Argonne NATIONAL LABORATORY

Exascale MPI: Making MPI Ready for Future Supercomputers

Yanfei Guo

Argonne National Laboratory

yguo@anl.gov

U.S. DEPARTMENT OF ENERGY

What is MPI?

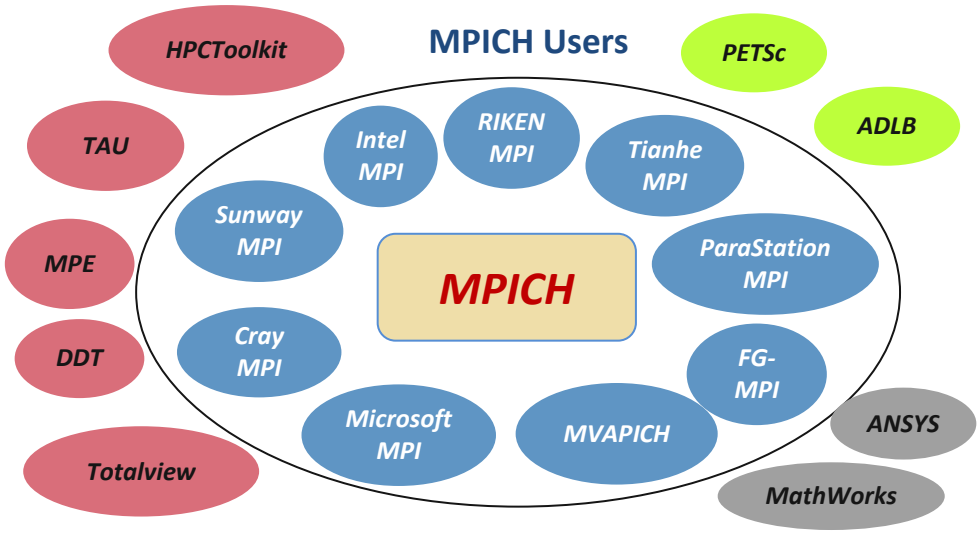
- MPI: Message Passing Interface
  - The MPI Forum organized in 1992 with broad participation by:
    - Vendors: IBM, Intel, TMC, SGI, Convex, Meiko
    - Portability library writers: PVM, p4
    - Users: application scientists and library writers
    - MPI-1 finished in 18 months
  - Incorporates the best ideas in a “standard” way
    - Each function takes fixed arguments
    - Each function has fixed semantics
      - Standardizes what the MPI implementation provides and what the application can and cannot expect
      - Each system can implement it differently as long as the semantics match
- MPI is not…
  - a language or compiler specification
  - a specific implementation or product

Reasons for Using MPI

- **Standardization** - MPI is the only message passing library which can be considered a standard. It is supported on virtually all HPC platforms. Practically, it has replaced all previous message passing libraries
- **Portability** - There is no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard
- **Performance Opportunities** - Vendor implementations should be able to exploit native hardware features to optimize performance
- **Functionality** – Rich set of features
- **Availability** - A variety of implementations are available, both vendor and public domain
  - MPICH is a popular open-source and free implementation of MPI
  - Vendors and other collaborators take MPICH and add support for their systems
    - Intel MPI, IBM Blue Gene MPI, Cray MPI, Microsoft MPI, MVAPICH, MPICH-MX

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 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)

Key Focus Areas

Work with vendors to ensure high performance MPI implementations for DOE supercomputer acquisitions

intel CRAY THE SUPERCOMPUTER COMPANY Mellanox TECHNOLOGIES

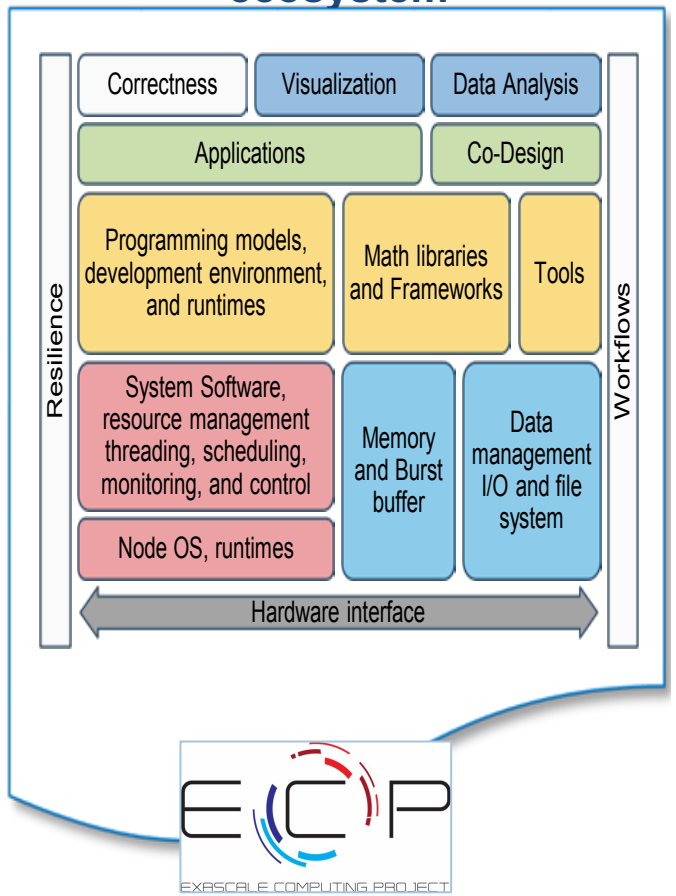
Influence the evolution of the MPI Standard to address ECP and DOE application needs

