



Accelerator Triggered Communication

Jim Dinan, NVIDIA
Hybrid & Accelerator WG
MPI Forum Virtual Meeting
February 16, 2022

MPI FORUM HYBRID & ACCELERATOR WORKING GROUP

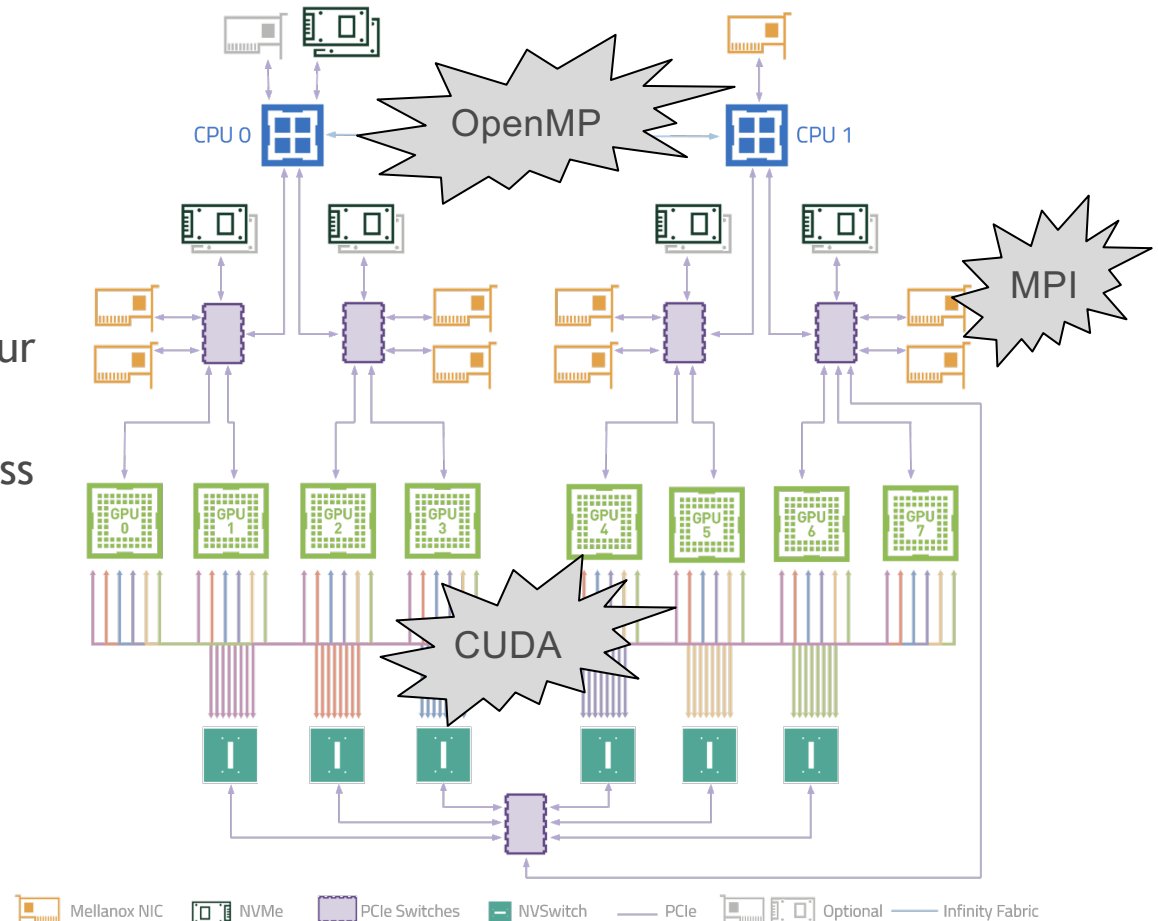
Goal: Integrate MPI communication with accelerator programming models

Why: Modern architectures require MPI + X + Y

- e.g. OpenMP, CUDA, HIP, OpenACC, etc.
- Transitions between programming models incur significant overheads (e.g. synchronization)
- Or expose overheads we want to hide, e.g. loss of pipelining to hide offload overheads

