The Message Passing Interface: Version 4.0 and Beyond

Martin Schulz, Technische Universität München
Chair of the MPI Forum

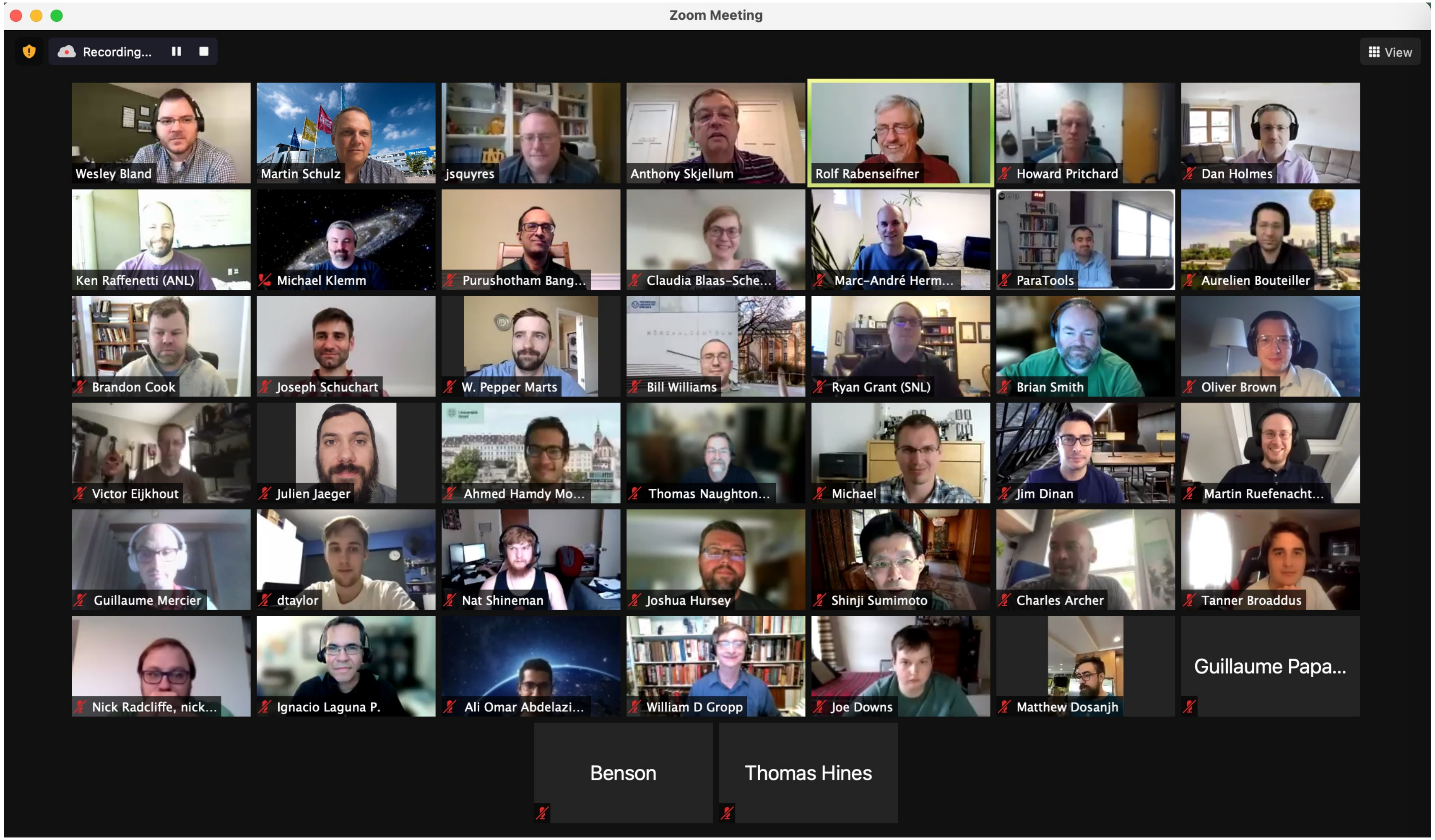
Panelists:
- Ignacio Laguna, LLNL
- Wesley Bland, Intel
- Howard Pritchard, LANL
- Jim Dinan, NVIDIA
- Ryan Grant, Queen's University, Canada
- Anthony Skjellum,
  University of Tennessee at Chattanooga

