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

Yanfei Guo, Ken Raffenetti, Hui Zhou and Rajeev Thakur

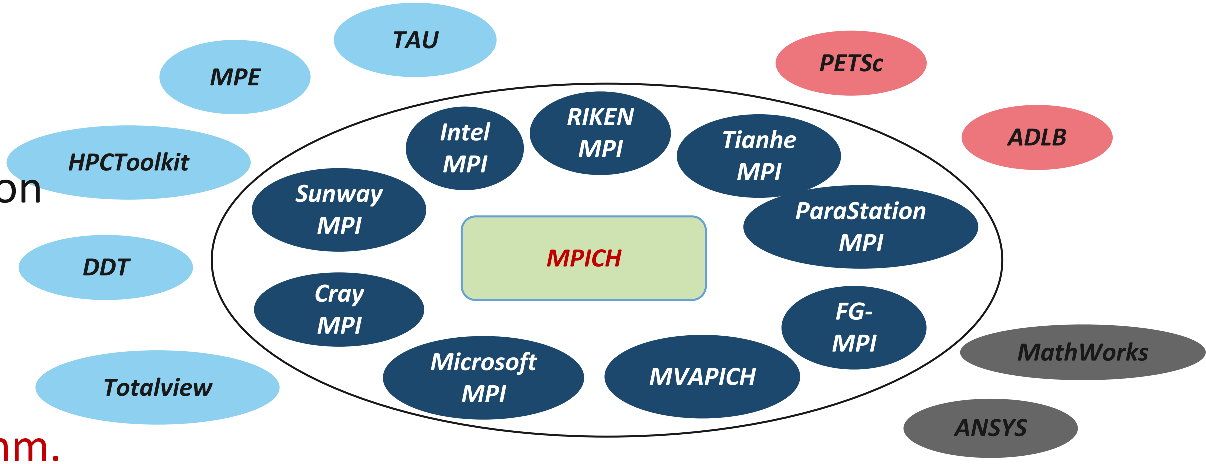
Assistant Computer Scientist

Argonne National Laboratory

MPICH turns 29

The MPICH Project

- Funded by DOE for 28 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



MPICH is not just a software

It's an Ecosystem

MPICH Adoption in Exascale Machines

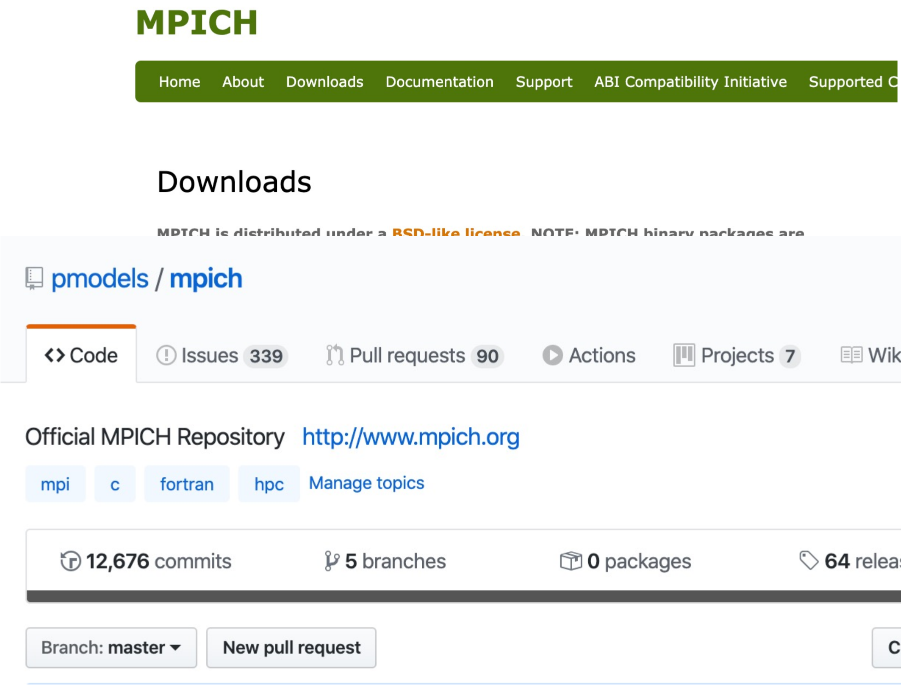
- Aurora, ANL, USA (MPICH)
- Frontier, ORNL, USA (Cray MPI)
- El Capitan, LLNL, USA (Cray MPI)