Active integration topics for MPI 4.1:

1. Stream and graph triggering
2. Kernel triggering using partitioned comms



CPU-INITIATED API CHALLENGES

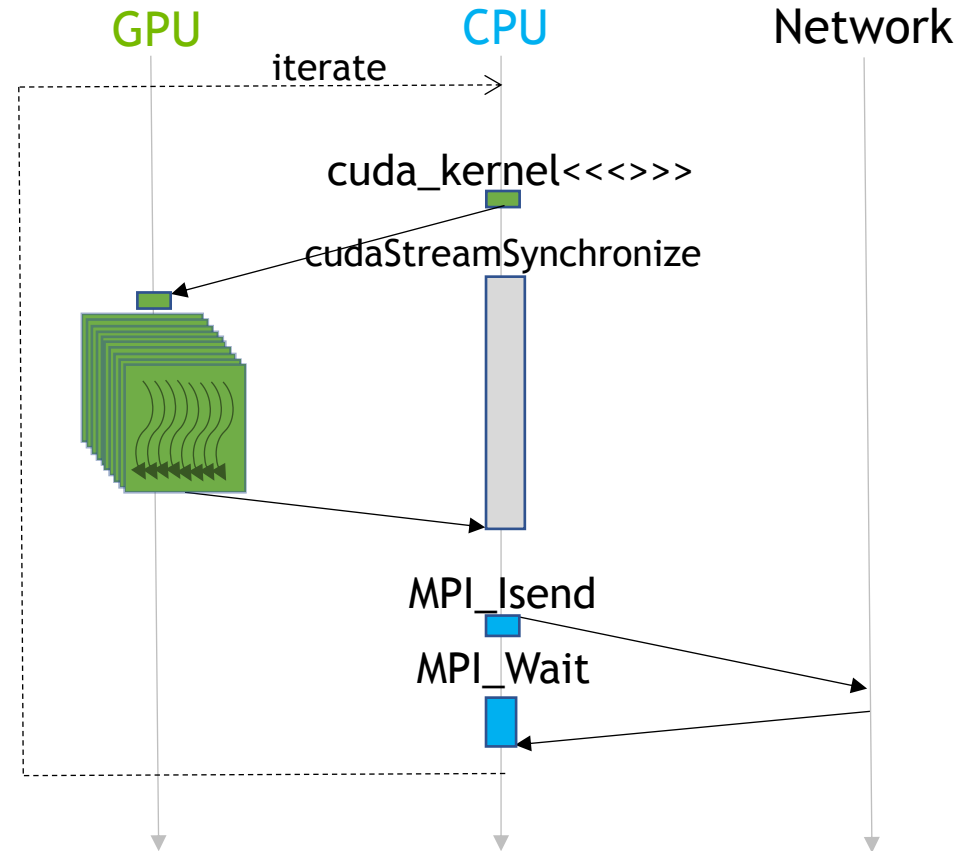
Offloading Overheads and Communication are Exposed

- **Compute** on GPU
- **Communication** from CPU
- Synchronization at boundaries