+ the entire MPI Forum

SC 2021 BoF, November 2021

MPI 4.0 got Ratified on June 9th 2021

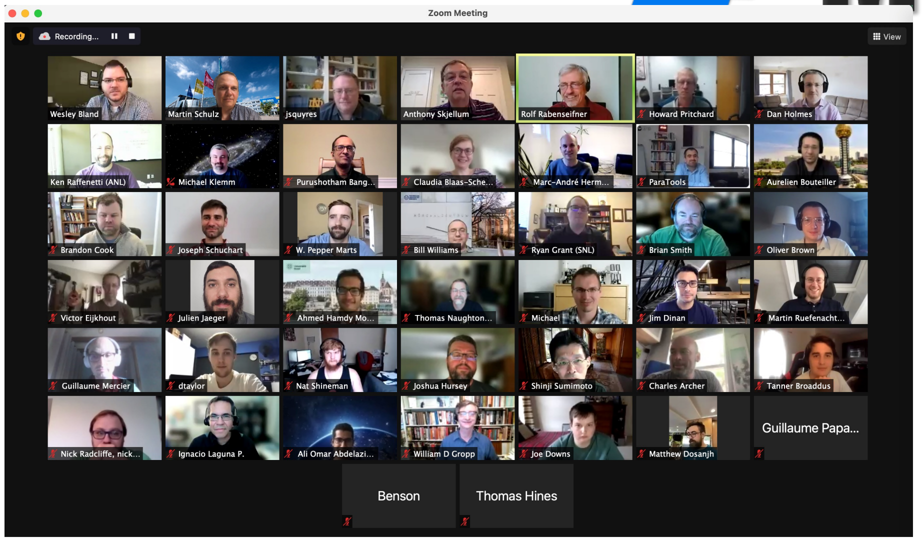
Available at http://www.mpi-forum.org/



MPI 4.0 (and what's Next)

Major additions for MPI 4.0

- Partitioned Communication
- New tool interface for events
- Solution for "Big Count" operations
- Persistent Collectives
- New init options via MPI Sessions
- Topology Solutions
- And much more …



MPI 4.0 Implementations in the Works

- The major implementations are already working towards MPI 4.0
- Several features already supported
- Full support across most implementations soon

The work of the MPI Forum Continues

- Next step: MPI 4.1 – minor changes/clarifications and cleanup/reorg
- Work on MPI 5.0 has begun as well
- http://www.mpi-forum.org/

Good Time to Join the MPI-Forum
The MPI-Forum is open to all interested in MPI.

Coarse-Grained Fault Recovery

Ignacio Laguna, Giorgis Georgakoudis

LLNL

Reinitialize MPI

Formal text has been drafted and is getting close to a plenary: https://reinit.github.io/reinit/

- Cleans up MPI state and jumps to a specified point in the code
- This constructs a global roll-back error recovery

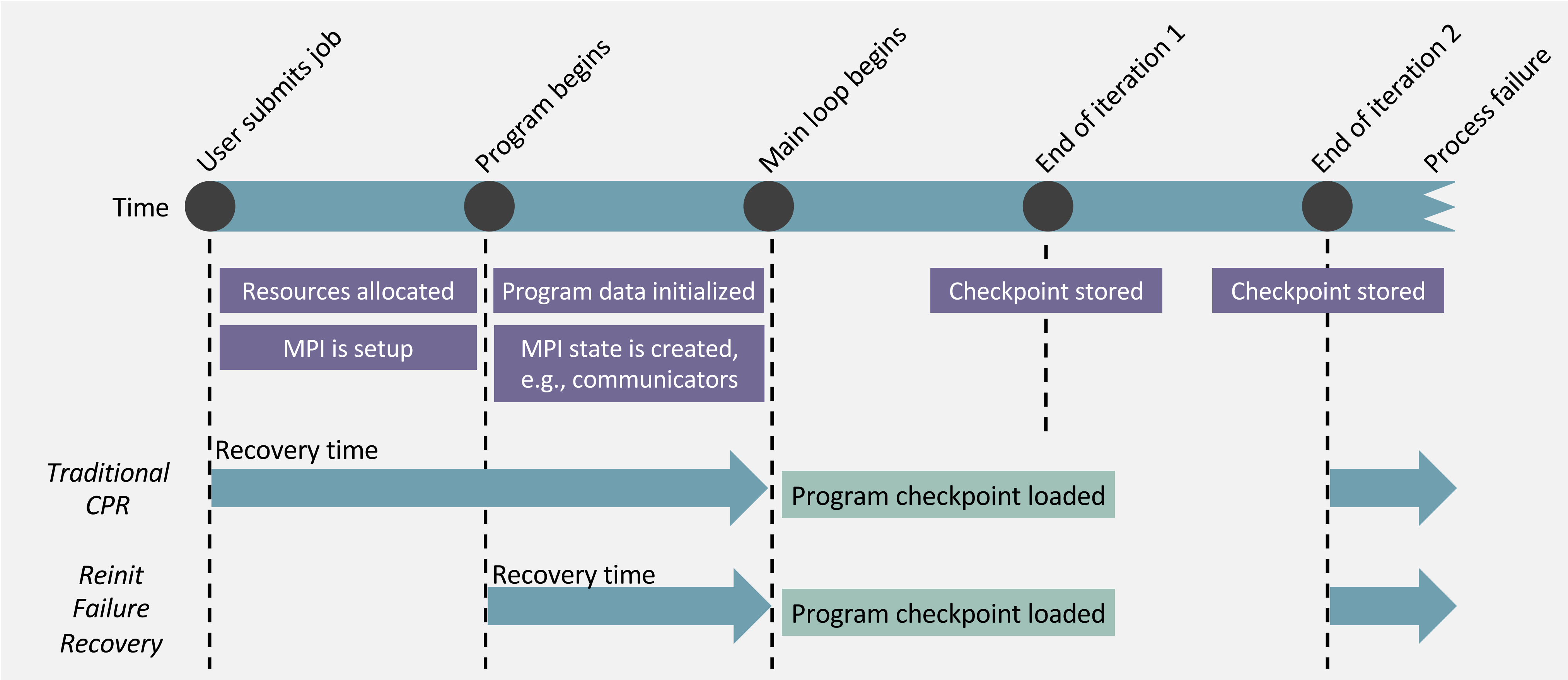
```
MPI_Init()
…initialize…
MPI_Reinit()

…do things…

MPI_Allreduce()
/* ERROR */
```

