STREAM / GRAPH TRIGGERED COMMUNICATION

James Dinan
September 28, 2022

STREAM TRIGGERING API EXTENSION

Accelerator Agnostic Work Queue Parameters

Two new function arguments:

1. `int queue_kind = {MPI_QUEUE_CUDA_STREAM, MPI_QUEUE_CUDA_GRAPH, …}`
   - Indicate the type of queue that the MPI operation is being appended to

2. `void *queue_args`
   - Provide “generic” arguments needed for the given queue type
   - Can be an input, e.g. to input a cudaStream_t
   - Can be an output, e.g. to output cudaGraph_t

STREAM AND GRAPH TRIGGERED COMMUNICATION

Non-Persistent Stream/Graph Triggered Operations

- New API: MPI operations that accept stream or graph argument
- Pro: What users are expecting
- Con: No separation of control and data planes

Persistent Stream/Graph Triggered Communication

- New API: start/wait operations that accept a stream argument
- Pro: Integrates with kernel triggered (partitioned) communication API
- Pro: Provides better separation of control and data planes
- Pro: Startall and Waitall can trigger/complete operations as a batch
- Con: Less flexible, e.g. when communication pattern changes over time

STREAM-TRIGGERED COMMUNICATION

Non-Persistent Communication APIs

```
int MPI_Isend_enqueue(const void *buf, int int tag, MPI_Comm comm, MPI_Request *request,
                      int queue_kind, void *queue_args);
int MPI_Irecv_enqueue(void *buf, int count, MPI_Datatype datatype, int source, int tag,
                      MPI_Comm comm, MPI_Request *request, int queue_kind, void *queue_args);
```

- Insert a nonblocking send/recv into a stream
- Data is ready to be sent or received when stream execution reaches this operation

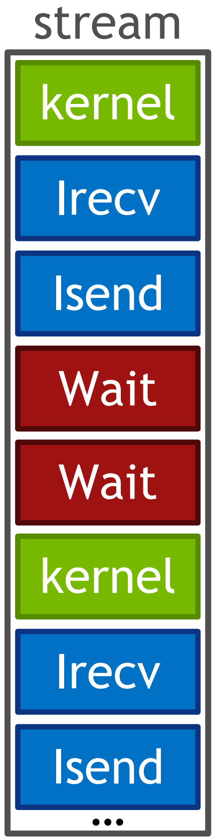
```
int MPI_Wait_enqueue(MPI_Request *req, MPI_Status *status, int queue_kind, void *queue_args);
int MPI_Waitall_enqueue(MPI_Count count, MPI_Request requests[], MPI_Status *status,
                        int queue_kind, void *queue_args);
```

- Wait for previously issued send and receive operations to complete
- Input is request arguments that were output by the enqueue operation
    - Subtle difference from existing MPI operations because the send/recv is not yet issued