MPI

Address Key Technical Challenges

- Performance & Scalability
- Heterogeneity
- Fault Tolerance
- MPI+X Hybrid Programming
- Topology Awareness

Work together with the ECP ecosystem



CH4 Design Goals

High-Level Netmod API

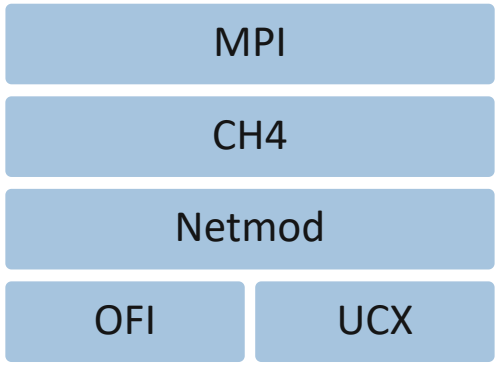
- Give more control to the network
  - `netmod_isend`
  - `netmod_irecv`
  - `netmod_put`
  - `netmod_get`
- Fallback to Active Message based communication when necessary
  - Operations not supported by the network

Provide default shared memory implementation in CH4

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

"Netmod Direct"

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

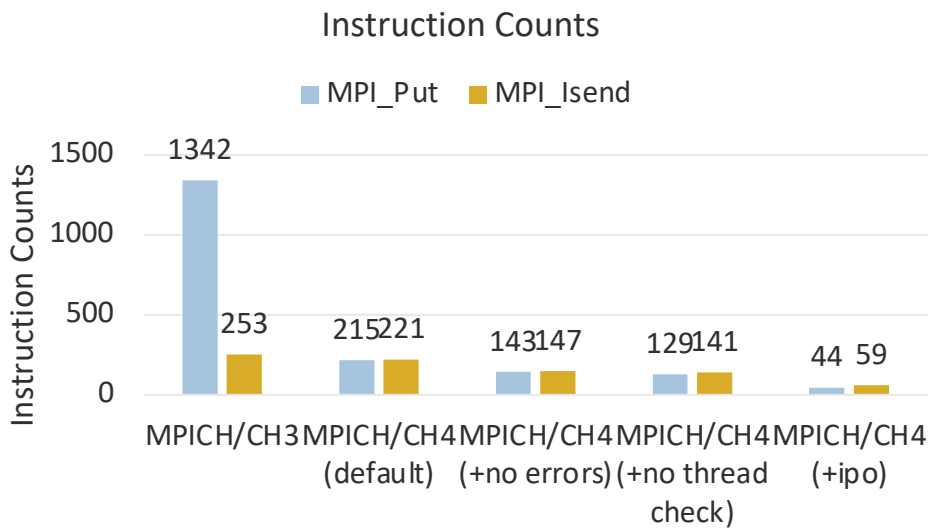


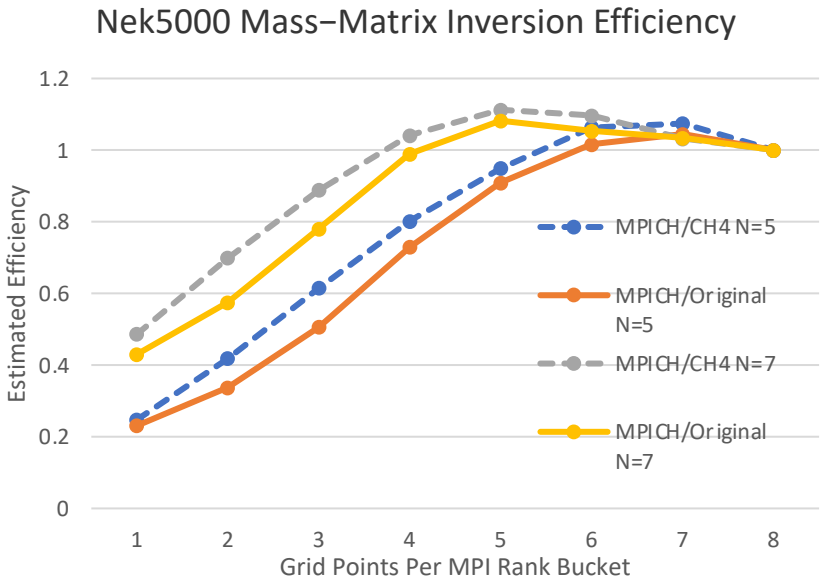
Minimal Per Process Data

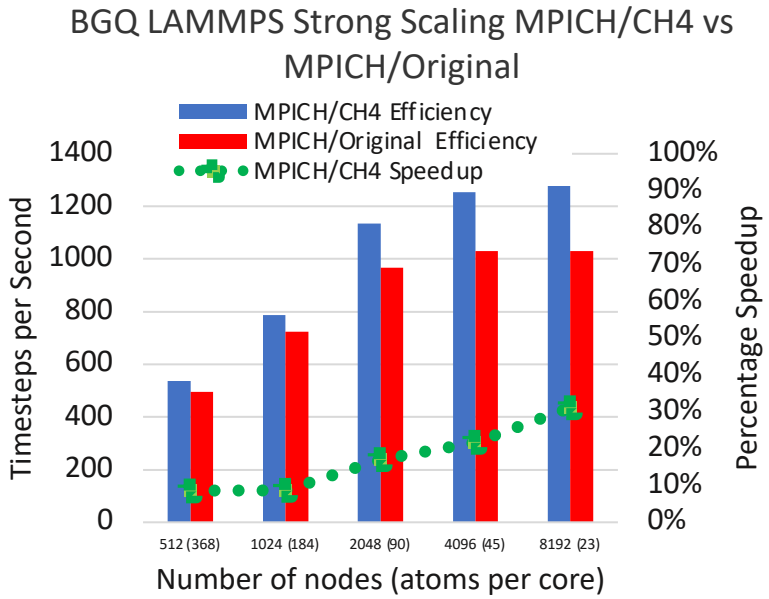
- Global address table
  - Contains all process addresses
  - Index into global table by translating `(rank+comm)`

