Argonne NATIONAL LABORATORY

Recent Progress in MPICH

Ken Raffenetti

Principal Software Development Specialist

Argonne National Laboratory

The MPICH Project

- MPICH and its derivatives are the world's most widely used MPI implementations
  - Supports all versions of the MPI standard including the recent MPI-3.1
- Funded by DOE for 26 years
- Has been a key influencer in the adoption of MPI
- Award winning project
  - DOE R&D100 award in 2005

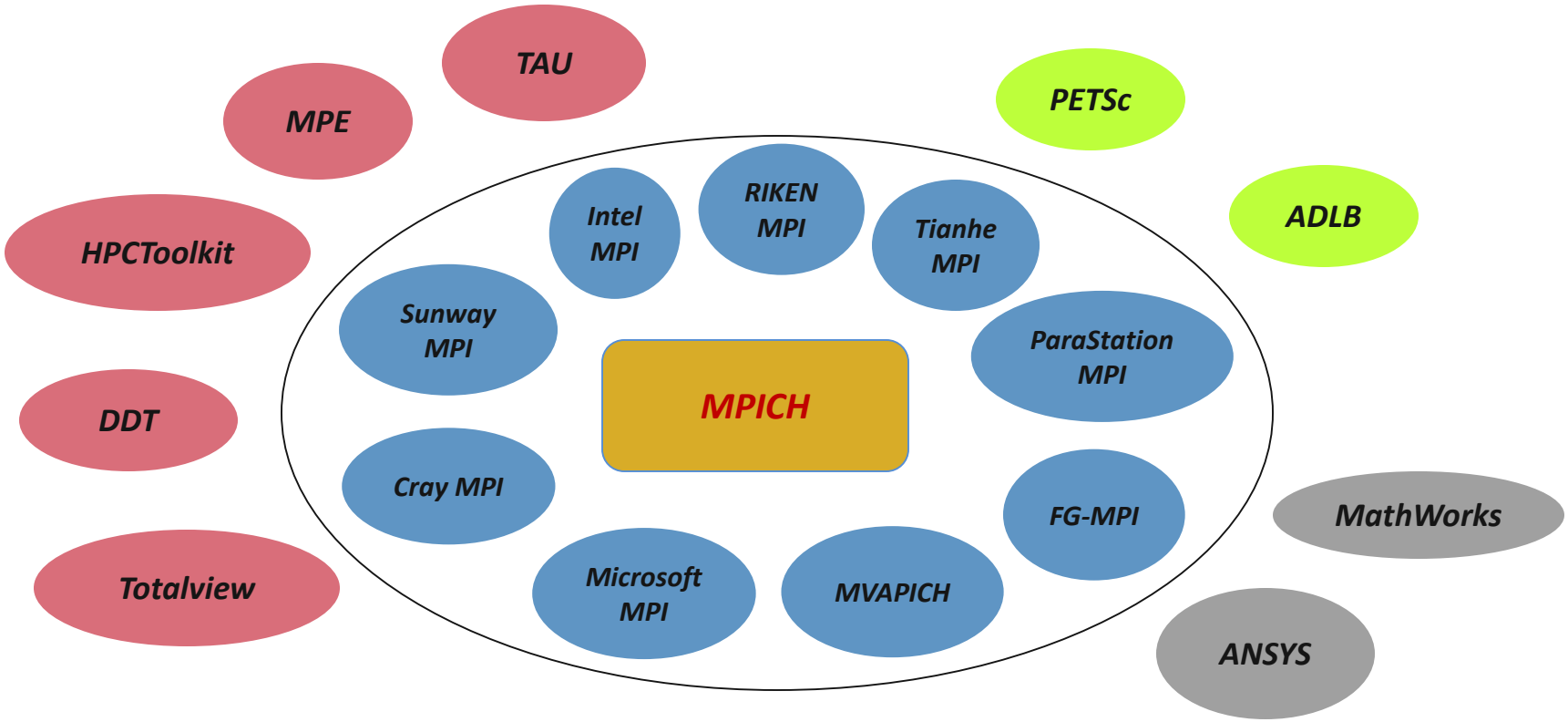
MPICH and it derivatives in the Top 10

1. **Summit (US): Spectrum MPI**
2. ***TaihuLight (China): Sunway MPI***
3. **Sierra (US): Spectrum MPI**
4. ***Tianhe-2A (China): MPICH-TH2***
5. ***ABCI (Japan): Intel MPI and MVAPICH***
6. ***Piz Daint (Germany): Cray MPI***
7. ***Titan (US): Cray MPI***
8. ***Sequoia (US): IBM PE MPI***
9. ***Trinity (US): Cray MPI***
10. ***Cori (US): Cray MPI***

MPICH and its derivatives power 8 of the top 10 supercomputers (Jun. 2018 Top500 rankings)

