MPICH for Exascale

Approved for public release

Yanfei Guo[1], Kenneth Raffenetti[1], Rob Latham[1], Marc Snir[2], Hui Zhou[1], Travis Koehring[1], Sudheer Chunduri[1], Xiaodong Yu[1], Rajeev Thakur[1]

1. Argonne National Laboratory

2. University of Illinois Urbana-Champaign

EXASCALE COMPUTING PROJECT

National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

Agenda

- [5 mins] Welcome - Ken Raffenetti
- [30 mins] MPICH Update - Yanfei Guo & Hui Zhou
  - Overview and 4.0, 4.1 release updates
- Partner updates

  [12 mins] Intel (Gengbin Zheng)
  [12 mins] MVAPICH2 (Hari Subramoni)
  [12 mins] HPE (Krishna Kandalla)
  [12 mins] Pilgrim (Chen Wang – UIUC)
- Q&A

- Please submit questions via Zoom chat
  - Speakers can answer questions live, if time permits
  - Chat responses welcome throughout the event

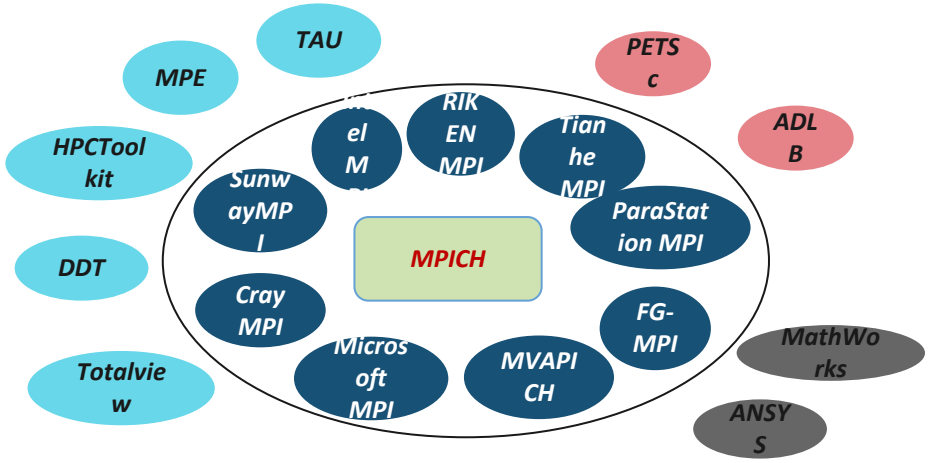
Exascale MPI (MPICH)

- Funded by DOE for 29 years
- Has been a key influencer in the adoption of MPI
  - First/most comprehensive implementation of every MPI standard
  - Allows supercomputing centers to not compromise on what features they demand from vendors
- DOE R&D100 award in 2005
- MPICH and its derivatives are the world's most widely used MPI implementations
  - Supports all versions of MPI including the recent MPI-3.1

- MPICH Adoption in US Exascale Machines
  - Aurora, ANL, USA (MPICH)
  - Frontier, ORNL, USA (Cray MPI)
  - El Capitan, LLNL, USA (Cray MPI)

MPICH is not just a software

It's an Ecosystem

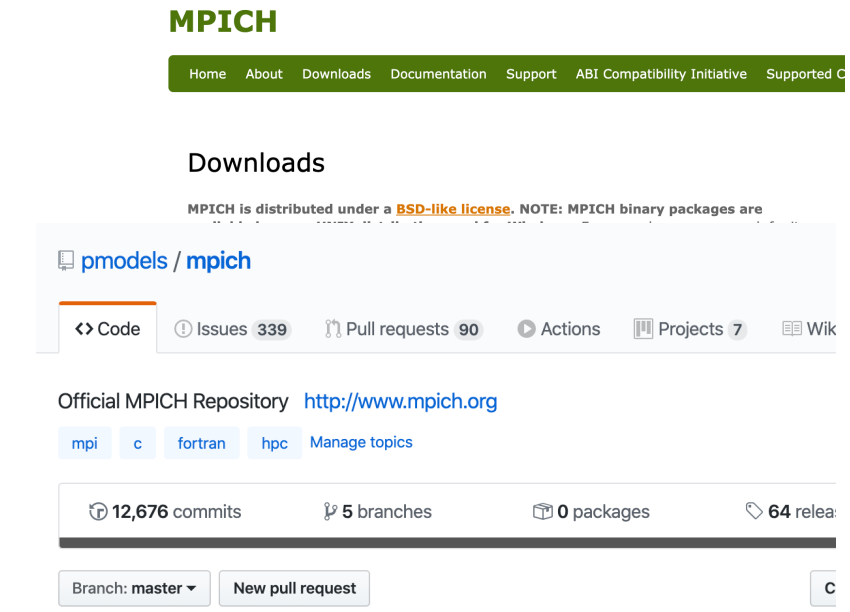


MPICH ABI Compatibility Initiative

- Binary compatibility for MPI implementations
  - Started in 2013
  - Explicit goal of maintaining ABI compatibility between multiple MPICH derivatives
  - Collaborators:
    - MPICH (since v3.1, 2013)
    - Intel MPI Library (since v5.0, 2014)
    - Cray MPT (starting v7.0, 2014)
    - MVAPICH2 (starting v2.0, 2017)
    - Parastation MPI (starting v5.1.7-1, 2017)
    - RIKEN MPI (starting v1.0, 2016)
- Open initiative: other MPI implementations are welcome to join
- http://www.mpich.org/abi