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

Lower Overheads = Better Strong Scaling

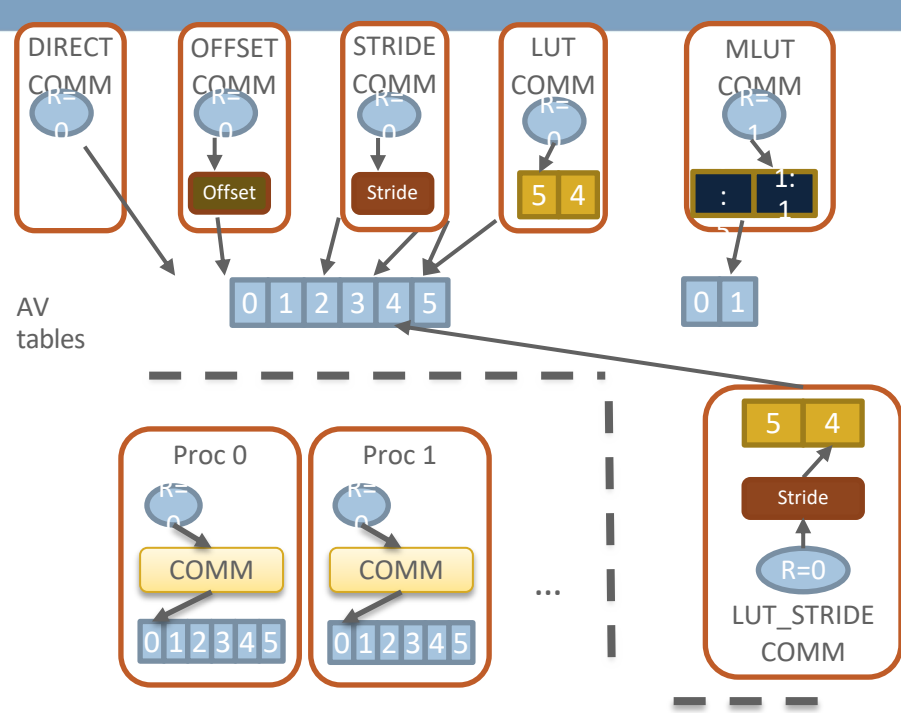


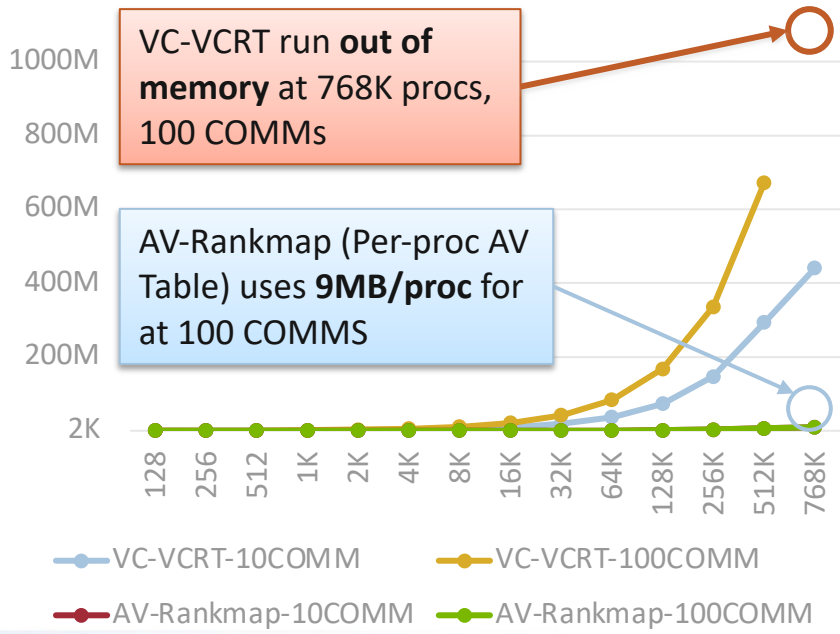




Memory Scalable Network Address Management

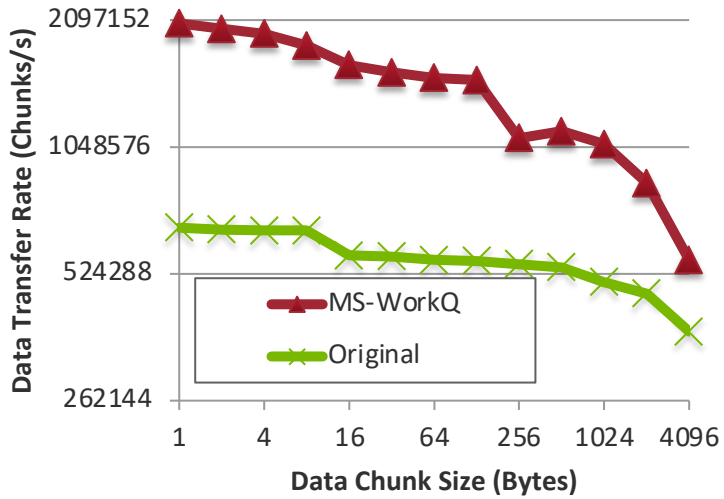
- AV Table: Compressing VC (480Bytes -> 12Bytes)
  - Compressing Multitransport Functionality
    - Function pointers are moved to a separate array
  - Deprioritizing Dynamic Processes
    - Process group information moved to COMM
- Rank Mapping Models
  - **Regular**: DIRECT, OFFSET, STRIDE, STRIDE_BLOCK
  - **Irregular**: LUT, MLUT
  - **Mixed**: LUT_STRIDE, LUT_STRIDE_BLOCK, etc.
- Shared AV Tables
  - AV Tables in shared memory for processes on the same node
  - Shared AV Table 0 (MPI_COMM_WORLD): created at init time, read-only, lock-free
  - Per-proc AV Tables (dynamic processes): avoid locking