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

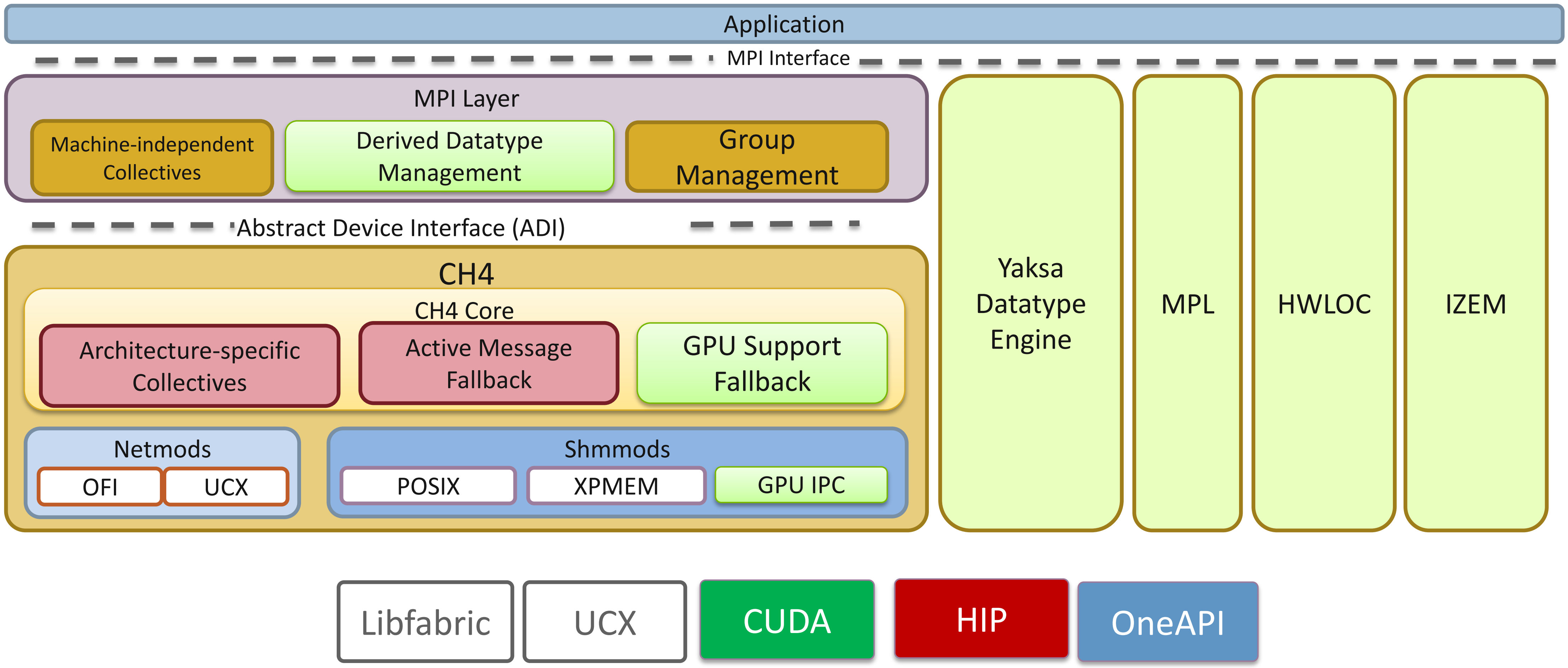
- Source Code Distribution
  - MPICH Website, Github
- Binary Distribution through OS Distros and Package Managers
  - Redhat, CentOS, Debian, Ubuntu, Homebrew (Mac)
- Distribution through HPC Package Managers
  - Spack, OpenHPC
- Distribution through Vendor Derivatives



MPICH Releases

- MPICH now aims to follow a 12-month cycle for major releases (4.x), down from 18 months previously
  - Minor bug fix releases for the current stable release happen every few months
  - Preview releases for the next major release happen every few months
  - Branching off when beta is released (feature freezed)
- Current stable release is in the 3.4.x series
  - mpich-3.4.2 was in May 2021
  - mpich-3.4.3 coming soon
- Upcoming major release is in the 4.0 series
  - mpich-4.0b1 released this week
  - rc1 and GA release coming soon

MPICH Layered Structure



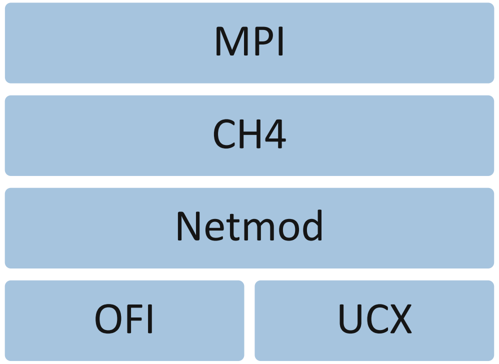
CH4 Design Goals

High-Level Netmod API

- Give more control to the network
  - `netmod_isend`
  - `netmod_irecv`
  - `netmod_put`
  - `netmod_get`
- Fallback to Active Message based communication when necessary
  - Operations not supported by the network

"Netmod Direct"

- Support two modes
  - Multiple netmods
    - Retains function pointer for flexibility
  - Single netmod with inlining into device layer
    - No function pointer overhead



