2022 OFA Virtual Workshop

STATUS OF OPENFABRICS INTERFACES (OFI) SUPPORT IN MPICH

Yanfei Guo, Assistant Computer Scientist

Argonne National Laboratory

OVERVIEW

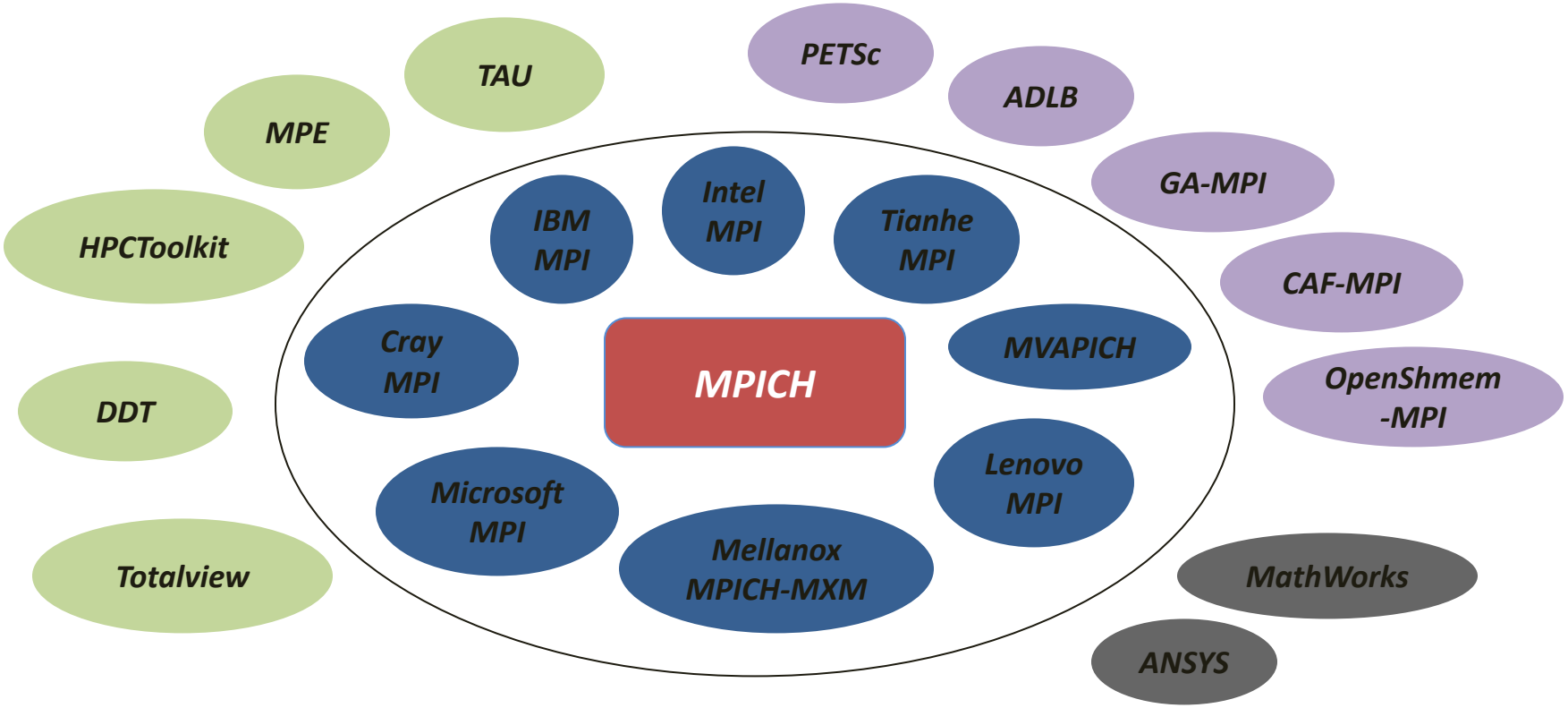
- What is MPICH?
- Why OFI?
- Current Support
- Future Plan

WHAT IS MPICH?