MPICH Distribution Model

- Source Code Distribution
  - MPICH Website, Github
- Binary Distribution through OS Distros and Package Managers
  - Redhat, CentOS, Debian, Ubuntu, Homebrew (Mac)
- Distribution through HPC Package Managers
  - Spack, OpenHPC
- Distribution through Vendor Derivatives



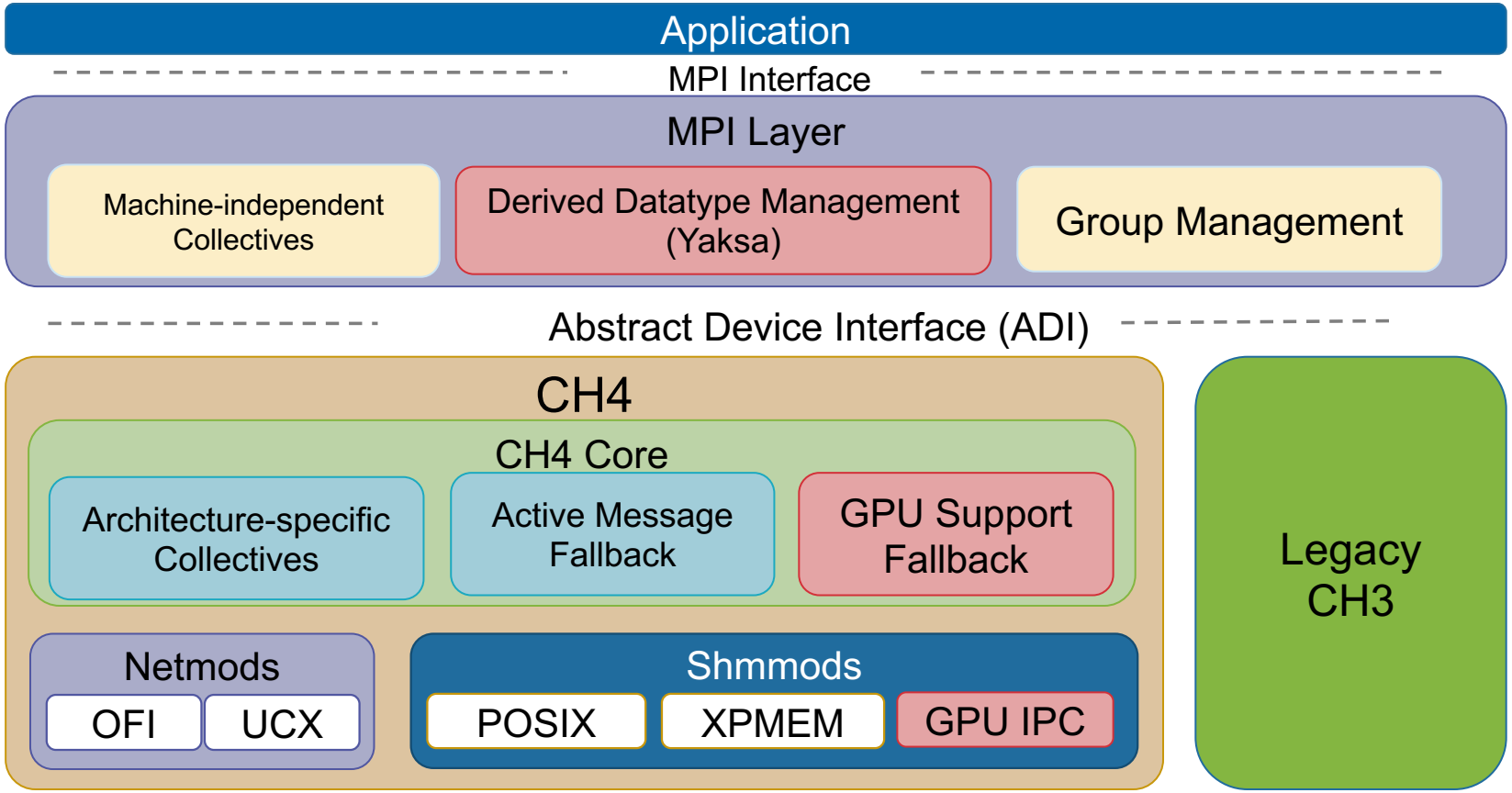
MPICH Releases

- **MPICH switched to a 12-month cycle for major releases (staring from 4.x), barring some significant releases**
  - Minor bug fix releases for the current stable release happen every few months
  - Preview releases for the next major release happen every few months
  - Branch as soon as beta release to allow vendors pick up early
- **Current stable release is in the 4.0 series**
  - mpich-4.0.2 was released in April 2022
- **Current major release is in the 4.1 series**
  - mpich-4.1a1 was released in last week

