Understanding MPI+X GPU Triggering APIs

(and some initial thoughts on next steps)

Patrick G. Bridges (UNM) , Anthony Skjellum (TTU),
Evan Suggs (TTU), Derek Shafer (UNM), and Puri Bangalore (UA)

CUP ECS Center for Understandable, Performant Exascale Communication Systems

THE UNIVERSITY OF NEW MEXICO

The community needs to start converging around MPI+X triggering interfaces

- Community Issue: How to connect MPI with ”+X” asynchronous computation APIs
- Done by a “triggering” interface
  - Expensive setup/synch done on host MPI
  - Queue or atomic update for MPI to +X coupling
  - Limits the complexity of the MPI to +X interface
- Lot of complicating issues, for example:
  - Diversity of “+X” environments (threads, GPUs, task-based parallelism, etc.)
  - MPI specification ambiguity and complicating historical decisions (e.g. persistence)
  - Proposals addressing other issues that are overloaded to address triggering in addition
- Community step 0: Where are we currently?

Understanding the existing proposals

- Identified 9 different MPIX triggering proposals
  - Includes both formal (e.g. papers) and informal (e.g. online discussions)
  - Some proposals included distinct triggering APIs that we studied as separate proposals
  - Successive proposals around same interface were aggregated into a single proposal
  - Don't claim to have found them all
- Step 1: Understand how to implement simple communication patterns in each of them
- Step 2: Identify categories with similarities, differences, and recurring issues
- Step 3: Classify proposals in categories
- Note: Didn't examine/compare with non-MPI triggered communication interfaces

List of implementations:

- MPI-GDS: stream-triggered send/recv interface for MVAPICH
- MCI-ACX Enqueued: stream-triggered portion of MPI-ACX
- MPICH Triggering: stream-triggering interface for MPICH
- HPE Send/Recv: stream-triggered send/recv interface for slingshot
- Project Delorean: Graph-sequenced, stream-triggered MPI
- HPE One-sided: stream-triggered PSCW interface for slingshot
- Partitioned Communication: aggregation of MPI partitioned communication with `Pbuf_prepare` and partitioned collectives proposals; partially implemented by MPI-ACX
- HPE Persistent: kernel-triggered interface for HPE slingshot
- Intel GPU-Initiated: GPU-initiated interface for Intel MPI from Intel documentation and communications with Dan Holmes

Example Implementation: MPICH Streams

```
MPIX_Stream mpistream;
MPI_Comm stream_comm;
MPI_Info info;

MPI_Info_create(&info);
MPI_Info_set(info, "type", "cudaStream_t");
MPIX_Info_set_hex(info, "value", &cudastream,
                    sizeof(cudastream));
MPIX_Stream_create(info, &mpistream);
MPI_Info_free(&info);
MPIX_Stream_comm_create(MPI_COMM_WORLD, mpistream,
                              &stream_comm);
```

```
for (int i = 0; i < niters; i++) {
    if (my_rank == 0) {
      MPIX_Send_enqueue(src_buf, 1, MPI_INT, 1, 123,
                        stream_comm);
      MPIX_Recv_enqueue(src_buf, 1, MPI_INT, 1, 123,
                        stream_comm, MPI_STATUS_IGNORE);
    } else if (my_rank == 1) {
      MPIX_Recv_enqueue(dst_buf, 1, MPI_INT, 0, 123,
                        stream_comm, MPI_STATUS_IGNORE);
      MPIX_Send_enqueue(dst_buf, 1, MPI_INT, 0, 123,
                        stream_comm);
    }
  }
  cudaStreamSynchronize(cudastream);
```

Example Implementation: MPICH Streams

```
MPIX_Stream mpistream;
MPI_Comm stream_comm;
MPI_Info info;

MPI_Info_create(&info);
MPI_Info_set(info, "type", "cudaStream_t");
MPIX_Info_set_hex(info, "value", &cudastream,
                  sizeof(cudastream));
MPIX_Stream_create(info, &mpistream);
MPI_Info_free(&info);
MPIX_Stream_comm_create(MPI_COMM_WORLD, mpistream,
                        &stream_comm);
```

```
for (int i = 0; i < niters; i++) {
    if (my_rank == 0) {
      MPIX_Send_enqueue(src_buf, 1, MPI_INT, 1, 123,
                        stream_comm);
      MPIX_Recv_enqueue(src_buf, 1, MPI_INT, 1, 123,
                        stream_comm, MPI_STATUS_IGNORE);
    } else if (my_rank == 1) {
      MPIX_Recv_enqueue(dst_buf, 1, MPI_INT, 0, 123,
                        stream_comm, MPI_STATUS_IGNORE);
      MPIX_Send_enqueue(dst_buf, 1, MPI_INT, 0, 123,
                        stream_comm);
    }
  }
  cudaStreamSynchronize(cudastream);
```