Jump and
“Clean up” MPI

Coarse-grained Recovery (Reinit)



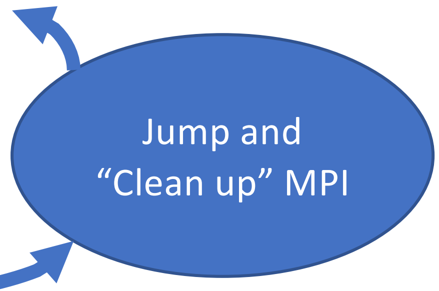
Checkpoint MPI State & Return to Previous State X

- More generic case of reinitializing MPI by allowing multiple reinitialization points
- Still in early discussions
- Likely to not be its own model, but will be a "building block" that can be used independently

```
MPI_Init()
…initialize…

for () {
  MPI_Save_state()
  …do things…

  MPI_Allreduce()
  /* ERROR */
}
```



Reinit Function

Specifies a
Rollback Location

```
int MPI_Reinit(resilient_fn, void *data)
```

IN	resilient_fn	user defined procedure (function)
IN	data	pointer to user defined data

The user-defined procedure should be in C, a function of type MPI_Reinit_function which is defined as:

```
typedef MPI_Reinit_fn void (*)(void *data));
```

The first argument is a user defined procedure, `resilient_fn`, which is called by the MPI_Reinit procedure. The second argument is a pointer to user defined data. This pointer is passed as an argument to the user defined procedure, `resilient_fn`, when the procedure is called. A valid MPI program must contain at most one call to the MPI_Reinit procedure. Calling MPI_Reinit more than one time results in undefined behavior.

More at https://reinit.github.io/reinit/

Error Handling

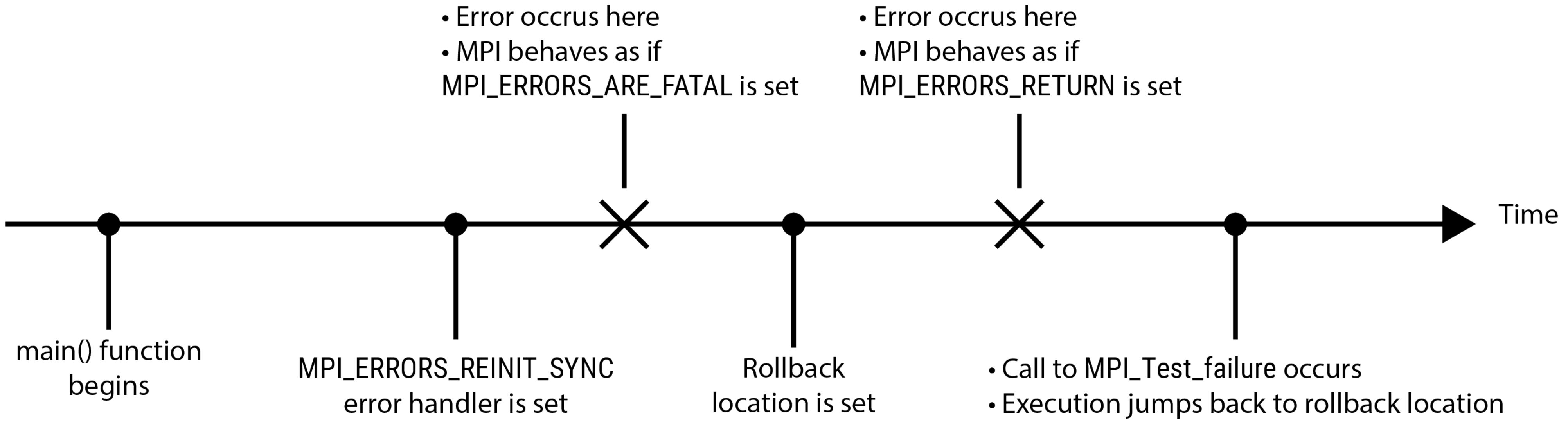
1

- **MPI_ERRORS_REINIT_ASYNC**
  a) The handler is called immediately after a process failure is detected
  b) Causes the execution of the program to resume at (or jump back to) the **active** rollback location

2

- **MPI_ERRORS_REINIT_SYNC**
  a) The handler has two effects.
  b) It enables the **MPI Test failure** function to cause the execution of the program to resume at (or jump back to) the active rollback location
  c) It returns the error code to the user.