MPICH: Goals and Philosophy

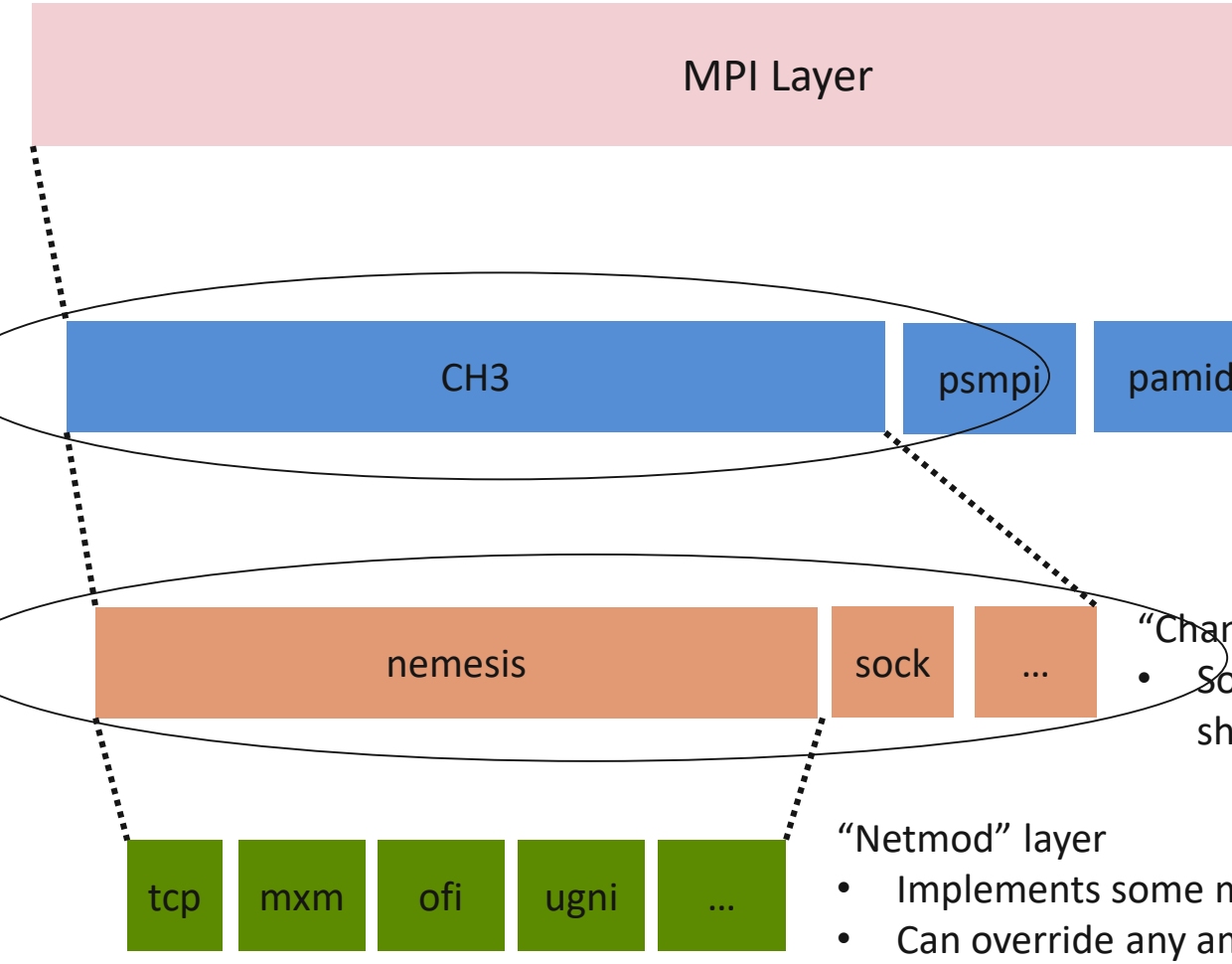
- MPICH continues to aim to be the preferred MPI implementations on the top machines in the world
- Our philosophy is to create an "MPICH Ecosystem"



MPICH-3.3 Features

1. New Device CH4: Low-instruction count communication
   - ***Thanks to Intel, Mellanox, and RIKEN for their significant contributions!***
   - Very lightweight communication
2. Support for very high thread concurrency (***partnership with Intel***)
   - Improvements to message rates in highly threaded environments
   - Move from the LOCK/WORK/UNLOCK model to a scalable ENQUEUE/DEQUEUE model
3. Memory Scalability Optimizations
4. Hardware Acceleration for MPI Atomics
5. Scalable job startup
6. New Collective Infrastructure (***partnership with Intel***)
   - New collective algorithms, more comprehensive algorithm selection
7. Topology-awareness improvements
8. Fault tolerance improvements
   - Non-catastrophic errors and limited abort scope
9. CUDA-awareness for contiguous GPU buffers (***partnership with Mellanox and NVIDIA***)

MPICH layered structure: Current and Planned



Platform independent code
- Collectives
- Communicator management

"Device" layer: Machine-specific functionality
- Derived datatype management
- RMA

"Channel" layer
- Some "core" functionality, such as shared memory

"Netmod" layer
- Implements some mandatory functionality
- Can override any amount of optional functionality (better bcast, better barrier)

CH4 Design Goals

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

High-Level Netmod API

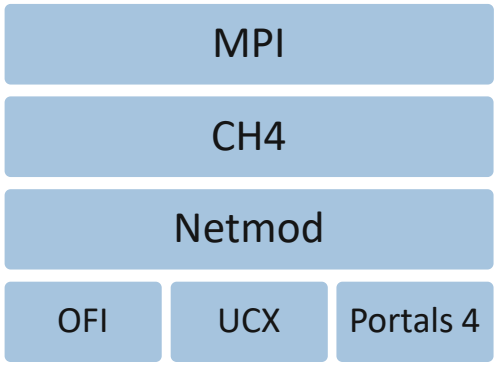
- Give more control to the network
  - `netmod_isend`
  - `netmod_irecv`
  - `netmod_put`
  - `netmod_get`
- Fallback to Active Message based communication when necessary
  - Operations not supported by the network