- **MPICH is a high-performance and widely portable open-source implementation of MPI**
- **It provides all features of MPI that have been defined so far (up to MPI-4.0)**
- **Active development lead by Argonne National Laboratory and University of Illinois at Urbana-Champaign**
  - Several close collaborators who contribute features, bug fixes, testing for quality assurance, etc.
    - IBM, Microsoft, Cray, Intel, Ohio State University, Queen's University, Mellanox, RIKEN AICS and others
- **www.mpich.org**

MPICH: GOAL AND PHILOSOPHY

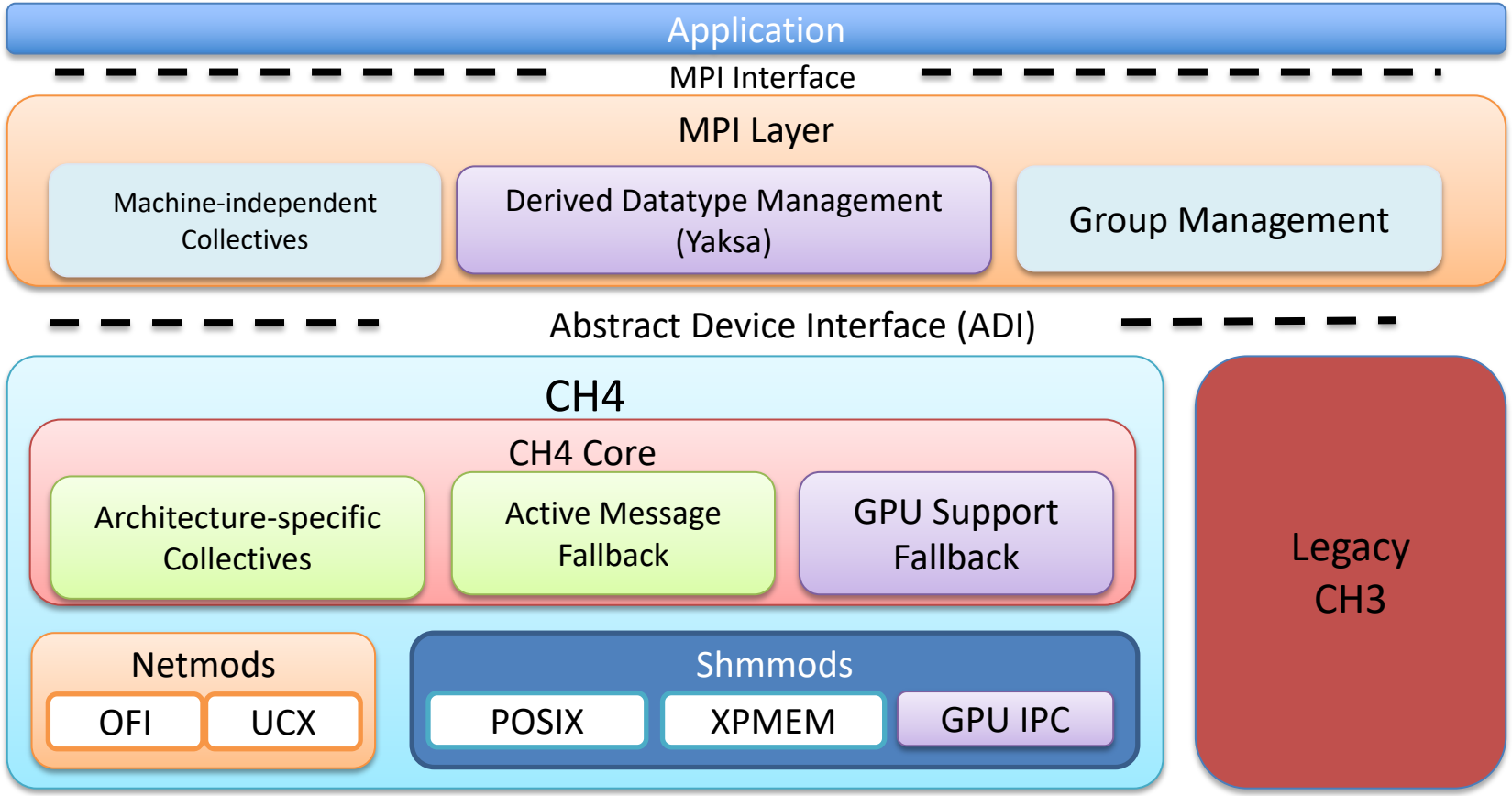
- MPICH aims to be the preferred MPI implementation on the top machines in the world
- Our philosophy is to create an "MPICH Ecosystem"