Spack Package Updates

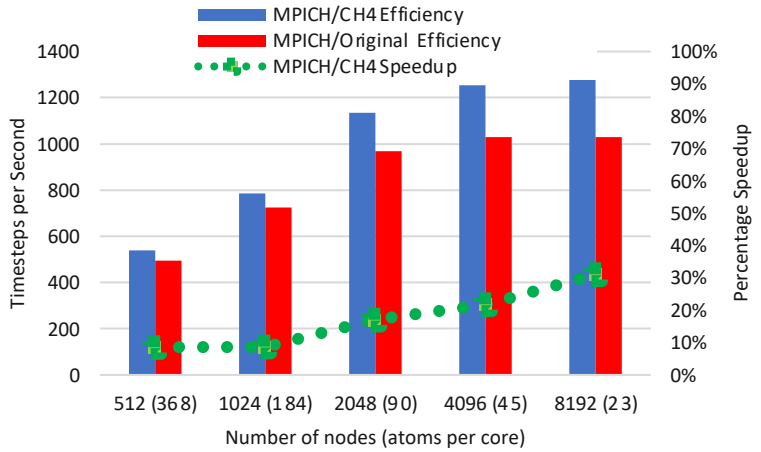
- Recently added GPU variants
  - CUDA (+cuda) supported with MPICH 3.4.x and up
  - ROCm (+rocm) supported with MPICH 4.0.x and up
  - Intel GPU variant in development
- VCI variant (+vci) for improved MPI+Thread performance
  - Supported with MPICH 4.0.x and up
- Argobots variant (+argobots) for supporting the Argobots user-level thread library
- Cray PMI variant (pmi=cray) supports running MPICH on Cray systems with aprun
- MPICH is part of the Extreme-scale Scientific Software Stack (E4S)

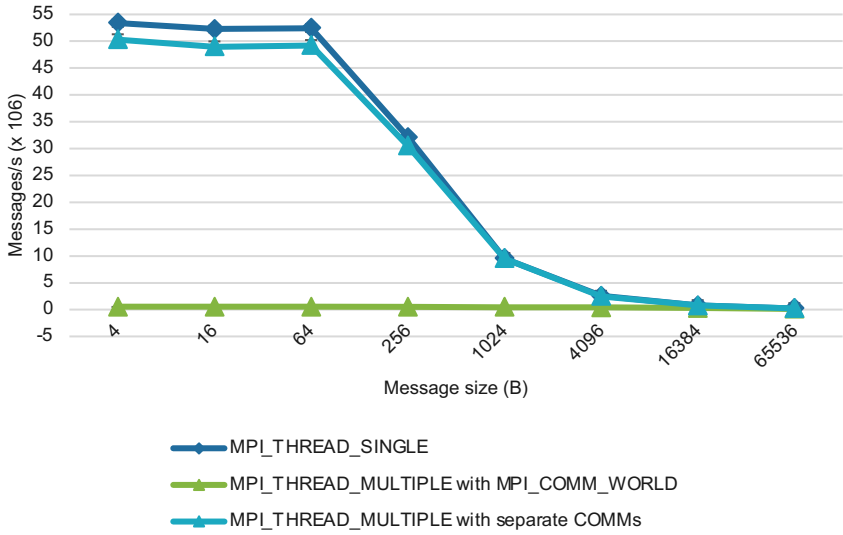
MPICH Architecture Overview



MPICH-4.0 – CH4 device

- Replacement for CH3 as default option, CH3 still maintained, but new features are implemented only in CH4
- Low-instruction count communication
  - Ability to support high-level network APIs (OFI, UCX)
  - E.g., tag-matching in hardware, direct PUT/GET communication
- VCI feature to support high thread concurrency
  - Improvements to message rates in highly threaded environments (MPI_THREAD_MULTIPLE)
  - Support for multiple network endpoints (THREAD_MULTIPLE or not)
- GPU-aware
  - CUDA, HIP, ZE
  - IPC, GPU Direct RDMA





The CH4 in MPICH is developed in close collaboration with vendor partners including Including AMD, Cray, Intel, Mellanox and NVIDIA

