Initial Pathfinding Formative Assessment Results

Patrick G. Bridges

Initial Pathfinding Research

- Preliminary work on application communication system requirements
  - Which calls does an application or framework use?
  - Tells us what to optimize to help unchanged applications
  - Described in assessment report
- More recently (since September) – examine the higher-level communication *patterns* used
  - What abstractions ***could*** applications be using?
  - What abstractions ***should*** we be optimizing (or creating)?

Initial Pathfinding Results

- Looked at LLNL Comb, LANL CLAMR, UNM/LANL Fiesta, various ECP frameworks (Cabana, Trilinos) and miniapps
- Discussions with lab staff about additional application needs and higher-level patterns
  - Olga Pearce (LLNL - Comb)
  - James Elliot (SNL – EMPIRE)
  - Bob Robey (LANL – CLAMR/xRage)
  - Galen Shipman (LANL)

Regular Halo Issues

- Optimization opportunities with multiple ranks per node
  - Coalescing messages to the same destination with other ranks on same node
  - Choosing the order in which to send messages
- Multi-threading and regular exchange patterns common
  - Allow threads to send and receive messages independently to reduce coupling
  - Leverage static communication structure and type to reduce setup costs
- GPU data movement very challenging
  - High overheads for packing data on the GPU
  - MPI Datatype packing often has horrendous performance
  - Complex tradeoffs in packing more buffers versus reducing pack/send/recv/unpack overlap
- HIGRAD, Fiesta, Comb all face these issues
- Comb captures GPU tradeoff issues really well

Irregular Halo Issues

- Application-level array scatter/gather is ubiquitous
  - Distributor and Halo classes in Cabana
  - Distributor class in Trilinos/Tpetra
  - L7 library in CLAMR (similar to xRage token library)
- Bulk synchronous application (and MPI!) abstractions
  - Tightly couples multiple independent threads
  - Caused by lack of good threaded communication primitives
  - Combining messages with neighbor collectives could worsen
  - Exacerbates imbalance problems in some important cases (MueLu solves in EMPIRE)

Lots of MPI abstractions and research on these issues

- Halo exchange optimization – neighbor collectives
  - Benefit: MPI can coalesce message exchanges between ranks
  - Cost: Reduces compute/communicate overlap between messages in the halo (all given to collective at once!)
  - Lots of benefits in complex halos, benefits less clear in more straightforward (e.g. 9 or 27 point) halos
- Persistent, partitioned communication
  - Set up communication paths once (e.g. the send or the collective, the datatypes being exchanged)
  - Give send or receive buffer to MPI when available
  - MPI can decide whether to send immediately or wait for more data
- GPU datatype implementation optimizations

How useful are these abstractions?

- CLAMR uses the L7 library to scatter/gather halo cells in its (1D) mesh array with neighbors
  - `L7_Setup/Push_Setup` constructs communication plan
  - `L7_Update/Push_Update` send and receives needed cells between neighbor processes
  - Multiple calls to update (with different types) per setup
  - One cell can be sent to multiple neighbors, sent data not necessarily contiguous but received data is contiguous
- Current `L7_Update` implementation
  - Manually packs send buffers to each neighbor
  - Irecv/Isend/Waitall for all incoming and outgoing messages
  - Very similar to Cabana distributor code we've already seen

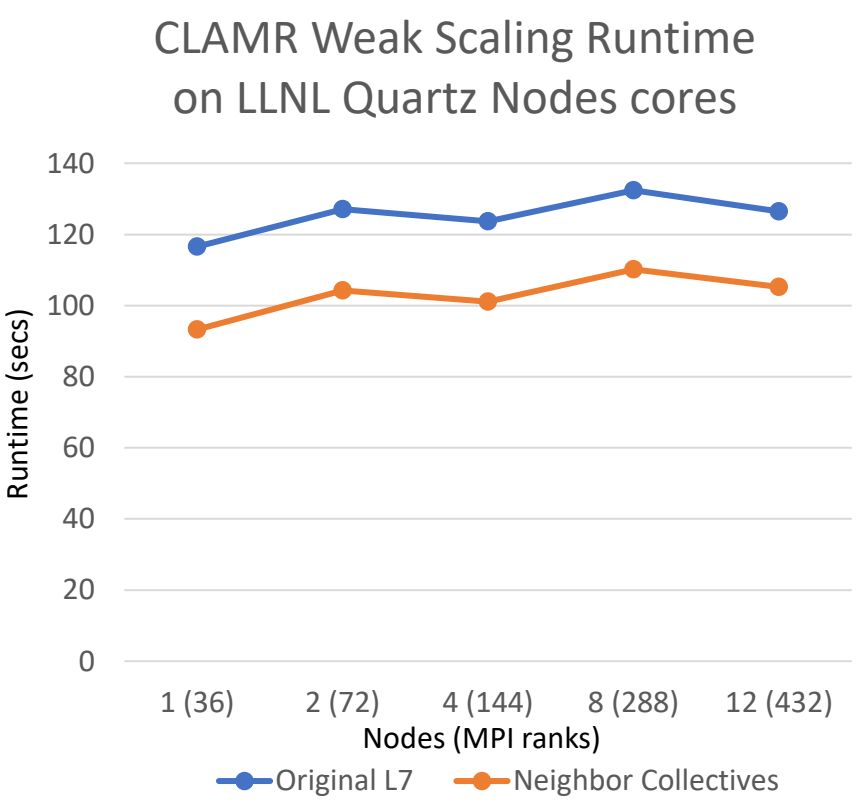
Neighbor Collective CLAMR/L7

- Goal: Initial evaluation, research platform
  - How well do neighbor collectives map to the array scatter/gather communication abstraction used in L7, Cabana, and Trilinos?
  - Evaluate performance of existing MPI implementations
  - Have an initial platform for testing additional optimizations
- Null hypotheses:
  - Easy to implement given existing information
  - Performance somewhat worse than L7 implementation but not catastrophically bad
    - Current neighbor collective implementations are naïve (basically just the `isend/irecv` loop L7 already has)
    - Leans heavily on derived datatypes, but may potentially get rid of unnecessary copies in the original application communication plan

