Welcome to the…
MPI BoF

Organized by the

2.3.1.07 Exascale MPI (Yanfei Guo, ANL, PI) and
2.3.1.17 OMPI-X: Open MPI for Exascale (David Bernholdt, ORNL, PI)

project teams, with presentations by

Matthew Dosanjh (SNL), Ken Raffenetti (ANL), Howard Pritchard (LANL), Yanfei Guo (ANL), Aurelien Bouteiller (UTK), and

Jim Dinan (NVIDIA) chair MPI Forum HACC WG

3:00-4:30pm ET Thursday 12 May 2022

Welcome and Introduction

- Goal: Provide an opportunity for ECP participants to discuss the MPI standard and key implementations

- Outline
  - Major MPI 4 features
    - Sessions, Persistent Collectives, Partitioned Communications – Matthew Dosanjh (SNL)
    - MPI_T, Hardware Topologies, Error Management – Ken Raffenetti (ANL)
  - MPI Forum next steps – Howard Pritchard (LANL)
  - MPI Hybrid & Accelerator Working Group Update– Jim Dinan (NVIDIA)
  - Recent highlights in MPICH – Yanfei Guo (ANL)
  - Recent highlights in Open MPI – Aurelien Bouteiller (UTK)
  - General Q&A and discussion – David Bernholdt (ORNL), moderator

MPI Sessions, Persistent Collectives, and Partitioned Communication

Matthew Dosanjh
Sandia National Laboratories

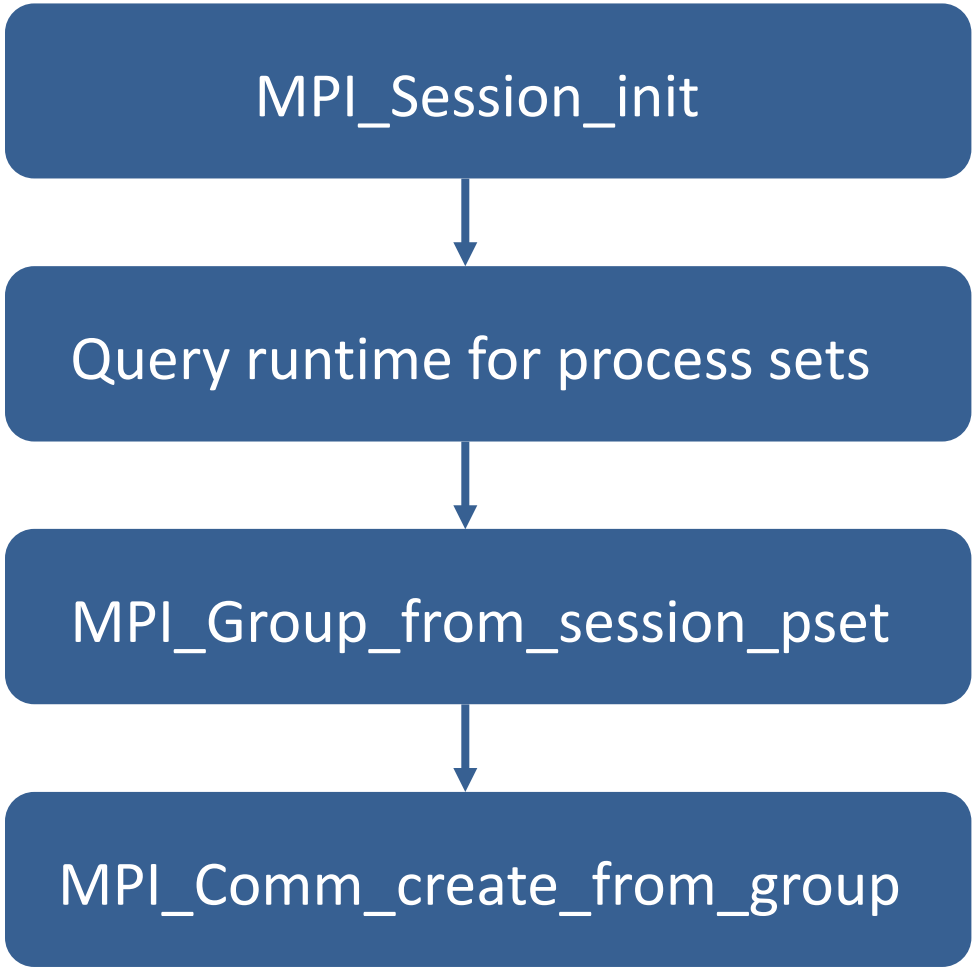
SAND2022-6299 C

Based on slides by
Howard Prichard
Los Alamos Laboratory

ECP EXASCALE COMPUTING PROJECT

MPI Sessions

- In the MPI 4.0 standard
- Sessions API supported in MPICH 4 release stream and Open MPI 5.0 release stream (not yet released)
- Complete implementation of MPI 4 Sessions API
- Requires PMIx 4 or newer (for Open MPI)
- Set of examples are available at https://github.com/hppritcha/mpi_sessions_tests



MPI Persistent Collectives

- Released in the MPI 4 standard at Barcelona MPI Forum Meeting (9/18)
- Aims to accelerate applications with repetitive collective operations
- Initialization call for a persistent collective operation is **non-local,** all members of the communicator being supplied to the initialization operation must eventually invoke this initialization call
- The info argument to these initialization calls can be used to specify MPI implementation specific optimizations.