MOTIVATION

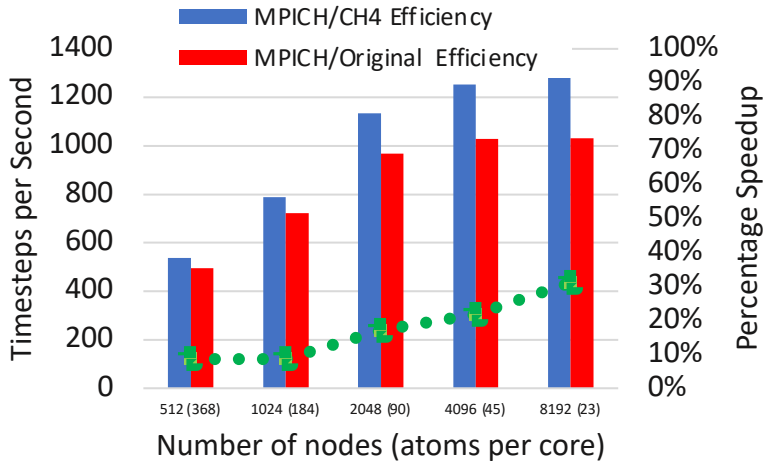
- **Why OFI/OFIWG?**
  - Support for diverse hardware through a common API
  - Actively, openly developed
    - Bi-weekly calls
    - Hosted on Github
  - Close abstraction for MPI
    - MPI community engaged from the start
  - Fully functional sockets provider
    - Prototype code on a laptop
  - Strong Vendor Support

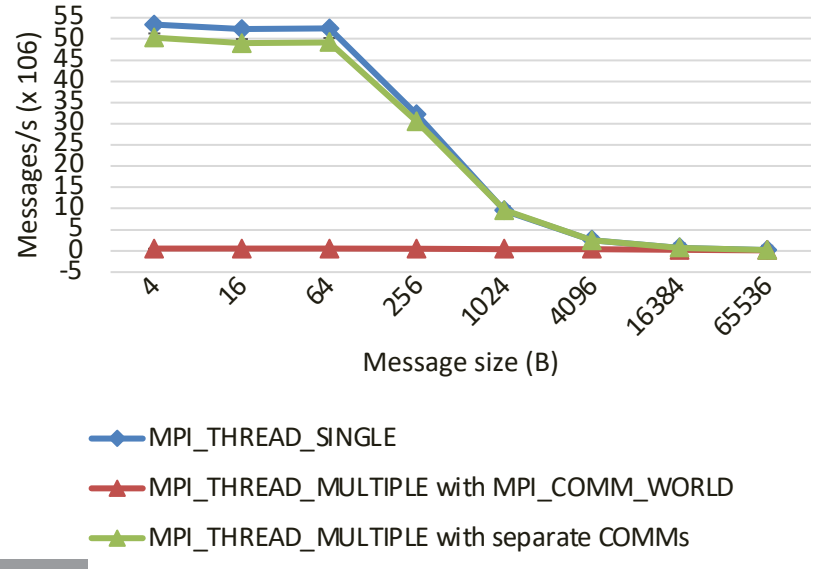
MPICH WITH CH4 DEVICE OVERVIEW



MPICH-3.4 SERIES

- **CH4 device being the default**
  - Replacement for CH3 as default option, CH3 still maintained till all of our partners have moved to CH4
  - Co-design effort
    - Weekly telecons with partners to discuss design and development issues
  - Three primary objectives:
    - Low-instruction count communication
      - Ability to support high-level network APIs (OFI, UCX)
      - E.g., tag-matching in hardware, direct PUT/GET communication
    - Support for very high thread concurrency
      - Improvements to message rates in highly threaded environments (MPI_THREAD_MULTIPLE)
      - Support for multiple network endpoints (THREAD_MULTIPLE or not)
    - Initial Support for GPU