MPICH-4.0 – Full support for MPI-4 standard

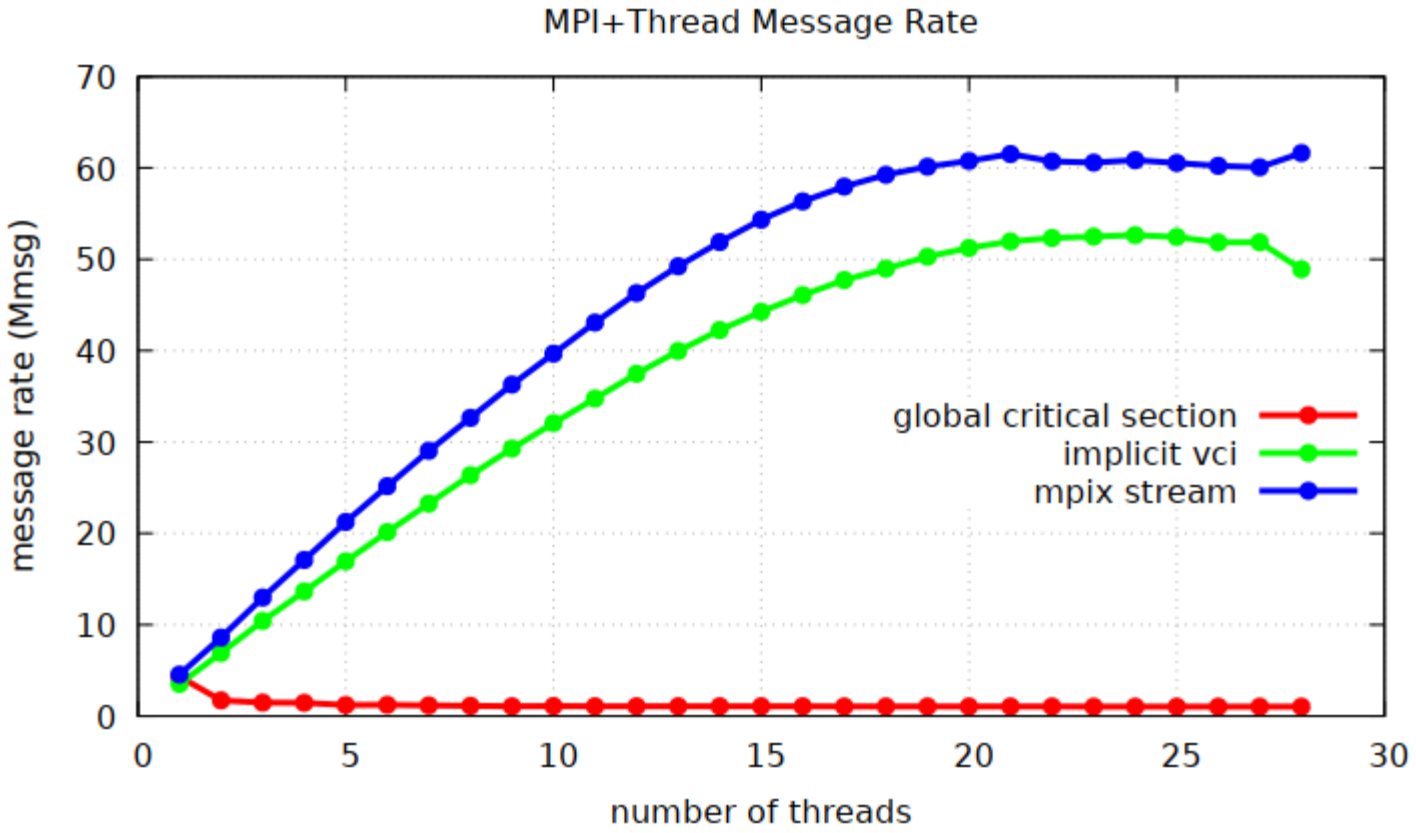
- MPI Forum released MPI 4.0 standard on June 9, 2021

- Major additions in MPI 4.0
  - Solution for “Big Count” operations
    - Use, e.g. `MPI_Send_c` with `MPI_Count` argument.
  - Persistent Collectives
    - For example, `MPI_Bcast_init`
  - Partitioned Communication
    - Splitting either send buffers or receive buffers into partions
    - Allow partial data transfers
  - MPI Sessions
    - A mother of all possibilities
  - New tool interface for events
    - Callback-driven event information
  - More: improved error handling, better `MPI_Comm_split_type`, standardized info hint assertions, improved info usages

The development is done in close collaboration with vendor partners including Including AMD, Cray, Intel, Mellanox and NVIDIA

