# MPI Forum HACC Working Group

September 28, 2022

# Agenda

(1 hour) Review proposals for integrating MPI communication with accelerators

(1) MPIX_Queue as a new MPI object (Naveen Ravichandrasekaran)
(2) MPIX_Queue as arguments to a function (Jim Dinan)
(3) MPIX_Stream (Hui Zhou)
(4) MPI Streams (Quincey Koziol)

(15 min) Review questions captured and pro/con
(15 min) Discuss approach we can use to reach a common proposal
Common evaluation criteria (e.g. implementation targets, application usage models, etc.)

# Proposals for Accelerator-Integrated MPI

## MPIX_Queue as a new MPI object (Naveen Ravichandrasekaran)

Summary:

- Add MPIX_Queue object that is associated with a GPU stream
  - There are two queues: One queue that belongs to the MPI library and another queue that belongs to the NIC
- Add MPIX_Enqueue_send/recv to append operations to a queue
- Add MPIX_Enqueue_start/wait to start all operations in the queue and wait for started operations to complete
- https://arxiv.org/pdf/2208.04817.pdf

Pro:

- Efficient hardware offloading:
  - Mapping to event counters and triggered operations, allows one counter to be used per stream
- Can batch multiple operations together
  - This reduces control overheads between CPU and GPU, as well as between GPU and NIC

- Allows implementation to handle intranode and internode communication separately
    - Important for HW offload

Con:

- Intranode performance is challenging because it requires a CPU progress thread to perform matching, etc.
    - Lose latency benefits of GPU integration

Questions:

- Is intranode vs internode management supposed to be made easier at the implementation level or at the application level? In the latter case, it's possible to use MPI shmem.
    - Can we get more specifics on the "MPI shmem"?
        - [GM] Yes: you can create a shared memory window and then perform load/sqtores in said window, thus bypassing the whole send/recv MPI stack. That, of course, is for application developers, not for MPI implementers.
- Function name changes to match MPI traditions:
    - MPI_Enqueue_send -> MPI_Enqueue_send_init
    - MPI_Enqueue_start -> MPI_Enqueue_start_all
    - MPI_Enqueue_wait -> MPI_Enqueue_wait_all
- Consider a sequence like this:
  (1) MPIX_Enqueue_send, (2) MPIX_Enqueue_start, (3) MPIX_Enqueue_recv, (4) MPIX_Enqueue_start, (5) MPIX_Enqueue_wait
    - Is this allowed? Does the wait operation wait on both send and recv?
      Answer: Yes this sequence is supported.
- Possible to extend API to include collectives/RMA triggered from GPU?
  Answer: Yes, we could also expand Stream-awareness to collectives and RMA operations. Specifically, the RMA operations seem like a better choice to avoid the message matching issues associated with the P2P operations associated with the intranode performance issue mentioned in the cons.

Notes:

# MPIX_Queue as arguments to a function (Jim Dinan)

Summary:

- Similar approach to previous but without explicit queue object, queues passed instead directly to Enqueue_recv/Enqueue_send
-

Pro:

- Includes both nonblocking and persistent communication APIs
- Fits with partitioned communication (MPI_Pready on device)
- Fits with future proposal for MPI_Prepare

Con:

- Assumes in-order stream (true for CUDA, not for SYCL)
    - [JD] The ordering model is defined by the external work scheduling model (FIFO for CUDA streams). We should write out an example using MPI_SYCL_QUEUE to show how we can support its ordering model.

Questions:

- Batching with startall/waitall doesn't seem to work for the requirement [Naveen]
    - We need batching info during operation enqueuing - meaning during send_int
    - Not during control path
    - Need more discussion
- How are unexpected messages handled? Do we always require a progress thread to handle the message matching?
- [DanH] MPI_Info keys might be better expressed as MPI_Channel? Partitioned communication already specifies this full channel semantic. What is the difference between MPI_Channel and MPI_Queue from the previous option? (Dan Holmes)
    - Idea is to have persistent communication operations that provide a full channel of communication. This has been discussed in the past in the context of the persistence WG and there might be another consumer of such a feature in solving problems HACC is working on.