MPI Persistent Collectives

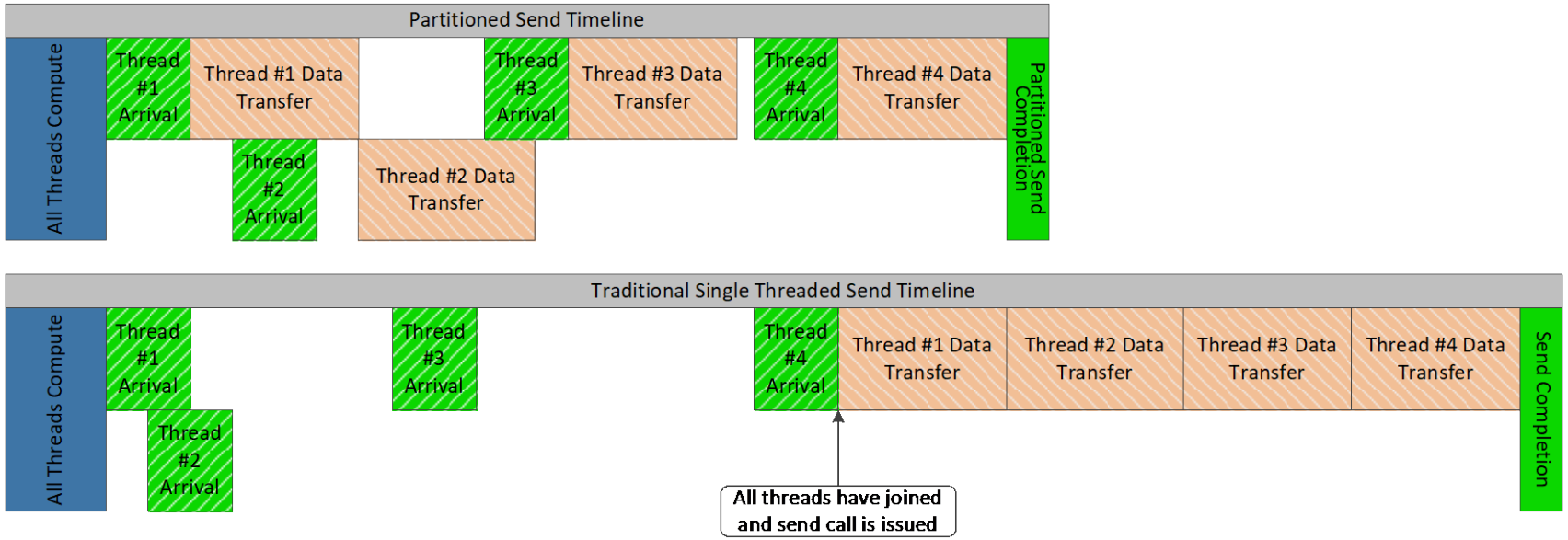
- Starting/completion semantics
- MPI_*_init creates a *request* and may be used zero or more times to start the corresponding collective operation using *MPI_Start* or *MPI_Startall*.
- This request must be inactive before starting a persistent collective operation
  - Starting a persistent collect operation makes it *active*
  - Only one outstanding collective operation on a request
- Loosens ordering constraints
  - Ordering matters for MPI_*_init not MPI_Start*.
- *MPI_Wait, MPI_Test,* etc. completes the operation but does not free the request

Partitioned Communication

- Share “ownership” of a buffer between MPI and user code
  - Similar to RMA Windows
- Uses a persistent buffer ‘partitioned’ into smaller chunks
- User informs MPI when data is ‘ready’ to send.
  - User code must manage memory epochs
  - Once a partition is marked ready, it’s data can not be changed
- Thread agnostic with a minimal synchronization overhead

Early Bird Communication

- New type of overlap
- Data can be transferred before the last compute thread is finished
- Can consolidate some messaging overhead
- Targeted for accelerator use in a future version of the MPI standard



Example (send side)

```
MPI_Psend_init(message, send_partitions, 1, send_type, dest, tag,
          info, MPI_COMM_WORLD, &request);
MPI_Start(&request);
#pragma omp parallel for shared(request) num_threads(NUM_THREADS)
for (int i=0; i<send_partitions; i++)
{
   /* compute and fill partition #i, then mark ready: */
   MPI_Pready(i, &request);
}
while(!flag)
{
   /* Do useful work */
   MPI_Test(&request, &flag, MPI_STATUS_IGNORE);
   /* Do useful work */
}
MPI_Request_free(&request);
```

Example (receive side)

```
MPI_Precv_init(message, recv_partitions, recv_partlength, MPI_DOUBLE,
          source, tag, info, MPI_COMM_WORLD, &request);
MPI_Start(&request);
#pragma omp parallel for shared(request) num_threads(NUM_THREADS)
for (int j=0; j<recv_partitions; j+=2)
{
   int part1_complete = 0;
   int part2_complete = 0;
   while(part1_complete == 0 || part2_complete == 0)
   {
      /* test partition #j and #j+1 */
      MPI_Parrived(&request, j, &flag);
      if(flag && part1_complete == 0)
      {
         part1_complete++;
         /* Do work using partition j data */
      }
      if (j+1 < recv_partitions) {
         MPI_Parrived(&request, j+1, &flag);
         if(flag && part2_complete == 0)
         {
            part2_complete++;
            /* Do work using partition j+1 */
         }
      }
      else {
          part2_complete++;
      }
   }
}
```

```
.
.
.
while(!flag)
{
   /* Do useful work */
   MPI_Test(&request, &flag, MPI_STATUS_IGNORE);
   /* Do useful work */
}
MPI_Request_free(&request);
```