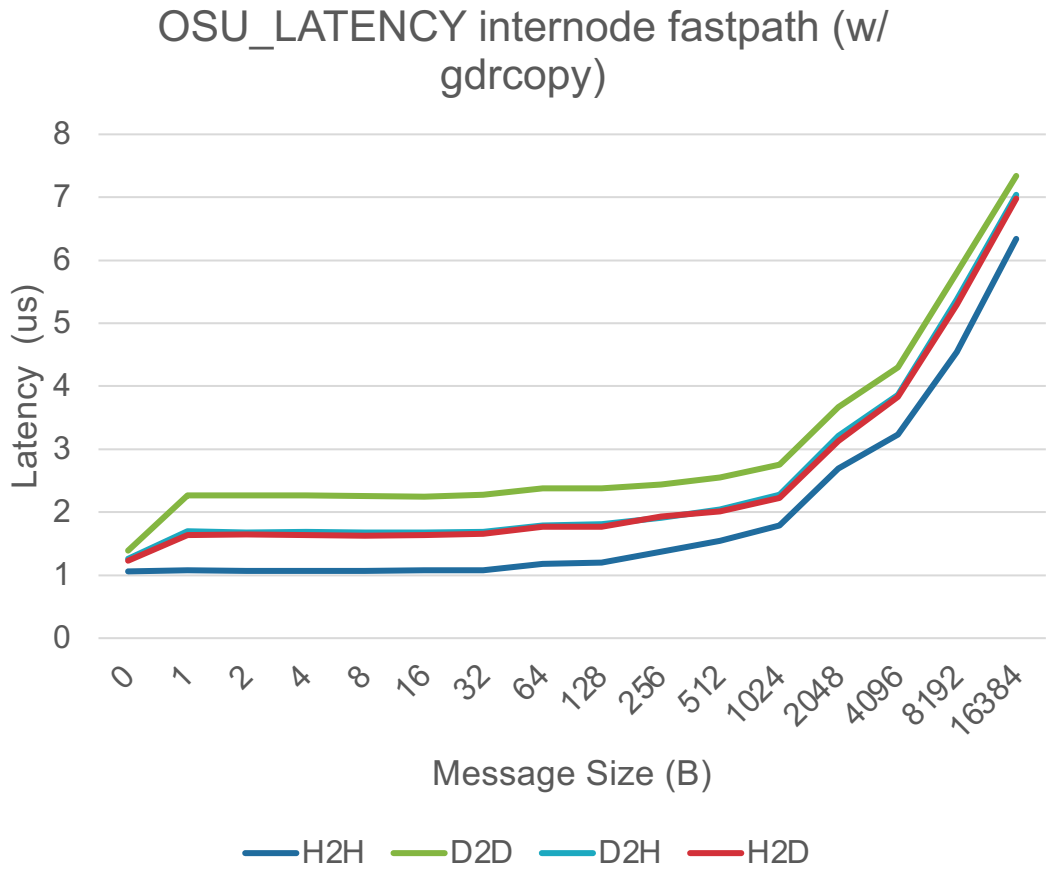
MPI+THREAD

- Previously, dismal performance with MPI_THREAD_MULTIPLE
- Implicit VCI mapping in MPICH-4.0 with potential performance
- Advice to users
  - Use different communicators
  - Same communicator, use different tags and set hints
- Explicit VCI coming in next release



MPI+GPU

- **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
  - New GPU-aware datatype engine