NON-PERSISTENT MPI STREAM TRIGGERED API

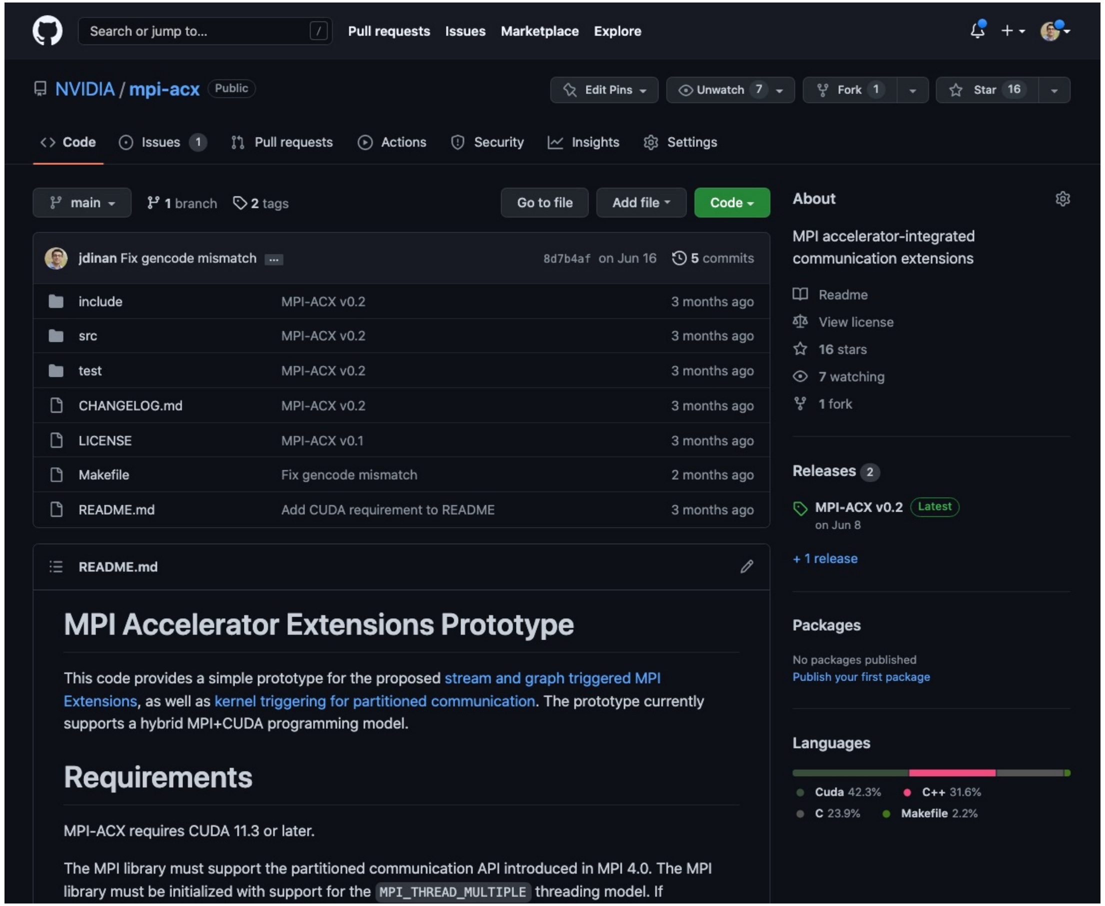
Simple Ring Exchange

```
MPI_Request send_req;
MPI_Request recv_req;

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);
  }
}
```



MPI-ACX

Prototype of Stream, Graph, and Kernel Triggered Operations



Proxy thread issues communication

- Calls the triggered MPI function
- One flag per operation in host registered memory

Supports:

- Stream/graph synchronous Isend, Irecv, and Wait
- Kernel triggered bindings for partitioned communication

Generic MPI Library

- Can be used with any MPI library, but might deadlock if MPI makes CUDA calls internally

https://github.com/NVIDIA/mpi-acx

PERSISTENT STREAM-TRIGGERED COMMUNICATION

Generic API Compatible with Multiple Accelerator Models

```
int MPI_Start_enqueue(MPI_Request *req, int queue_kind, void *queue_args);
int MPI_Wait_enqueue(MPI_Request *req, MPI_Status *status, int queue_kind, void *queue_args);

int MPI_Startall_enqueue(MPI_Count count, MPI_Request requests[], int queue_kind, void *queue_args);
int MPI_Waitall_enqueue(MPI_Count count, MPI_Request requests[], MPI_Status *status,
                        int queue_kind, void *queue_args);
```

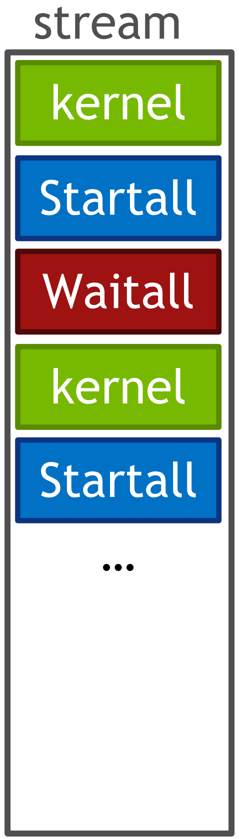
1. Persistent init APIs called on CPU, set up future communication
   - Allows MPI library to manage control and data planes separately

