Partitioned Communication CUDA Bindings

Updated 11/3/21

Contact: Jim Dinan <jdinan@nvidia.com>

Approach

Define Kernel Triggered bindings for MPI partitioned operations

Approach is to define as part of MPI CUDA language bindings

- `__device__` APIs
- Callable by kernels (`__global__` functions)
  - K<<<grid_dim, block_dim, smem, stream>>>(arg0, arg1, …);

Try to make them generic enough that similar bindings can be used with other accelerator languages / language extensions

MPI 4.0 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

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

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

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

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

Add device bindings

MPI Request Objects in Device Context

MPI handle must be valid in the device context

- Internal MPI state must be accessible by device (e.g. device, pinned host memory)

Option 1: Unified Memory

- Allocate MPI object in managed memory, accessible to host/device, handle value is unified pointer
  - Pages containing MPI requests migrate on access by host/device, or can set *cudaMemAdviseSetPreferredLocation* to avoid migration
  - Alternatively, MPI can update requests in start call and then pin to the device via *cudaMemAdviseSetReadMostly*

Option 1.5: Unified Virtual Addressing

- Allocate MPI object in cudaHostAlloc'd memory, accessible to host/device, handle value is unified pointer
  - Host/device pointer are same when device supports canUseHostPointerForRegisteredMem property
  - Lives in pinned host memory, which adds PCIe access latency from GPU

Option 2: Export request handle to the device, handle value is device pointer

- New export operation exports MPI request so it can be accessed by device
- Improves portability and can optimize the object for access from the device

Request Export

```
__host__ int MPI_Prequest_create(MPI_Request req,
                        MPI_Info info,
                        MPI_Prequest *preq /* host ptr */)

__host__ int MPI_Prequest_free(MPI_Prequest *preq)

MPI_PREQUEST_NULL
```

- `MPI_Prequest` objects are only valid for use in device functions
- The `MPI_Prequest` object is created on the currently selected device
  - MPI_Request must be in the non-started state
  - Can optionally use the info argument to specify the device
- `MPI_Prequest` for an MPI_Psend operation may contain:
  - libibverbs: Pointer to list of pre-prepared RDMA write WQEs, pointer to QP (SQ, DBR, DB)
  - libfabric: Reference to event counter with associated triggered send operation
- `MPI_Prequest` for an MPI_Precv operation may contain:
  - Pointer to flag in memory
- Starting `req` also starts any `preqs` that were created on `req`
  - E.g. posts triggered operations to the NIC

Discussion 8-18-2021:

Ryan: Could add a device side request query function to convert a request object

```
__device__
MPI_Request_query(
   MPI_Request req_in,
   MPI_Request *req_out);
```

- Import request to device
- Host creates “exportable” request object so it can be imported, not able to predict whether device will call query
- Could require memory allocation from device

Whit: Could make sense to have both prequest_create on host and query on the device

Request Export (Still Under Discussion)

```
__host__ int MPI_Prequest_create(MPI_Request req,
                        MPI_Info info,
                        MPI_Prequest *preq /* host ptr */)

__host__ int MPI_Prequest_free(MPI_Prequest *preq)

MPI_PREQUEST_NULL
```

- `MPI_Prequest` objects are only valid for use in device functions
- The `MPI_Prequest` object is created on the currently selected device
  - MPI_Request must be in the non-started state
  - Can optionally use the info argument to specify the device
- `MPI_Prequest` for an MPI_Psend operation may contain:
  - libibverbs: Pointer to list of pre-prepared RDMA write WQEs, pointer to QP (SQ, DBR, DB)
  - libfabric: Reference to event counter with associated triggered send operation
- `MPI_Prequest` for an MPI_Precv operation may contain:
  - Pointer to flag in memory
- Starting `req` also starts any `preqs` that were created on `req`
  - E.g. posts triggered operations to the NIC

Alternative: Request Import (Still Under Discussion)

```
__device__ MPI_Request_query(MPI_Request req_in,
                             MPI_Request *req_out);
```

Device-side MPI Request query function generates device-accessible request

- Host creates “exportable” request object so it can be imported on the device

Challenges:

- Not able to predict whether device will call query on a given request
- Could require memory allocation or other communication setup from device

Discussion:

- Could make sense to have both prequest_create on host and request_query on the device

Device Ready

```
__device__ int MPI_Pready(MPI_Prequest prequest, int partition)

__device__ int MPI_Pready_range(int part_low, int part_high,
                MPI_Prequest prequest)

__device__ int MPI_Pready_list(int length,
                const int array_of_partitions[],
                MPI_Prequest prequest)
```

- `MPI_Pready` operation may involve
  - ibverbs: Copy pre-prepared WQE to SQ and ring doorbell
  - libfabric: Increment event counter
- Allow functions to be inlined
- Could replace __device__ with a generic thing like MPI_DEVICE_QUALIFIER

Device Arrived

```
__device__ int MPI_Parrived(MPI_Prequest preq,
                            int partition, int *flag)
```

- `MPI_Parrived` operation may involve
  - libibverbs: Polling a CQ, e.g. if write with immediate is used
  - Checking flags for completion of the given operation
- May generate more efficient code to return flag
- Allow functions to be inlined
- Q: Should we return an MPI error code?
  - What can the device do with it? Hand it back to the CPU.

Questions

Do we want warp/block variants of ready/arrived functions?

- E.g. may want to perform a memcpy in `MPI_Pready` for P2P transfers

Martin: Does this require MPI_THREAD_MULTIPLE or some other (new?) threading model in the MPI library?

- There are concurrency in two forms:
  - Multiple threads running on the device calling ready/arrived APIs
  - Host and device performing MPI operations at the same time
- Ryan: MPI_THREAD_PARTITIONED was part of the original partitioned communication proposal - Only parrived/pready called by multiple threads, otherwise it's MPI_THREAD_SERIALIZED

Questions (11/3/2021)

Should we make specifying the device in MPI_Prequest more general?

- Could add device ID to the MPI_Prequest creation function
- Could introduce an opaque object for device handle
  - Would need MPI functions to create a device handle

Namespacing of language extensions

- Desire to allow multiple language extensions to be used in the same program
  - E.g. Shipping a binary that can be used with any vendor's GPU
- Would need namespacing so that compilation and linking work properly

Need to specify interactions between MPI_Request and MPI_Prequest

- Two handles to the same operation, could clash if accessed at the same time
  - E.g. MPI_Parrived on prequest while waiting on request or MPI_Pready called from both host/device for different partitions

Host Code

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

Ensure all partitions marked ready before calling wait

Device Code

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

All threads working on partition *i* call MPI_Pready. Could filter out one thread or add MPI_Pready_block or MPI_Pready_warp.