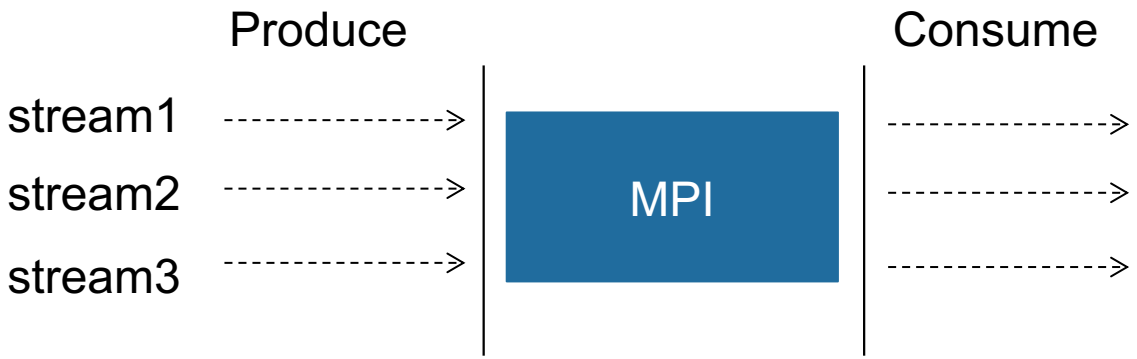


On Summit with MPICH 4.0, UCX 1.11.0, CUDA 11.4.2, GDRCOPY 2.3

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

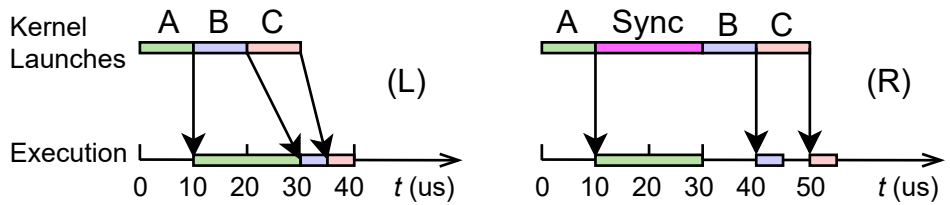
GPU-Stream-Aware MPI

- 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



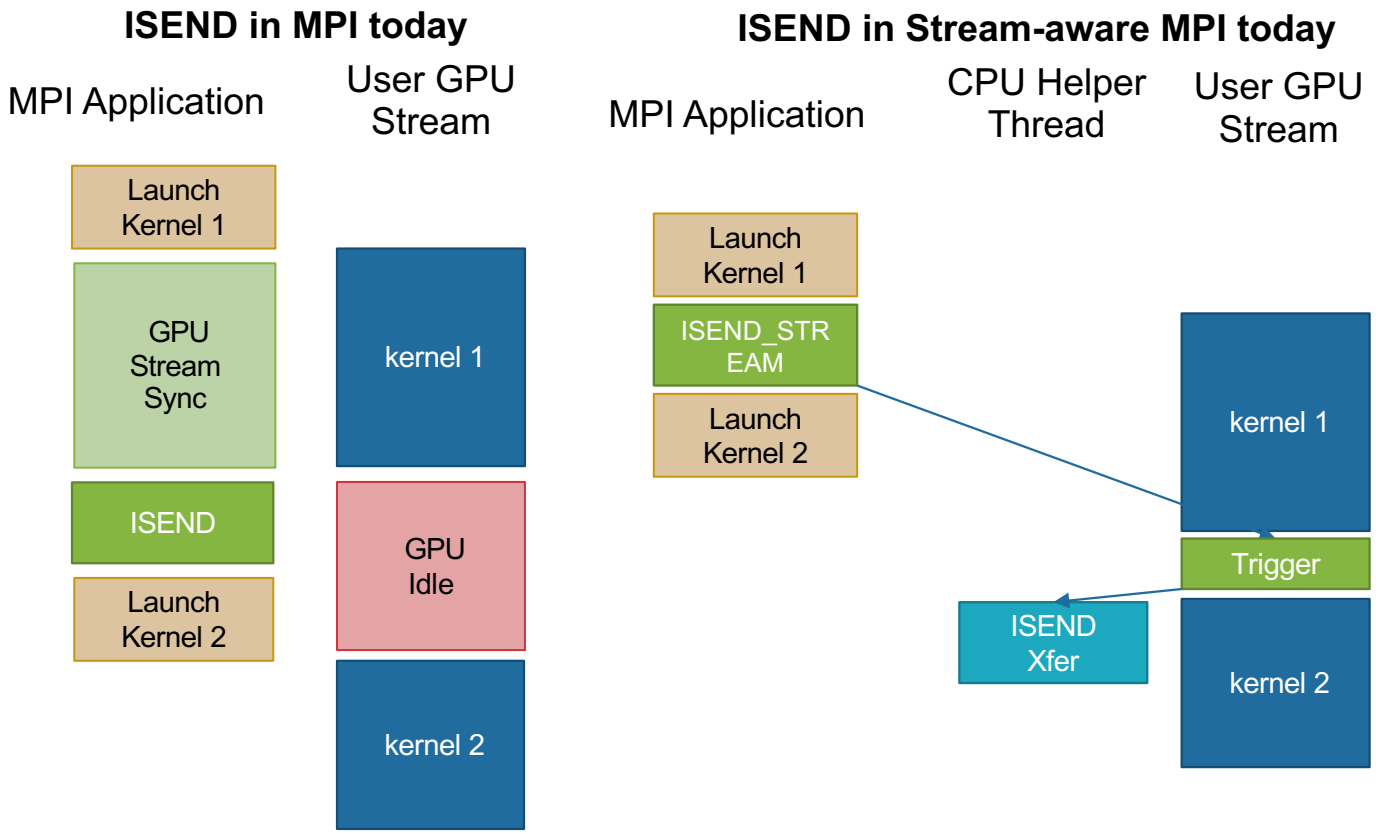
User has to sync the device to make sure data is ready for MPI to send

MPI has to sync the device to make sure receive data is ready to be consumed on any stream



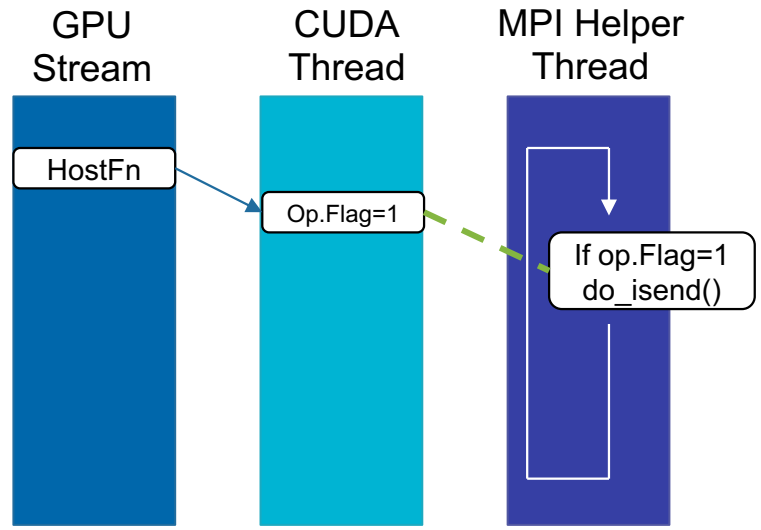
Triggering MPI Operation from GPU Streams