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

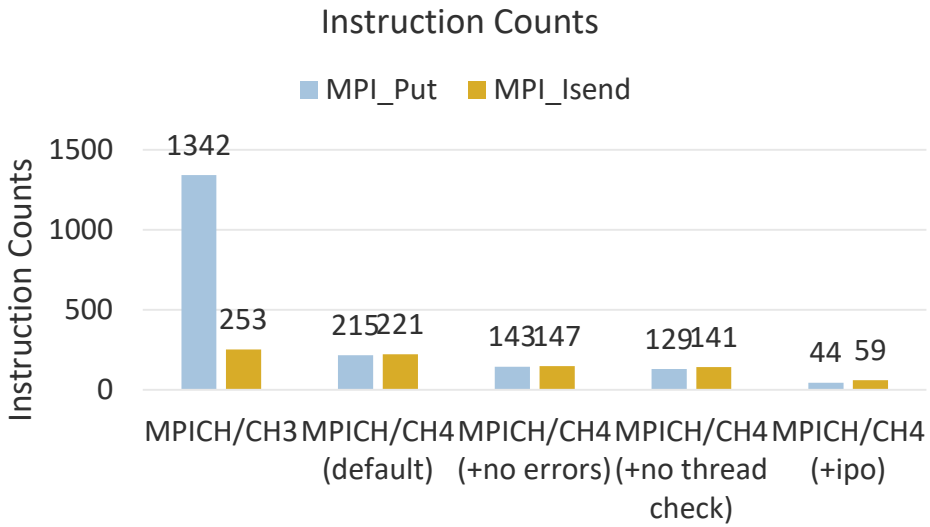


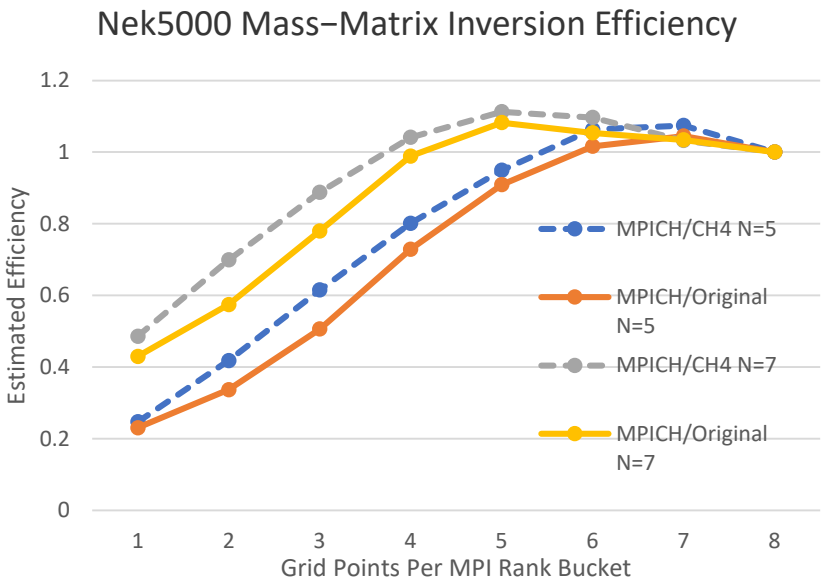
Minimal Per Process Data

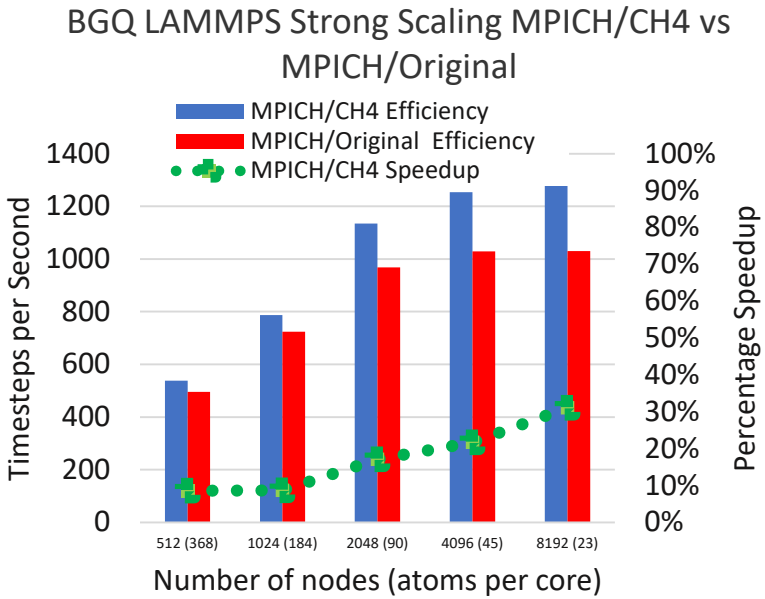
- Global address table
  - Contains all process addresses
  - Index into global table by translating `(rank+comm)`