2. Communication request is started and waited on from stream
   - Captures data dependence through stream ordering
   - Can be restarted to repeat the same operation again enabling CUDA graph replay

MPI PERSISTENT STREAM TRIGGERED API

Simple Ring Exchange

```
MPI_Request req[2];
MPI_Status status[2];

MPI_Send_init(&send_buf, …, &req[0]);
MPI_Recv_init(&recv_buf, …, &req[1]);

for (i = 0; i < NITER; i++) {
  if (i > 0)
    MPI_Waitall_enqueue(2, req, status, MPI_CUDA_STREAM, stream);

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

  if (i < NITER - 1)
    MPI_Startall_enqueue(2, req, MPI_CUDA_STREAM, stream);
}
```



KERNEL TRIGGERED COMMUNICATION USAGE

Paired with Stream Triggered Wait/Start

Host Code

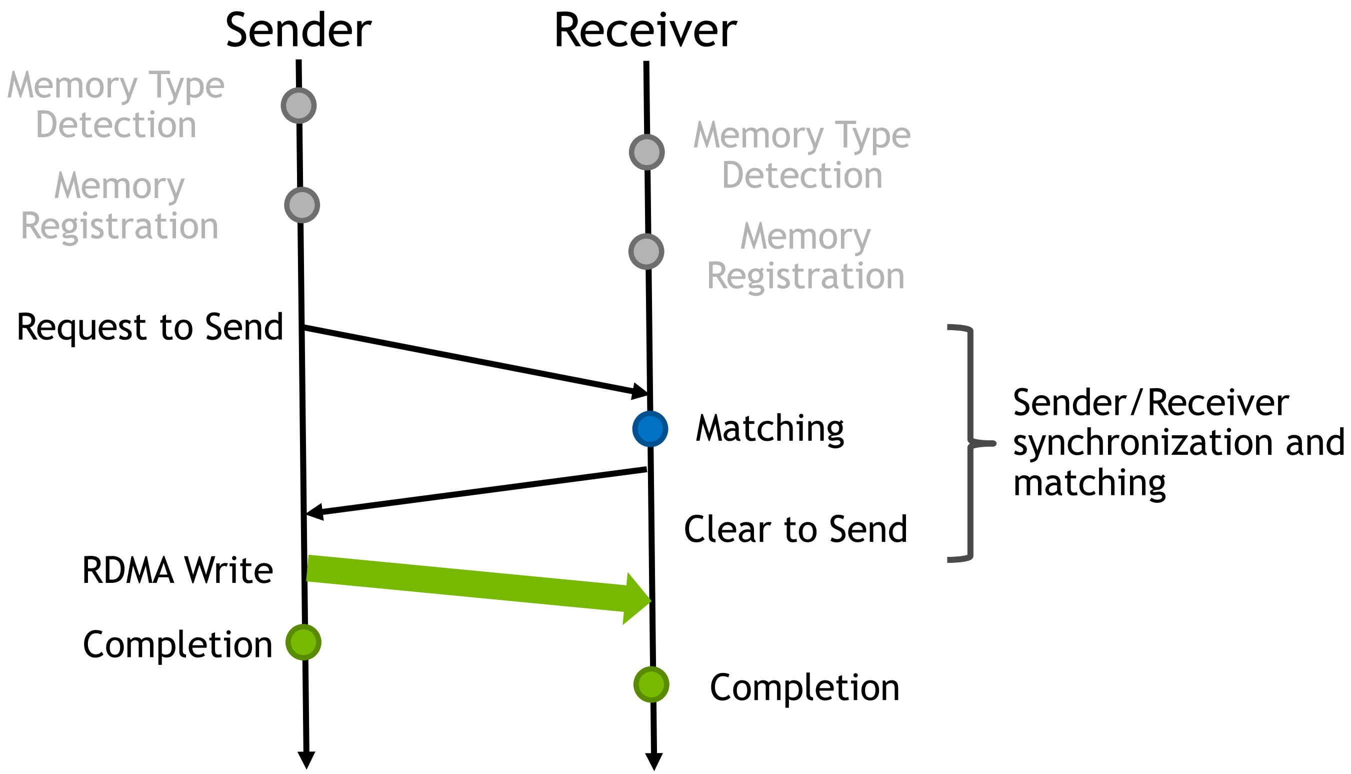
```
MPI_Request req[2]; MPI_Status s[2];
MPI_Psend_init(..., &req[0]);
MPI_Precv_init(..., &req[1]);
while (...) {
  MPI_Startall_enqueue(2, req, stream);
  MPI_Prepare_all_enqueue(2, req, stream);
  kernel<<<..., stream>>>(..., req);
  MPI_Waitall_enqueue(2, s, req, stream);
}
cudaStreamSynchronize(stream);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

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]);
}
```