## MPIX_Stream (Hui Zhou)

Summary:

- MPI_Stream captures generalized "serial execution context"
    - E.g. a thread, a CUDA stream, SYCL queue, etc.
- Add MPIX_Stream_comm_create to create a communicator whose work is performed with respect to a local stream
- Add MPIX_*_enqueue operations (send, recv, isend, irecv, wait, etc.) to enqueue operations on the stream communicator

Pro:

- Provides more than just accelerator support, also provides thread-isolation for non-GPU setups
- Extends MPI communicators, which can provide a familiar/convenient API for users
- Can allow streams to map to separate network endpoints (VCs) at both source and destination

- Orthogonal concept to queue or graph and is complementary to those proposals

Con:

- Binary value encoded as string into MPI_Info value.
- Enqueueing nonlocal MPI operations onto a device, forces the device to poll for completion, i.e. the device must do full MPI progress or delegate back to the CPU. [Applies to all three options so far, except partitioned communication.]
    - Communication is non-local in nature, right? The MPI_Wait is the synchronization for that.
    - [DanH] in a bulk synchronous algorithm, sure: everything needs to be blocked until the synchronization is done. There are other patterns of communication, though. Often, I'd rather have the CPU polling for completion (because it wasn't doing intense computation anyway) than a GPU (which has to stop computing to poll).

Questions:

- Is MPI_Stream intended to be scoped with one MPI session?
  Yes. It shares the same as all the other local MPI objects
- We should have thread_block and warp (work group and work sub-group for SYCL) versions of all these communication_enqueue functions, right? [Applies to all three options so far.]
  [HZ] Is it possible to annotate those with info hints to MPIX_stream_create?
    - [DanH] If the enqueue procedure results in a kernel being submitted to the device, then we probably need to say how big that kernel is – how many threads/groups/warps/etc. If a previous kernel submitted by the user produced the data that will be sent (or a later kernel that will be submitted by the user will consume the data that will be received) then it might make sense to have the enqueued kernel match the size of the user kernel, but it might not. It probably does not make sense for it to request one GPU thread. How will the enqueued send/receive work be split between the threads?
- [MG]Are you mapping each communicator and stream combination to a different network endpoint, so that you can have concurrency with different comm&stream combinations?
  [HZ}That is left as option/policy. For MPI+thread, that is desirable. For MPI+GPU, that may be less desirable. Implementation can adopt different default based on the context.

# Project Delorean: MPI Graphs & Streams (Quincey Koziol)

Summary:

- Add MPI interfaces that allow users to define a graph of operations. Once defined, the graph can be executed (e.g. by a call to MPIX_Graph_execute)
    - Can allow for optimization by “collapsing” multiple operations into a semantically identical single/set of operation(s), since they are all “known” to the MPI implementation before the graph is executed
- Add MPI interfaces that allow users to define a stream of MPI operations
- Define a way to make strong progress for “true” async operations, with / without offload

Pro:

- Generalized to allow for many different kinds of hardware offload
- Can capture an MPI workflow and execute it asynchronously (e.g. offloaded to some execution agent)
- Supports batching

Con:

-

Questions:

- How are tasks in a stream or graph scheduled for execution? What is monitoring dependencies and dispatching work?
    - Can be offloaded to a SmartNIC, e.g. that has generic cores which can be used to execute the described work
        - QK: Yes - a “very smart NIC”
- Is the request returned/created by MPI_thing_def like a persistent request? Inactive on creation, can be later activated/started, etc.
    - QK: Yes, similar to the persistent operations, but generalized to allow for more MPI operations (e.g. File_open, Comm_create, etc)
        - We could update the persistent operations to be fully comp
- [DanH] Has some similarities to user schedules, but is more general/abstract.
    - QK: Yes - there is a “user callback” operation (which I didn’t speak to today) that can be inserted into a graph/stream, so provide a superset of generalized requests / user schedules
    - [DanH] No, I mean https://www.semanticscholar.org/paper/Extending-the-Message-Passing-Interface-(MPI)-with-Schafer-Ghafoor/af2789f307a27c20992a36e333c7ab30430102f5

Notes:

- QK: Would benefit from Jim’s “matching” info keys

# How to Reach a Common Proposal

## Goals/Criteria

1. Must enable taking advantage of today's hardware/platform features
    a. Support for GPU/Accelerator work scheduling/offloading models
    b. Support for GPU streams (must support flexible ordering, e.g. allowed by SYCL)
    c. Support for CUDA graphs (why CUDA here?)  why not graphs in the general case?
2. Must be abstract and general enough to be useful a decade in the future
    a. Must not be so abstract or general that we fail to meet goal (1)
3. Support for batch management of operations to reduce overheads
4. Understanding inter-node/intra-node implementation issues