Partnership with Intel, Mellanox, RIKEN

Lower Overheads = Better Strong Scaling



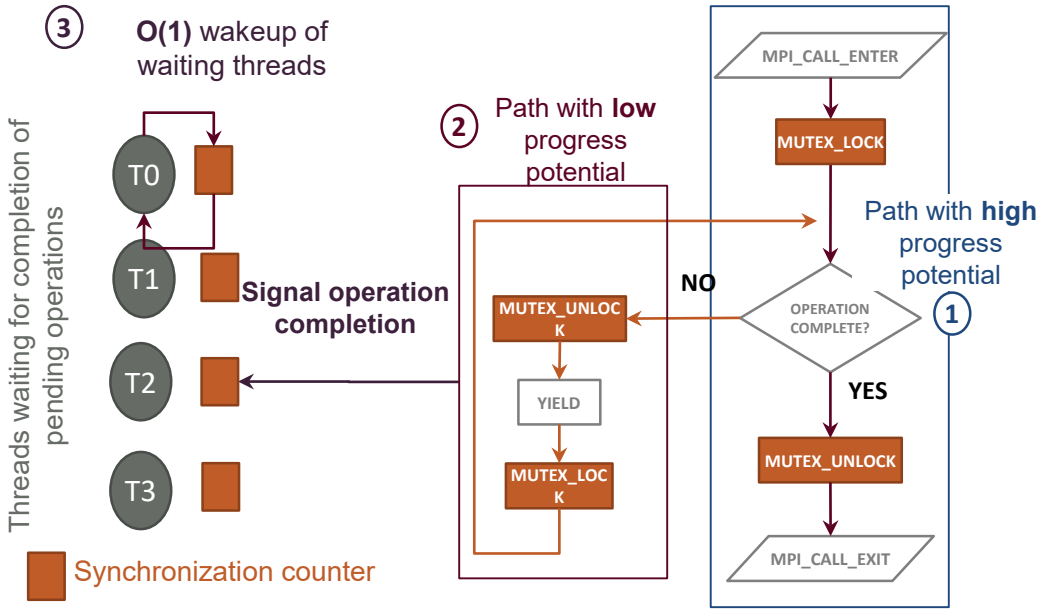


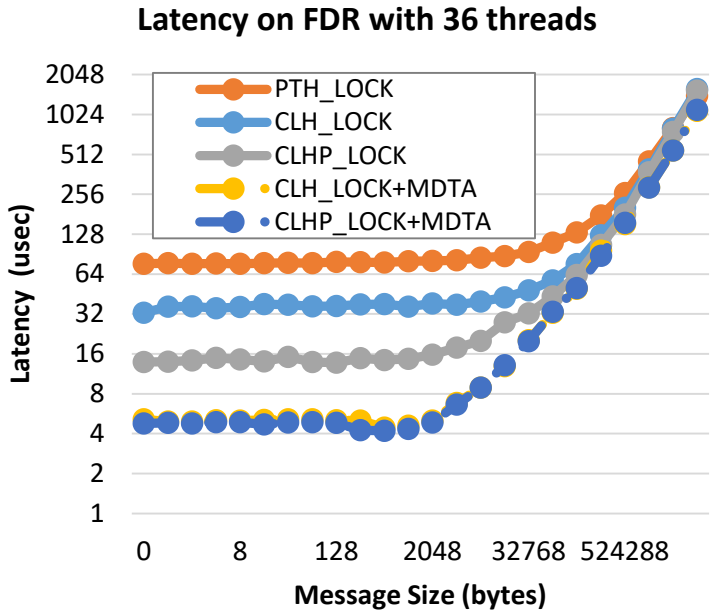


Improving MPI_THREAD_MULTIPLE

- Leveraging hardware knowledge for scalable locking practices
- MPI-knowledge driven thread synchronization
  - Work (e.g., message reception) driven thread reactivation on blocking operations
  - Prioritizing execution paths with the highest progress