Different Scenarios for SYNC Error Handling



Reinit Specification Document

More at https://reinit.github.io/reinit/

Fault Tolerance WG Mission Statement

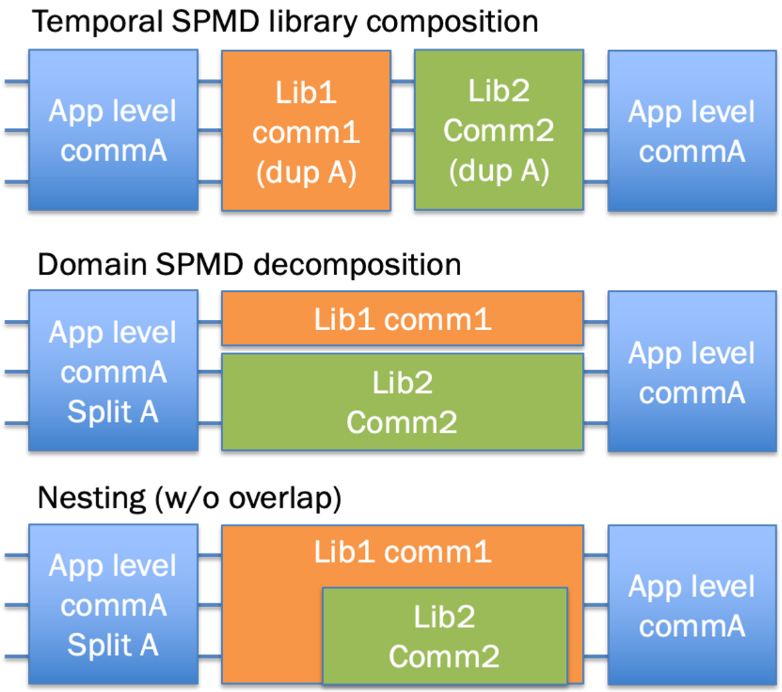
- Commissioned to work on fault tolerance.
- Work has expanded to include all error handling.
- The focus includes more than just the well-known ULFM proposal:
  - Finer control on what gets aborted after an error
  - Let programs fallback to TCP/other if MPI has an error; **increase the appeal to non-HPC folks**
  - Clarification of what the state of the MPI library should be after an error (i.e., **POSIX-like error handling**)
  - Consult on error management in new additions (MPI Sessions, MPI_INFO before MPI_INIT, etc.)

New Error Handling Features is MPI 4.0

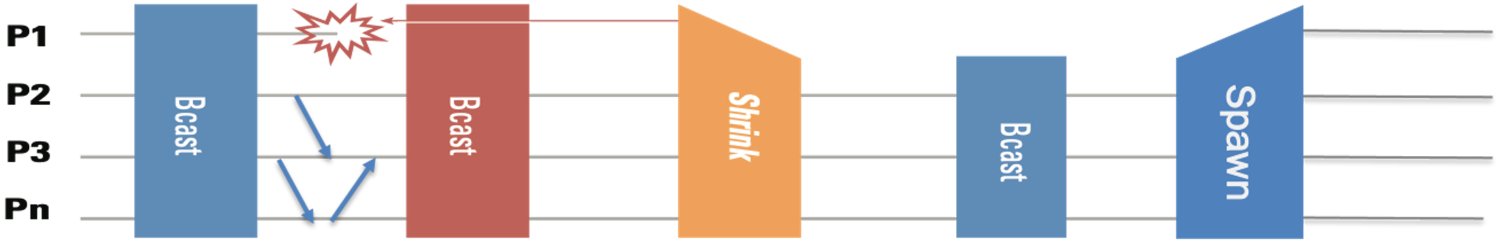
- New MPI Error Handler - **MPI_ERRORS_ABORT**
- Add **MPI_ERR_PROC_ABORTED** error code.
- Localize error impact of some MPI operations, raise an **error on MPI_COMM_SELF**, not MPI_COMM_WORLD
- **Errors do not ”break MPI”** but indicate the operation didn’t work. Other operations may still succeed.
- Specify that MPI_SUCCESS indicates only the result(s) of the operation, not the state of the MPI library.
- Allow the user to specify the **default error handler at `mpiexec`** time.

Levels of Composability

- Level 0 – Models coexist but do not interoperate
- Level 1 – Models can be used in the same application, but not at the same time.
  - E.g., Use fine-grained recovery, then coarse-grained, then fine-grained again
- Level 2 – Models used in the same application, but not all processes are using the same models
  - E.g., One communicator uses coarse-grained recovery, another uses fine-grained
- Level 3 – Models are fully integrated and can be used interchangeably.