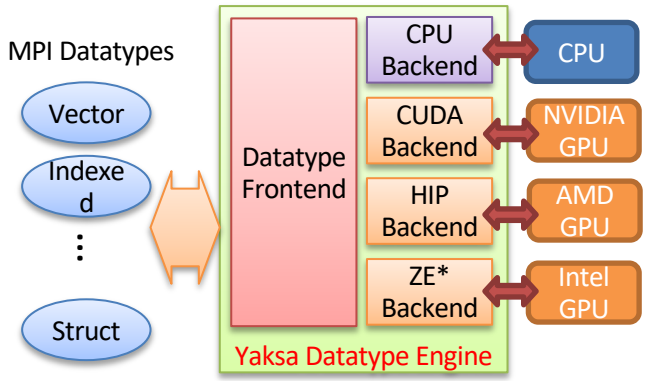


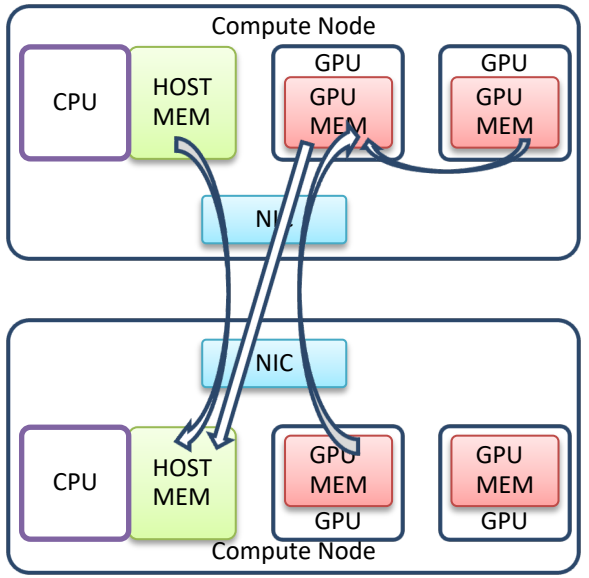
MPICH-4.0 SERIES – GPU SUPPORT

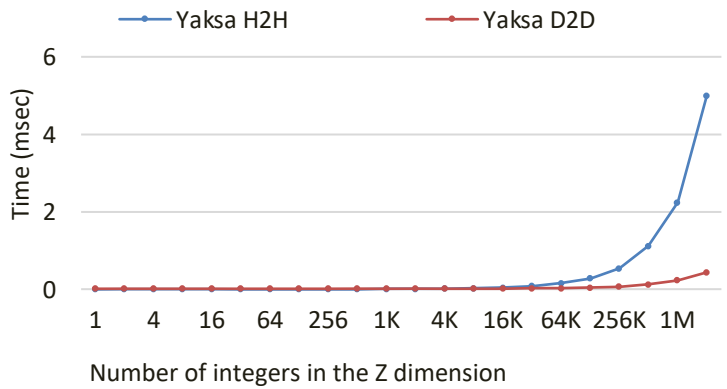
Enhanced GPU Support

- **Native GPU Data Movement**
  - Multiple forms of "native" data movement
  - GPU Direct RDMA is generally achieved through Libfabrics or UCX (we work with these libraries to enable it)
  - GPU Direct IPC is integrated into MPICH
- **GPU Fallback Path**
  - GPU Direct RDMA may not be available due to system setup (e.g. library, kernel driver, etc.)
  - GPU Direct IPC might not be possible for some system configurations
  - GPU Direct (both forms) might not work for noncontiguous data
  - Datatype and Active Message Support
- **Other Improvements**
  - Reduced Overhead for CPU buffer in a GPU-enabled MPICH build
  - Avoiding unnecessary buffer location checking

- **Supporting Multiple GPU Node**
  - Data movement between GPU devices
  - Utilizing high bandwidth inter-GPU links (e.g. NVLINK)
- **GPU-IPC Communication via Active Message**
  - Create IPC handles for GPU buffers
  - Send IPC handles to target process
  - Receiver initiate Read/Write using the IPC handle
- **Fallback Path in General SHM Active Message**
  - When IPC is not available for the GPU-pair