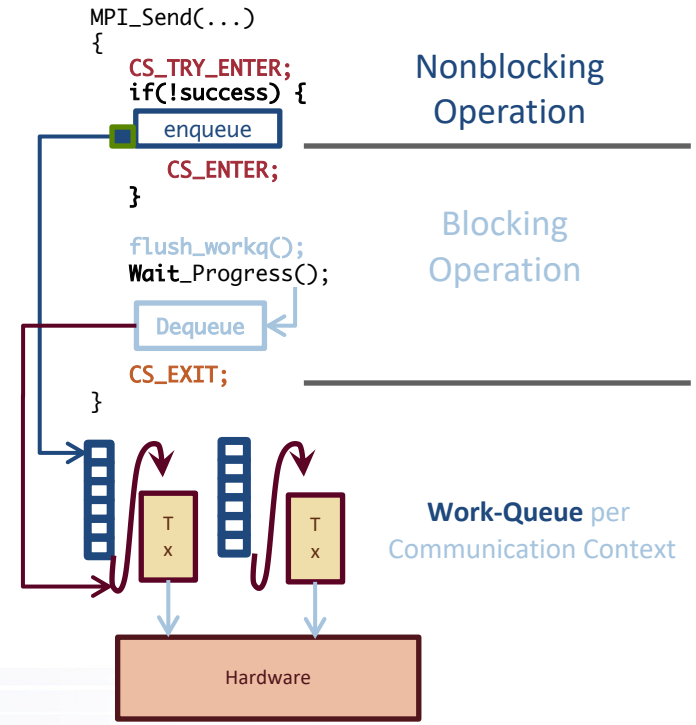




Multithreaded MPI Work-Queue Model

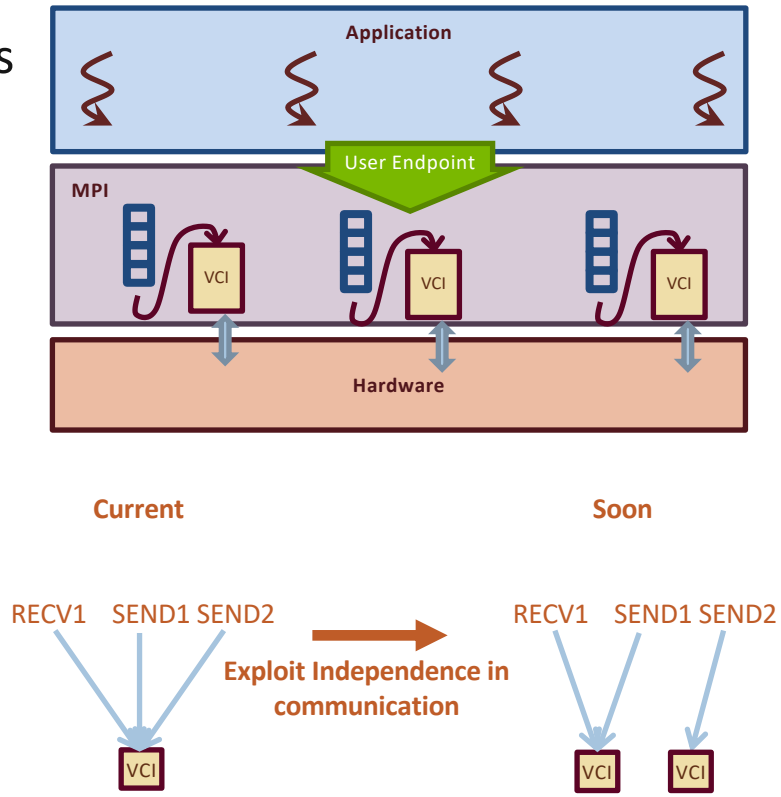
- **Context**
  - Existing lock-based MPI implementations **unconditionally** acquire locks
  - **Nonblocking** operations may **block** for a lock acquisition
    - **Not** truly nonblocking!
- **Consequences**
  - Nonblocking operations may be slowed by blocking ones from other threads
  - Pipeline stalls: higher latencies, lower throughput, and less communication-computation overlapping
- **Work-Queue Model**
  - One or multiple **work-queues per endpoint**
  - Decouple blocking and nonblocking operations
  - Nonblocking operations enqueue **work descriptors** and leave if critical section held
  - Threads issue work on behalf of other threads when acquiring a critical section