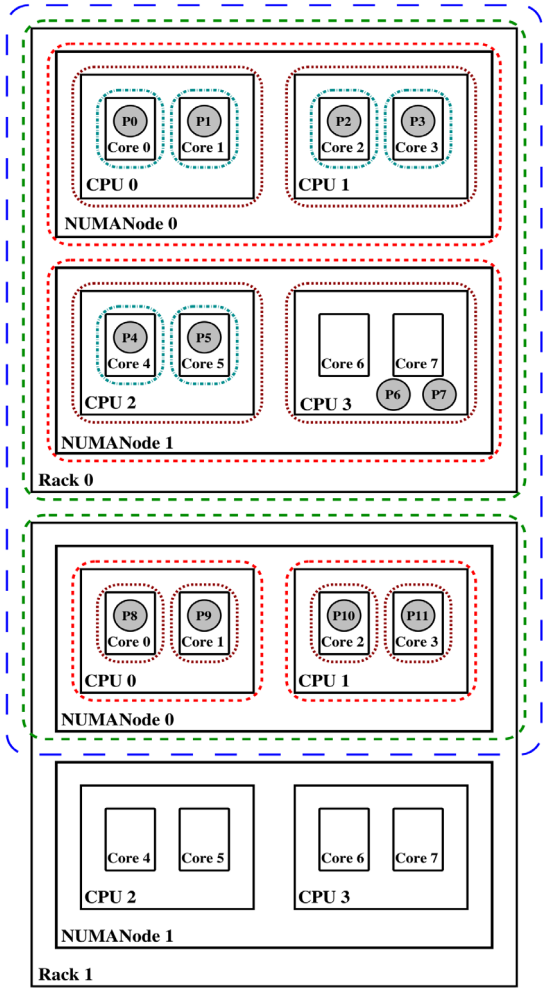
Partitioned communication (MPI 4.0) implementation status

- General implementation layered on top of MPI
  - https://github.com/tonyskjellum/MPIPCL
- Available in MPICH since v4.0a1 release
- Available in Main branch of OpenMPI

Hardware Topologies, Error Handling, and MPI_T Events

Ken Raffenetti
Argonne National Laboratory

New Ways to Adapt to Hardware Topologies



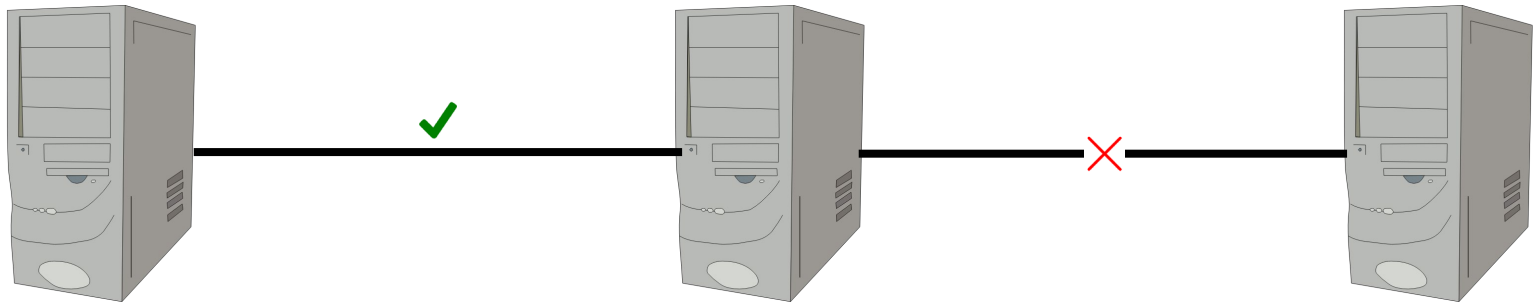
- New systems are increasingly hierarchical
  - Mapping of MPI processes to hardware resources is critical for efficiency and performance
  - Desire for hardware topology-aware communicators

- New values for MPI_COMM_SPLIT_TYPE
  - Guided Mode (MPI_COMM_TYPE_HW_GUIDED)
    - User can provide info key-value to specify hardware level
  - Unguided Mode (MPI_COMM_TYPE_HW_UNGUIDED)
    - Start from input communicator (e.g. MPI_COMM_WORLD)
    - Step-wise go to lower/deeper levels
    - Iterative until leaf is reached

Improved Error Handling

- Goal: allow applications to limit impact of failures to avoid termination
  - Specify that MPI_SUCCESS indicates only the result of the operation, not the state of the MPI library
  - Localize error impact of some MPI operations. (e.g. MPI_ALLOC_MEM will now raise an error on MPI_COMM_SELF, not MPI_COMM_WORLD)
  - Specify that MPI should avoid fatal errors when the user does not use MPI_ERRORS_ARE_FATAL
  - New MPI Error Handler - MPI_ERRORS_ABORT
  - Allow the user to specify the default error handler at `mpiexec` time

- What can you do with this?
  - Point-to-point communication with sockets-like error handling.
  - Enables manager/worker and other non-traditional types of applications.
  - Enterprise applications that want to move from sockets to MPI can do so.