Commonly used model, but -

- Offload latencies in critical path
- Communication is not overlapped

Hiding increases code complexity,
Not hiding limits strong scaling



STREAM TRIGGERED COMMUNICATION

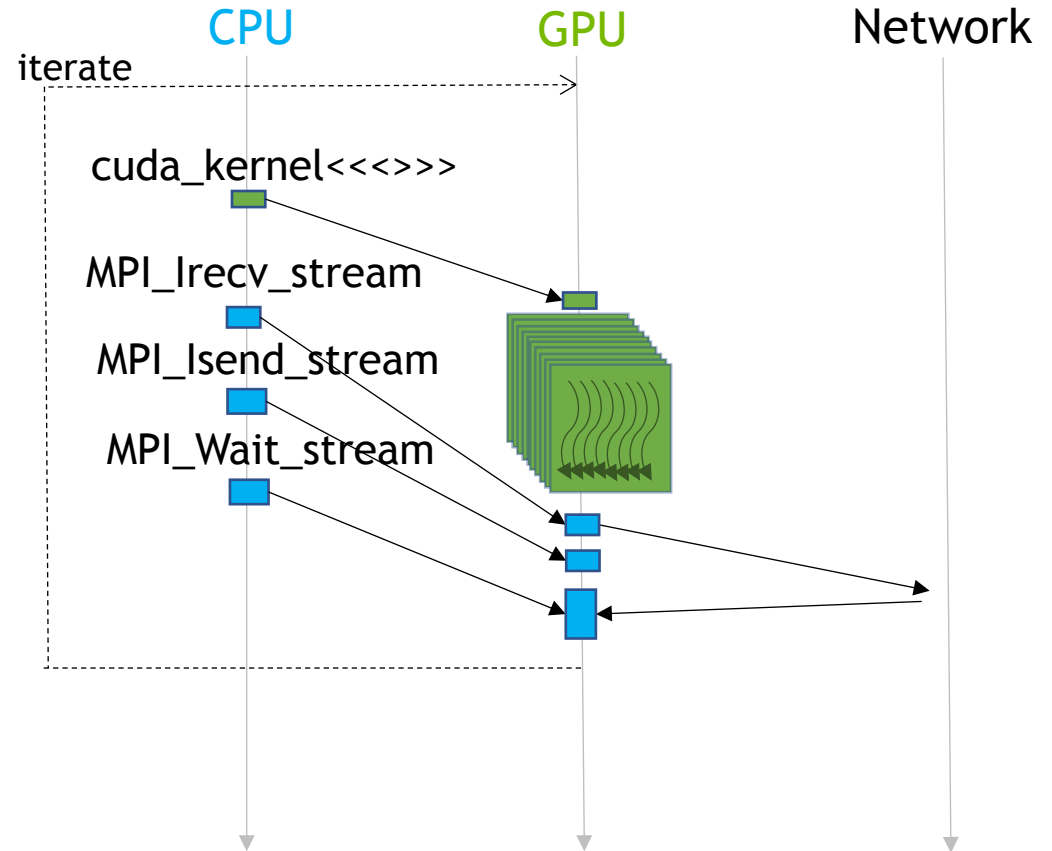
Hides offloading overheads

Eliminates transfer of control between CPU and GPU

“Triggering” avoids running MPI progress engine on the accelerator

Communication on streams can be recorded and embedded in graphs

Computation and communication are distinct tasks, don't naturally overlap



KERNEL TRIGGERED COMMUNICATION

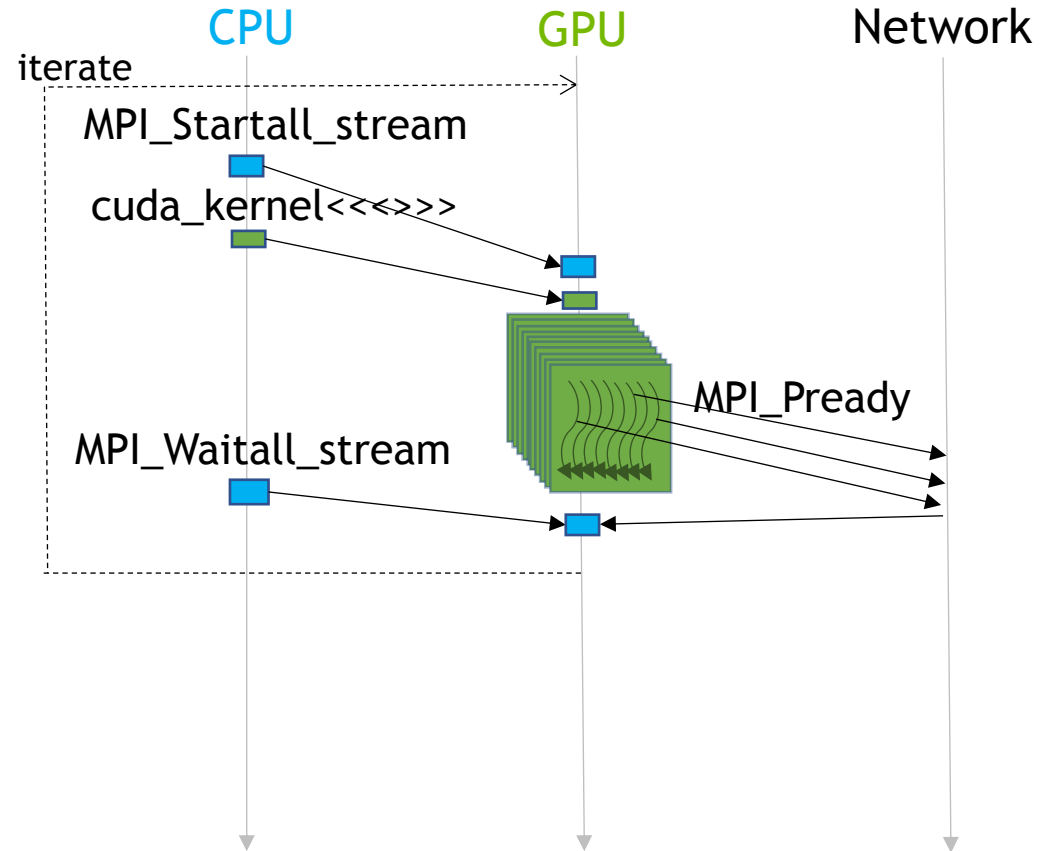
Overlaps Communication and Computation

Use MPI-4 partitioned communication to trigger communication from kernel

Stream-based start/wait eliminates offloading overheads

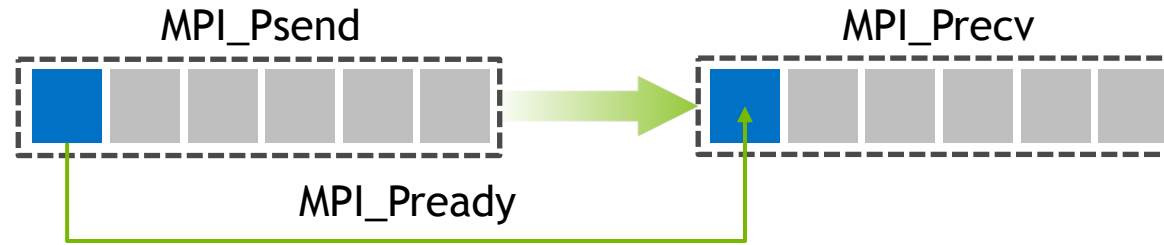
Marking partitions as ready overlaps computation and communication

Communication on streams can be recorded and embedded in graphs



ACCELERATOR TRIGGERED COMMUNICATION

Using the MPI 4.0 Persistent Partitioned Communication API



Send/Recv data buffers are broken into equal-sized partitions

- **MPI_Pready**: mark partition as ready to send
- **MPI_Parrived**: query if partition has arrived

Partitioned operations match once in own matching space based on order of init calls

Extending to accelerators: language bindings for accelerator models (unify if possible)

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)
```

Host

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

Host + Stream

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

Device

```
__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)
```

KERNEL TRIGGERED COMMUNICATION USAGE

Partitioned Neighbor Exchange

Host Code

Device 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);
    kernel<<<..., s>>>(..., preq);
    cudaStreamSynchronize(s);
    MPI_Waitall(2, req);
}
MPI_Prequest_free(&preq);
MPI_Request_free(&req[0]);
MPI_Request_free(&req[1]);
```

```
__global__ kernel(..., MPI_Prequest preq) {
    int i = my_partition(...);
    // Compute and fill partition i
    // then mark i as ready
    MPI_Pready(i, preq);
}
```