  - Nonblocking operations **are truly** nonblocking
- **Multiple network endpoints**
  - Both user visible and hidden





Multi-VCI Communication

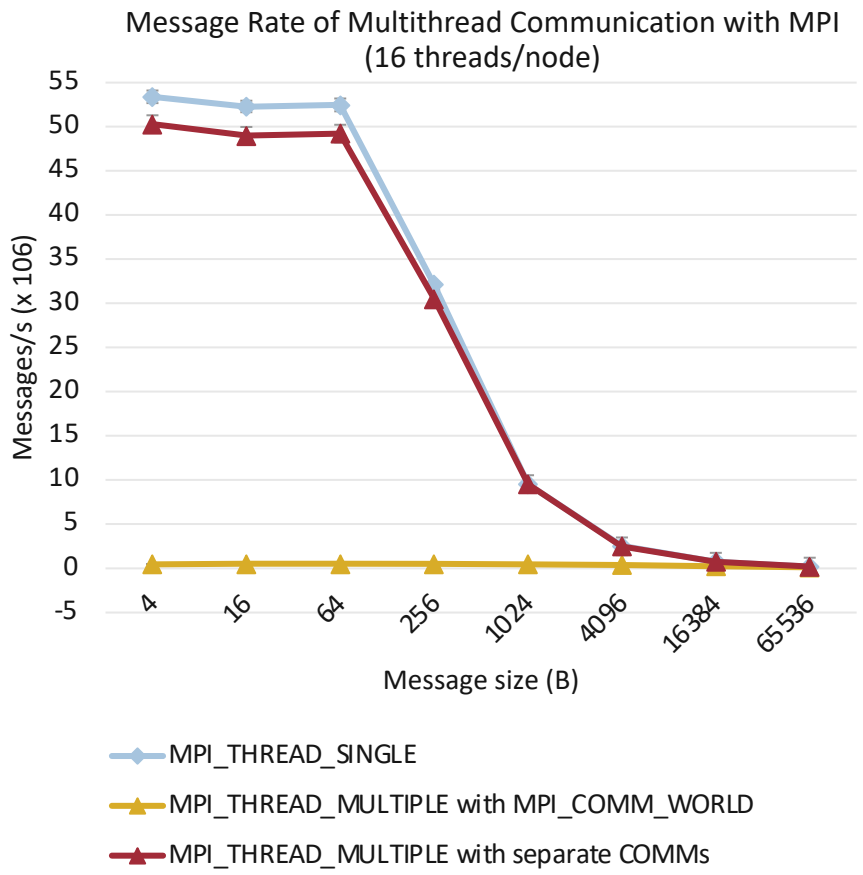
- **Current MPICH code**
  - Single VCI (Virtual Communication Interface) being used
    - Serializes all traffic
    - Does not fully exploit network hardware contexts
- **Proposed solution: Multi-VCI communication**
  - Each VCI encapsulates/abstracts network resources
  - Isolation between VCIs
  - Transparent to the user
  - Exploit independence in communication paths
    - Separate VCIs per communicator
    - Separate VCIs per RMA window
    - Distribute traffic between VCIs with respect to ranks, tags, and generally out-of-order communication