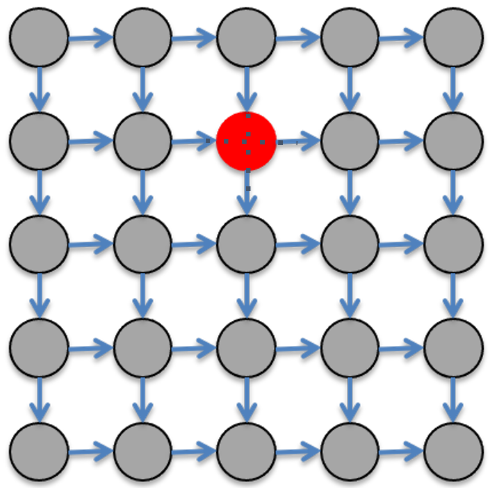
ULFM MPI Crash Recovery (Background)



- Some applications can **continue** w/o recovery
- Some applications are **malleable**
  - Shrink creates a new, smaller communicator on which collectives work
- Some applications are ***not* malleable**
  - Spawn can recreate a "same size" communicator
  - It is easy to reorder the ranks according to the original ordering
  - Pre-made code snippets available

- Failure Notification
- Error Propagation
- Error Recovery
- Respawn of nodes
- Dataset restoration

Not all recovery strategies require all of these features, that's why the interface should split notification, propagation and recovery.



Who should be notified of a failure?
What is the scope of a failure?
What actions should be taken?

- Adds **3 error codes** and **5 functions to manage process crash**
  - **Error codes:** interrupt operations that may block due to process crash
  - MPI_COMM_FAILURE_ACK / GET_ACKED: continued operation with ANY-SOURCE RECV and observation known failures
  - MPI_COMM_REVOKE lets applications interrupt operations on a communicator
  - MPI_COMM_AGREE: synchronize failure knowledge in the application
  - MPI_COMM_SHRINK: create a communicator excluding failed processes
  - More info on the MPI Forum ticket #20: https://github.com/mpi-forum/mpi-issues/issues/20

Tools - Function Interception
Current State of the Art

- Name-shifted interface (PMPI)
  - Relatively simple to use
  - Supports a single tool at a time without resorting to non-standard workarounds (P^nMPI)

- Tools implement their own versions of functions and intercept MPI calls with tricks like weak symbols
  - Calls PMPI_<foo> when done

Tools - Function Interception
Desired Features

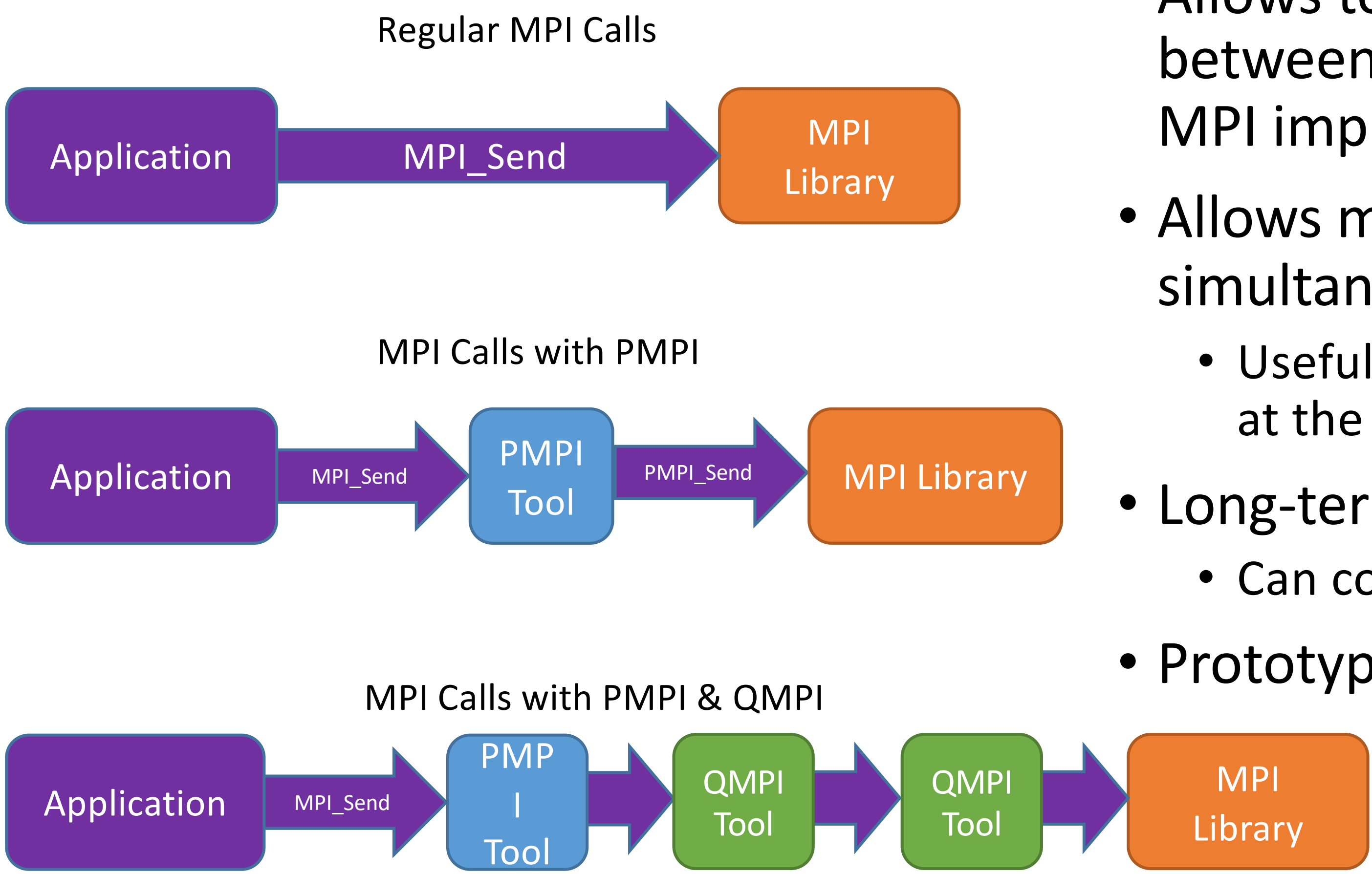
- Support for multiple, simultaneous tools
- Support for multiple copies of the same tool (e.g., one instance for rows and another for columns)
- A way for users to specify the set and order of tools
- Intercept all functions provided by an MPI library (including non-standard functions
- Interoperability with existing “PMPI” tools