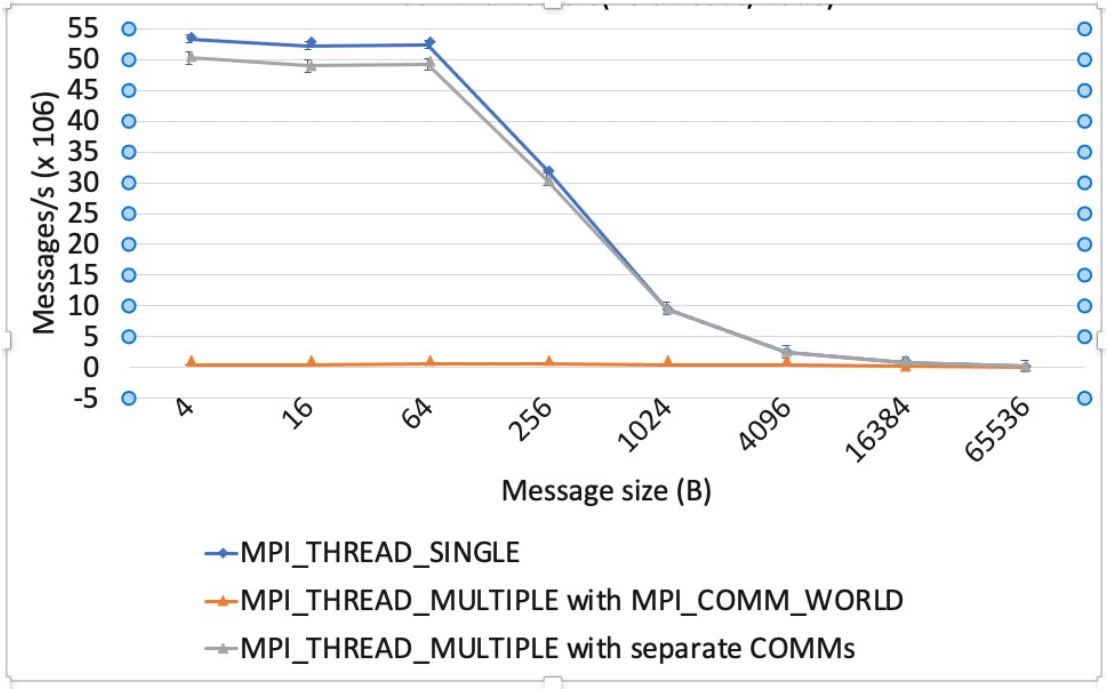


The GPU support in MPICH is developed in close collaboration with vendor partners including Including AMD, Cray, Intel, Mellanox and NVIDIA

MPICH-4.0 SERIES – MULTI-THREADED MPI

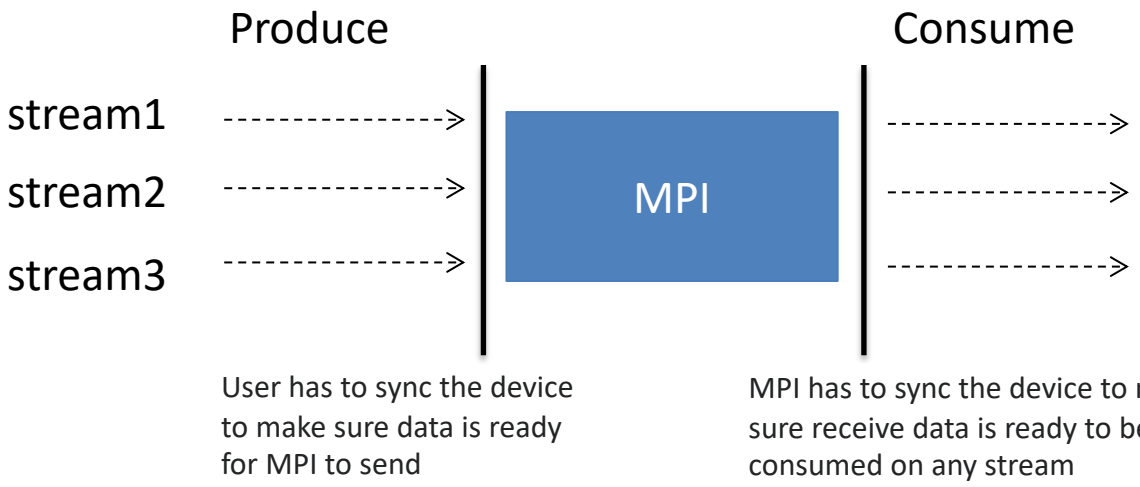
- Enable strong scaling with multiple VCIs (virtual communication interface)
- Multi-VCI for Point-to-point implemented in 3.4.0
- Multi-VCI for RMA added in 4.0
- Multi-VCI for Active Messages in 4.0

