MPICH: Status and Upcoming Releases
http://www.mpich.org

Yanfei Guo

Assistant Computer Scientist

Argonne National Laboratory

MPICH turns 30

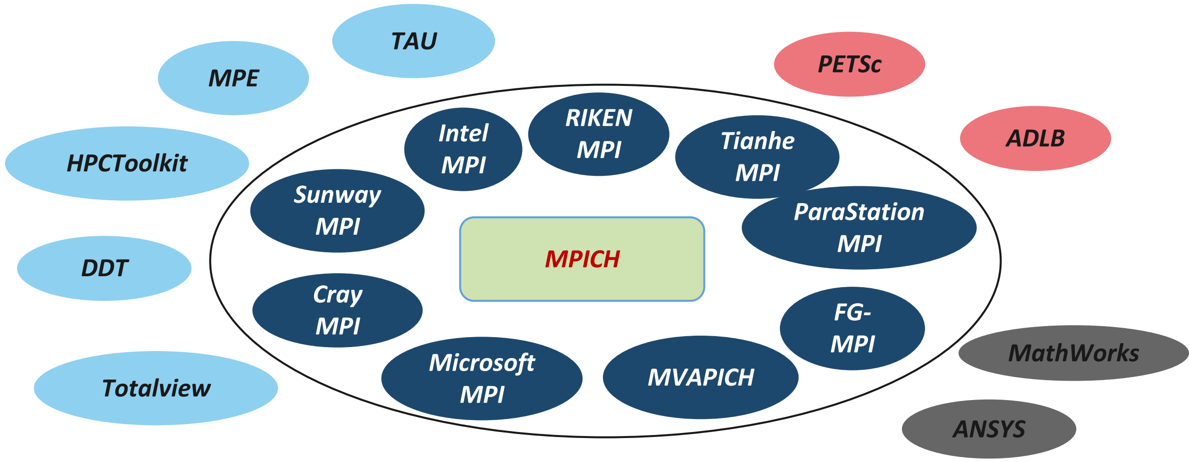
The MPICH Project

- Funded by DOE for 30 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 for MPICH
- DOE R&D100 award in 2019 for UCX (MPICH internal comm. layer)
- MPICH and its derivatives are the world's most widely used MPI implementations
- **Adoption in US Exascale Systems**

  Aurora, ANL, USA (MPICH)

  Frontier, ORNL, USA (Cray MPI)

  El Capitan, LLNL, USA (Cray MPI)



MPICH is not just a software

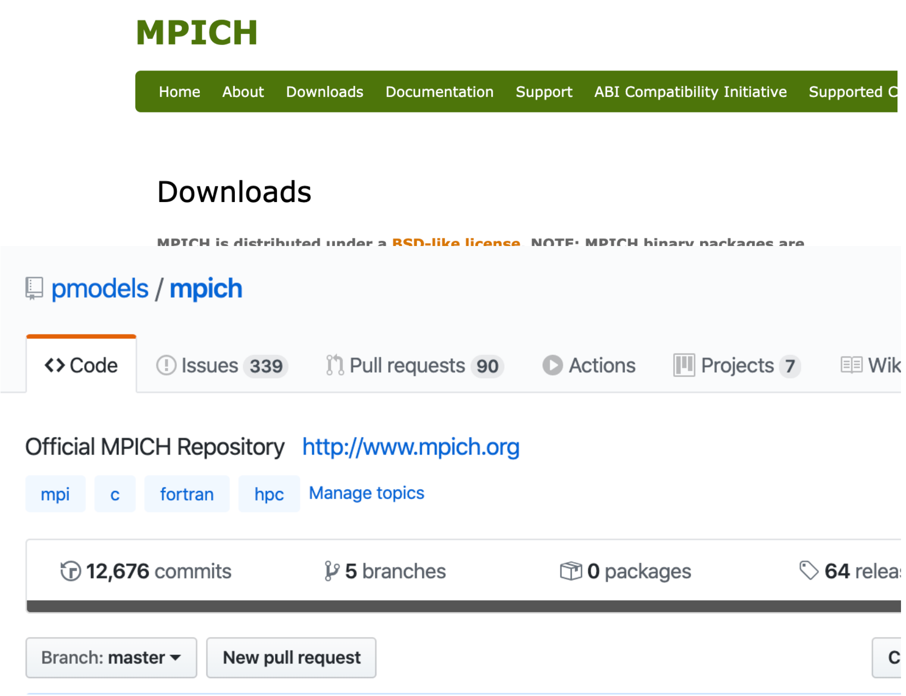
It's an Ecosystem

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)
- Open initiative: other MPI implementations are welcome to join
- http://www.mpich.org/abi

MPICH Distribution Model

- Source Code Distribution
  - MPICH Website, Github
  - BSD-2 compatible license
- 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