Desired Features

- ✓Support for multiple, simultaneous tools
- ✓Support for multiple copies of the same tool (e.g., one instance for rows and another for columns)
- ✓A way for users to specify the set and order of tools
- ✓Intercept all functions provided by an MPI library (including non-standard functions
- ✓Interoperability with existing "PMPI" tools

QMPI!

Function Pointer Interception (QMPI)



- Allows tools to insert themselves between the application and the MPI implementation
- Allows multiple tools to be used simultaneously
  - Useful to layer complementary tools at the same time.
- Long-term Replacement for PMPI
  - Can co-exist with PMPI short-term
- Prototype available in MPICH

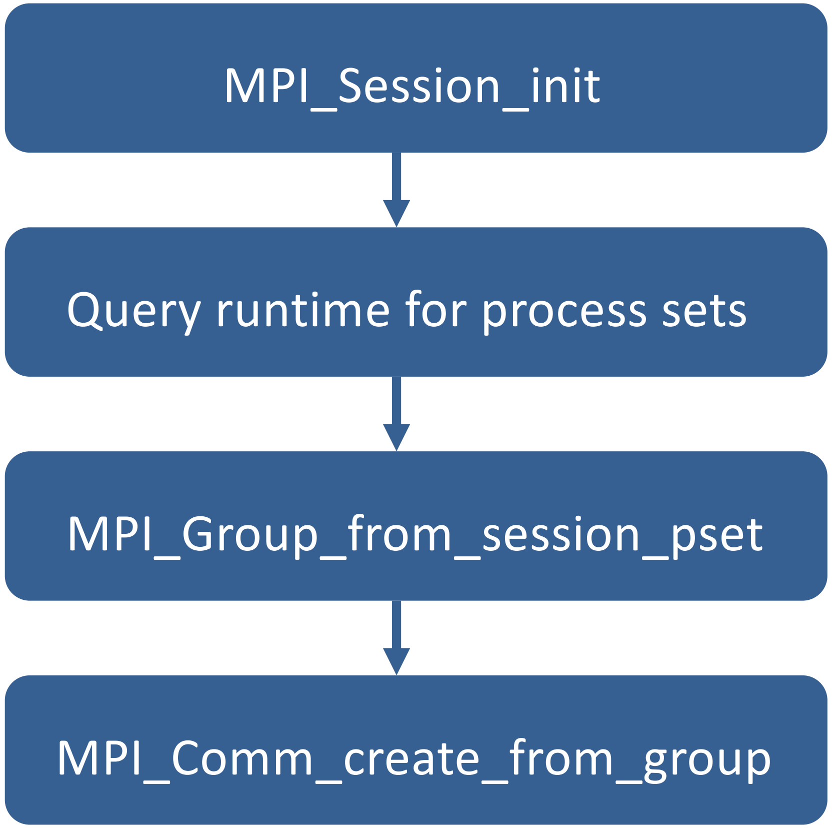
MPI Sessions WG Update

Howard Pritchard
Los Alamos National Laboratory

LA-UR-21-31331

MPI Sessions – current state

- In the MPI 4.0 standard
- Consider this as first step for Sessions



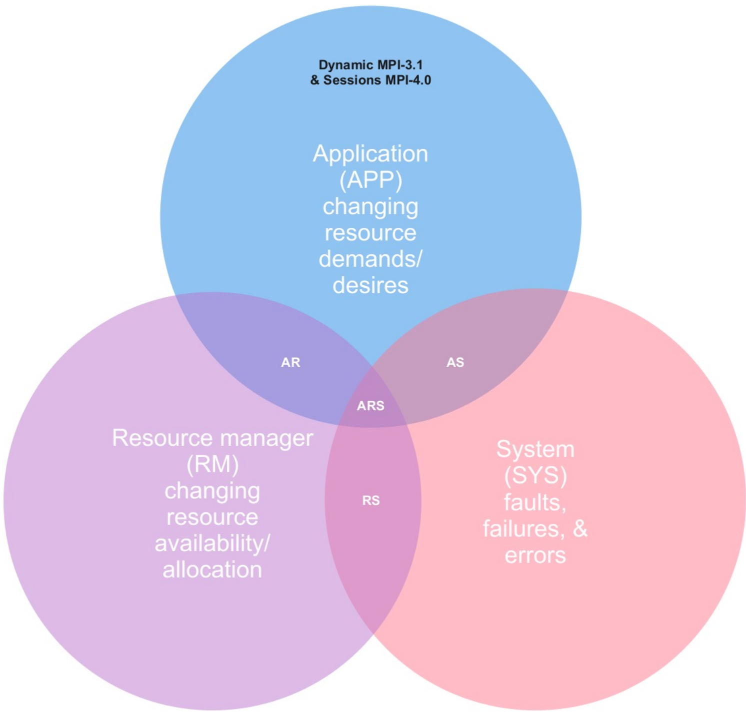
MPI Sessions – current state in MPI implementations

- Available in MPICH 4.0 release stream
- Prototype based on Open MPI is available at https://github.com/hpc/ompi/tree/sessions_pr (this branch is subject to rebasing!)
- Set of simple tests are available at https://github.com/open-mpi/ompi-tests-public

MPI WG Sessions – current activities

- For MPI 4.1 standard - clarifications of Sessions related text

- For MPI 5.0 - investigating requirements for more dynamic use of Sessions:
  - Presentation of available process sets in a manner more suitable for dynamic environments
  - Mechanisms for runtime notifying application of resource changes
  - Mechanisms for application to notify runtime about changing resource requirements
  - Adding/removing MPI processes (beyond MPI_Comm_spawn)
  - Working with FT WG to develop FT approaches that leverage Sessions functionality



MPI HYBRID & ACCELERATOR WORKING GROUP UPDATE

James Dinan, NVIDIA
HACC WG Chair
SC '21 MPI Forum BoF

HYBRID & ACCELERATOR WORKING GROUP

Mission: Improve interoperability of MPI with other programming models

Active topics:

1. Continuations proposal #6
2. Clarification of thread ordering rules #117