POC: Halim Amer <aamer@anl.gov>

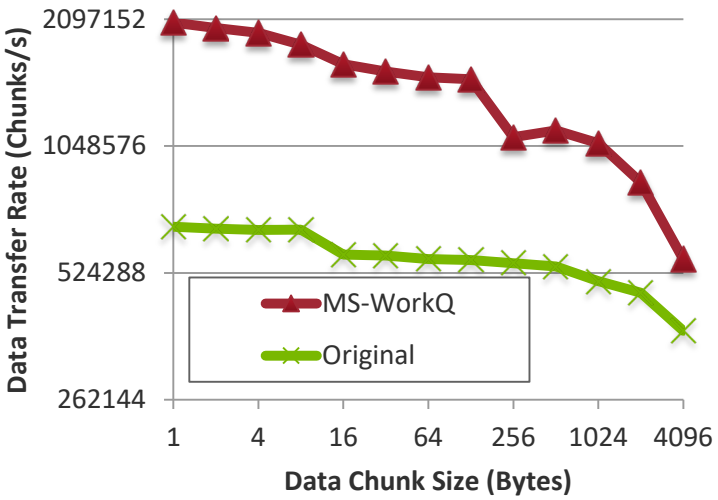


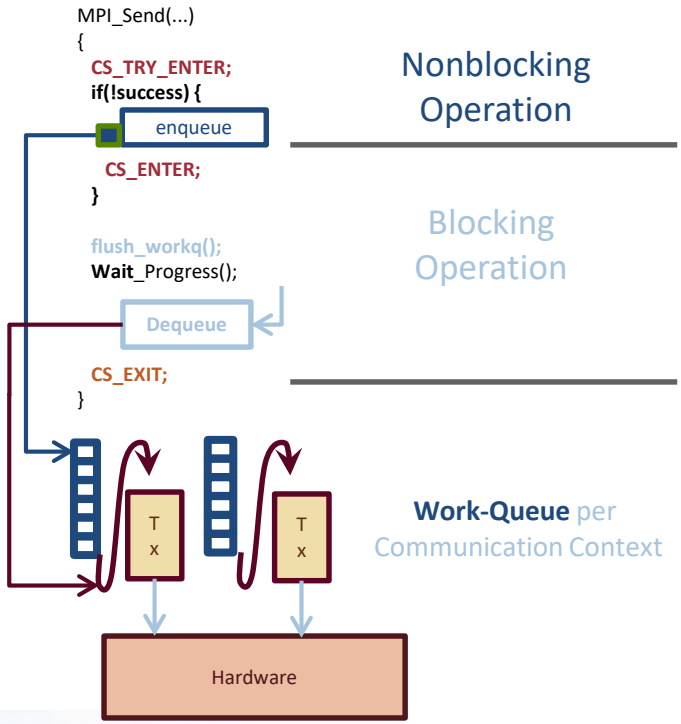


Multithreaded MPI Work-Queue Model

- **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

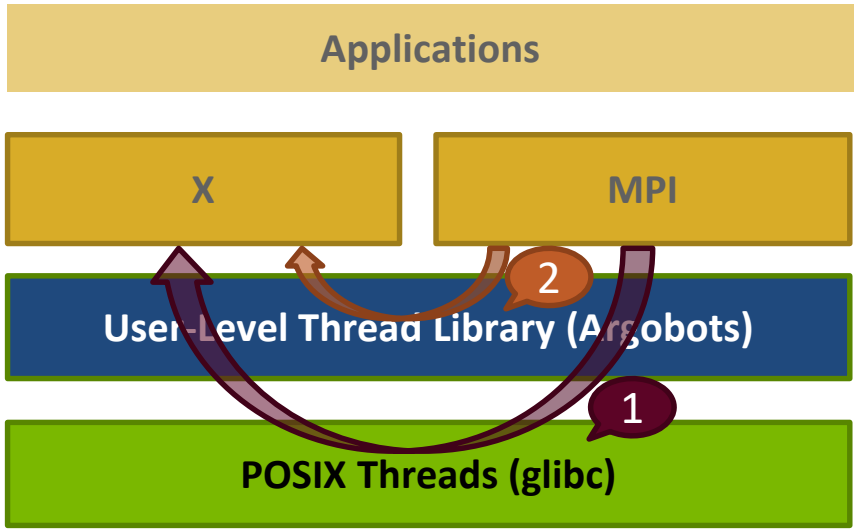
Partnership with Intel

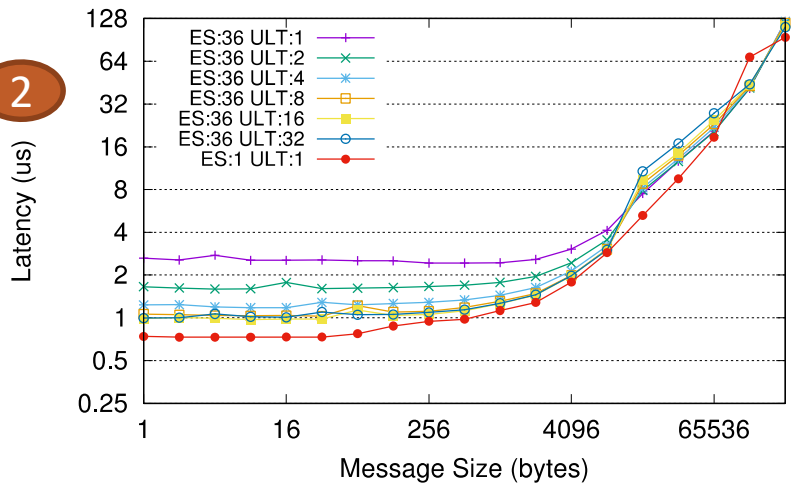




Interoperability with X Through User-Level Threads

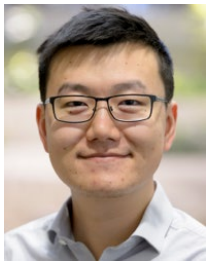
- MPI interoperates with shared-memory programming systems (X) either
  - Directly, if X is a threading runtime
  - Indirectly, through it's threading runtime
- MPICH can interoperate with/through the OS-level threading layer (e.g., Pthreads package) **(heavy)**
- MPICH can also interoperate with X through the **Argobots** package **(light)**
- Interoperation with lightweight threads allows
  - Better communication/computation overlapping
  - Reduced thread synchronization costs
  - Latency hiding of blocking operations





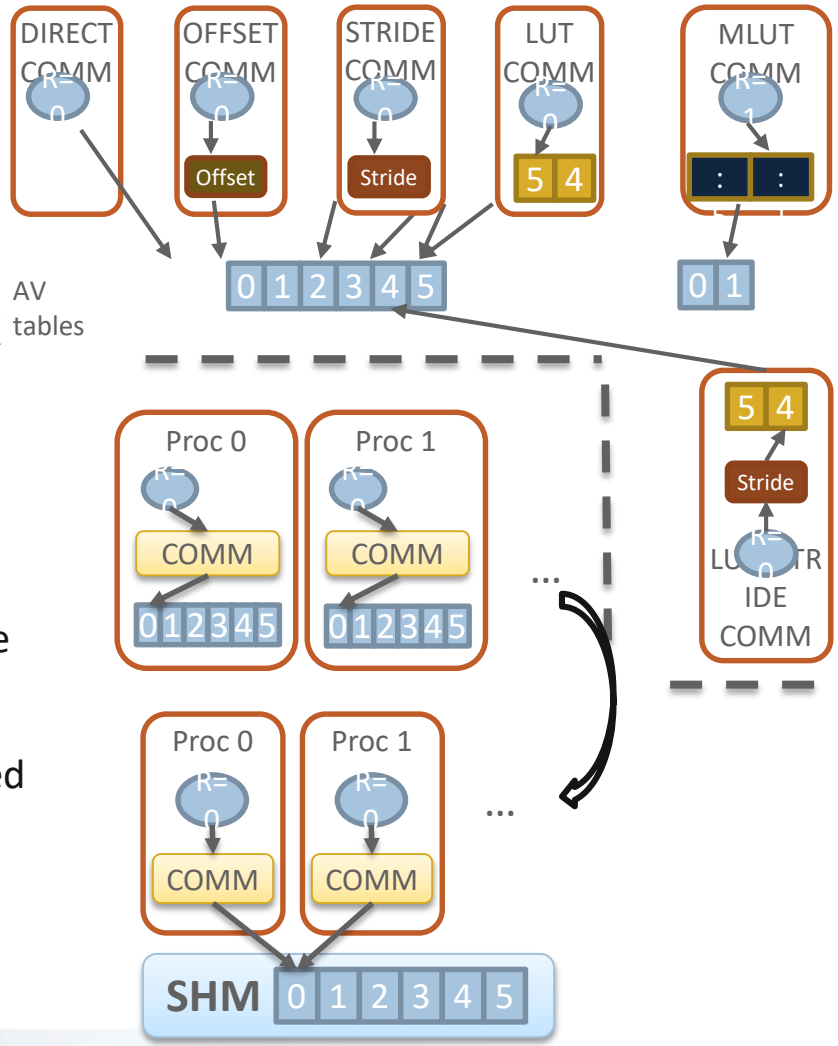
- Point-to-point latency with 36 Haswell Cores and Mellanox FDR
- Proven better latency hiding than Pthreads-level interaction