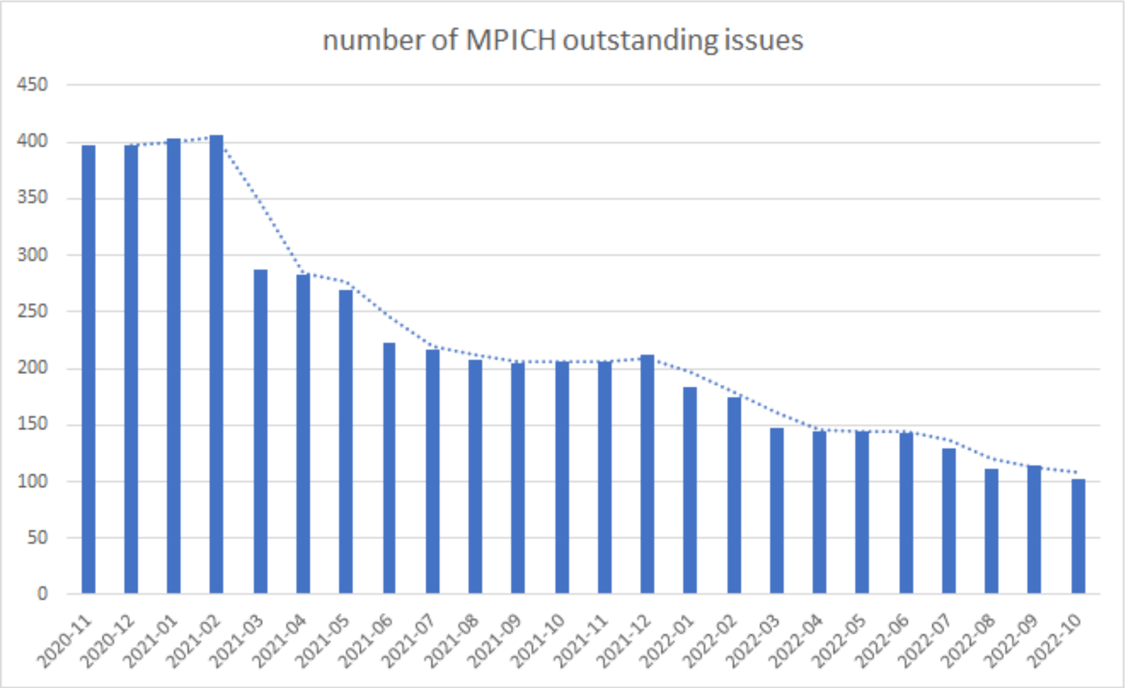


MPICH Releases

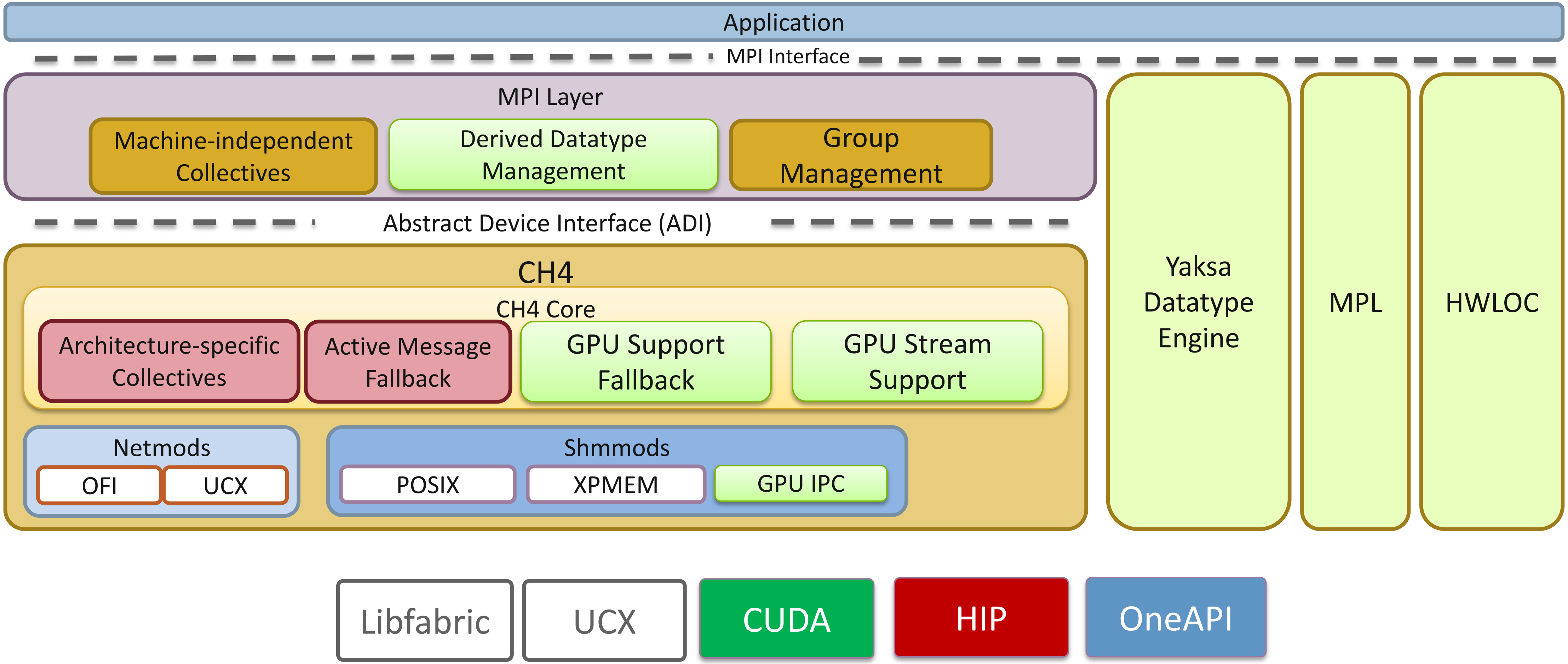
- MPICH typically follows a 12-month cycle for major releases (3.x/4.x), barring some significant releases
  - Minor bug fix releases for the current stable release happen every few months
  - Preview releases for the next major release happen every few months
- Current stable release is in the 4.0.x series
  - mpich-4.0.3 released on 11/08/2022
- Upcoming major release is in the 4.1 series
  - mpich-4.1b1 released last week
  - RC1 and GA release coming soon

Following and Participating MPICH Development

- MPICH main repo on Github at https://github.com/pmodels/mpich
- Join our development call every Thursday at 9am (central): https://bit.ly/mpich-dev-call
- Submit a Github issue
  - Github issue is the preferred way for bug reports