Provide default shared memory implementation in CH4

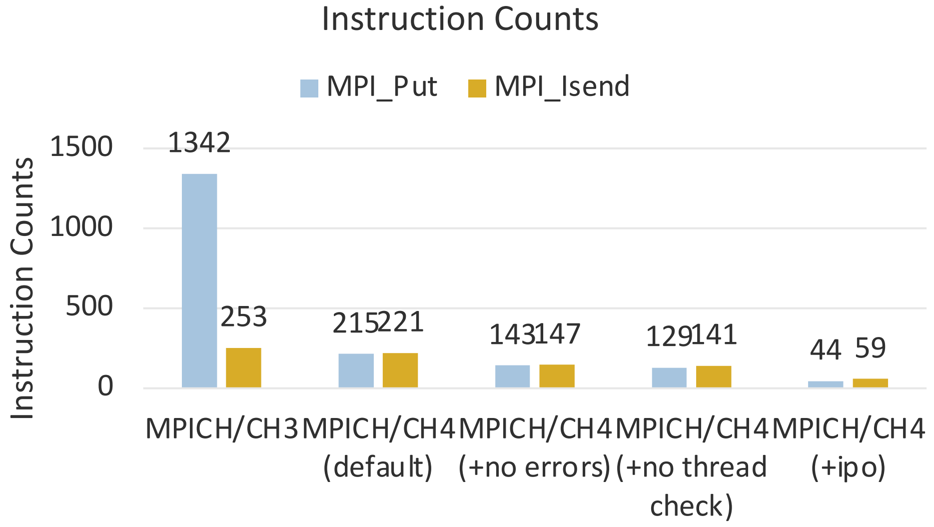
- Disable when desirable
  - Eliminate branch in the critical path
  - Enable better tuned shared memory implementations
  - Collective offload

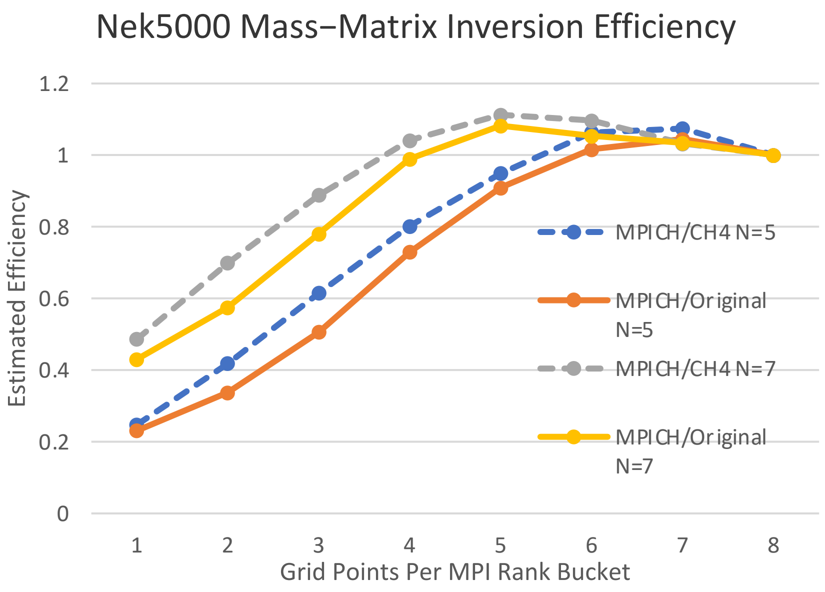
Minimal Per Process Data

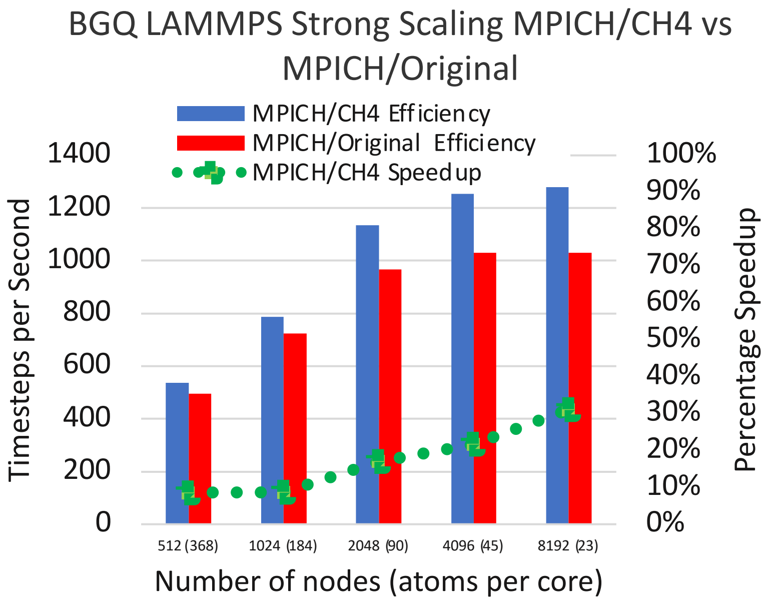
- Global address table
  - Contains all process addresses
  - Index into global table by translating `(rank+comm)`