MPI_T Events: Callback-driven event information

- Motivation
  - PMPI does not provide access to MPI internal state information
  - MPI_T performance variables only show aggregated information
- New interface to query available runtime event types
  - Follows the MPI_T variable approach
  - No specific event types mandated by standard
  - Event structure can be inferred at runtime
  - Need community input!
- Register callback functions to be called by the MPI runtime
  - Runtime may defer callback invocation (tool can query event time)
  - Runtime may reduce restrictions on callback functions per invocation
  - Callback can query event information individually or copy data en bloc

```
/* 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)
```

MPI Forum next steps

Howard Pritchard
Los Alamos National Laboratory

NNSA National Nuclear Security Administration
U.S. DEPARTMENT OF ENERGY | Office of Science

What's happening in MPI Forum

- **MPI Standard document related activities**
  - MPI 4.0 standard was released June, 2021
  - Preparing a 4.1 release
  - Improving maintainability of the Latex source code
  - Improving readability of the document, e.g. bettering highlighting of examples
  - Semantics terms supplementary document https://www.mpi-forum.org/docs/mpi-4.0/addendum-Semantics.pdf

MPI Forum Working group highlights

- **Sessions WG**
  - MPI 4.1 clarifications/corrections
  - Using sessions model in dynamic resource environments
    - Versioning of process sets (maybe opaque to application)
    - Considering various options for application notification of process set changes
- **RMA WG**
  - Corrections/clarifications for the MPI 4.1 standard
  - Conformance to the semantic terms document
  - Upcoming workshop(June) - https://mpiwg-rma.github.io/forma22/
- **Tools WG**
  - QMPI / MPI function pointer interception -- Next-generation of MPI Profiling (PMPI) interface
- **Hardware Topologies**
  - Deprecate MPI_COMM_TYPE_HW_GUIDED, to be replaced by MPI_COMM_TYPE_RESOURCE_GUIDED
  - New methods based on process sets

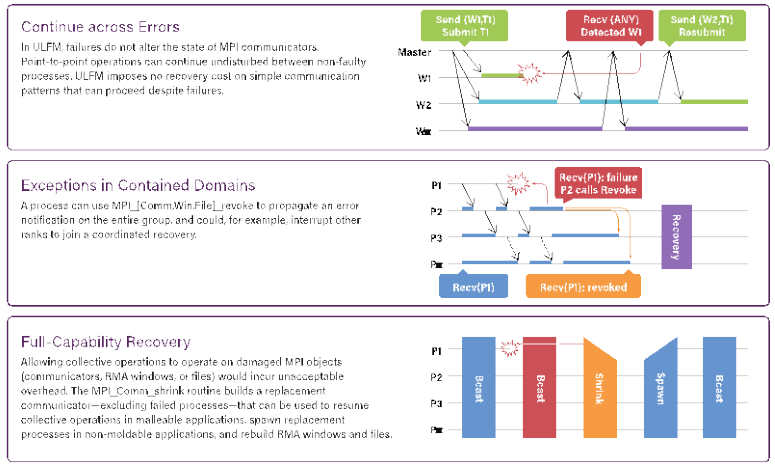
MPI Forum Working Group Highlights (cont.)

- **Hybrid / Accelerators WG**
  - Accelerator bindings for partitioned communication https://github.com/mpiwg-hybrid/hybrid-issues/issues/4
  - Accelerator info keys https://github.com/mpiwg-hybrid/hybrid-issues/issues/3
  - Stream/Graph Based MPI Operations https://github.com/mpiwg-hybrid/hybrid-issues/issues/5
  - Continuations https://github.com/mpiwg-hybrid/hybrid-issues/issues/6

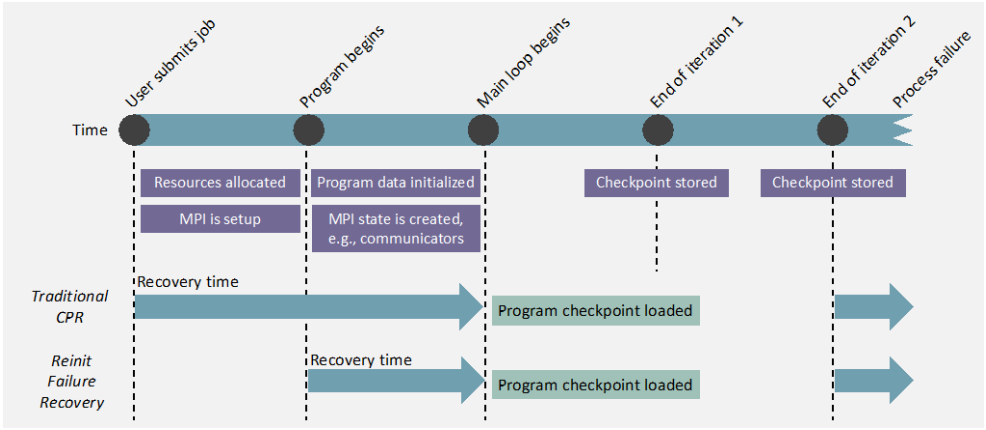
- **Fault Tolerance WG**
  - MPI-5: Broad directions
    - POSIX-like error handling gives fallback for errors (crude fault tolerance) but no MPI fault recovery
    - **Fine-grained recovery**: “Mini-ULFM” – Led by Aurelien Bouteiller
    - **Course-grained recovery**: “Reinit” – Led by Ignacio Laguna
  - Convergence Ideas to support all models
    - Interrupting Error Handlers (trigger outside MPI calls)
    - Implicit Propagation of Errors (avoid need to call MPI_COMM_REVOKE in common use cases)
    - Resilient Sessions - Sessions used to restore MPI objects after errors (alternative to MPI_COMM_SHRINK)
    - Online switching of models (use ULFM, then use Reinit, then use ULFM again)

Items brewing in working groups (cont.)