MPICH Layered Structure

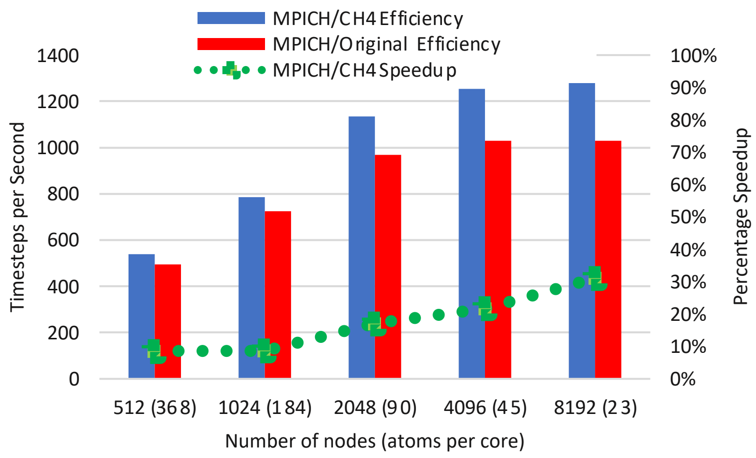


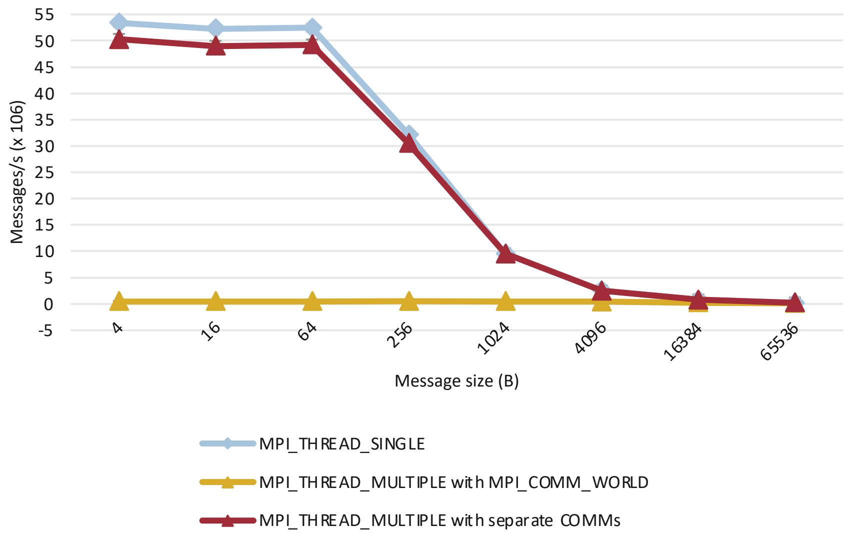
Goal of CH4 Device

- Lightweight Layer for High-Performance Hardware
  - Getting out-of-the way of high-performance hardware
  - Minimizing software overhead for HW supported operations.
- High Scalability
  - Minimizing per-process footprint
  - Scalable MPICH internal data structures
- Optimized Multi-threaded Performance
  - Reduce contentions on multi-threaded MPI
  - Multiple virtual communication interfaces (VCIs)
- Support for Heterogeneous Hardware Architecture
  - GPU Support

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

Full support for MPI-4 standard

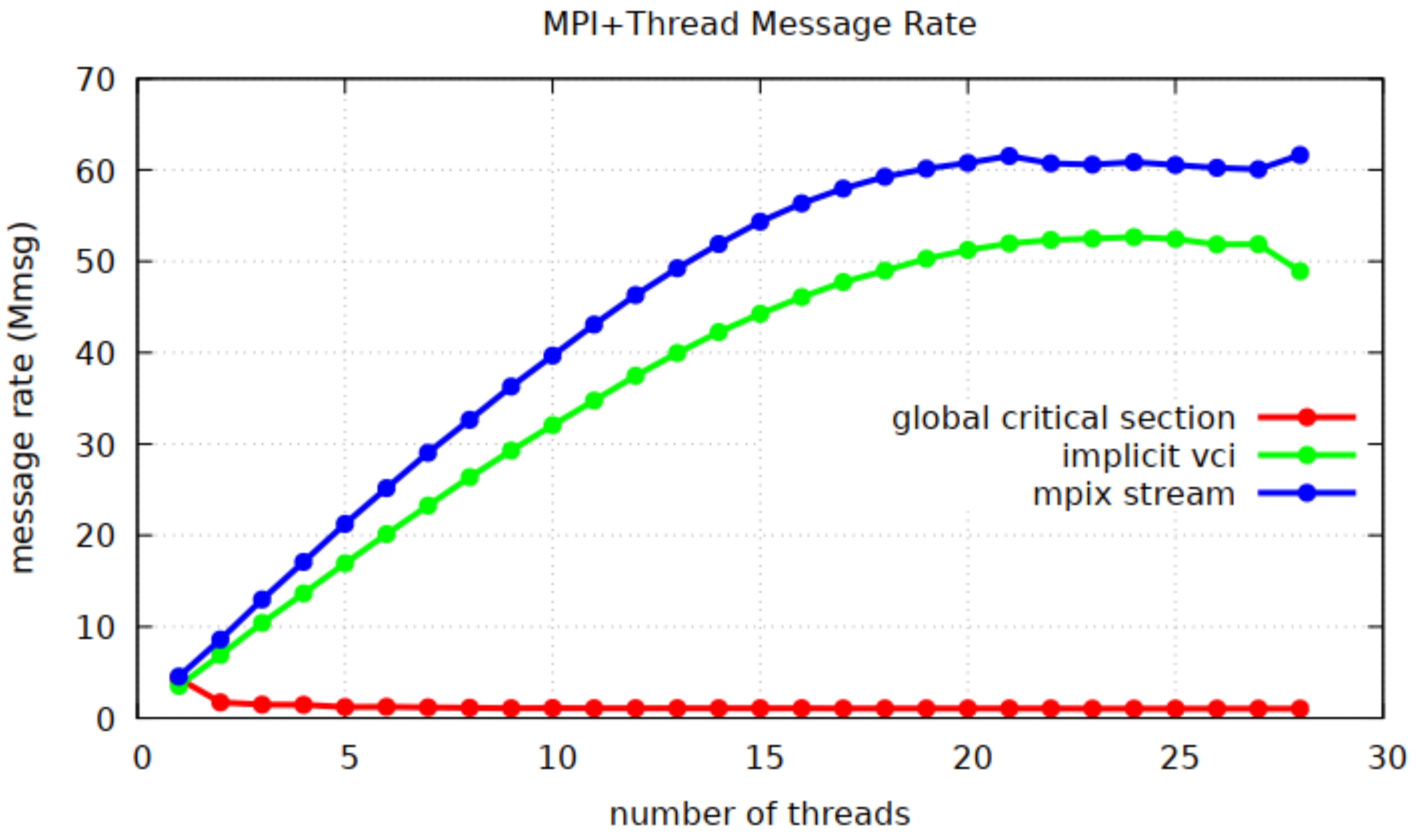
- MPI Forum released MPI 4.0 standard on June 9, 2021