L7_Update with Neighbor Collectives (pseudocode)

```
L7_Update(void *data_buffer, int type) {
        ierr = MPI_Neighbor_alltoallw((void *)data_buffer,
                        l7_id_db->mpi_send_counts,
                        l7_id_db->mpi_send_offsets,
                        update_datatype->out_types,
                        (void *)data_buffer,
                        l7_id_db->mpi_recv_counts,
                        l7_id_db->mpi_recv_offsets,
                        update_datatype->in_types,
                        l7_id_db->comm);
        return ierr;
}
```

Neighbor Collectives Performance in the CLAMR L7 Library



- LLNL Quartz cluster nodes
- Weak scaled 1 node test case:
  - 2048x2048 coarse mesh
  - Max 2 levels of refinement
  - MPI only – 36 ranks per node
  - MVAPICH2/2.3, Intel compiler
- Unexpected 20% across the board *performance improvement* on Quartz

Quartz Performance Improvement

- Original L7:
  - L7_Push_Update very slow on Quartz – need to understand why
  - Perhaps cache/NUMA problems in the face of imbalance

```
CPU: Device compute           time was  126.4586    126.4861    126.5058   s min/median/max
CPU:   state_timer_finite_difference    15.1772     15.2837     15.7394   s min/median/max
CPU:   mesh_timer_calc_neighbors        90.9144     90.9344     90.9453   s min/median/max
CPU:     mesh_timer_push_boundary        0.4344      3.4630     23.9418   s min/median/max
CPU:     mesh_timer_setup_comm          16.9817     39.3236     57.0550   s min/median/max
CPU:   mesh_timer_load_balance           7.9643     11.3517     11.6579   s min/median/max
```

- Neighbor Collective L7:

```
CPU: Device compute           time was  105.1840    105.2063    105.2242   s min/median/max
CPU:   state_timer_finite_difference    15.1898     15.3055     15.7579   s min/median/max
CPU:   mesh_timer_calc_neighbors        69.0902     69.1312     69.1685   s min/median/max
CPU:     mesh_timer_push_boundary        0.3659      0.7189      0.9569   s min/median/max
CPU:     mesh_timer_setup_comm           6.4681     19.4915     33.7506   s min/median/max
CPU:   mesh_timer_load_balance           8.1205     11.8475     12.1148   s min/median/max
```

But initial results from Lassen are completely different!

- Our first runs on Lassen with Spectrum MPI show 20%+ *worse CLAMR performance!*
  - Original L7_Push_Update didn't take significant time
  - Neighbor collective versions slower under Spectrum MPI
  - Lassen has more memory, caches than Quartz
- Slowdowns likely due to the datatypes implementation
  - Good research on datatype engines from mpich, mvapich, and openmpi teams
  - Demonstrations of datatypes optimization value important to push vendors to adopt these optimizations
- This fits our original null hypothesis – worse but not terrible
- These wild performance swings and unpredictable performance are exactly the problem we seek to address

Initial Neighbor Collective Assessment Wrapup

- Graph topology was easy to create - L7 already had the data needed
- Initial performance results as expected, need to dig in more
  - Reasonable performance with a solid datatype implementation
  - Anomalous performance on Quartz worth examining to understand tradeoffs
- Concern: getting the datatypes right was complicated
  - Neighbor calls require byte offsets, not type offsets
  - Each source and destination pair had its own datatype
  - Microbenchmark performance varied noticeably between different MPI implementations
  - Reliance on datatypes a possible performance pitfall
- Additional neighbor collective advantages
  - Original L7_Update didn't work on GPU-resident buffers
  - Switching to in-place MPI-based send/receive *should* allow it to work directly using a CUDA-aware MPI (not yet tested)
  - Derived datatypes support working with complex application data types – most irregular halo abstractions don't support this

Next Assessment Step Plans

- Three relevant abstractions (in addition to datatype infrastructure)
  - Neighbor communication – All halo exchanges, but potential tradeoffs
  - Persistent communication - Mostly static communication patterns (not CLAMR/xRage)
  - Partitioned communication – Halos on heavily threaded and GPU systems
- Neighbor collectives and Irregular Halos
  - Understand and optimize what's going on in CLAMR/L7
  - Test GPU Update performance
  - Implementing same strategy in Cabana or Trilonos would provide more diverse testcases
  - Integrate with persistent communication to allow more optimization when communication pattern is static
  - Examine message scheduling optimizations in neighbor communciations [Ghazimirsaeed 2019], AMG codes [Bienz 2019]

Next Assessment Step Plans (Cont'd)

- Neighbor Communication and Regular Halos
  - Is more effective message scheduling and handling (e.g. packing/unpacking/pipelining) worth it with simpler exchanges?
  - Partitioned communication and GPUs an attractive starting point in regular halos, with later combination with neighbor communication
- These Applications
  - CLAMR – Non-persistent Neighbor Collectives
  - Cabana/Trilinos – Persistent Neighbor Collectives, GPU support
  - HIGRAD/Fiesta – Partitioned Communication (Threads, then GPUs)
  - Comb
    - Both neighbor communication and partitioned communication are options
    - Persistent neighbor collectives from GPUs is the current thinking

Questions?