3. Integration with accelerator programming models:
    1. Accelerator info keys #3
    2. Stream/Graph Based MPI Operations #5
    3. Accelerator bindings for partitioned communication #4
    4. Partitioned communication buffer preparation (shared with Persistence WG) #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

```
MPI_Request cont_req;
MPIX_Continue_init(&cont_req);

omp_event_handle_t event;
int value;
#pragma omp task depend(out:value) detach(event)
{
  MPI_Request req;
  MPI_Irecv(&value, ..., &req);
  MPIX_Continue(&req, &release_event, event, MPI_STATUS_NULL, cont_req);
}

#pragma omp task depend(in: value)
{
  // process value
}
```

```
void release_event(MPI_Status status, void *data)
{
  omp_event_handle_t event = (omp_event_handle_t)(uintptr_t)data;
  omp_fulfill_event(event);
}
```

"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.

STREAM TRIGGERED NEIGHBOR EXCHANGE

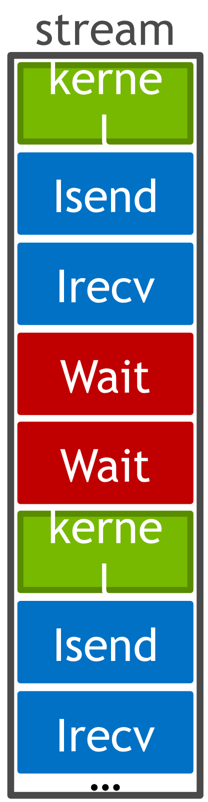
Simple Ring Exchange Using a CUDA Stream

```
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);
```



CUDA BINDINGS FOR MPI PARTITIONED APIS

```
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 to stream eliminates overhead from stream synchronization

NVIDIA

Thank you!

Wednesdays 10-11am US Eastern Time

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

Partitioned Communication

SC21 MPI BoF

Ryan Grant, Queen’s University, Canada

New in MPI 4.0

MPI Partitioned Communication Concepts

- Many actors (threads) contributing to a larger operation in MPI
  - Same number of messages as today!
  - No new ranks – no need to understand target thread
- Many threads work together to assemble a message
  - MPI only has to manage knowing when completion happens
  - These are actor/action counts, not thread level collectives
- Persistent-style communication
  - Init…(Start…test/wait)…free
- No heavy MPI thread concurrency handling required
  - Leave the placement/management of the data to the user
- No more complicated packing of data, send structures when they become available

Partitioned Communication & GPUs

Partitioned Communication aimed at multi-threaded multi-core devices – improve for GPUs

Host (CPU) side