PERSISTENCE HELPS, BUT …

Persistent Ops match again each time they are started



MPI INFO TO THE RESCUE?

mpi_assert_persistent_op_matches_another_persistent_op

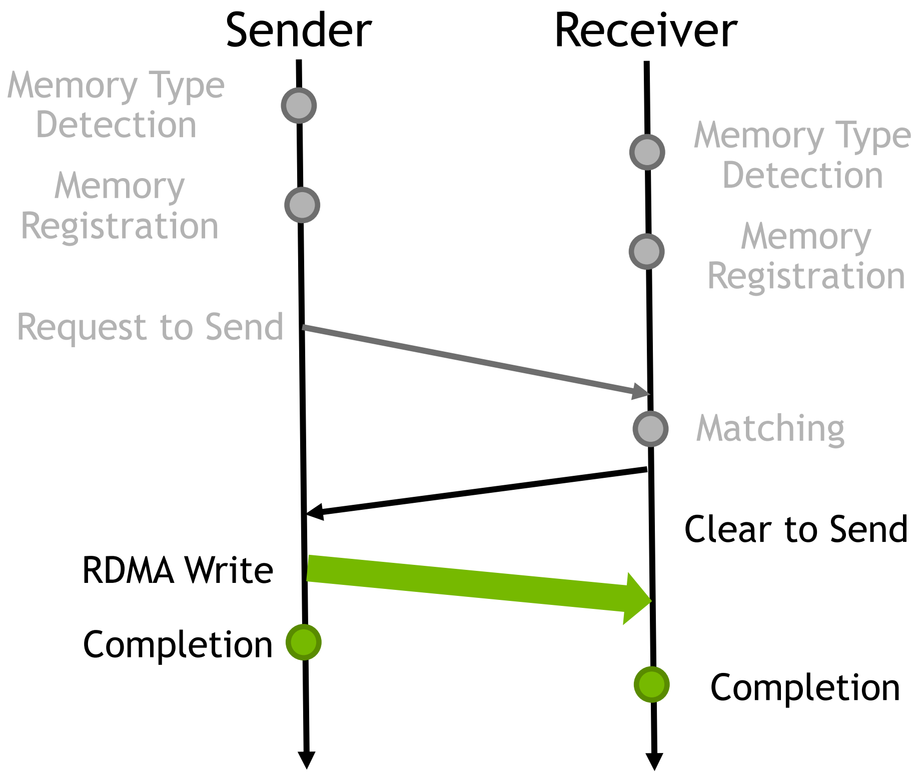
… and …

mpi_assert_always_matches_the_same_op

… and …

MPI_Prepare(req);