- 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

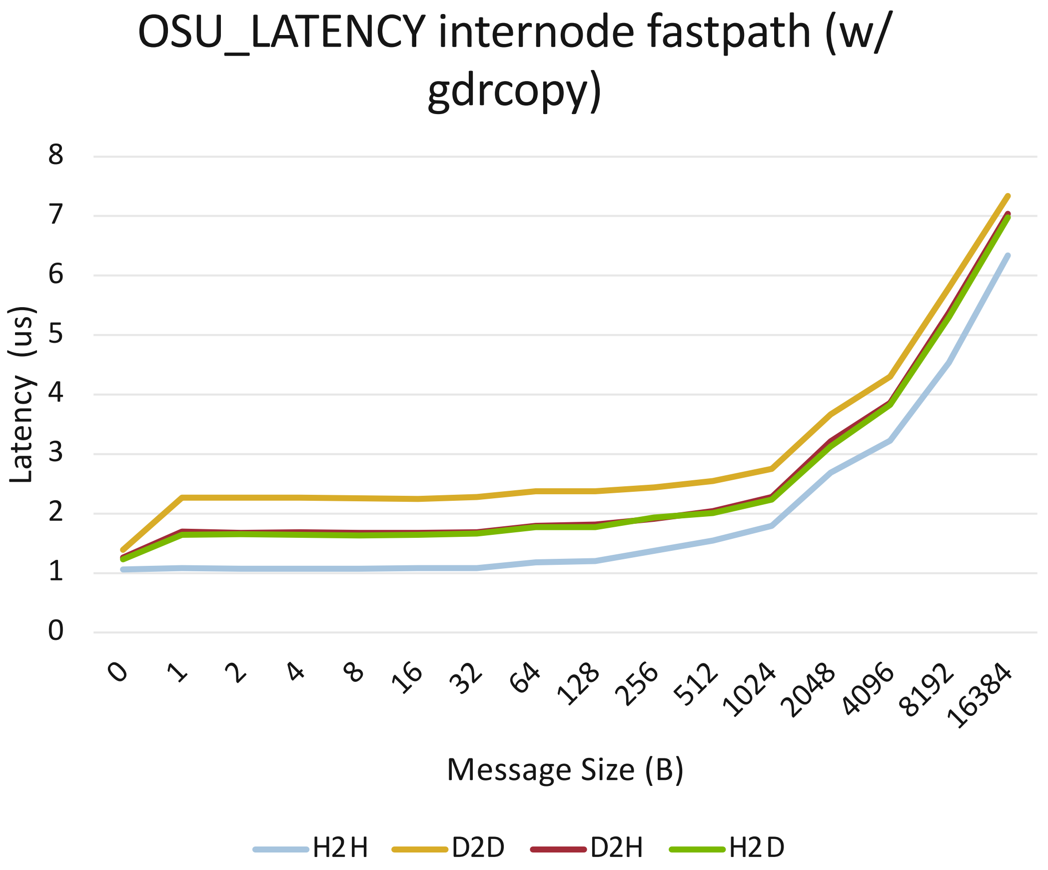
- Previously, dismal performance with MPI_THREAD_MULTIPLE
- Implicit VCI mapping in MPICH-4.0 with potential performance
- Advice to users
  - Use different communicators
  - Same communicator, use different tags and set hints
- Explicit VCI coming in MPICH-4.1



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

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



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

MPICH 4.1 Release Series

- **MPIX Stream prototype**
- **Standalone PMI Library**
- **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 – 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

- `MPIX_Stream` identifies a serial execution context

```
int MPIX_Stream_create(MPI_Info info, MPIX_Stream *stream)
int MPIX_Stream_free(MPIX_Stream *stream)
```

- `info` can be `MPI_INFO_NULL`, identifies a generic thread context
- In the case of threads, it is the application's responsibility to ensure access to an MPIX_Stream is serialized. Essentially MPI_THREAD_SERIAL, but at the object-level, rather than all of MPI.

Hui Zhou, Ken Raffenetti, Yanfei Guo, and Rajeev Thakur. 2022. MPIX Stream: An Explicit Solution to Hybrid MPI+X Programming. In Proceedings of the 29th European MPI Users' Group Meeting (EuroMPI/USA'22).