Memory Scalable Network Address Management



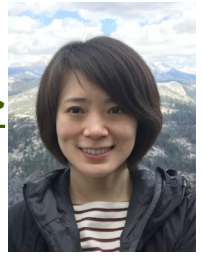
POC: Yanfei Guo <yguo@anl.gov>

- 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



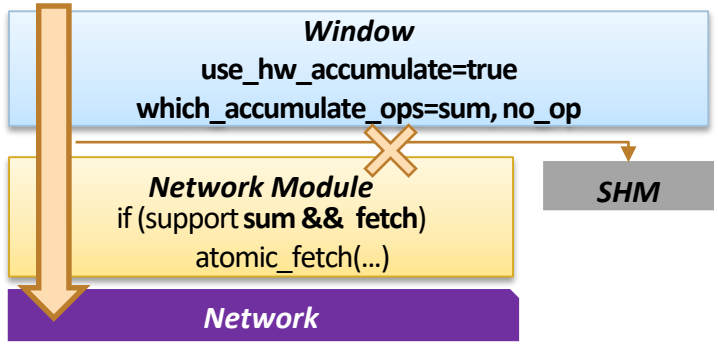
Hardware Acceleration for MPI Atomics

Min Si
<msi@anl.gov>

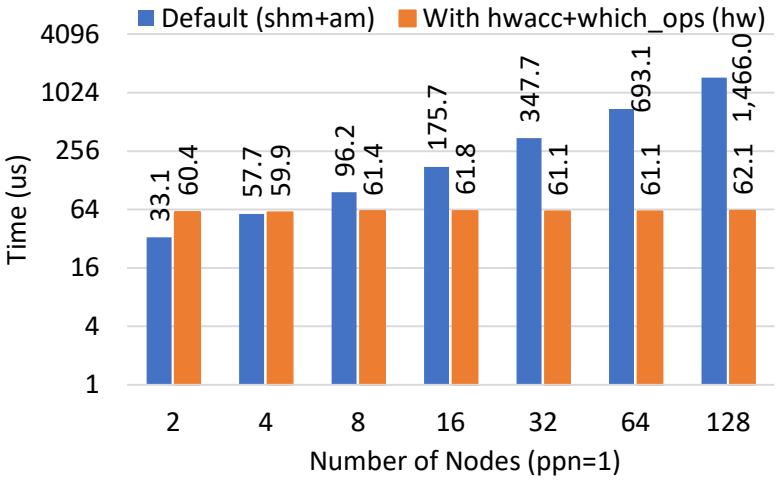


- MPI cannot naturally use hardware atomics for RMA accumulates:
  - Enabling SHM-based accumulates breaks atomicity with network hardware atomics
  - Mismatching between MPI semantics and hardware capability (e.g., maxloc, minloc)
- Optimizations:
  - New info **"disable_shm_accumulate"**:
    - Only use network-based atomics for accumulates in specific windows
  - New info **"which_accumulate_ops"**:
    - Limit the need of atomic support at runtime
  - User specifies all accumulate reduce ops (e.g., "sum, replace") used over a window
  - Runtime queries their hardware support before communication in order to utilize hardware atomics if possible

Fetch_and_op(,...,MPI_NO_OP)



Hardware accelerated RMA atomics with info hints in NXTVAL* kernel on Theta/KNL (cache-quad)



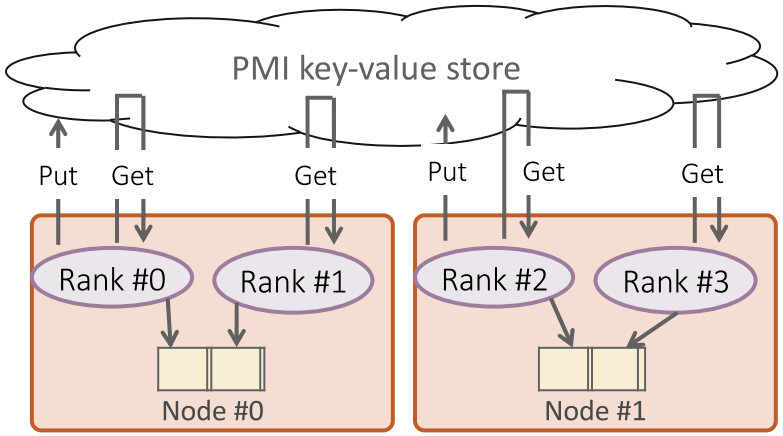
** NXTVAL is the task scheduling module used in NWChem where every process issues MPI_Fetch_and_op to update a global counter with long type.*

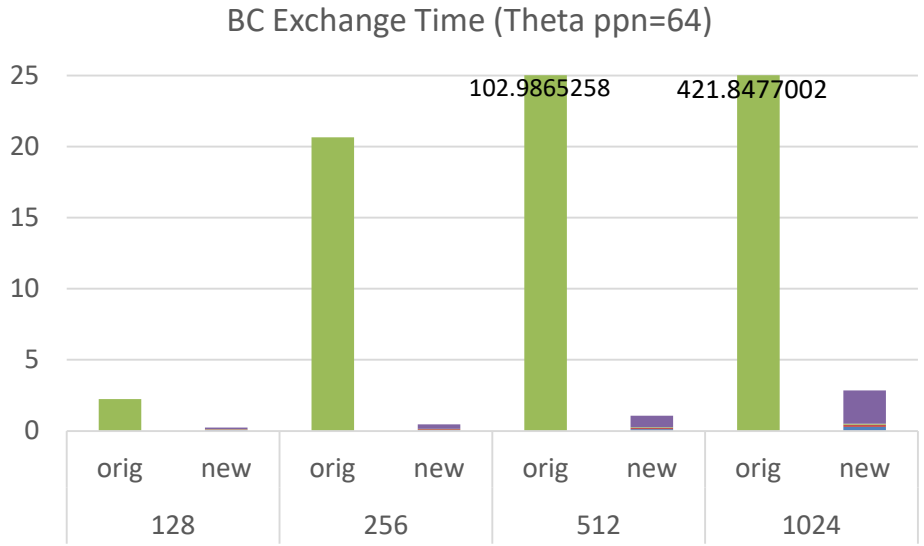
Scalable Job Startup

Using shared memory and MPI collectives to reduce startup time

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

- Only node-root processes do PMI_KVS_Put
  - All processes on the node do PMI_KVS_Get into shared memory segment
  - No redundant lookups at the node level
- Remaining business cards are exchanged with MPIR_Allgather using the node-roots communicator, again into shared memory.
- BC exchange time reduced from 421 -> ~3 seconds on 1024 KNL nodes, 64 ppn on ALCF Theta (MPICH + OFI/gni)
- Launch time of ~8 seconds on the full OFP machine (8192 KNL nodes/64 ppn)





New Collectives Infrastructure

- Thanks to Intel for the significant work on this infrastructure
- Two major improvements:
  - 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)

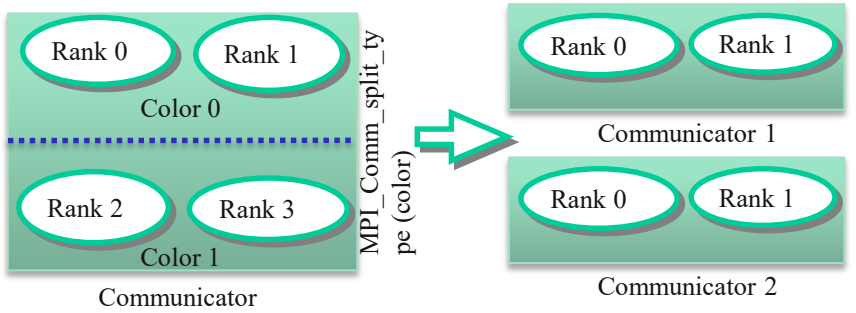
POC: Taru Doodi <taru.doodi@intel.com>

Contributed by Intel (with some minor help from Argonne)