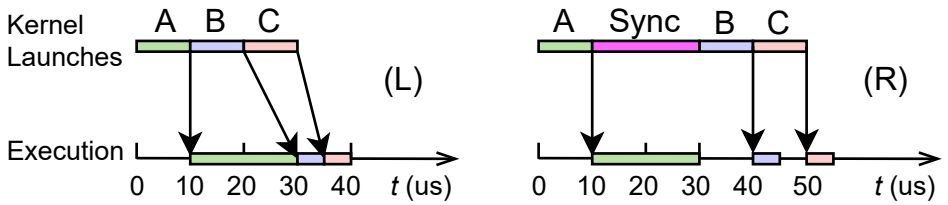
- Parallel semantics based on `communicator/rank/tag`
- Explore MPIX for direct threading semantics



GPU-STREAM-AWARE MPI

- **Mismatch between MPI communication and GPU computation**
- **MPI routines do not take a stream argument and do not know**
  - Which stream the send data is produced on
  - Which stream the receive data will be consumed on
- **Syncing with stream to do MPI can be inefficient**
  - Launching/Sync cost
  - Missed opportunity for computation/communication overlapping

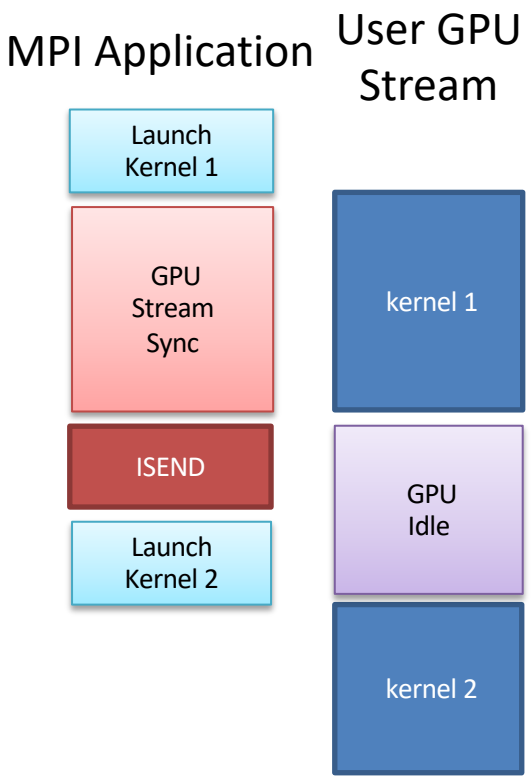




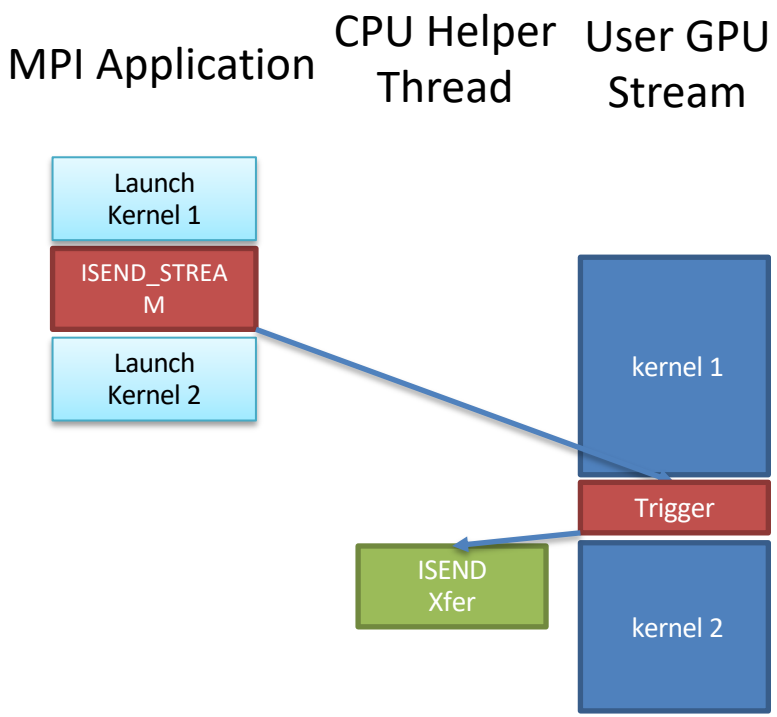
GPU-STREAM-TRIGGERED MPI OPERATION

- **Allowing point-to-point MPI to be "prepared and enqueued"**
- **GPU stream triggers transmission**
- **GPU-stream-aware interface**