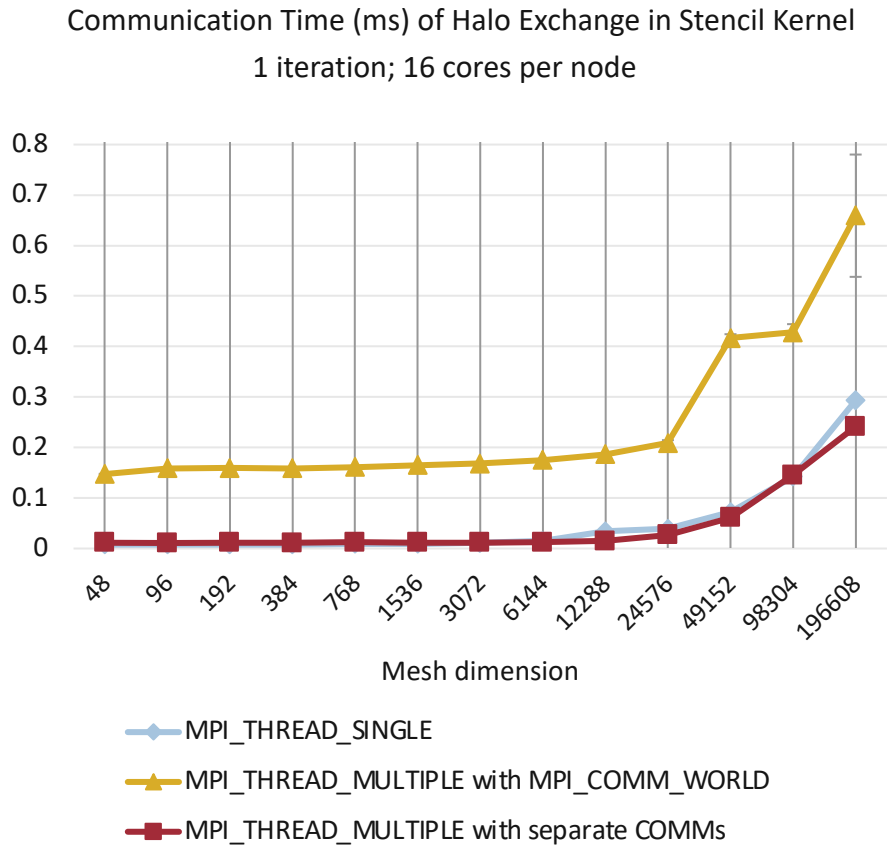


Partnership with Intel

Multi-VCI Performance

- Evaluation of the prototype with multithreaded stencil kernel that model from ECP applications.

