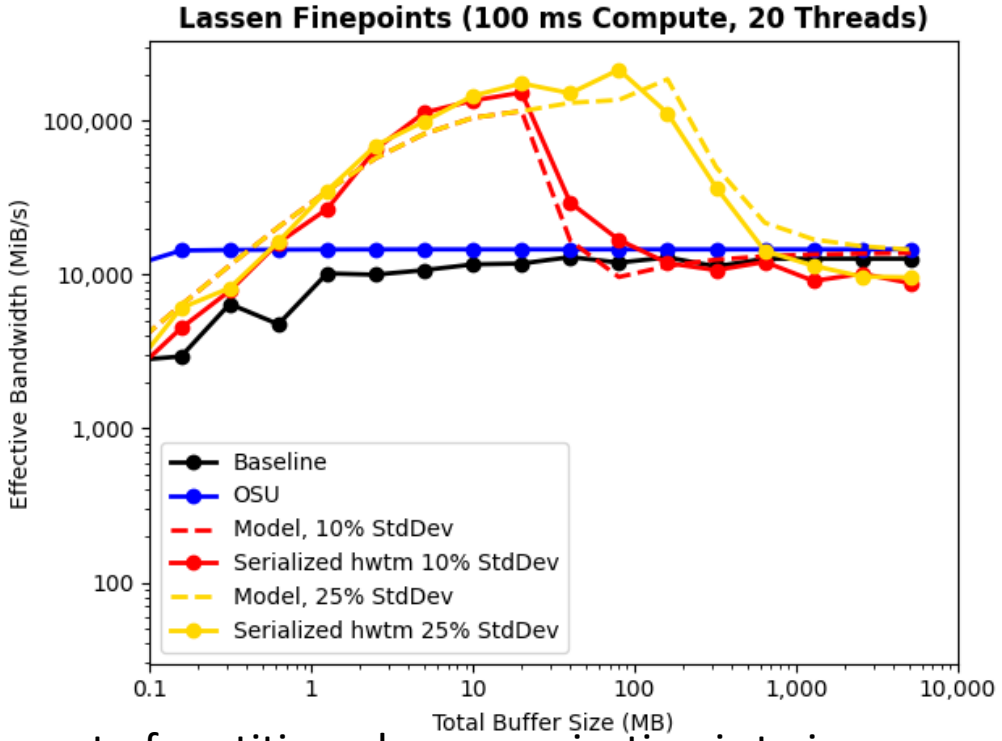
- Higher-level primitives may be useful such as alternative ways to describe data layouts (replace derived datatypes)
- Direct C++ Language Interface for MPI – enable greater collaboration and optimization with packages like Kokkos/Raja/etc
- REACH: One-sided (RMA) persistence---experimental idea for MPI-5

GPU Data Exchange Benchmarking

- Motivation: Need better tools to understand GPU communication costs
  - Current communication benchmarks either trivial (OSU ping-pong) or fairly complex (Comb)
  - Tying GPU profiling to MPI and applications not always trivial
- Created ping-pong test using Fiesta/Kokkos 4D state array
  - Examine data exchange options on more realistic application data structures
  - Vary size, dimension being communicated as well as MPI communication strategy
- Integrating and testing various NVIDIA profiling tools into communication system
- Encouraging initial results on ability of benchmark to evaluate
  - Costs/benefits of different MPI data exchange strategies
  - Ability of GPU profiling tools to capture communication costs
- More details and results on poster by Keira Haskins from SNL internship (mentors Kurt Ferreira and Scott Levy)

Partitioned Communication Modeling

- Partitioned communication promising for improving performance but has complicated performance tradeoffs
  - Performance change vs. partition size
  - Amount of compute/communicate overlap achievable
- Developed model of partitioned communication performance using model of when threads reach partitioned communication call
- Accurately predicts partitioned communication performance on simple benchmarks on LLNL Lassen system
- Work on extending model to partitioned application on of SNL summer internship
- Additional details in poster presentation by Jered Domingue—Trujillo (SNL mentors: Ryan Grant and Matthew Dosanjh)



Impact of partitioned communication is to increase effective bandwidth though overlap for medium-sized message. Model accurately predicts this when tested on LLNL Lassen cluster

Rationale for why modern primitives are helpful… and achievements thus far

- Persistent communication –- see Gerald's poster
- Partitioned communication has complex performance tradeoffs – see Jered's modeling
- MPI Datatypes issue – Fiesta and Rei's ping pong numbers (poster sessions), and that there are more improvements in the works here
- Partitioned communication – MPIPCL
  - Published papers, useful source code, integration/collaboration with Sandia
  - Component of the *MPI Advance* project now

Next steps

- Combine neighbor collectives and partitioning
  - Collaborations in our center to define use cases and specific APIs
  - MPI Forum WG on Partitioned and Persistent and Collective ops
- Optimize neighbor collective halo exchanges to take into account GPU costs
- Study/tweak interactions of Kokkos+MPI (ex: stream vs. device sync)
- Demonstrate value of asynchronous (strong) progress

Next steps, II

- Optimize neighbor collective halo exchanges to take into account GPU costs
  - A100 experiments, embedding in progress engines
  - Correlating with partitioned offload of kernels
  - Correlating with performance modeling that's been done already
  - Leverage ExaMPI test implementation and MPI Advance prototype
- Testing capabilities of CUDA Graphs, libmp

Lifecycle of concept to standard to production in MPI+X

- Concepts derived from NNSA codes
- Prototypes defined and demonstrated to help
- Close interactions with MPI Standard for future standardization
  - MPI-4 successes: Persistent Collectives, Partitioned Point-to-point
  - On-going: Collective/Partitioned/Persistent/HACC working groups
- MPI Advance supports
  - Credibility, proof of concept, long-haul application support
  - early access (e.g., MPI-5 will publish in 2026-27)
  - Means to demonstrate achievable performance, establish new best practices
  - Baseline code for production MPI implementations

Q&A