Partnership with Intel, Mellanox, Cray, RIKEN, NVIDIA and AMD

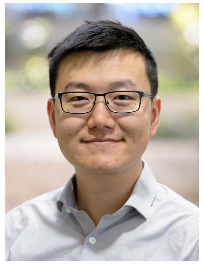
Lower Overheads = Better Strong Scaling







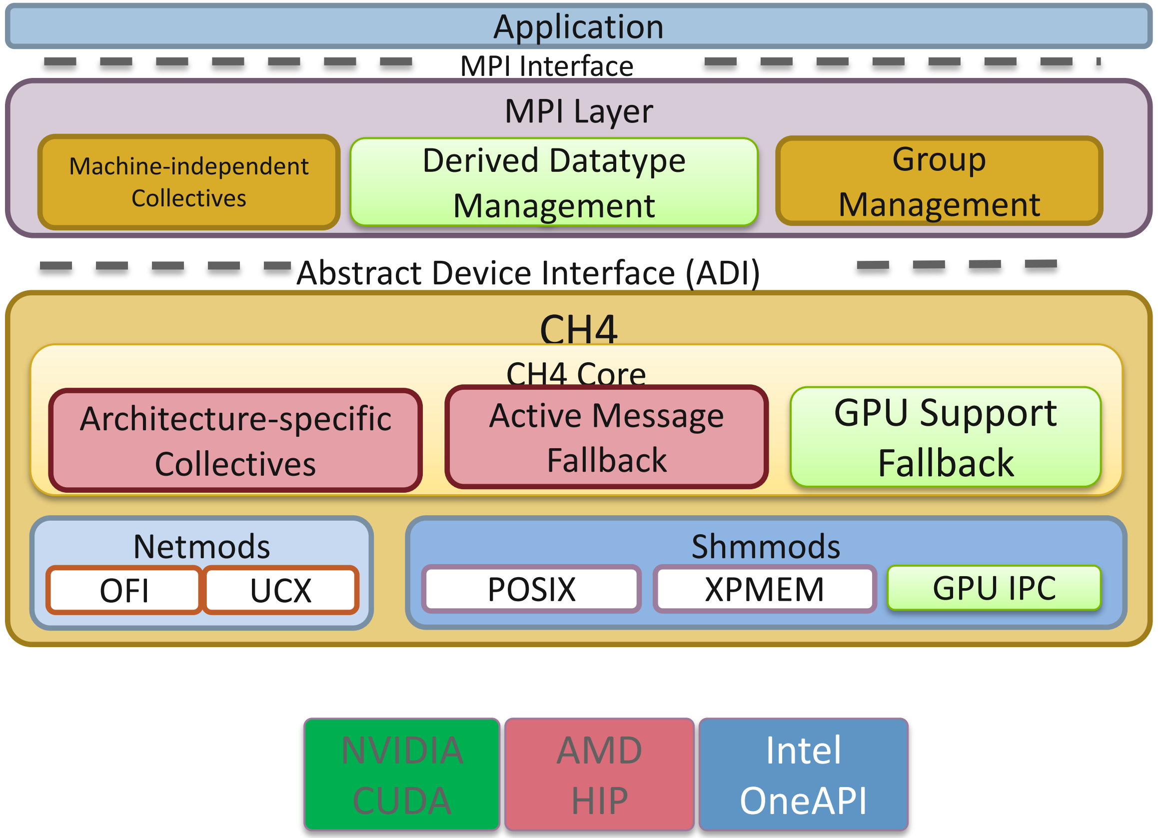
Supporting GPU in MPI Communication (1/3)



POC: Yanfei Guo
<yguo@anl.gov>

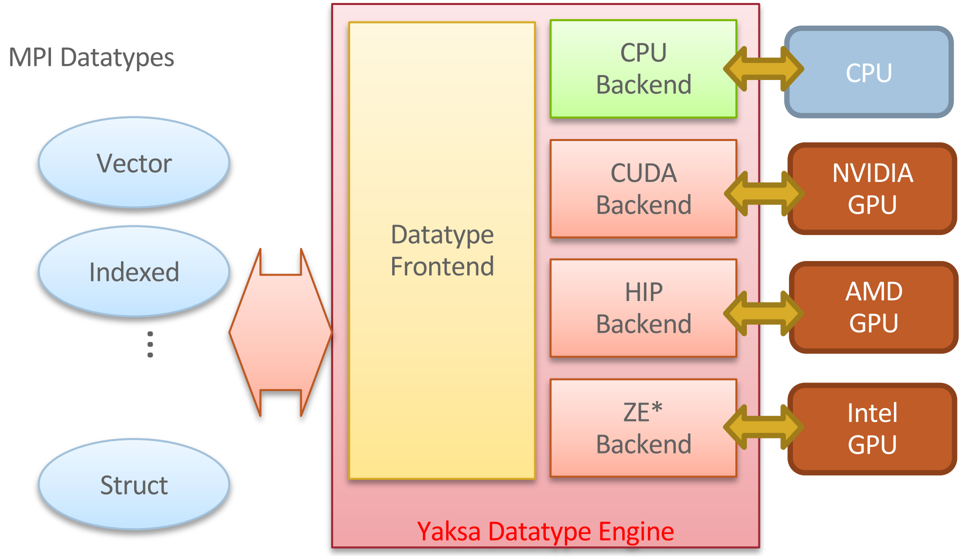
- **Native GPU Data Movement**
  - Multiple forms of “native” data movement
  - GPU Direct RDMA is generally achieved through Libfabrics or UCX (we work with these libraries to enable it)
  - GPU Direct IPC is integrated into MPICH
- **GPU Fallback Path**
  - GPU Direct RDMA may not be available due 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