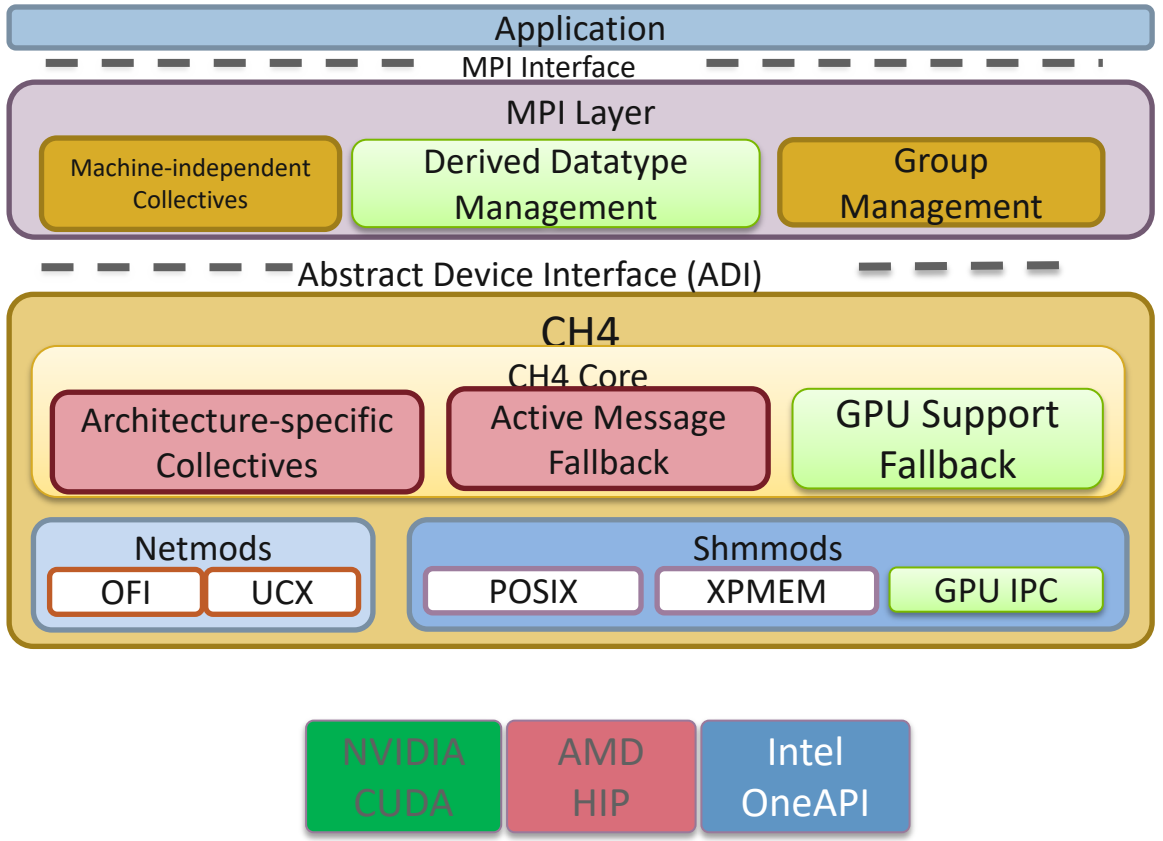




Supporting GPU in MPI Communication (1/3)

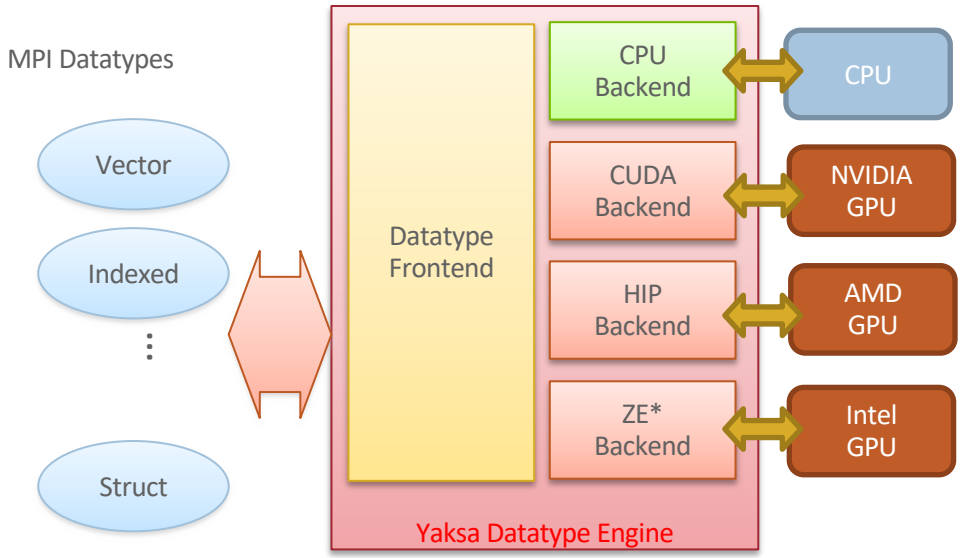
- **Native GPU Data Movement**
  - Multiple forms of “native” data movement
  - GPU Direct RDMA is generally achieved through Libfabrics or UCX (we work with these libraries to enable it)
  - GPU Direct IPC is integrated into MPICH
- **GPU Fallback Path**
  - GPU Direct RDMA may not be available due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