All partitions must be marked ready before calling wait

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 Virtual 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 migrate 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

- Export operation sets up MPI request so it can be accessed by device
- Improves portability
- Can optimize the MPI request object for access from the device

REQUEST EXPORT

```
int MPI_Prequest_create(MPI_Request req, MPI_Info info, MPI_Prequest *preq);  
  
int MPI_Prequest_free(MPI_Prequest *preq);  
  
MPI_PREREQUEST_NULL
```

- Prequest object is valid on device in and MPI_Prequest_free
- The MPI_Prequest object is created for use by 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 counters with associated triggered send operations
- MPI_Prequest for an MPI_Precv operation may contain:
 - Pointer to flags in device memory
- Starting req also starts any preqs that were created on req
 - E.g. posts triggered operations to the NIC

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
 - peer-to-peer: memcpy (e.g. PCIe or NVLink)
 - ibverbs: Copy pre-prepared WQE to SQ and ring doorbell
 - libfabric: Increment event counter
- Allow functions to be inlined
- Allow for functions with thread granularity (e.g. warp/block) to optimize memcpy's
- Could replace __device__ with a generic thing like MPI_DEVICE_QUALIFIER

DEVICE ARRIVED

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

- MPI_Parrived operation may involve
 - peer-to-peer: Checking a flag
 - libibverbs: Polling a CQ
 - libfabric: Checking an event counter
- Allow functions to be inlined
- Allow for functions with thread granularity (e.g. warp/block) to optimize memcpy's
 - E.g., When using a receiver copys protocol
- Q: Should we return an MPI error code?
 - What can the device do with it?
 - Hand it back to the CPU, to be dealt with with the next operation on the request.

KERNEL TRIGGERED COMMUNICATION USAGE

Partitioned Neighbor Exchange

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);
    kernel<<<..., s>>>(..., preq);
    cudaStreamSynchronize(s);
    MPI_Waitall(2, req);
}
MPI_Prequest_free(&preq);
MPI_Request_free(&req[0]);
MPI_Request_free(&req[1]);
```

Device Code

```
__global__ kernel(..., MPI_Prequest preq) {
    int i = my_partition(...);
    // Compute and fill partition i
    // then mark i as ready
    MPI_Pready(i, preq);
}
```

All partitions must be marked ready before calling wait

KERNEL TRIGGERED COMMUNICATION USAGE

Partitioned Neighbor Exchange

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_enqueue(2, req, ..., s);
    kernel<<<..., s>>>(..., preq);
    cudaStreamSynchronize(s);
    MPI_Waitall_enqueue(2, req, ..., s);
}
MPI_Prequest_free(&preq);
MPI_Request_free(&req[0]);
MPI_Request_free(&req[1]);
```

Device Code

```
__global__ kernel(..., MPI_Prequest preq) {
    int i = my_partition(...);
    // Compute and fill partition i
    // then mark i as ready
    MPI_Pready(i, preq);
}
```

On-stream operations eliminate
stream synchronization

KERNEL TRIGGERED COMMUNICATION USAGE

Kernel Calls MPI_Parrived

Host Code

```
MPI_Request req[2];
MPI_Prerequest preq;
MPI_Psend_init(..., &req[0]);
MPI_Precv_init(..., &req[1]);
MPI_Prerequest_create(req[0], ..., &preq);
while (...) {
    MPI_Startall_enqueue(2, req, ..., s);
    sender<<<..., s>>>(..., preq);
    receiver<<<..., s>>>(..., preq);
    MPI_Waitall_enqueue(2, req, ..., s);
}
MPI_Prerequest_free(&preq);
MPI_Request_free(&req[0]);
MPI_Request_free(&req[1]);
```

Device Code

```
__global__ sender(..., MPI_Prerequest preq) {
    int i = my_partition(...);
    sender_compute(..., i);
    MPI_Pready(i, preq);
}

__global__ receiver(..., MPI_Prerequest preq) {
    int arrived;
    int i = my_partition(...);
    do {
        MPI_Parrived(preq, i, &arrived);
    } while (!arrived);
    receiver_compute(..., i);
}
```

