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](http://www.mpich.org)
MPICH aims to be the preferred MPI implementation on the top machines in the world
Our philosophy is to create an “MPICH Ecosystem”
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

- Application
- MPI Interface
- MPI Layer
  - Machine-independent Collectives
  - Derived Datatype Management (Yaksa)
  - Group Management

Abstract Device Interface (ADI)

CH4
- CH4 Core
  - Architecture-specific Collectives
  - Active Message Fallback
  - GPU Support Fallback

Netmods
- OFI
- UCX

Shmmods
- POSIX
- XPMEM
- GPU IPC

Legacy
CH3

© OpenFabrics Alliance
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
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

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

Other Improvements
- Reduced Overhead for CPU buffer in a GPU-enabled MPICH build
- Avoiding unnecessary buffer location checking

The GPU support in MPICH is developed in close collaboration with vendor partners including including AMD, Cray, Intel, Mellanox and NVIDIA.
Enable strong scaling with multiple VCIs (virtual communication interface)
- Multi-VDI for Point-to-point implemented in 3.4.0
- Multi-VDI for RMA added in 4.0
- Multi-VDI 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

<table>
<thead>
<tr>
<th>Stream</th>
<th>Produce</th>
<th>Consume</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream1</td>
<td></td>
<td><img src="image1" alt="Kernel Launches" /></td>
</tr>
<tr>
<td>stream2</td>
<td></td>
<td><img src="image2" alt=" Execution" /></td>
</tr>
<tr>
<td>stream3</td>
<td>User has to sync the device to make sure data is ready for MPI to send</td>
<td>MPI has to sync the device to make sure receive data is ready to be consumed on any stream</td>
</tr>
</tbody>
</table>
- Allowing point-to-point MPI to be “prepared and enqueued”
- GPU stream triggers transmission
- GPU-stream-aware interface

**GPU-STREAM-TRIGGERED MPI OPERATION**

**ISEND in MPI today**

- User GPU Stream
- Launch Kernel 1
- GPU Stream Sync
- ISEND
- Launch Kernel 2

**MPI Application**

**ISEND in Stream-aware MPI today**

- CPU Helper Thread
- User GPU Stream
- Launch Kernel 1
- ISEND_STREAM
- Launch Kernel 2
- Trigger
- ISEND_Xfer

**MPI today**

- Kernel 1
- Kernel 2
- Kernel 1
- Kernel 2

© OpenFabrics Alliance
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

```
if op.Flag=1
do_isend()
```

`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,I.send,I.recv}_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
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);

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
THANK YOU!

Yanfei Guo, Assistant Computer Scientist

Argonne National Laboratory