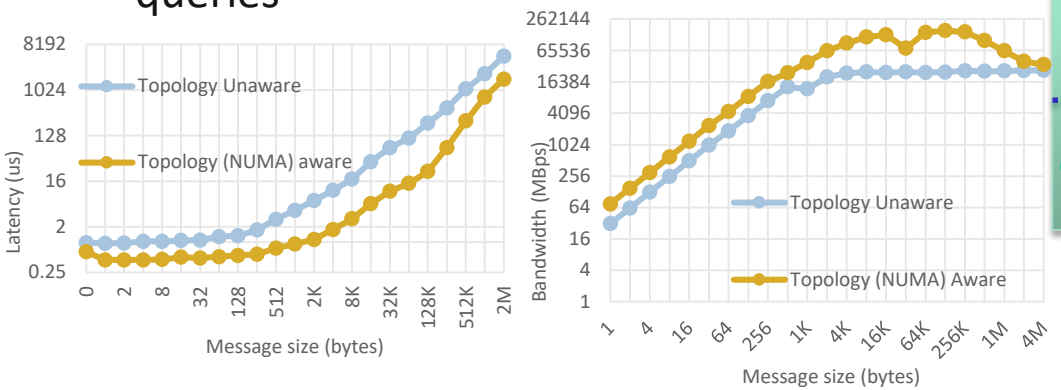
Topology optimizations for MPI_Comm_split_type

- MPI_Comm_split_type partitions a communicator into multiple comms
  - Takes a color argument and creates communicators with the same color
  - Communication local to the processes in the same communicator
- Topology awareness: Communicator split such that the new communicators created are local to processes sharing hardware resources
- User-provided info hints for shared memory optimization hwloc for bindset and topology queries

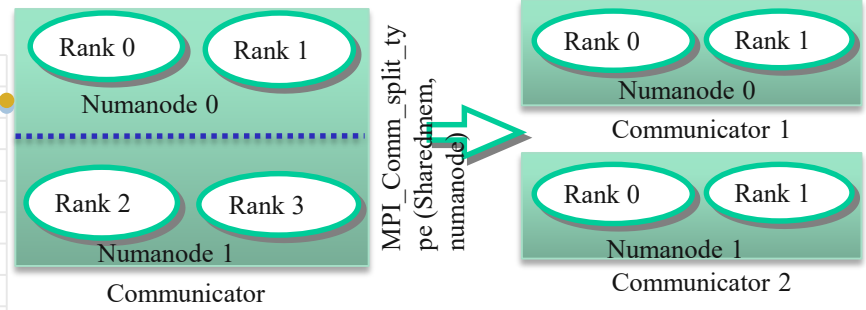
POC: Kavitha Madhu <kmadhu@anl.gov>



MPI communicator split with user defined color



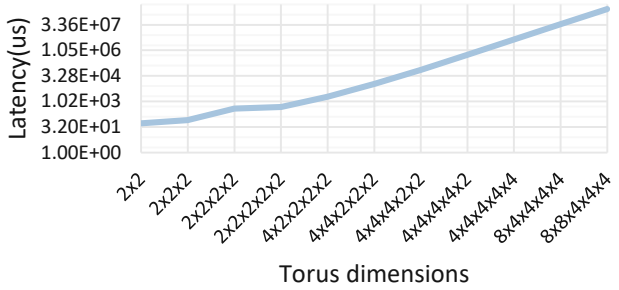
OSU Multi latency and Bandwidth tests on a single Skylake node



MPI communicator split for numa nodes

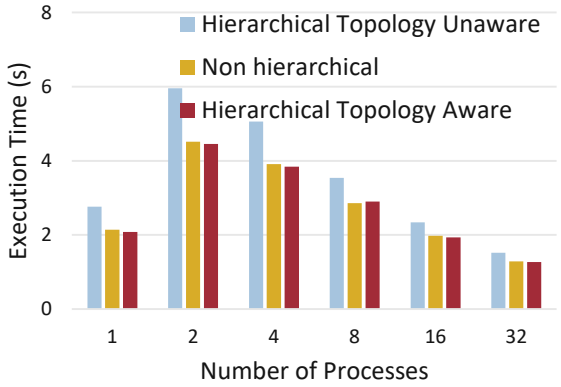
New Developments

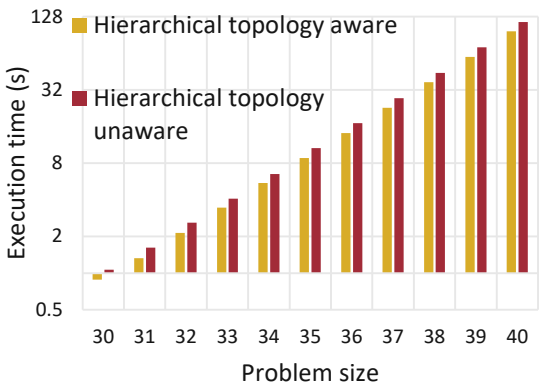
- Network Topology parsed using netloc, topology detection and classification at init time
  - Currently supported network toplogies: Fat tree, Torus



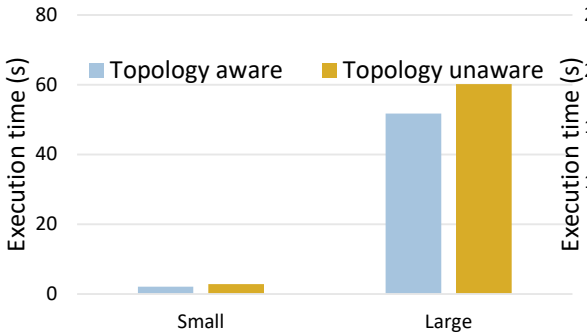
Topology Detection Latency: Torus network