ISEND in MPI today

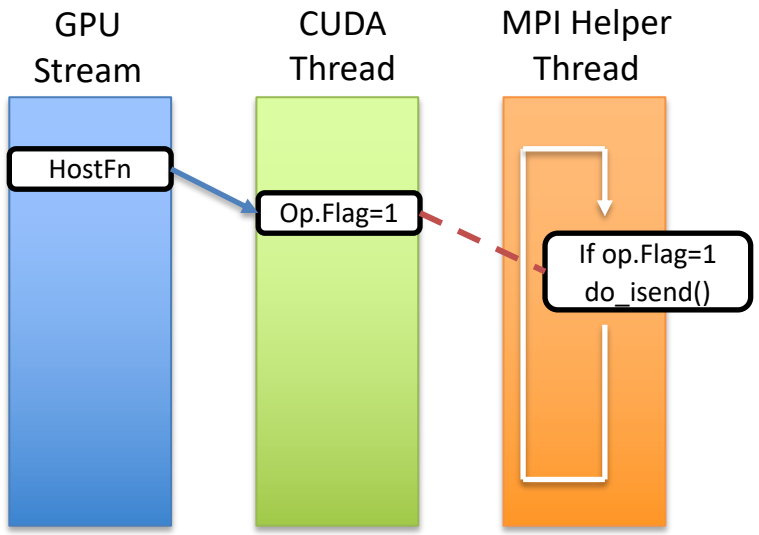


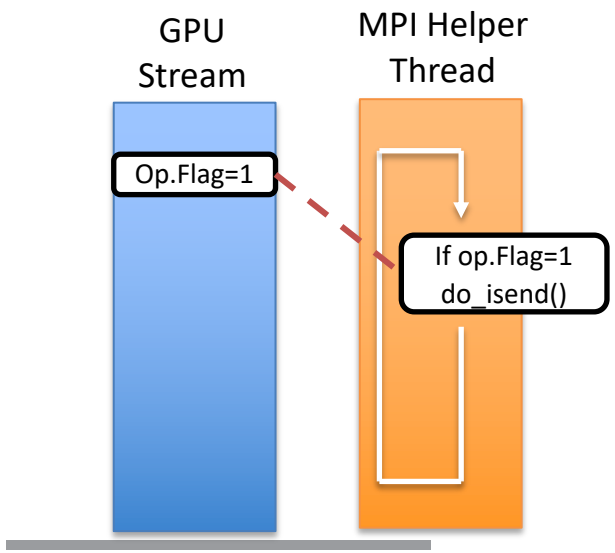
ISEND in Stream-aware MPI today



DIFFERENT WAYS OF TRIGGERING

- **Triggering Process is Essentially a Lightweight CPU-GPU Synchronization**
  - SET: GPU stream triggers ops on CPU
  - WAIT: GPU stream waits ops on CPU
  - Can also do a CPU-GPU BARRIER(-like) operation on theory
- **3 Ways for Implementation**
  - Launch Host Function / Stream Callback Function – NVIDIA, AMD
  - GPU Kernels – NVIDIA, AMD, Intel
  - Stream Mem OP (CUDA Driver API, Kernel Driver Option Required) - NVIDIA