DISCUSSION

Protocol Challenges With Partitioned API

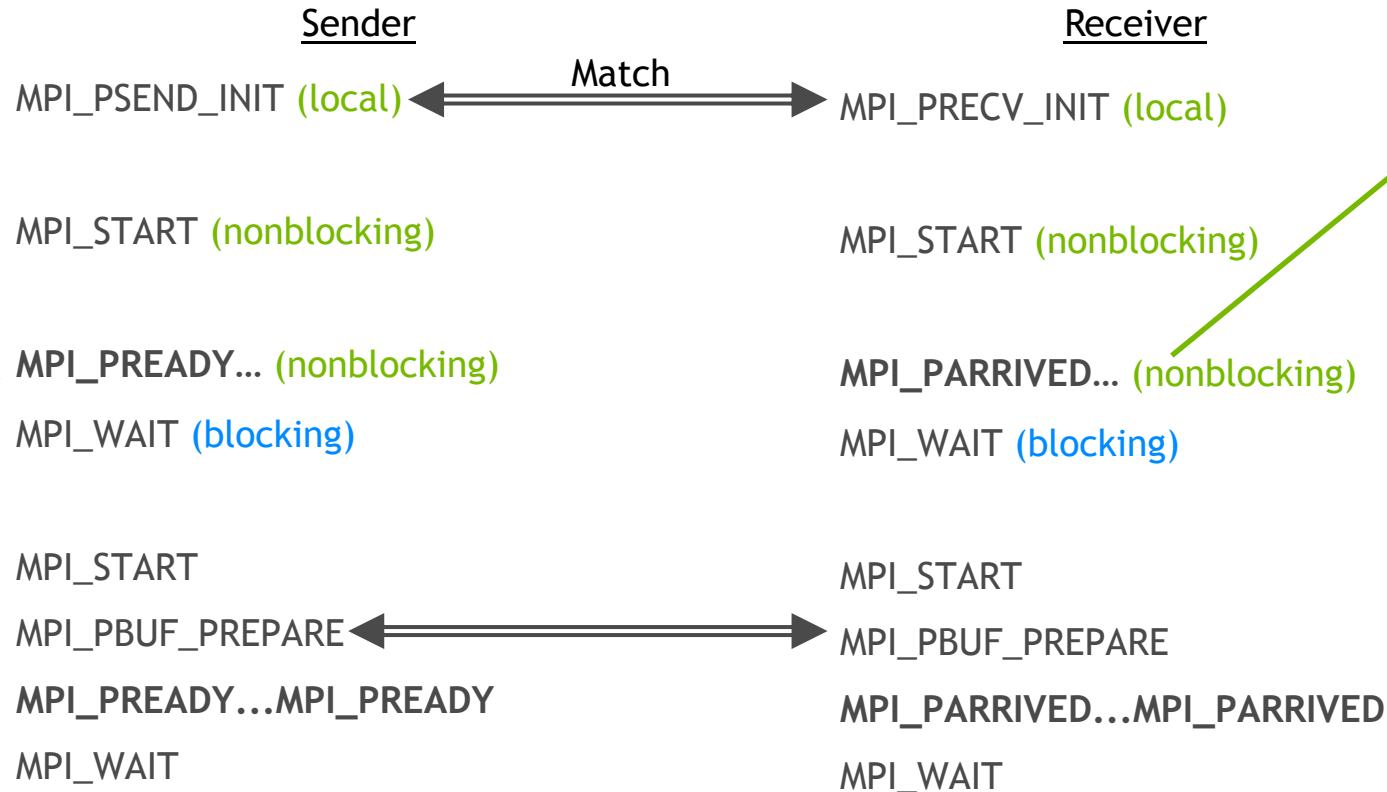
Protocol challenges:

- Establishing the half channel: matching psend with precv
- Determining if receiver is ready

Design goal: Accelerator should not need to interact with the MPI progress engine

SENDER/RECEIVER INTERACTION

Protocol Challenges

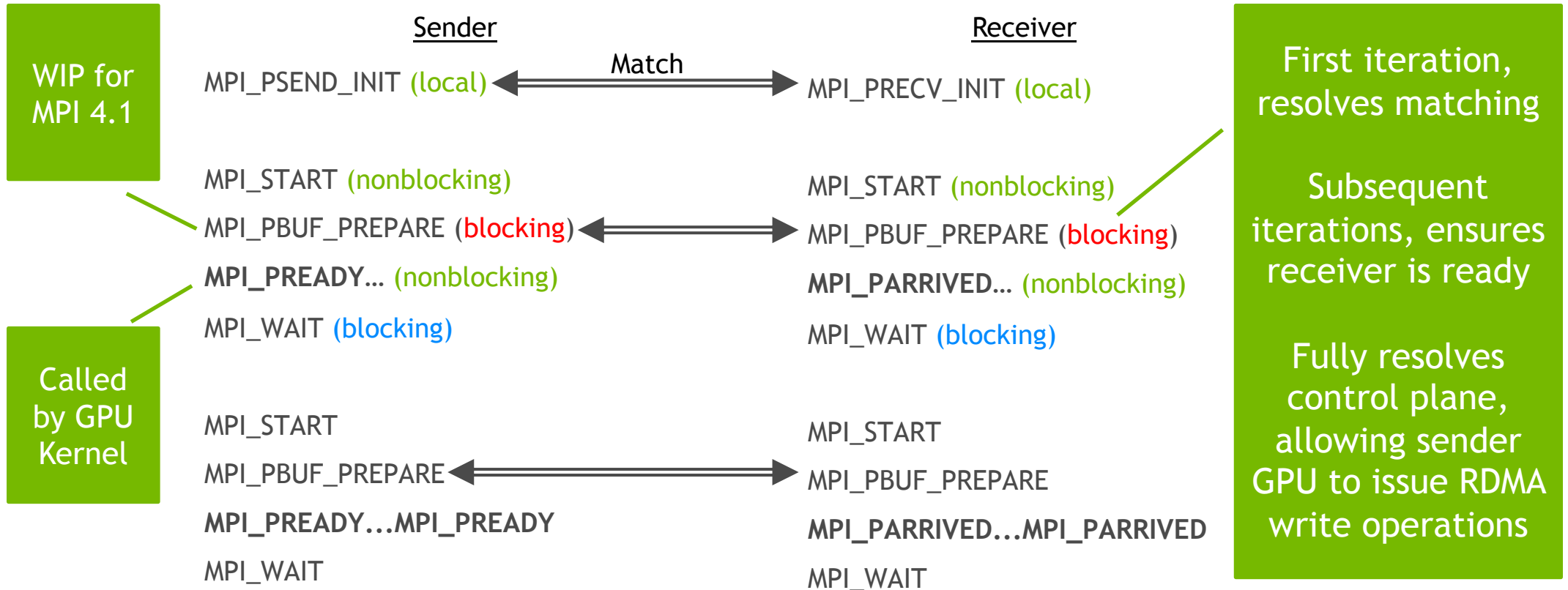


First iteration,
matching may not
have occurred

Subsequent
iterations, is
receiver ready?

SENDER/RECEIVER INTERACTION

Buffer Preparation Step



HYBRID & ACCELERATOR WG

Discussion

Partitioned communication API incorporated in MPI 4.0

Partitioned communication provides a path to kernel triggered communication

- CUDA and SYCL language extensions for MPI_Pready and MPI_Parrived
- Proposed partitioned buffer preparation function to advance protocols
 - Collapse send/recv to RDMA and memcpy

Steam/Graph integration is under active discussion

- “Greatest Common Denominator” approach

Thank you!

Wednesdays 10-11am US Eastern Time

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