- **Fault Tolerance WG**
  - *ULFM* for execution of fault tolerant applications
    - Use error handlers to react and repair effects of failures
    - Now directly available in Open MPI 5.0 (as well as MPICH)
  - *Reinit* for global-restart fault tolerance
    - Automatic recovery of process and node failures
    - Minimum changes to MPI applications
    - Focused on bulk synchronous codes with checkpoint/restart
    - Draft specification released (v0.1.3) & under discussion



ULFM Example use cases: continue without repair, repair in domains, respawn missing processes



Reinit Example: showing the rollback upon failure to reduce overheads on recovery of codes with C/R support.

Joining Working Groups

Active Working Groups

https://www.mpi-forum.org/mpi-40/#active-working-groups

- **Fault Tolerance WG**
  - Mondays @ 12pm ET (bi-weekly)
  - https://github.com/mpiwg-ft/ft-issues/wiki
- **Sessions WG**
  - Mondays @ 1pm ET
  - https://github.com/mpiwg-sessions/sessions-issues/wiki
- **Tools WG**
  - Thursdays @ 11am ET
  - https://github.com/mpiwg-tools/tools-issues/wiki/Meetings
- **Persistent/Partitioned/… WG & Hybrid / Accelerators WG (share timeslot)**
  - Wednesdays @ 10am ET
  - https://github.com/mpiwg-hybrid/hybrid-issues/wiki

MPI HYBRID & ACCELERATOR WORKING GROUP UPDATE

James Dinan, NVIDIA
HACC WG Chair
ECP MPI BoF ‘22

HYBRID & ACCELERATOR WORKING GROUP

Mission: Improve interoperability of MPI with other programming models

Active topics:

1. Continuations proposal [Joseph Schuchart, UTK] #6
2. Clarification of thread ordering rules [Daniel Holmes, Intel] #117
3. Integration with accelerator programming models:
    1. Accelerator info keys [Jim Dinan, NVIDIA] #3
    2. Stream/Graph Based MPI Operations [Jim Dinan, NVIDIA] #5
    3. Accelerator bindings for partitioned communication [Jim D., NVIDIA + Maria Garzaran, Intel] #4
    4. Partitioned communication buffer preparation [Ryan Grant, Queen's U.] #264

More information: https://github.com/mpiwg-hybrid/hybrid-issues/wiki

COMPLETION CONTINUATIONS

Treat the completion of an MPI operation as continuation of some activity

- Interoperability with asynchronous and multithreaded programming models
- Register callbacks that continue the activity upon completion of an MPI operation

```
11  MPI_Request cont_req;
12  MPIX_Continue_init(&cont_req);
13
14  omp_event_handle_t event;
15  int value;
16  #pragma omp task depend(out:value) detach(event)
17  {
18    MPI_Request req;
19    MPI_Irecv(&value, ..., &req);
20    MPIX_Continue(&req, &release_event, event, MPI_STATUS_NULL, cont_req);
21  }
22
23  #pragma omp task depend(in: value)
24  {
25    // process value
26  }
```

```
31  void release_event(MPI_Status status, void *data)
32  {
33    omp_event_handle_t event = (omp_event_handle_t)(uintptr_t)data;
34    omp_fulfill_event(event);
35  }
```

"Callback-based completion notification using MPI Continuations,"
Joseph Schuchart, Christoph Niethammer, José Gracia, George Bosilca, Parallel Computing, 2021.

"MPI Detach - Asynchronous Local Completion,"
Joachim Protze, Marc-André Hermanns, Ali Demiralp, Matthias S. Müller, Torsten Kuhlen. EuroMPI '20.

CLARIFICATION OF THREAD ORDERING

MPI-3.1 Section 3.5: If a process has a single thread of execution, then any two communications executed by this process are ordered. On the other hand, if the process is multithreaded, then the semantics of thread execution may not define a relative order between two send operations executed by two distinct threads. The operations are logically concurrent, even if one physically precedes the other. In such a case, the two messages sent can be received in any order. Similarly, if two receive operations that are logically concurrent receive two successively sent messages, then the two messages can match the two receives in either order.

Option 1: Operations from different threads are unordered, even if app. enforces ordering

- Pro: Can enable MPI libraries to optimize performance for multithreaded applications
- Con: Hard to get ordering, and MPI doesn't know what the user considers to be a thread
  - E.g., A user-level thread can be migrated to a different shepherd thread. Does MPI see it?

Option 2: MPI must respect an order across threads if the user enforces one

- Pro: Probably what users expect, can relax ordering with info assertions and per-thread comms
- Con: Removes a (questionable) performance optimization opportunity