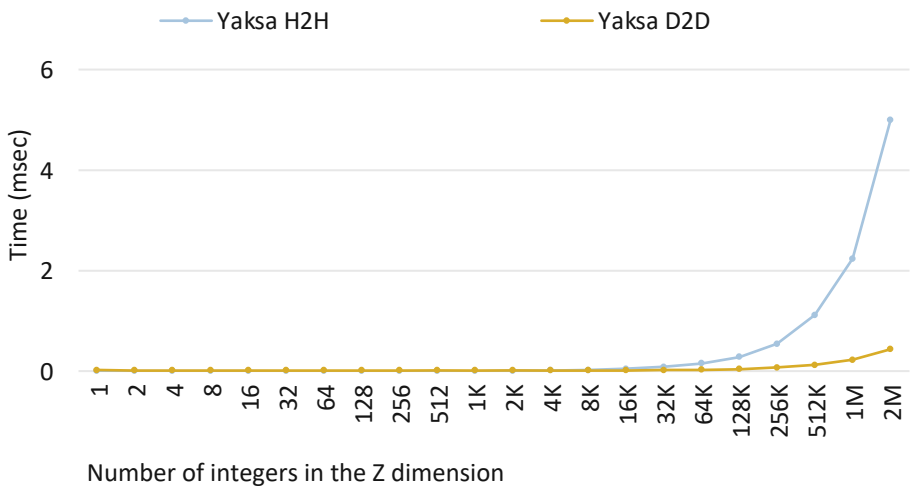
The GPU support in MPICH is developed in close collaboration with vendor partners including Including AMD, Cray, Intel, Mellanox and NVIDIA



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

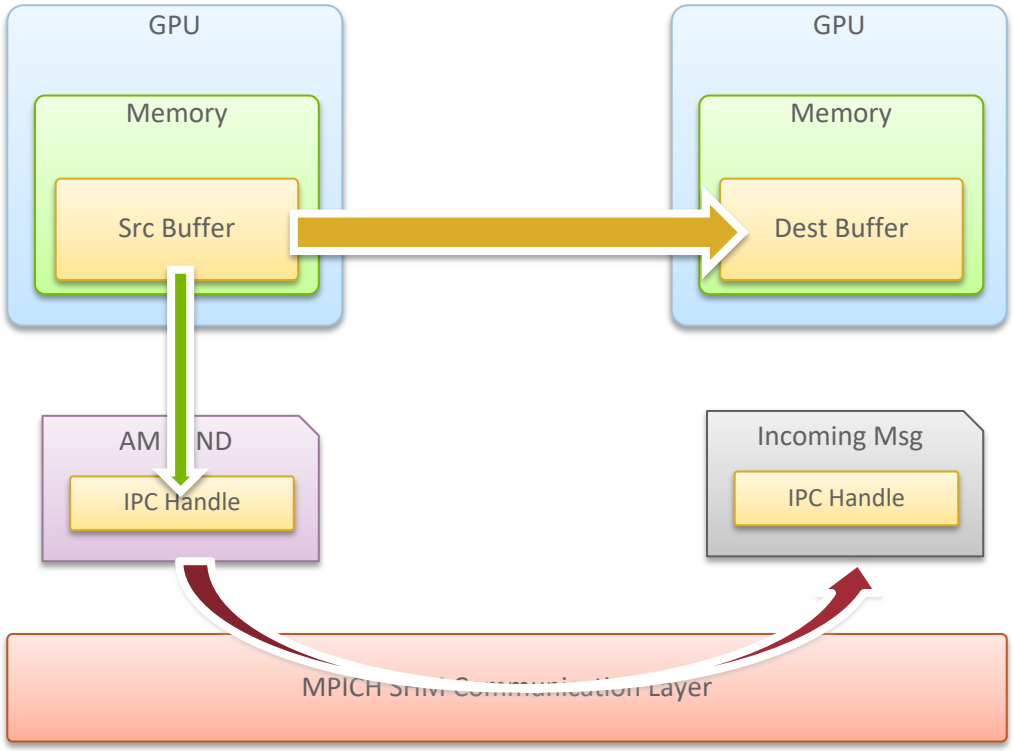




The GPU support in MPICH is developed in close collaboration with vendor partners including Including AMD, Cray, Intel, Mellanox and NVIDIA

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

Other Research and Optimization for MPI

- Collective Algorithm
  - Optimized Algorithms for different message size, process groups, HW acceleration
- Topology Awareness
  - Topology-aware communication
- Job Launching Scalability
- Heterogeneous Memory
- GPU-stream-triggered Operation
  - Synchronization Strategy between CPUs and GPUs
  - Vendor independent abstraction for GPU interoperability
- MPI-4 Standard

Our Projects

- MPICH
  - https://www.mpich.org/
- OSHMPI
  - https://pmodels.github.io/oshmpi-www/
- Argobots
  - https://www.argobots.org/

Thank you!

Yanfei Guo

Argonne National Laboratory

yguo@anl.gov