New calls/abstractions (stream communicators, send/recv_enqueue)
New semantics (MPI Streams) but clearly defined in terms of MPI concurrency model
Requires MPI matching in the data movement path

Example Implementation: HPE One-Sided

```
/* Normal MPI communicator, window, group, and  */
 /* buffers assumed to exist  */
 for(int i = 0; i < niters; i++) {
   if(rank == 0){
     /* Send ping */
     MPI_Win_start(group, MPI_MODE_STREAM, win);
     MPI_Put(src,n,MPI_INT,1, disp, n, MPI_INT, win);
     /* Puts triggered here */
     MPIX_Win_complete_stream(win,stream);
     /* Receive pong */
     MPIX_Win_post_stream(group, win, stream);
     MPIX_Win_wait_stream(win,stream);
   }
   else { /* Receive ping */
     MPIX_Win_post_stream(group, win, stream);
     MPIX_Win_wait_stream(win, stream);
     /* Send pong */
     MPI_Win_start(group, MPI_MODE_STREAM, win);
     MPI_Put(src, n, MPI_INT, 0, disp, n,
             MPI_INT,win);
     /* Puts triggered here */
     MPIX_Win_complete_stream(win, stream);
   }
 }
 cudaStreamSynchronize(stream);
```

Example Implementation: HPE One-Sided

```
/* Normal MPI communicator, window, group, and  */
 /* buffers assumed to exist  */
 for(int i = 0; i < niters; i++) {
   if(rank == 0){
     /* Send ping */
     MPI_Win_start(group, MPI_MODE_STREAM, win);
     MPI_Put(src,n,MPI_INT,1, disp, n, MPI_INT, win);
     /* Puts triggered here */
     MPIX_Win_complete_stream(win,stream);
     /* Receive pong */
     MPIX_Win_post_stream(group, win, stream);
     MPIX_Win_wait_stream(win,stream);
   }
```

```
else { /* Receive ping */
     MPIX_Win_post_stream(group, win, stream);
     MPIX_Win_wait_stream(win, stream);
     /* Send pong */
     MPI_Win_start(group, MPI_MODE_STREAM, win);
     MPI_Put(src, n, MPI_INT, 0, disp, n,
             MPI_INT,win);
     /* Puts triggered here */
     MPIX_Win_complete_stream(win, stream);
   }
 }
 cudaStreamSynchronize(stream);
```

Mix of new and existing abstractions, reusing RMA windows for stream triggering
N need for complex matching during data movement – just put/get
Slightly different (strenghthened!) RMA semantics

Many other interesting variants!

- MVAPICH version that changed the semantics of MPI_Send
- Partitioned Communication Kernel Triggering
- Intel’s SYCL-oriented Kernel Triggering
- and even more… (please read the paper!)

Identified Categories

- Area 1: GPU control path used: Stream or Kernel
- Area 2: API Design Considerations
  - Reuses Existing MPI APIs or abstractions: Yes or No
  - Changes Existing MPI API Semantics: No, Strengthens, or Weakens
  - Separate MPI Operation Initialization and Starting: Yes or No
  - GPU MPI Operation Completion Support: All, Some, or None
  - Collective Communication Support: Full, Partial, Group, or None
- Area 3: Ordering and Concurrency Considerations
  - MPI Operation Sequencing Abstraction: Yes or No
  - Sequencing Abstraction Semantics: Full or Partial
  - MPI Concurrency Standard Integration: Explicit, Implicit, or Unspecified
- Area 4: Implementation Considerations
  - GPU/NIC Progress: Yes or No
  - Available Implementation or Evaluation: Yes or No
  - Multi-architecture Support: Yes, GPU, NIC, or No

The full classification of 9 triggering proposals in these 12 categories is in the paper

