MPI HYBRID & ACCELERATOR WORKING GROUP UPDATE

James Dinan, NVIDIA
HACC WG Chair
December 6, 2021

HYBRID & ACCELERATOR WORKING GROUP

Mission: Improve interoperability of MPI with other programming models

Active topics:

1. Continuations proposal #6
2. Clarification of thread ordering rules #117
3. Integration with accelerator programming models:
    1. Accelerator info keys #3
    2. Stream/Graph Based MPI Operations #5
    3. Accelerator bindings for partitioned communication #4
    4. Partitioned communication buffer preparation (shared with Persistence WG) #264

More information: https://github.com/mpiwg-hybrid/hybrid-issues/wiki

COMPLETION CONTINUATIONS

Treat the completion of an MPI operation as continuation of some activity

- Interoperability with asynchronous and multithreaded programming models
- Register callbacks that continue the activity upon completion of an MPI operation

```
11  MPI_Request cont_req;
12  MPIX_Continue_init(&cont_req);
13
14  omp_event_handle_t event;
15  int value;
16  #pragma omp task depend(out:value) detach(event)
17  {
18    MPI_Request req;
19    MPI_Irecv(&value, ..., &req);
20    MPIX_Continue(&req, &release_event, event, MPI_STATUS_NULL, cont_req);
21  }
22
23  #pragma omp task depend(in: value)
24  {
25    // process value
26  }
```

```
31  void release_event(MPI_Status status, void *data)
32  {
33    omp_event_handle_t event = (omp_event_handle_t)(uintptr_t)data;
34    omp_fulfill_event(event);
35  }
```

"Callback-based completion notification using MPI Continuations,"
Joseph Schuchart, Christoph Niethammer, José Gracia, George Bosilca, Parallel Computing, 2021.

"MPI Detach - Asynchronous Local Completion,"
Joachim Protze, Marc-André Hermanns, Ali Demiralp, Matthias S. Müller, Torsten Kuhlen. EuroMPI '20.

STREAM TRIGGERED NEIGHBOR EXCHANGE

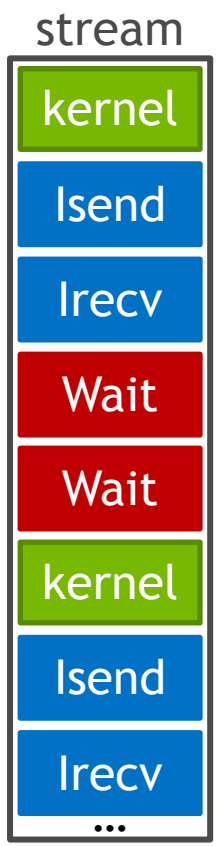
Simple Ring Exchange Using a CUDA Stream

```
MPI_Request send_req, recv_req;
MPI_Status  sstatus,  rstatus;

for (i = 0; i < NITER; i++) {
  if (i > 0) {
    MPI_Wait_enqueue(recv_req, &rstatus, MPI_CUDA_STREAM, stream);
    MPI_Wait_enqueue(send_req, &sstatus, MPI_CUDA_STREAM, stream);
  }

  kernel<<<..., stream>>>(send_buf, recv_buf, …);

  if (i < NITER – 1) {
    MPI_Irecv_enqueue(&recv_buf, …, &recv_req, MPI_CUDA_STREAM, stream);
    MPI_Isend_enqueue(&send_buf, …, &send_req, MPI_CUDA_STREAM, stream);
  }
}
cudaStreamSynchronize(stream);
```



ACCELERATOR AWARE INFO

Improve Interoperability With Accelerator Models

Challenges for users of accelerator-aware MPI today:

- Check whether accelerator awareness is supported
- Discover whether MPI is using a performance path for accelerator memory (e.g. system has the right config to do direct communication with the accelerator)
- Reduce overhead from hybrid programming

Possible solution:

- *mpi_hybrid_model = { “CUDA”, “HIP”, “SYCL”, “OpenMP”, etc. }*
- *mpi_accelerator_zero_copy = { “true”, “false” }*
- *mpi_assert_accelerator = { device_ids }*
- *mpi_assert_memory_kind = { “host”, “device” }*

CUDA BINDINGS FOR MPI PARTITIONED APIS

```
int MPI_Psend_init(const void *buf, int partitions, MPI_Count count,
                   MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Info info,
                   MPI_Request *request)

int MPI_Precv_init(void *buf, int partitions, MPI_Count count,
                   MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Info info,
                   MPI_Request *request)

int MPI_[start,wait][_all](...)
```

Keep host only

```
__device__ int MPI_Pready(int partition, MPI_Request request)

__device__ int MPI_Pready_range(int partition_low, int partition_high, MPI_Request request)

__device__ int MPI_Pready_list(int length, const int array_of_partitions[], MPI_Request request)

__device__ int MPI_Parrived(MPI_Request request, int partition, int *flag)
```

Add device bindings

KERNEL TRIGGERED COMMUNICATION USAGE

Partitioned Neighbor Exchange

Host Code

```
MPI_Request req[2];
MPI_Psend_init(..., &req[0]);
MPI_Precv_init(..., &req[1]);
while (...) {
  MPI_Startall(2, req);
  MPI_Pbuf_prepare_all(2, req);
  kernel<<<..., s>>>(..., req);
  cudaStreamSynchronize(s);
  MPI_Waitall(2, req);
}
MPI_Request_free(&req[0]);
MPI_Request_free(&req[1]);
```

Device Code

```
__device__
void MPI_Pready(int idx, MPI_Request req);

__global__ kernel(..., MPI_Request *req) {
  int i = my_partition(...);
  // Compute and fill partition i
  // then mark i as ready
  MPI_Pready(i, req[0]);
}
```

KERNEL & STREAM TRIGGERED COMMUNICATION USAGE

Partitioned Neighbor Exchange

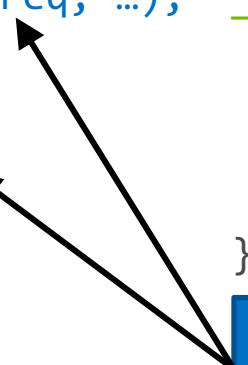
Host Code

```
MPI_Request req[2];
MPI_Psend_init(..., &req[0]);
MPI_Precv_init(..., &req[1]);
while (...) {
  MPI_Startall_enqueue(2, req, …);
  MPI_Pbuf_prepare_all_enqueue(2, req, …);
  kernel<<<..., s>>>(..., req);
  cudaStreamSynchronize(s);
  MPI_Waitall_enqueue(2, req, …);
}
MPI_Request_free(&req[0]);
MPI_Request_free(&req[1]);
```

Device Code

```
__device__
void MPI_Pready(int idx, MPI_Request req);

__global__ kernel(..., MPI_Request *req) {
  int i = my_partition(...);
  // Compute and fill partition i
  // then mark i as ready
  MPI_Pready(i, req[0]);
}
```



Moving to stream eliminates overhead from stream synchronization

Thank you!

Wednesdays 10-11am US Eastern Time

https://github.com/mpiwg-hybrid/hybrid-issues/wiki