Stream communicator

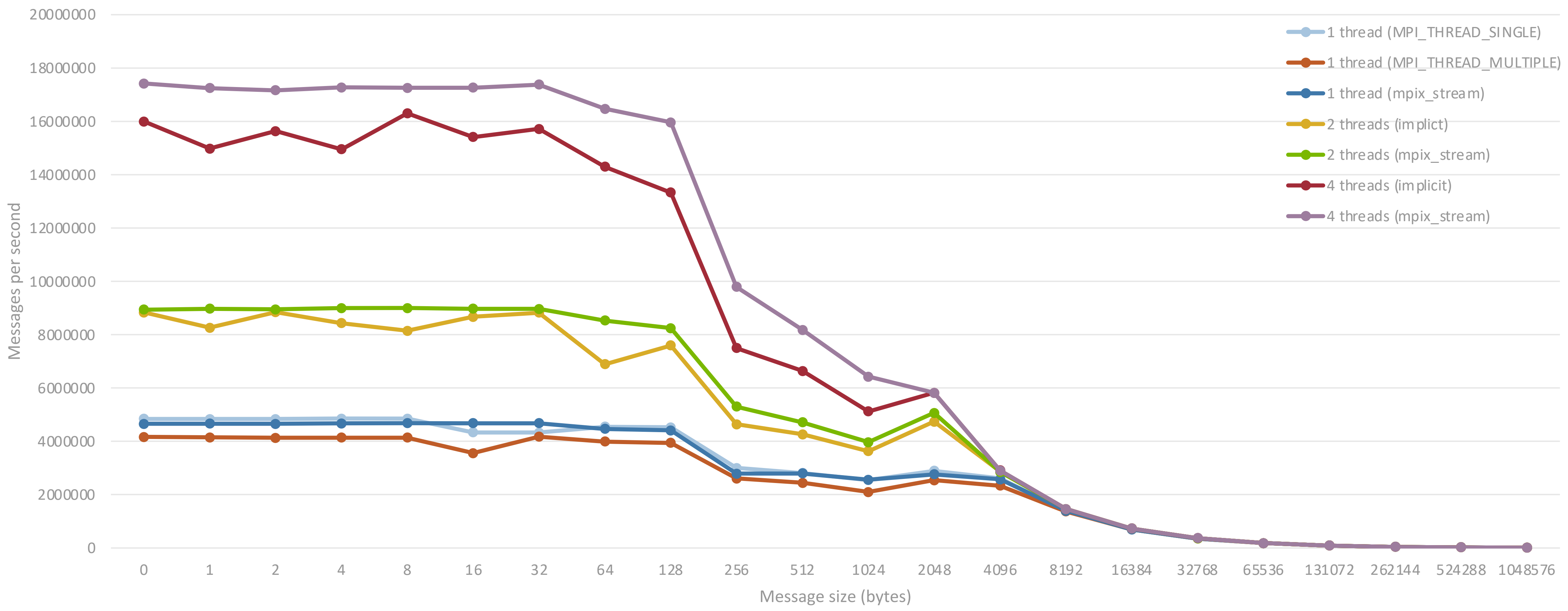
- Stream communicator is a communicator with local streams attached.

```
int MPIX_Stream_comm_create(MPI_Comm parent_comm,
        MPIX_Stream stream, MPI_Comm *stream_comm)
```

- MPIX streams are local, but communications are between pairs of them
- Otherwise, synchronization is unavoidable at receiver or sender.
- It okay for `stream` to be `MPIX_STREAM_NULL`.
- Conventional communicators are the same as stream communicators with `MPIX_STREAM_NULL` on every process.

MT.COMB Benchmark (PT2PT MSG RATE)

Intel Xeon Platinum 8180M, ConnectX-6, nodes=2, ppn=1



MPIX Stream for Progress

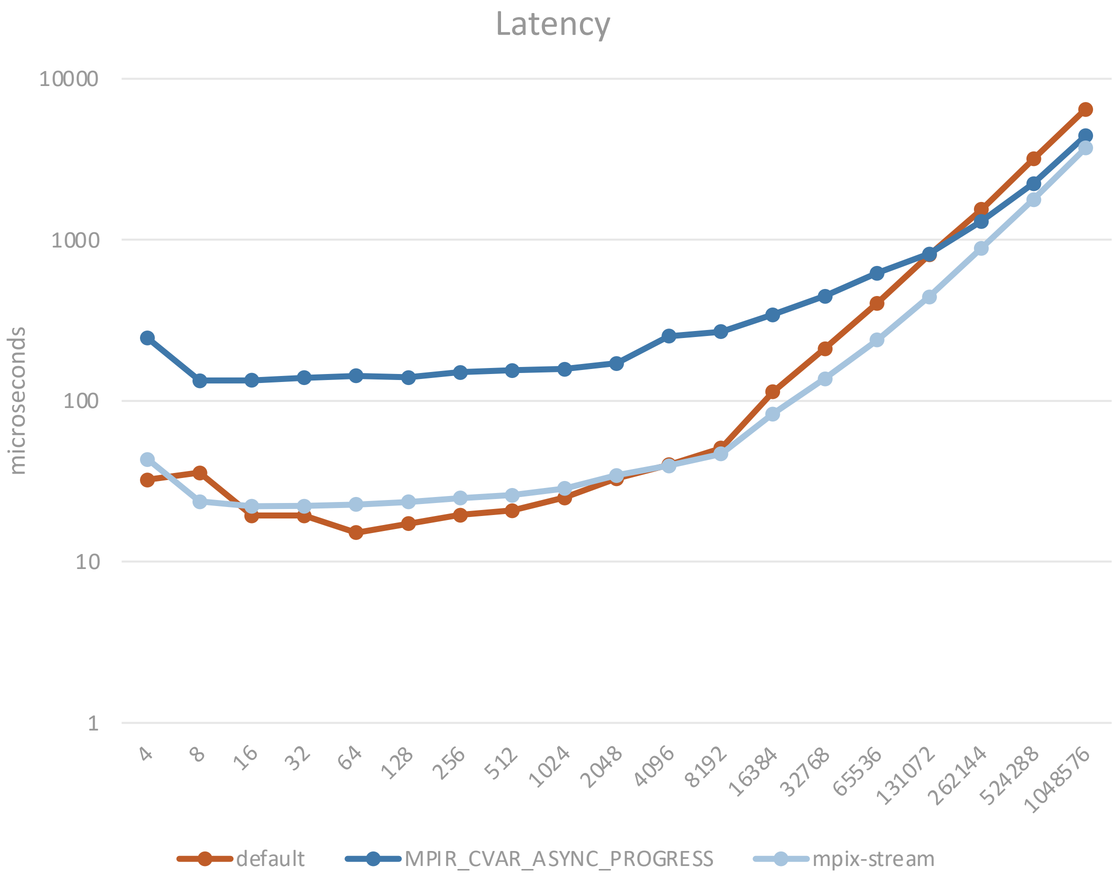
- Progress communication on a stream instead of individual request(s)