SIMPLIFYING PROTOCOLS VIA INFO ASSERTIONS



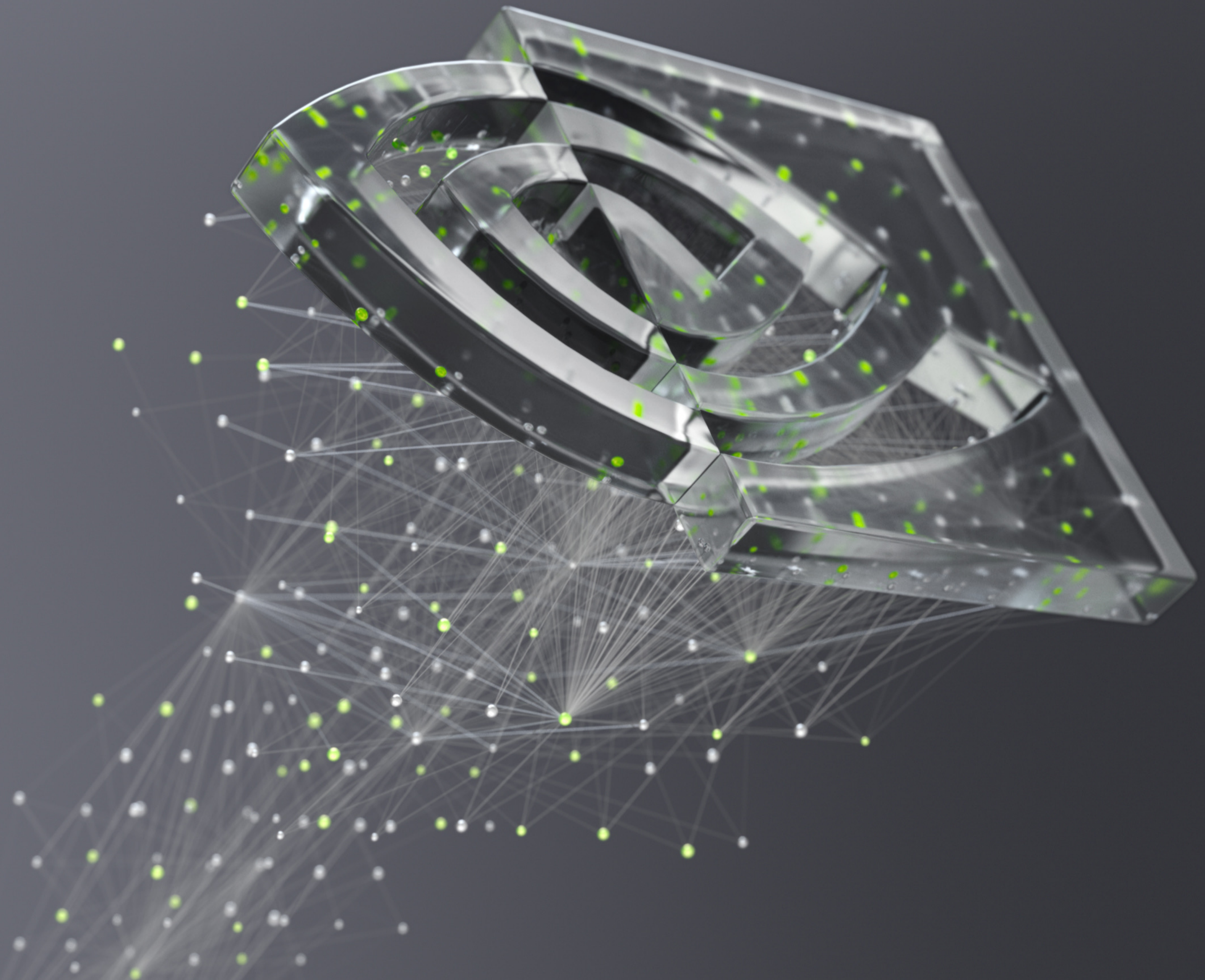
CONCLUSIONS

Add accelerator queue arguments to MPI operations

- Pro: A lot of flexibility
- Con: A lot of flexibility (harder to optimize)
- Con: Can’t trigger in batches

Persistent communication

- Pro: Don’t need to add very many new APIs
- Pro: Simplifies protocols (a lot if we add info)
- Pro: Can trigger in batches (startall, waitall)



NVIDIA