STREAM TRIGGERED NEIGHBOR EXCHANGE

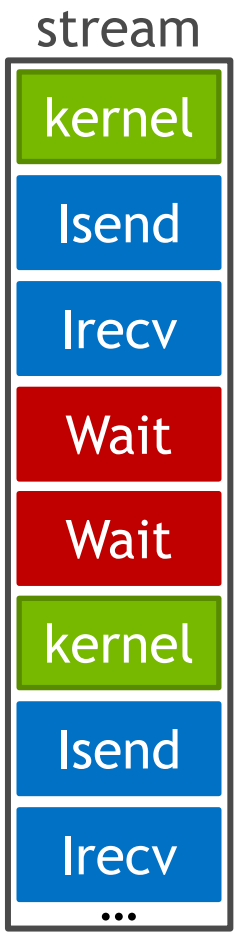
Simple Ring Exchange Using a CUDA Stream

```
MPI_Request send_req, recv_req;
MPI_Status  sstatus,  rstatus;

for (i = 0; i < NITER; i++) {
  if (i > 0) {
    MPI_Wait_enqueue(recv_req, &rstatus, MPI_CUDA_STREAM, stream);
    MPI_Wait_enqueue(send_req, &sstatus, MPI_CUDA_STREAM, stream);
  }

  kernel<<<..., stream>>>(send_buf, recv_buf, …);

  if (i < NITER – 1) {
    MPI_Irecv_enqueue(&recv_buf, …, &recv_req, MPI_CUDA_STREAM, stream);
    MPI_Isend_enqueue(&send_buf, …, &send_req, MPI_CUDA_STREAM, stream);
  }
}
cudaStreamSynchronize(stream);
```



ACCELERATOR BINDINGS FOR MPI PARTITIONED APIS

CUDA and SYCL Language Bindings Under Exploration

```
int MPI_Psend_init(const void *buf, int partitions, MPI_Count count,
                   MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Info info,
                   MPI_Request *request)

int MPI_Precv_init(void *buf, int partitions, MPI_Count count,
                   MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Info info,
                   MPI_Request *request)

int MPI_[start,wait][_all](...)
```

Keep host only

Add device bindings

```
__device__ int MPI_Pready(int partition, MPI_Request request)

__device__ int MPI_Pready_range(int partition_low, int partition_high, MPI_Request request)

__device__ int MPI_Pready_list(int length, const int array_of_partitions[], MPI_Request request)

__device__ int MPI_Parrived(MPI_Request request, int partition, int *flag)
```

KERNEL TRIGGERED COMMUNICATION USAGE

Partitioned Neighbor Exchange

Host Code

```
MPI_Request req[2];
MPI_Psend_init(..., &req[0]);
MPI_Precv_init(..., &req[1]);
while (...) {
  MPI_Startall(2, req);
  MPI_Pbuf_prepare_all(2, req);
  kernel<<<..., s>>>(..., req);
  cudaStreamSynchronize(s);
  MPI_Waitall(2, req);
}
MPI_Request_free(&req[0]);
MPI_Request_free(&req[1]);
```

Device Code

```
__device__
void MPI_Pready(int idx, MPI_Request req);

__global__ kernel(..., MPI_Request *req) {
  int i = my_partition(...);
  // Compute and fill partition i
  // then mark i as ready
  MPI_Pready(i, req[0]);
}
```

KERNEL & STREAM TRIGGERED COMMUNICATION USAGE

Partitioned Neighbor Exchange

Host Code

```
MPI_Request req[2];
MPI_Psend_init(..., &req[0]);
MPI_Precv_init(..., &req[1]);
while (...) {
  MPI_Startall_enqueue(2, req, …);
  MPI_Pbuf_prepare_all_enqueue(2, req, …);
  kernel<<<..., s>>>(..., req);
  cudaStreamSynchronize(s);
  MPI_Waitall_enqueue(2, req, …);
}
MPI_Request_free(&req[0]);
MPI_Request_free(&req[1]);
```

Device Code

```
__device__
void MPI_Pready(int idx, MPI_Request req);

__global__ kernel(..., MPI_Request *req) {
  int i = my_partition(...);
  // Compute and fill partition i
  // then mark i as ready
  MPI_Pready(i, req[0]);
}
```

Moving control ops to stream eliminates stream synchronization overhead

KERNEL & STREAM TRIGGERED COMMUNICATION USAGE

Partitioned Neighbor Exchange

Host Code

```
MPI_Request req[2];
MPI_Psend_init(..., &req[0]);
MPI_Precv_init(..., &req[1]);
while (...) {
  MPI_Startall_enqueue(2, req, …);
  MPI_Pbuf_prepare_all_enqueue(2, req, …);
  kernel<<<..., s>>>(..., req);
  cudaStreamSynchronize(s);
  MPI_Waitall_enqueue(2, req, …);
}
MPI_Request_free(&req[0]);
MPI_Request_free(&req[1]);
```

Device Code

```
__device__
void MPI_Pready(int idx, MPI_Request req);

__global__ kernel(..., MPI_Request *req) {
  int i = my_partition(...);
  // Compute and fill partition i
  // then mark i as ready
  MPI_Pready(i, req[0]);
}
```

Allows the sender to wait until receiver is ready, so Pready becomes RDMA write

Thank you!

Wednesdays 10-11am US Eastern Time

https://github.com/mpiwg-hybrid/hybrid-issues/wiki

MPICH Updates

Yanfei Guo (ANL)

EXASCALE COMPUTING PROJECT

National Nuclear Security Administration

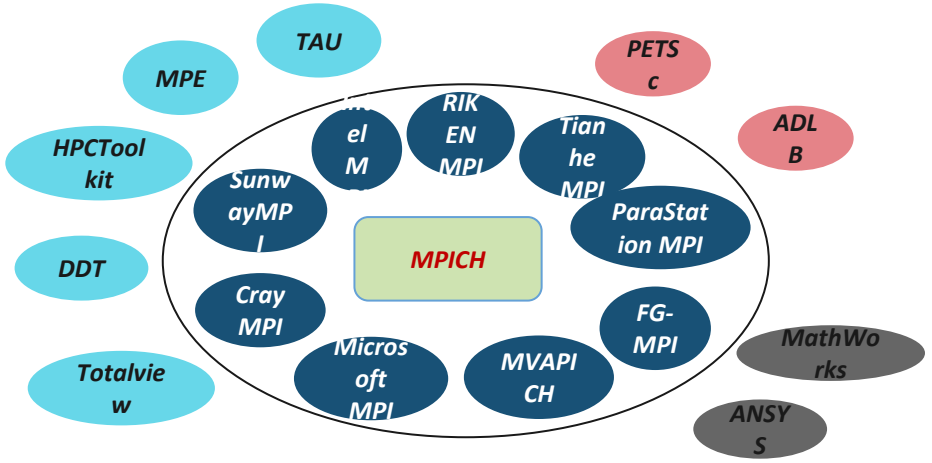
U.S. DEPARTMENT OF ENERGY | Office of Science