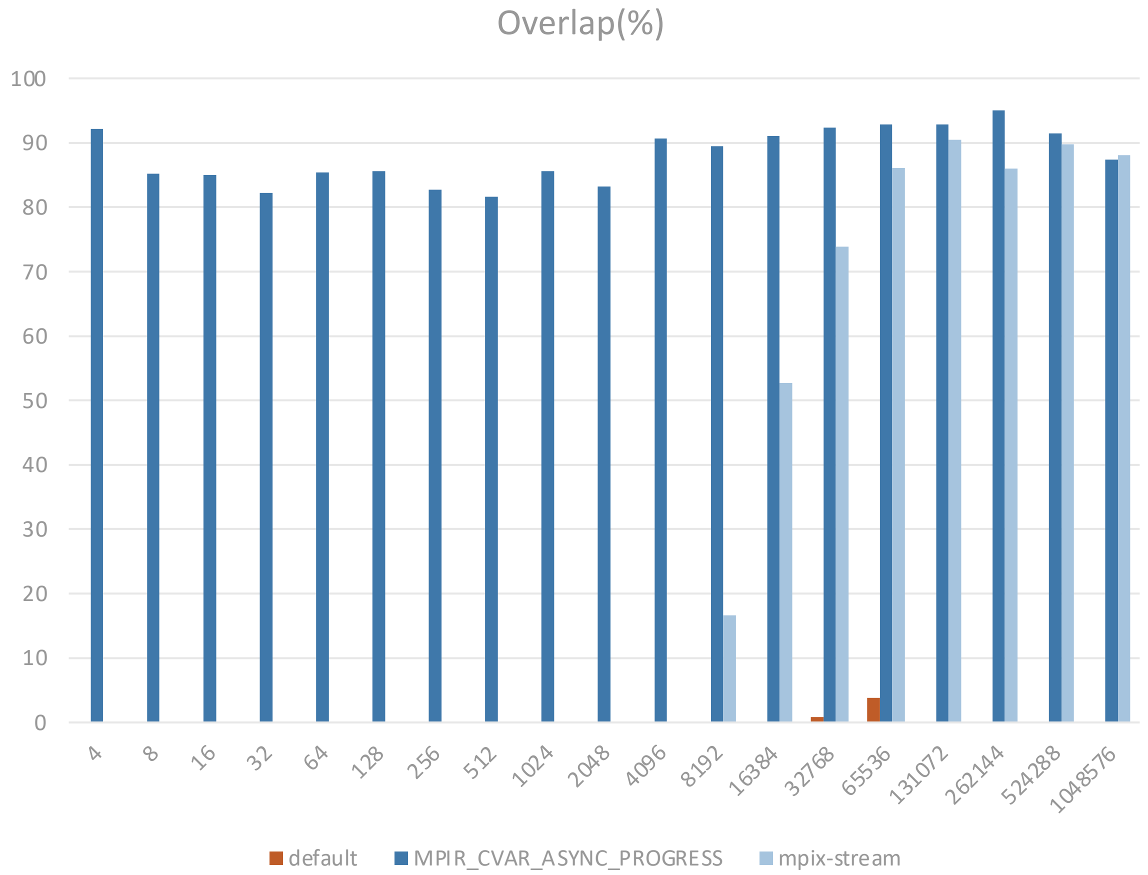
```
int MPIX_Stream_progress(MPIX_Stream stream)
```

- Application can create/join progress threads as needed
- No longer relies on `MPI_Wait/MPI_Test` for progress
- Coordinated stream progress needs no additional thread-safety from the implementation
- Progress thread does not need to be aware of outstanding requests

OSU Microbenchmarks MPI_Iallreduce + MPIX_Stream_progress

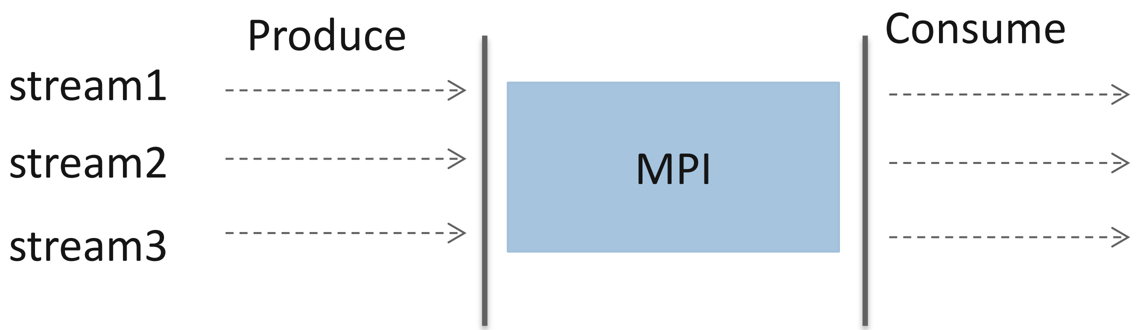
Intel Xeon Platinum 8180M, Connect-X6, nodes=2, ppn=28