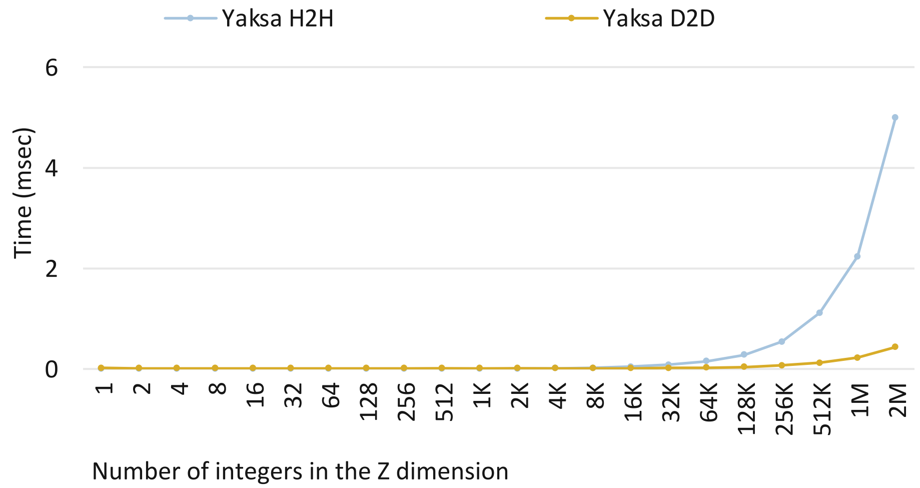
The GPU support in MPICH is developed in close collaboration with vendor partners including Including AMD, Cray, Intel, Mellanox and NVIDIA



Supporting GPU in MPI Communication (2/3)

- **MPICH support for using complex noncontiguous buffers with GPU**
  - Buffer with complex datatype is not directly supported by the network library
  - Packing complex datatype from GPU into contiguous send buffer
  - Unpacking received data back into complex datatype on GPU
- **Yaksa: A high performance datatype engine**
  - Used for internal datatype representation in MPICH
  - Front-end provide interface for MPI datatypes
  - Multiple backend to leverage different hardware for datatype handle
  - Generated GPU kernels for packing/unpacking

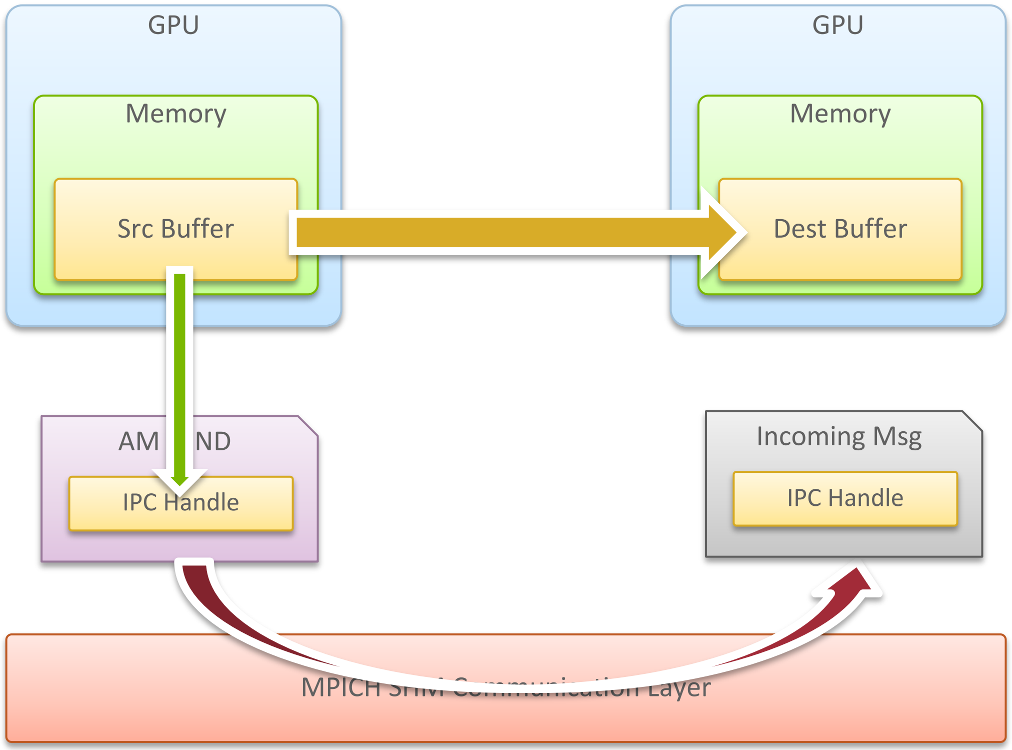




The GPU support in MPICH is developed in close collaboration with vendor partners including Including AMD, Cray, Intel, Mellanox and NVIDIA