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 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

MPICH-4.0 – Full support for MPI-4 standard

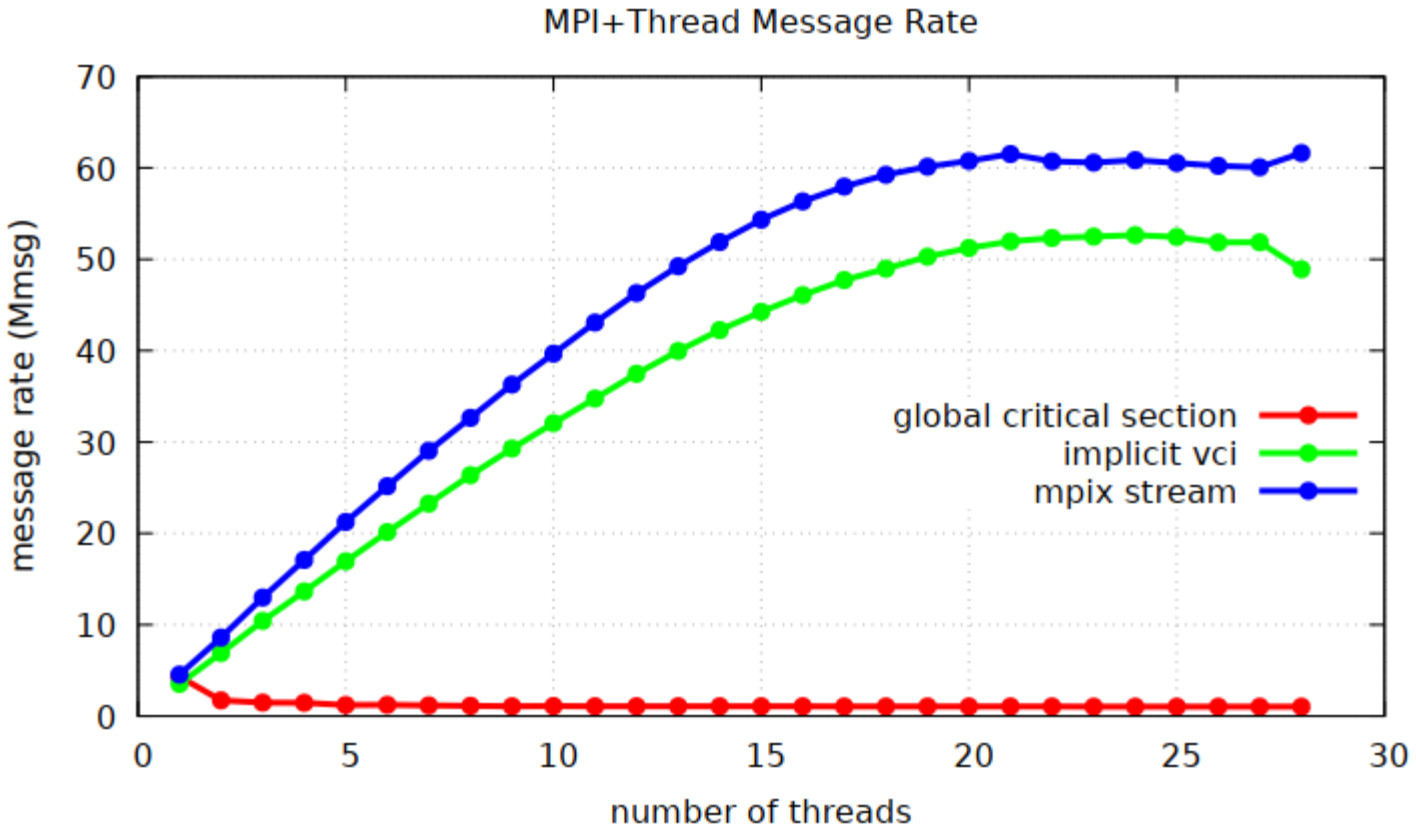
- MPI Forum released MPI 4.0 standard on June 9, 2021