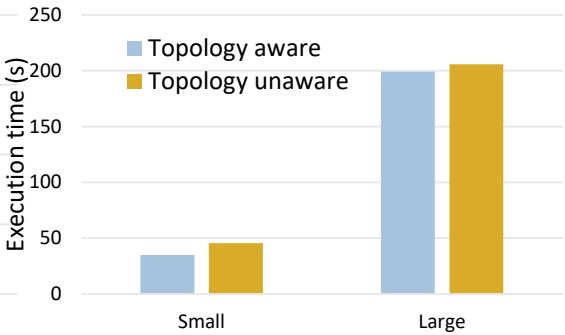
- MPI_Comm_split_type arguments:
  - Shared memory key
  - Supported info hints:
  - Node level hints: Nodes, sockets (packages), numa nodes, cache
  - Network level hints: Switch level, torus dimension
  - Proximity hints: IO devices, GPU devices
  - Resource limit hints: Subcommunicator minimum size, Subcommunicator minimum memory size





Topology aware Work stealing scalability and speedup with Fibonacci in one Skylake node





Topology aware Work stealing speedup with UTS on BG/Q

Fault Tolerance Improvements

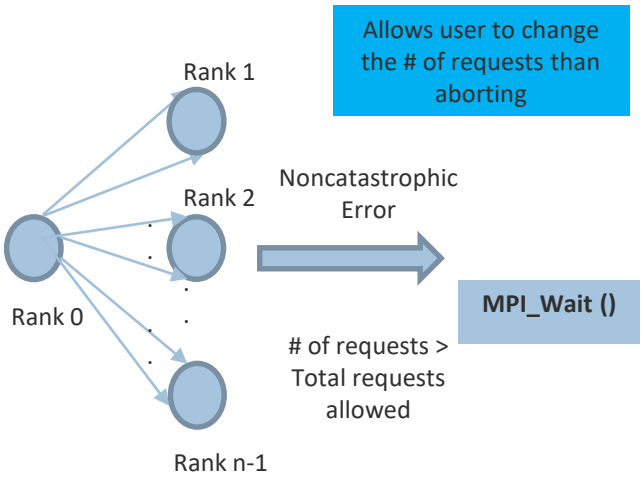
POC: Yanfei Guo
<yguo@anl.gov>

Non-catastrophic Errors

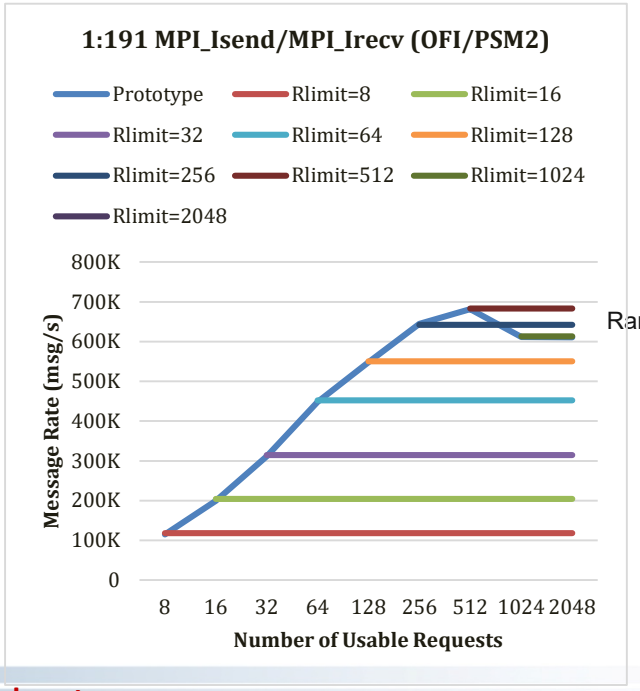
- Defined the state of MPI after errors and identified non-catastrophic error such as resource exhaustions, shared memory allocation failure and no-async progress
- A prototype implementation is carried out to handle non-catastrophic error

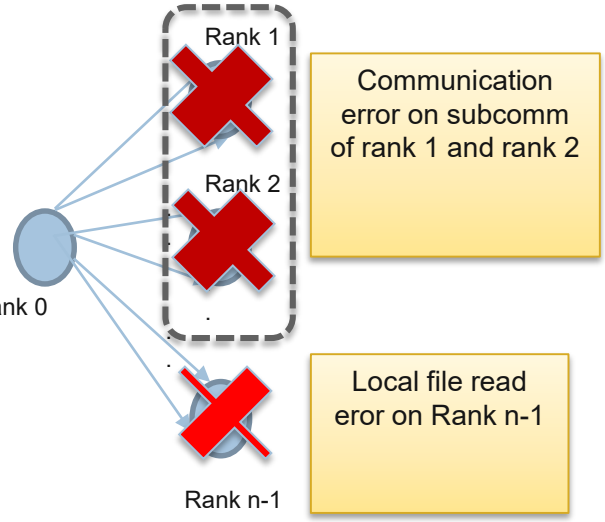
Definition of Error Scope

- Aborting applications is too expensive in large applications
- To avoid always aborting an application, proposed work clearly defined the scope of error and added prototype implementation to allow error handler for only aborting connected processes in a sub-communicator



MPI processes do not abort on noncatastrophic error





Aborting MPI_Processes based on the location of error occurrence

CUDA Awareness

POC: Giuseppe Congiu <gcongiu@anl.gov>

- Enable use of CUDA buffers in MPI communications
  - Predefined and Contiguous datatypes supported through UCX
    - CUDA IPC and cudaMemcpy for intranode communication (multi-process/single GPU)
    - GPUDirect for internode communication

Partnership with Mellanox and NVIDIA

MPICH 3.4 Planned Features

1. Support for heterogeneous memory (e.g., MCDRAM)
2. Expanded CUDA-awareness (***partnership with Mellanox and NVIDIA***)
   - Reductions, RMA accumulate
3. Dataloop extensions and optimizations
   - Optimized code paths for common datatype compositions
4. Topology-awareness improvements
   - Virtual topology functionality, neighborhood collectives
5. Multi-WorkQ/Multi-VNI communication (***partnership with Intel***)
   - Utilizing network hardware parallelism, MPI Endpoints prototype
6. ULFM Prototype
7. Hydra rewrite (***partnership with Intel***)
   - More scalable launch, optimized internal protocol