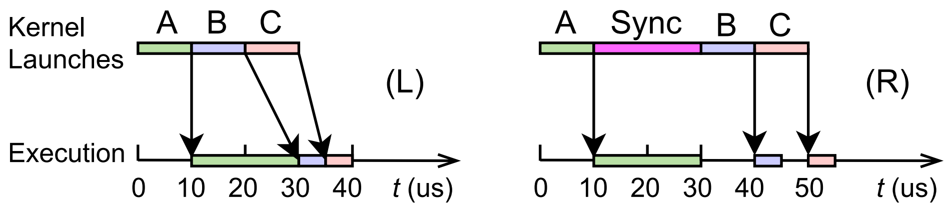
MPIX_Stream as A GPU-Stream-Aware MPI

- 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



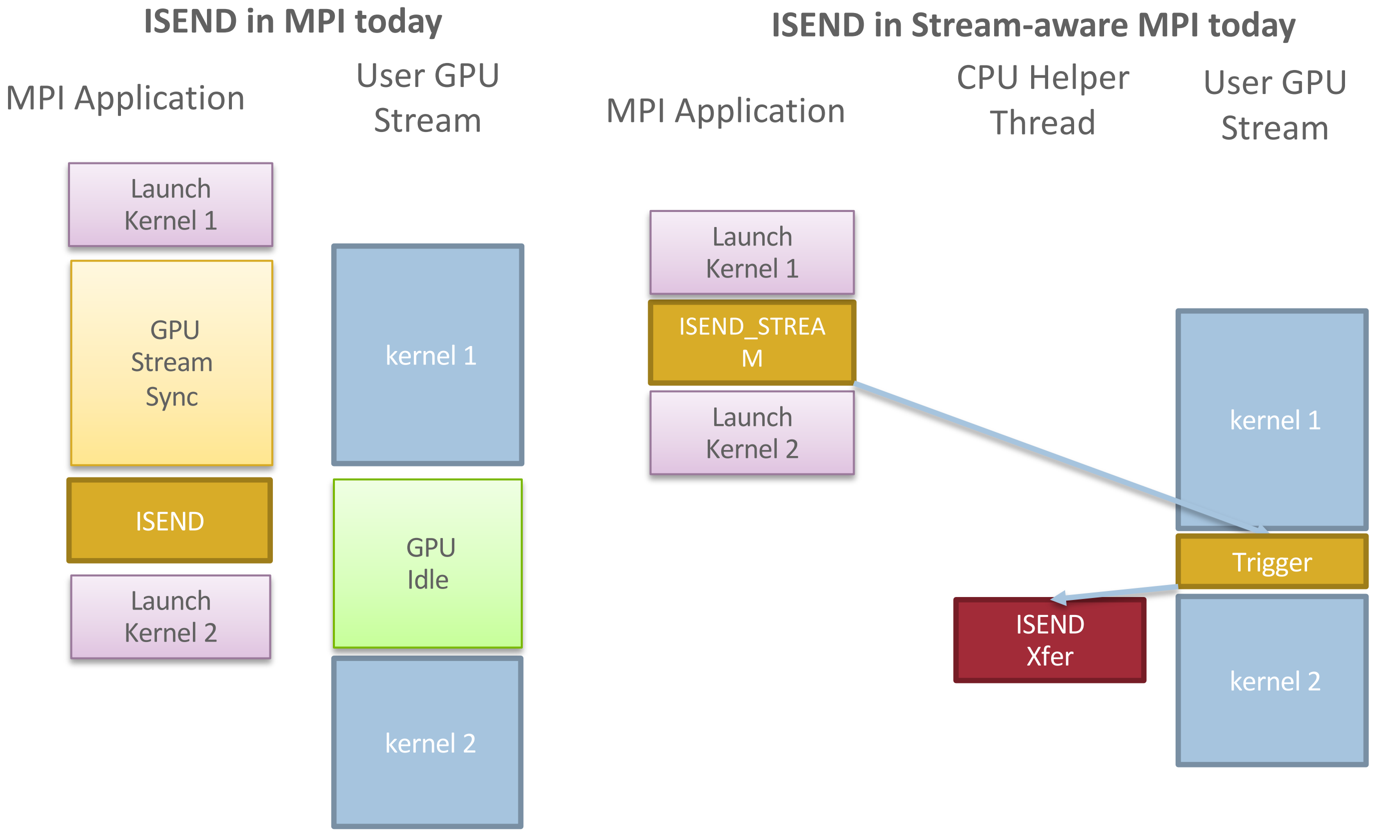
User has to sync the device to make sure data is ready for MPI to send

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



Proposed MPIX_Stream Interfaces: GPU Specific Operations

- `int MPIX_Stream_create(MPI_Info info, MPIX_Stream *stream)`
  - For CUDA stream – `MPI_Info_set(info, “type”, “cudaStream_t”);`
    `MPIX_Info_set_hex(info, “value”, &stream, sizeof(stream));`
- Stream_comm implies enqueueing on GPU stream
- `MPIX_Stream_progress` for helper thread or HW offloading capability
- 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 - GPU Stream Triggered Ops

```
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);
```

Future Plan for MPIX_Stream

- Exploring different techniques for GPU triggering
  - MPICH-4.1 based on stream launched host function
    - Performing MPI_Isend inside the host function
  - Triggering with GPU kernel and stream mem ops
    - Fine-grained control
    - Trade-off between cost and functionality triggered ops
- Better utilizing hardware features
  - Triggering NIC directly
    - Efficiency
    - Better overlapping

Standalone PMI

- PMI remains an internal component in MPICH
- Supporting both PMI-1 and PMI-2 is confusing
  - PMI-1 is the default in MPICH/Hydra, well tested
  - PMI-2 is/was 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 that 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