- 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

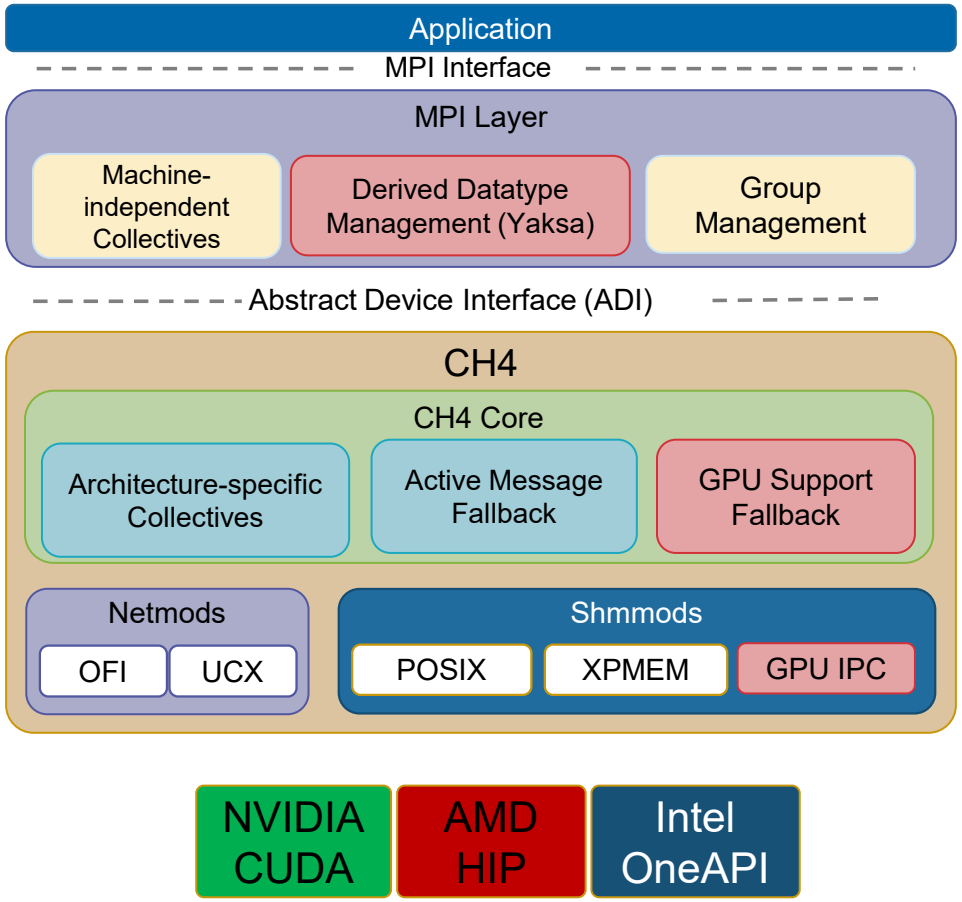
- Previously, dismal performance with MPI_THREAD_MULTIPLE
- Implicit VCI mapping in MPICH-4.0 with potential performance
- Advice to users
  - Use different communicators
  - Same communicator, use different tags and set hints
- Explicit VCI coming in next release



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

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



MPICH 4.1 Release Plan

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

Open MPI Update

Aurelien Bouteiller (UTK)

EXASCALE COMPUTING PROJECT

National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

Recent updates in Open MPI stable (4.x)

- OMPI 4.1.x (4.1.3, March 22) stable has a lot of updates
  - Full support for MPI 3.1
  - New documentation website (readthedoc, manpages) https://docs.open-mpi.org/en/main/in
  - PMIx update with better compatibility with external PMIx (including PMIx 4.x)
  - Bug fixes with large datatypes (and counts)
  - MPI I/O updates (OMPIO and ROMIO 3.4)
  - Lots of bug fixes for the CUDA components
- Performance related updates
  - Added new MPI collectives implementations (HAN, ADAPT) providing support for architecture aware collectives.
  - Many multi-threaded improvements, including UCX support for one- and two- sided communications, WAITALL
  - MPI_Op support for vectorial ISA (x86 with AVX2, AVX512)
  - MPI Software Performance Counters available via MPI_T or PAPI (message rate, instant bandwidth, pending messages, expected and unexpected queues and many more)

Updates in the next Open MPI release (5.0)

- OMPI 5.0.0rc6 (Apr 2022)
  - Builds as a monolithic library with plugins directly linked into the library (single .so file), a more file system friendly approach at scale
  - Removed the C++ bindings
  - Removed support for MPIR, now tools support via PMIx tools interface
  - Revamped and refreshed documentation on readthedocs.io
  - PRRTE, upgraded to PMIx 4.2 (many fixes to the mpiexec option parsing)
- New features:
  - MPI 4.x readiness (persistent collectives, partitioned communications (#8641), sessions, and more)
  - User Level Fault Mitigation (ULFM) support
  - FP16 support (MPIX extension)
  - Extended MPI_Op support for vectorial ISA (x86 but also ARM64)

Updates in the next Open MPI release (5.0)

- Performance improvements
  - Architecture-aware collective (HAN) will become the default instead of tuned
  - Additional multi-threaded improvements for one and two sided communications
  - Atomic operations (ARM64, PPC, C11)
  - MPI non-contiguous datatypes optimizations
  - CUDA+OFI support
  - ABI compatible with 4.1.x
  - Ready for anything from HPC to cloud (containers, AWS, …)

Welcome and Introduction

- Goal: Provide an opportunity for ECP participants to discuss the MPI standard and key implementations

- Outline
  - Major MPI 4 features
    - Sessions, Persistent Collectives, Partitioned Communications – Matthew Dosanjh (SNL)
    - MPI_T, Hardware Topologies, Error Management – Ken Raffenetti (ANL)
  - MPI Forum next steps – Howard Pritchard (LANL)
  - MPI Hybrid & Accelerator Working Group Update– Jim Dinan (NVIDIA)
  - Recent highlights in MPICH – Yanfei Guo (ANL)
  - Recent highlights in Open MPI – Aurelien Bouteiller (UTK)
  - **General Q&A and discussion** – David Bernholdt (ORNL), moderator