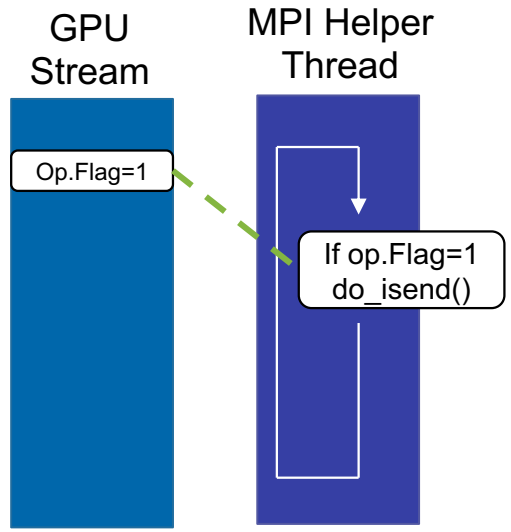
- Allowing point-to-point MPI to be “prepared and enqueued”
- GPU stream triggers transmission
- GPU-stream-aware interface



Different Ways of Triggering

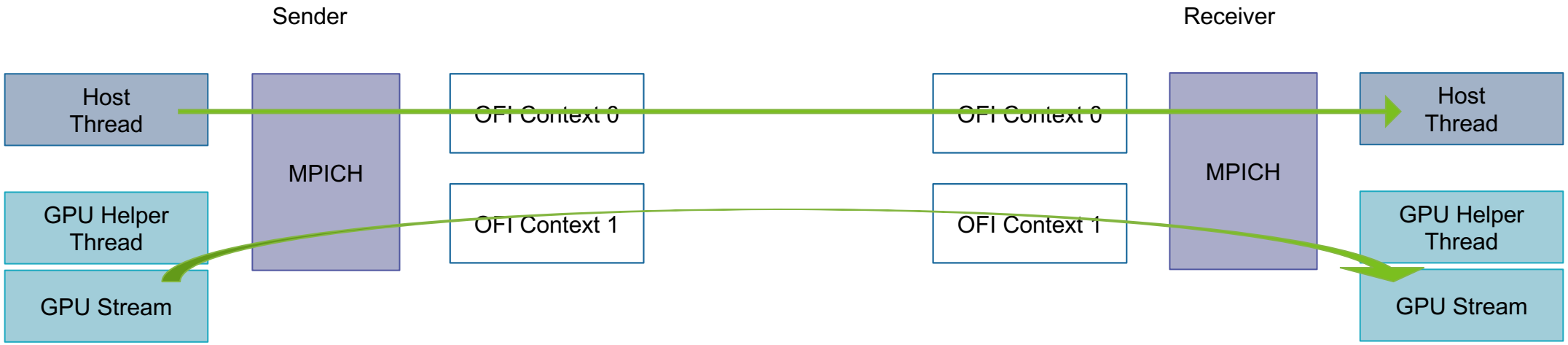
- Triggering Process is Essentially a Lightweight CPU-GPU Synchronization
  - SET: GPU stream triggers ops on CPU
  - WAIT: GPU stream waits ops on CPU
  - Can also do a CPU-GPU BARRIER(-like) operation on theory
- 3 Ways for Implementation
  - Launch Host Function / Stream Callback Function – NVIDIA,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





GPU-Stream-Aware MPI is Multi-Threaded MPI

- 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



MPICH 4.1 Release Series

MPICH 4.1 Release Series

- MPICH Testsuite
  - Comprehensive testsuite for MPI implementations in general
  - Now available as separate release target
- Accelerate CI builds
  - CI is key for productivity, we do hundreds of CI builds daily
  - Projects are getting more complex, and slower to build
  - Option to prebuild submodules, `./autogen.sh -quick` to avoid repeated rebuild
- MPIX Stream prototype
- Standardize PMI interface

MPIX Stream – the missing link in MPI+X

- MPI+Thread
  - MPI is a process execution model
  - "*When a thread is executing one of these routines, if another concurrently running thread also makes an MPI call, the outcome will be as if the calls executed in some order*"
  - If application expresses parallelism "correctly", implementations can reserve concurrency
  - How do you do so when MPI does not have thread concept? That is a good question!
- MPI+GPU
  - Accelerator runtime introduces yet another execution context, e.g. CUDA stream
  - It is an always async, serial execution context
  - It is critical for minimizing the CPU/GPU launching and synchronization cost
  - How do we pass the GPU stream into MPI?
  - What happens when we mix conventional MPI calls with MPI operations enqueued to a GPU stream?