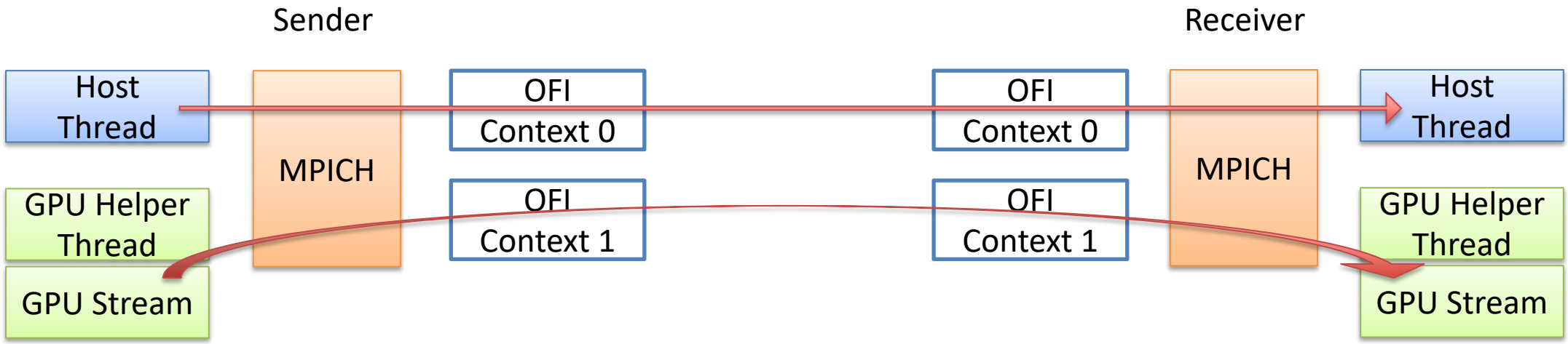


do_isend() can be
1. Entire MPI_Isend()
2. OFI Triggered Operation

We thank Jim Dinan @ NVIDIA for his suggestions and insights.

GPU-STREAM-AWARE MPI AND MPI + THREADS

- **GPU-Stream-aware MPI is multi-threaded MPI**
  - Generally MPI_THREAD_MULTIPLE
  - Can optimize to work with MPI_THREAD_SERIALIZE
- **Performance Consideration**
  - Contention between host thread and helper thread, or between multiple helper threads
  - Can utilize multiple VCIs for optimization



MPIX_ EXTENSION TODAY

- **MPIX_Stream**
  - A serialized context – all operation from this context is serialized
  - Can represent a GPU stream, or even a CPU thread
- **MPIX_Stream_comm_create(comm, user_strm, *stream_comm);**
  - Collectively create a stream comm
  - Each rank associate one stream to the comm, MPIX_STREAM_NULL for not using stream
  - Exchange VCI info at comm creation
- **MPIX_Stream_comm_create_multiplex(comm, user_strm, *stream_comm);**
  - Shared stream comm between multiple MPIX_streams
- **MPI_{Send,Recv,Isend,Irecv}_enqueue(…, stream_comm, …);**
  - Enqueues operation to the stream associated with the stream comm
- **MPI_{Wait,Waitall}_enqueue();**
  - Blocks the stream until WAIT completes

https://github.com/pmodels/mpich/discussions/5908

EXAMPLE

Sender

```
cudaStreamCreate(&cuda_st);
MPIX_Stream(cuda_st, &mpix_st);
MPIX_Stream_comm_create(MPI_COMM_WORLD, mpix_st, &stream_comm);
…
MPIX_Isend_enqueue(…, stream_comm, &req);
MPIX_Wait_enqueue(&req, MPI_STATUS_IGNORE);
…
cudaStreamSynchronize(cuda_st);
```

Receiver

```
cudaStreamCreate(&cuda_st);
MPIX_Stream(cuda_st, &mpix_st);
MPIX_Stream_comm_create(MPI_COMM_WORLD, mpix_st, &stream_comm);
…
MPIX_Irecv_enqueue(…, stream_comm, &req);
MPIX_Wait_enqueue(&req, MPI_STATUS_IGNORE);
…
cudaStreamSynchronize(cuda_st);
```

NEXT STEPS

- **MPIX_Stream planned in MPICH-4.1 series (4.1a1 coming soon)**
  - Optional, can be disable at configure time
  - A prototype to understand how user would use it
- **API can change in the future**
  - May change based on users' response
- **Proposal to the MPI-Forum Hybrid Programming Model WG**
- **Expanding support for other GPU vendors**
- **OFI Triggered Operation**
  - Keeping the triggered work to the minimal for maximum overlapping

2022 OFA Virtual Workshop

THANK YOU!

Yanfei Guo, Assistant Computer Scientist

Argonne National Laboratory