Standalone PMI -- available in MPICH-4.1b1

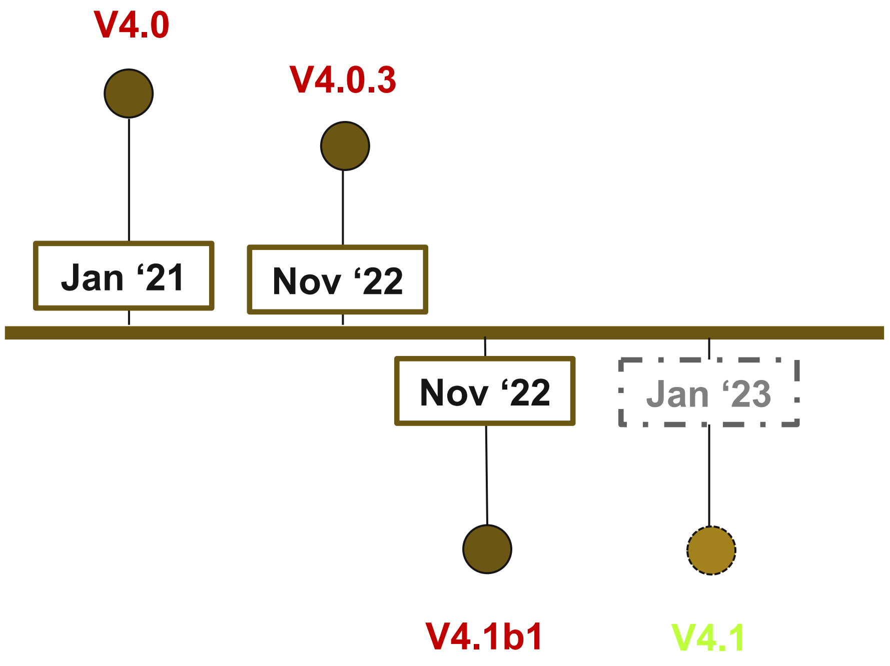
- Better configure options
  - `--with-pmi={pmi1,pmi2,pmix}`
  - `--with-pmilib={mpich,slurm,cray,pmix}`
  - `--with-pm={no,hydra,gforker,remshell}`
  - `--with-pmi={slurm,cray} also works`
- Separate release targets
  - `pmi-4.1b1.tar.gz` and `hydra-4.1b1.tar.gz`
- Consistent PMI headers
  - Third party 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

Standalone 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 `#include <pmi.h>` and `libpmi.so`, use functions with `PMI_` prefix
  - 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 Roadmap

- MPICH-4.1a1 released in May
- MPICH-4.1b1 released last week
  - 4.1.x branch is created
- GA release in early 2023
- Critical bug fixes will be backported to 4.0.x



Other On-going Projects

- ML-based Performance Tuning for Collective
  - Better collective algorithms and algorithms selection
- MPI with Compression
  - Lossy compression for MPI collectives
  - Reduction in msg size => reduction in latency
- Optimization Collective for GPUs
  - Leveraging GPU IPC for collective
  - Better management of intermediate buffers
- DPU Offloading

Programming Models and Runtime Systems Group

Current Staff Members

- Yanfei Guo (assistant scientist)
- Thomas Gillis (postdoc)
- Rob Latham (software developer)
- Ken Raffenetti (software developer)
- Rajeev Thakur (senior scientist)
- Xiaodong Yu (postdoc)
- Junchao Zhang (postdoc)
- Hui Zhou (software developer)

Past Staff Members

- Pavan Balaji (senior scientist)
- Abdelhalim Amer (assistant scientist)
- Neelima Bayyapu (postdoc)
- Wesley Bland (postdoc)
- Darius T. Buntinas (developer)
- Sudheer Chunduri (assistant scientist)
- Giuseppe Congiu (postdoc)
- James S. Dinan (postdoc)
- Huansong Fu (predoc)
- David J. Goodell (developer)
- Ralf Gunter (research associate)
- Shintaro Iwasaki (postdoc)
- Travis Koehring (predoc)
- Huiwei Lu (postdoc)
- Kavitha Madhu (postdoc)
- Lena Oden (postdoc)
- Antonio Pena (postdoc)
- Min Si (assistant scientist)
- Sangmin Seo (assistant scientist)
- Min Tian (visiting scholar)
- Yanjie Wei (visiting scholar)
- Yuqing Xiong (visiting scholar)
- Jian Yu (visiting scholar)
- Xiaomin Zhu (visiting scholar)

Current and Recent Students

- Ashwin Aji (Ph.D.)
- Abdelhalim Amer (Ph.D.)
- Md. Humayun Arafat (Ph.D.)
- Seonmyeong Bak (Ph.D.)
- Alex Brooks (Ph.D.)
- Adrian Castello (Ph.D.)
- Yanhao Chen (Ph.D.)
- Dazhao Cheng (Ph.D.)
- James S. Dinan (Ph.D.)
- Piotr Fidkowski (Ph.D.)
- Huansong Fu (Ph.D.)
- Priyanka Ghosh (Ph.D.)
- Sayan Ghosh (Ph.D.)
- Ralf Gunter (B.S.)
- Jichi Guo (Ph.D.)
- Yanfei Guo (Ph.D.)
- Marius Horga (M.S.)
- John Jenkins (Ph.D.)
- Feng Ji (Ph.D.)
- Shintaro Iwasaki (Ph.D.)
- Ping Lai (Ph.D.)
- Palden Lama (Ph.D.)
- Yan Li (Ph.D.)
- Huiwei Lu (Ph.D.)
- Jintao Meng (Ph.D.)
- Ganesh Narayanaswamy (M.S.)
- Qingpeng Niu (Ph.D.)
- Poornima Nookala (Ph.D.)
- Ziaul Haque Olive (Ph.D.)
- Kaiming Ouyang (Ph.D.)
- David Ozog (Ph.D.)
- Renbo Pang (Ph.D.)
- Nikela Papadopolou (Ph.D.)
- Sreeram Potluri (Ph.D.)
- Sarunya Pumma (Ph.D.)
- Li Rao (M.S.)
- Gopal Santhanaraman (Ph.D.)
- Thomas Scogland (Ph.D.)
- Min Si (Ph.D.)
- Shunpei Shiina (M.S.)
- Brian Skjerven (Ph.D.)
- Rajesh Sudarsan (Ph.D.)
- Hengjie Wang (Ph.D.)
- Xi Wang (Ph.D.)
- Lukasz Wesolowski (Ph.D.)
- Shucai Xiao (Ph.D.)
- Chaoran Yang (Ph.D.)
- Rohit Zambre (Ph.D.)
- Boyu Zhang (Ph.D.)
- Xiuxia Zhang (Ph.D.)
- Xin Zhao (Ph.D.)

Thank you!

https://www.mpich.org

Also join our development call every Thursday at 9am (central): https://bit.ly/mpich-dev-call