MPIX Stream -- proposal

- **`int MPIX_Stream_create(MPI_Info info, MPIX_Stream *stream)`**
  - `MPI_INFO_NULL` is OK
  - For CUDA stream – `MPI_Info_set(info, "type", "cudaStream_t");`
    `MPIX_Info_set_hex(info, "value", &stream, sizeof(stream));`
- **`int MPIX_Stream_comm_create(MPI_Comm oldcomm, MPIX_Stream stream,`**
  `MPI_Comm *stream_comm)`
- Use stream_comm normally for MPI operations, and a local serial context applies
  - A two-way promise!
- For CUDA stream, additional "enqueue" APIs
  - **`int MPIX_{Send,Isend,Recv,Irecv,Wait,Waitall}_enqueue(…)`**
- **`int MPIX_Stream_comm_create_multiplex(oldcomm, n, streams[],`**
  `&multiplex_comm)`
  - **`MPIX_Stream_{Send,Isend,Recv,Irecv}(…, src_stream_idx, dst_stream_idx)`**

Code Example

```
MPI_Info_create(&info);
MPI_Info_set(info, “type”, “cudaStream_t”);
MPIX_Info_set_hex(info, “value”, &cuda_stream, sizeof(cuda_stream));
MPIX_Stream_create(info, &mpi_stream);

MPIX_Stream_comm_create(MPI_COMM_WORLD, mpi_stream, &stream_comm);

if (rank == sender_rank) {
    /* … */
    MPIX_Send_enqueue(…, stream_comm);
    /* … */
} else if (rank == receiver_rank) {
    /* … */
    MPIX_Irecv_enqueue(…, stream_comm, &req);
    /* … */
    MPIX_Wait_enqueue(&req, &status);
    /* … */
}
cudaStreamSynchronize(cuda_stream);
```

Updating PMI -- issues

- PMI remain an internal component in MPICH
- Supporting both PMI-1 and PMI-2 is confusing
  - PMI-1 is still the default in MPICH/Hydra, well tested
  - PMI-2 remain experimental, not feature-complete, less stable
  - Slurm documents PMI-2, but supports PMI-1
  - Cray supports PMI-2
- Interest in using PMI/Hydra independently from MPICH
  - PMI interface is a universal interface works everywhere MPI works
  - Hydra is a robust and versatile launcher
  - PMI/Hydra works well for multi-process runtimes, e.g. OpenSHMEM, NVSHMEM
- Need to extend PMI/Hydra to support modern PMI features
  - To (partially) support PMIx

Updating PMI -- available in MPICH-4.1a1

- Better configure options
  - `--with-pmi={pmi1,pmi2,pmix}`
  - `--with-pmilib={mpich,slurm,cray,pmix}`
  - `--with-pm={no,hydra,gforker,remshell}`
  - `--with-pmi={slurm,cray}` also works
- Separate release targets
  - `pmi-4.1a1.tar.gz` and `hydra-4.1a1.tar.gz`
- Standard PMI interfaces
  - Third party PMI implementation should support the same `pmi.h` and `pmi2.h`
- Internal refactoring
  - PMI-1 and PMI-2 are internally unified
  - Wire protocol layer and semantic layer are separated

Updating PMI – future plans

- Extend PMI-1 and PMI-2 to a superset
  - PMI-1 backward compatible
  - PMI-2 feature compatible, backward compatible with function aliases or thin wrappers
  - Independent wire protocols
- Deprecating PMI-2
  - Just `PMI_` prefix and `#include <pmi.h>` and `libpmi.so`
  - Always backward compatible
  - New API extensions tracked by `PMI_VERSION` and `PMI_SUBVERSION`
- Extend PMI toward PMIx
  - KVS scopes
  - KVS value types, in particular, binary values
  - Predefined/reserved KVS keys with `PMI_` prefix

MPICH 4.1 Release Plan

- MPICH-4.1 alpha
  - Released last week – 4.1a1
- MPICH 4.1 beta
  - Planed 11/2022
  - Development for MPICH 4.2 start
- MPICH 4.1 GA
  - End of 2022 or early 2023

Keep In Touch With Us

- MPICH Development Update Meeting
  - Every Thursday 9am central
  - Microsoft Teams
- Mailing list: discuss@mpich.org
- GitHub: https://github.com/pmodels/mpich
  - Report issues
  - Contribute pull requests
  - Join discussions

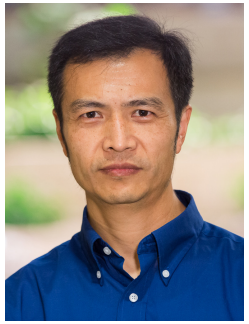
Current MPICH Team

Yanfei Guo
Project Lead

Rajeev Thakur

Marc Snir

Kenneth Raffenetti

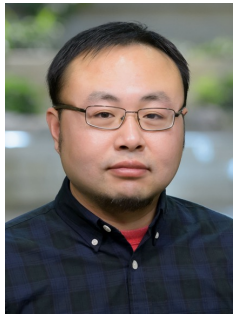


Hui Zhou

Robert Latham

Sudheer Chunduri

Travis Koehring



Xiaodong Yu