Supporting GPU in MPI Communication (3/3)

- **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

Review of Major MPICH 3.4 Features

POC: Ken Raffenetti
<raffenet@mcs.anl.gov>

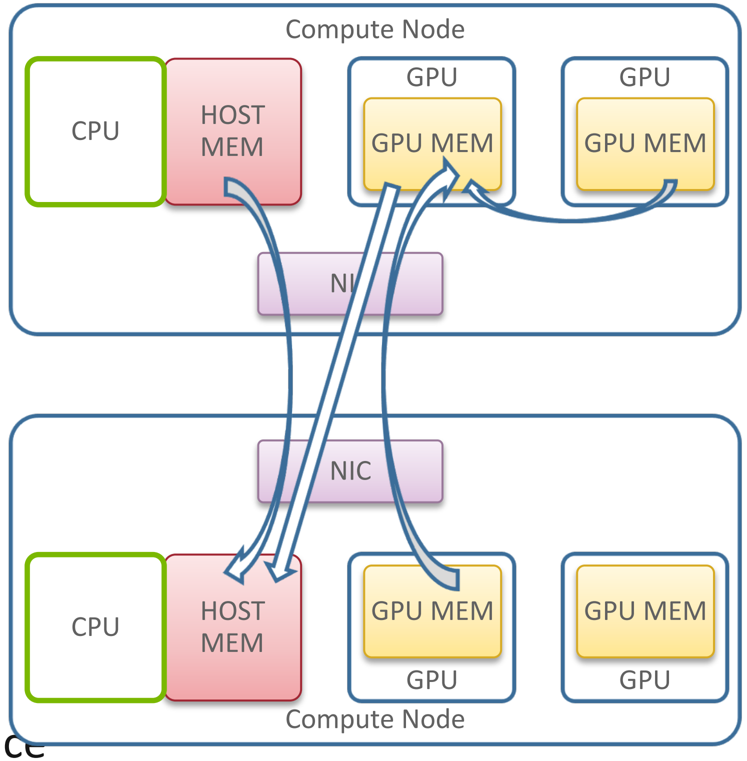
1. Extending CUDA Awareness and Support for other GPUs (***partnership with Mellanox, NVIDIA, AMD, Intel and Cray***)
   - Multiple Vendors Support: NVIDIA, AMD and Intel
2. Multi-VCI communication (***partnership with Intel***)
   - Utilizing network hardware parallelism
3. Collective Selection Framework (***partnership with Intel***)
   - A Framework supporting different strategies of collective algorithm selection
4. Datatype engine extensions and optimizations
   - Optimized code paths for common datatype compositions

Focuses in MPICH 4.0 Release Series

POC: Hui Zhou
<zhouh@anl.gov>

- Full implementation of MPI-4.0 specification
  - MPI Sessions
  - Partitioned Communications
  - Persistent Collectives, Tool Events, Large Count, and more
  - https://www.mpi-forum.org/docs/
- Enhance GPU and threading support
  - Make the support more stable and user experience smoother
  - Push the production performance to our research projection