Category Highlights

- Area 1: GPU control path used: Stream or Kernel
- Area 2: API Design Considerations
  - **Reuses Existing MPI APIs or abstractions:**
    - HPE One-side, Partitioning, MPI-GDS: Yes
    - HPE Two-sided, MPICH: No
  - **Separate Operation Initialization and Starting**
    - MPICH, MPI-GS, Intel: No
    - Most others: Yes
  - Note: Most stream-triggered APIs rely on +X fence on enqueued waits; legality of host wait on stream-triggered request often not clear
- Area 3: Ordering and Concurrency Considerations
  - **MPI Operation Sequencing Abstraction**
    - HPE Two-sided, MPICH: Yes
    - HPE One-sided, Most others: No
  - **MPI Concurrency Standard Integration: Only MPICH provides additional clarity**
- Area 4: Implementation Considerations
  - **GPU/NIC Progress: Yes or No**
    - HPE One-sided, Partitioning: Yes
    - HPE Two-sided, most others: No

The full classification of 9 triggering proposals in these 12 categories is in the paper.

Key Gaps/Takeaways/Insights

From paper:

1. Persistent operation semantics cause proposals to re-specifying existing MPI operations (e.g. MPIX_Enqueue_send)
2. Need a consistent way to cause or assert that remote partners are ready for communication
3. Almost all APIs lack a clear, well-specified concurrency model between triggered operations and triggered/host operations
4. Limited support for GPU-triggered MPI collective communication (traditional or neighbor)
5. Completion checking of kernel-triggered operations highly dependent on features of the programming environment

Additional thoughts:

1. Different people mean different things by “GPU triggering” – API features vs. implementation features
2. Need exemplar triggered communication application abstractions and benchmarks to make sure APIs are usable
3. The detailed semantics of these APIs matter for both programmability and performance and are still often not well-defined
4. APIs must take into account *shared* capabilities of current *and future* network interfaces and programming environments

Next Steps/Future Work

- Collect and curate documentation and source code on stream triggered APIs, prototypes, and supporting fabric interfaces
- Identify application and framework APIs (e.g. Trilinos::Distributor Cabana::Halo, Kokkos::Comm) to drive MPI-level APIs
- Propose alternatives and converge on stream-triggered APIs that leverage lessons learned from existing proposals

If I were to design a triggering API…

Philosophy:

1. Focus on a two-sided stream triggering API that still allows one-sided data movement
2. Enqueuing starts and waits on operations is a pretty natural API (e.g. MPI-ACX, MPICH, HPE Two-sided)
3. Specify the operations to enqueue by building on/fixing persistence whenever possible
4. Leverage existing ideas to enable one-sided data movement for two-sides operations (e.g. ”prepare”)
5. Include neighbor collectives from the beginning to enable aggregation of trigger/synch overheads
6. Define in tandem with development of C++ abstractions, benchmarks, and proxies for real networking hardware

Current Draft:

1. Provide a local (not communicator-wide) queue/stream abstraction
   - `MPI_Enqueue_start(req)`, `MPI_Enqueue_startall(reqs)` `MPI_Enqueue_waitall(reqs, statuses)`
   - Define specific concurrency semantics similar to MPIX_Stream
2. Enable persistence for enqueueing by providing matching two-sided persistent operations *prior* to `MPI_Start`:
   - `MPI_Match(req)`, `MPI_Imatch(req, &req2)`
   - Once per initialization, meant for host-based operation
3. Generalize MPI_Pbuf_prepare for persistent requests to guarantee one-sided data movement (RTS/CTS)
   - `MPI_Prepare(req)`, `MPI_Enqueue_prepare(req)`
   - Called or enqueued prior to enqueuing a two-sided comm. req
   - Considering ways to assert a request is already prepared

Currently working out details for Cabana/Kokkos regular and irregular halo exchanges with libfabric triggered operations

Acknowledgements

- Research Collaborations and Discussions: Amanda Beinz (UNM), Matthew Dosanjh (Sandia), Ryan Grant (Queen's University), Carl Pearson (Sandia)
- Stream Triggering Authors, Designers, and Implementers
  - Dan Holmes (Intel)
  - Hui Zhou (Argonne)
  - Jim Dinan (NVIDIA)
  - Larry Kaplan (HPE)
- MPI Hybrid Working Group members
- Anonymous paper reviewers
- Funding: U.S. DOE PSAAP-III Award (DE-NA0003966)

Questions?

Email: patrickb@unm.edu