Enhanced GPU Support

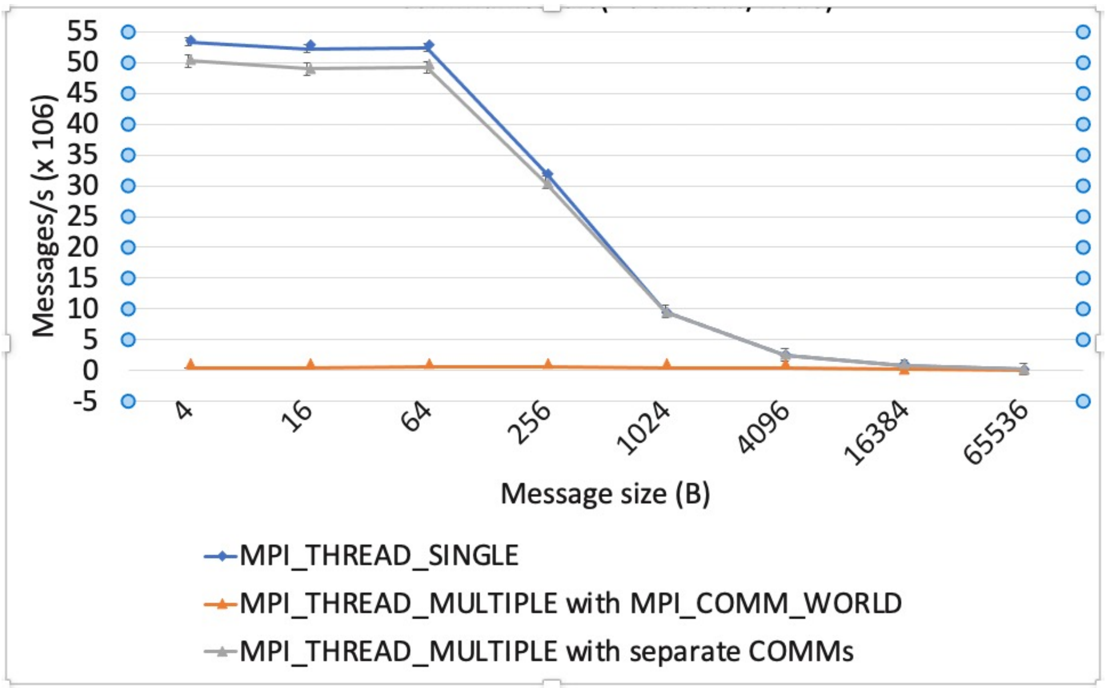


- MPICH is fully GPU-aware since 3.4.0
- But …
  - GPU initialization cost, GPU pointer query cost, …
  - GPU testing takes 8 hours! Not fully green.
  - Need GPU-aware API
- Enhancements
  - `MPIR_CVAR_ENABLE_GPU=0` should recover full CPU-only performance
  - Full GPU CI testing, green!
  - GPU direct IPC for NVIDIA and Intel GPUs
- In Progress
  - GPU IPC for AMD GPUs, …

Better Threading Support

- Enable strong scaling with multiple VCI (virtual communication interface)
- Multi-VCI for Point-to-point implemented in 3.4.0
- Multi-VCI for RMA added in 4.0a1
- Multi-VCI for Active Messages added in 4.0b1

- Parallel semantics based on `communicator/rank/tag`



C Binding Generation

- + 3,000 lines of Python script
- - 40,000 lines of C

- API extracted from mpi-standard repo

- Generates –
  - Profiling interface
  - API documentation
  - Parameter validation
  - Handle object pointer conversion

- Fortran binding generation will be updated to Python and unified
  - F08 binding generation 80% done

```
MPI_Bcast_init:

    .desc: Creates a persistent request for broadcast
```

MPI Session

```
void library_foo_init(void)
{
    MPI_Session_init(info, errhan, &session);
    MPI_Group_from_session_pset(session, "mpi://world", &group);
    MPI_Comm_create_from_group(group, "foo (string tag)",
                               info, errhan, &comm);
    ...
}
```

- Libraries to keep MPI usage opaque to user
- Basic implementation internally initializes "world" in the first `MPI_Session_init/MPI_Init`
- World initialization to delayed to the creation of first world-comm
- Fully *correct* implementation need to support first-class dynamic processes

Partitioned Communication

- In-between two-sided (pt2pt) and one-sided (RMA) communication
- Basic implementation done, plenty of optimization opportunity ahead!

```
if (rank == sender) {
  MPI_Psend_init(buf, parts, cnt, datatype,
                 recver, tag, comm, info, &req);
  MPI_Start(&req);
  ...
  MPI_Pready(i);
  ...
  MPI_Wait(&req, MPI_STATUS_IGNORE);
  MPI_Request_free(&req);
}
```

```
if (rank == recver) {
  MPI_Precv_init(buf, parts, cnt, datatype,
                 sender, tag, comm, info, &req);
  MPI_Start(&req);
  ...
  MPI_Parrived(req, i, &flag);
  ...
  MPI_Wait(&req, MPI_STATUS_IGNORE);
  MPI_Request_free(&req);
}
```

Large Count API

- A large count version for every API that has a ”count” or “displacement” argument (guess how many?)
- No more work-arounds!
- API use `MPI_Count`, internally we use `MPI_Aint` where-ever possible

```
MPI_Type_contiguous_c(10000000000, MPI_INT, &my_type);
```

```
MPI_Send_c(buf, 10000000000, MPI_INT, dest, tag, comm);
```

MPI_T Events

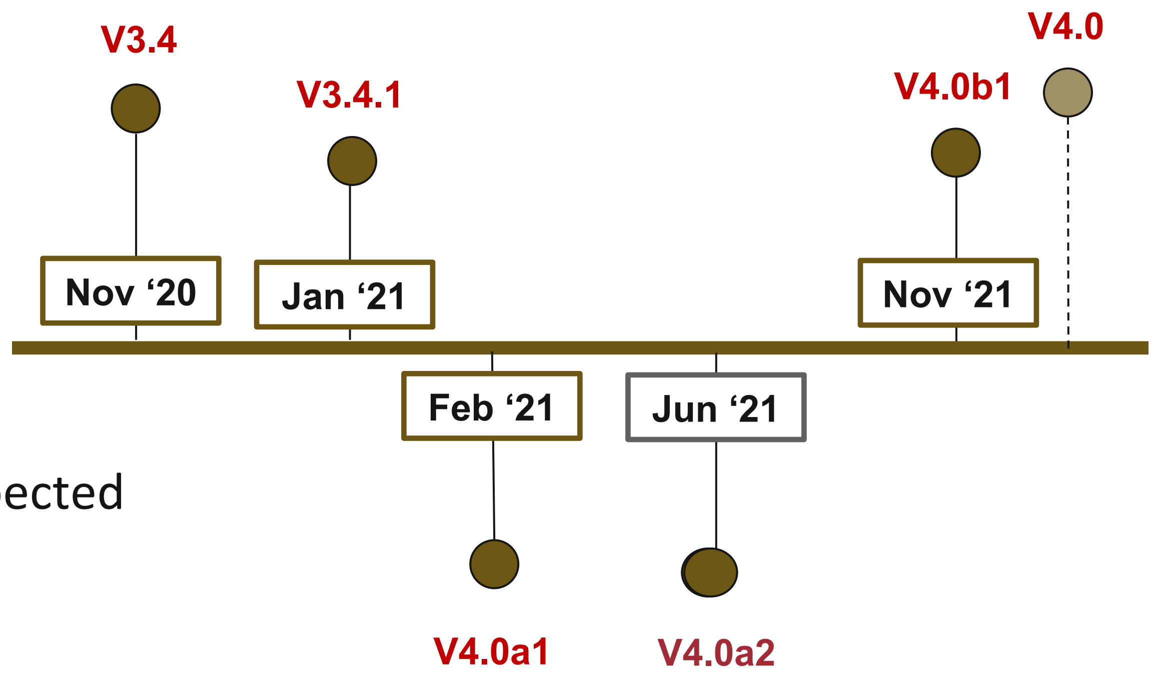
- Callback-based interface for tools to get information on internal library events
- Infrastructure and example events complete. We welcome community feedback to define useful events!

```
/* discover and register for events  */
int MPI_T_event_get_num(int *num_events);
int MPI_T_event_get_info(int event_index, char *name, int *name_len, int *verbosity,
                         MPI_Datatype*array_of_datatypes, MPI_Aint*array_of_displacements,
                         int *num_elements, MPI_Aint *extent, MPI_T_enum *enumtype,
                         MPI_Info*  info,  char *desc, int *desc_len,  int *bind);
int MPI_T_event_handle_alloc(int event_index, void *obj_handle, MPI_Infoinfo,
                             void *user_data, MPI_T_event_cb_functionevent_cb_function,
                             MPI_T_event_registration*event_registration)
```

```
/* callback prototype and API to read info from event instance handle */
typedef void (*MPI_T_event_cb_function)(MPI_T_event_instance event_instance,
                                        MPI_T_event_registration event_registration,
                                        MPI_T_cb_safety cb_safety, void *user_data);
int MPI_T_event_read(MPI_T_event_instance event_instance, int element_index, void *buffer)
```

MPICH 4.0 Roadmap

- MPICH-4.0a1 released in February
  - Majority of the MPI-4.0 API implemented
- MPICH-4.0a2 released in June
  - Synchronized to MPI Forum meeting with the expected official ratification of MPI-4.0 standard
  - Full implementation of MPI-4.0 API
  - More stable GPU/threading support
- MPICH-4.0b1 released this week
  - 4.0.x branch is created
- GA release in late 2021/early 2022
- Critical bug fixes are backported to 3.4.x



MPICH 4.1 Plans (RFC)

- Enhancing GPU Support
  - Collective, RMA
  - Stream awareness extensions
- Extend PMI 1 and PMI 2 interface / Enhance hydra
  - PMI 1 ages very well and still kicking!
  - Support PMIx capability with backward compatibility and simplicity of PMI 1 and 2
  - Enhance hydra with tree-launching capability
- Improve useability
  - Explore MPIX space for more natural/direct semantics
  - Fixing issues – we shrank outstanding issus from ~450 down to 200 this year! We all cut it in half again next year.

Programming Models and Runtime Systems Group

Current Staff Members

- Yanfei Guo (assistant scientist)
- Sudheer Chunduri (assistant scientist)
- Travis Koehring (predoc)
- Rob Latham (software developer)
- Ken Raffenetti (software developer)
- Rajeev Thakur (senior scientist)
- Xiaodong Yu (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)
- Giuseppe Congiu (postdoc)
- James S. Dinan (postdoc)
- Huansong Fu (predoc)
- David J. Goodell (developer)
- Ralf Gunter (research associate)
- shintaro Iwasaki (postdoc)
- 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)
- Junchao Zhang (postdoc)
- 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
- Mailing list: discuss@mpich.org
- Issues and Pull requests: https://github.com/pmodels/mpich
- Weekly development call every Thursday at 9am (central): https://bit.ly/mpich-dev-call

Backup Slides

Collective Selection Framework

- Choose Optimal Collective Algorithms
  - Optimized algorithm for certain communicator size, message size
  - Optimized algorithm using HW collective support
  - Making decision on each collective call
- Pre-generated Decision Tree
  - JSON file describing choosing algorithms with conditions
  - JSON file created by profiling tools
  - JSON parsed at MPI_Init time and applied to the library

Contributed by Intel (with some minor help from Argonne)

New Collective Infrastructure

- Thanks to Intel for the significant work on this infrastructure
- Two major improvements:
  - Dependency-based scheduling
  - C++ Template-like structure (still written in C)
    - Allows collective algorithms to be written in template form
    - Provides “generic” top-level instantiation using point-to-point operations
    - Allows device-level machine specific optimized implementations (e.g., using triggered operations for OFI or HCOLL for UCX)
  - Several new algorithms for a number of blocking and nonblocking collectives (performance tuning still ongoing)

Contributed by Intel (with some minor help from Argonne)