```
MPI_Psend_init(..., &request);
for (...) {
  MPI_Start(&request);
  kernel<<<...>>>(..., request);
  MPI_Wait(&request);
}
MPI_Request_free(&request);
```

Kernel:

```
__device__ kernel(..., MPI_Request request)
{
  int i = my_partition[my_id];
  /* Compute and fill partition i then mark
ready: */
  MPI_Pready(i, request);
}
```

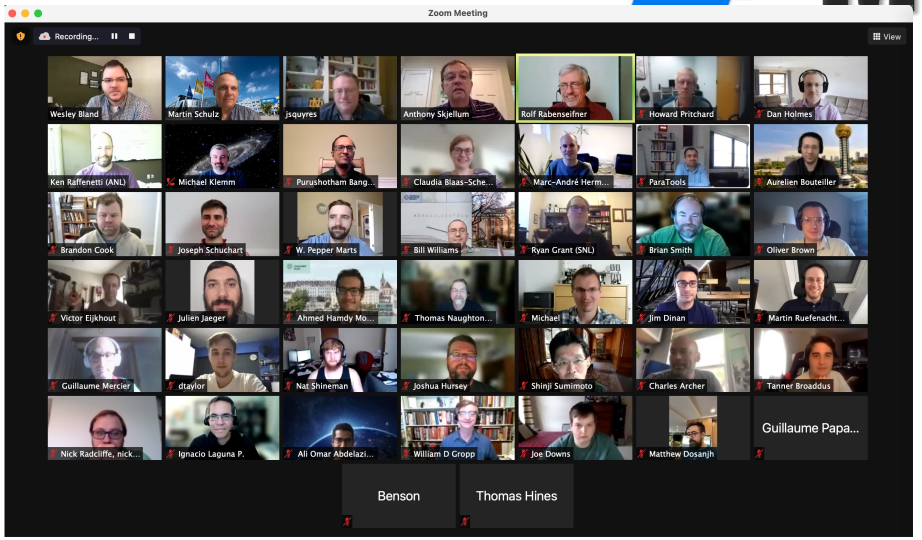
Note: CPU does communication setup and completion steps for MPI. Setup commands on NIC and poll for completion of entire operation. Kernel just indicates when NIC/MPI can send data. Ideally want to trigger communication from GPU to fire off when data is ready without communication setup/completion in kernel

Proposed for MPI 4.1

Pbuf_prepare Example

Send-side	receive-side
MPI_PSEND_INIT	MPI_PRECV_INIT
MPI_START	MPI_START
MPI_PBUF_PREPARE (blocking/non-local)	MPI_PBUF_PREPARE (blocking/non-local)
MPI_PREADY…(nonblocking)	Optional - parrived (nonblocking)
MPI_WAIT (completing)	MPI_WAIT (completing)
MPI_START, MPI_PSYNC	MPI_START, MPI_PSYNC
MPI_PREADY...MPI_PREADY	MPI_PARRIVED...MPI_PARRIVED
MPI_WAIT	MPI_WAIT

MPI: Version 4.0 and Beyond – Q&A



Major additions for MPI 4.0

- Partitioned Communication
- New tool interface for events
- Solution for “Big Count” operations
- Persistent Collectives
- New init options via MPI Sessions
- Topology Solutions
- And much more …

MPI 4.0 Implementations in the Works

- The major implementations are already working towards MPI 4.0
- Several features already supported
- Full support across most implementations soon

The work of the MPI Forum Continues

- Next step: MPI 4.1 – minor changes/clarifications and cleanup/reorg
- Work on MPI 5.0 has begun as well
- http://www.mpi-forum.org/

Good Time to Join the MPI-Forum
The MPI-Forum is open to all interested in MPI.

The MPI Forum Drives MPI

Standardization body for MPI

- Discusses additions and new directions
- Oversees the correctness and quality of the standard
- Represents MPI to the community

Organization consists of:

- Chair (Martin Schulz, TUM/LRZ)
- Secretary (Wesley Bland, Intel)
- Treasurer (Brian Smith, ORNL)
- Editor (Bill Gropp, UIUC/NCSA)

Open membership

- Any organization is welcome to participate
- Consists of working groups and the actual MPI forum (plenary)
- Voting (plenary) meetings 4 times each year (3 in the US, one with EuroMPI/Asia/USA)
- Voting rights depend on attendance

The Bulk of Work is in the Working Groups

Collective Communication, Topology, Communicators, Groups
- Torsten Hoefler, Andrew Lumsdaine and Anthony Skjellum

Fault Tolerance
- Wesley Bland, Aurélien Bouteiller

HW Topologies
- Guillaume Mercier

Hybrid and Accelerator Programming
- Jim Dinan

Language Bindungs
- Martin Ruefenacht

Persistence
- Anthony Skjellum

Point to Point Communication
- Rich Graham and Dan Holmes

Remote Memory Access
- Bill Gropp and Rajeev Thakur

Semantic Terms
- Rolf Rabenseifner and Purushotham Bangalore

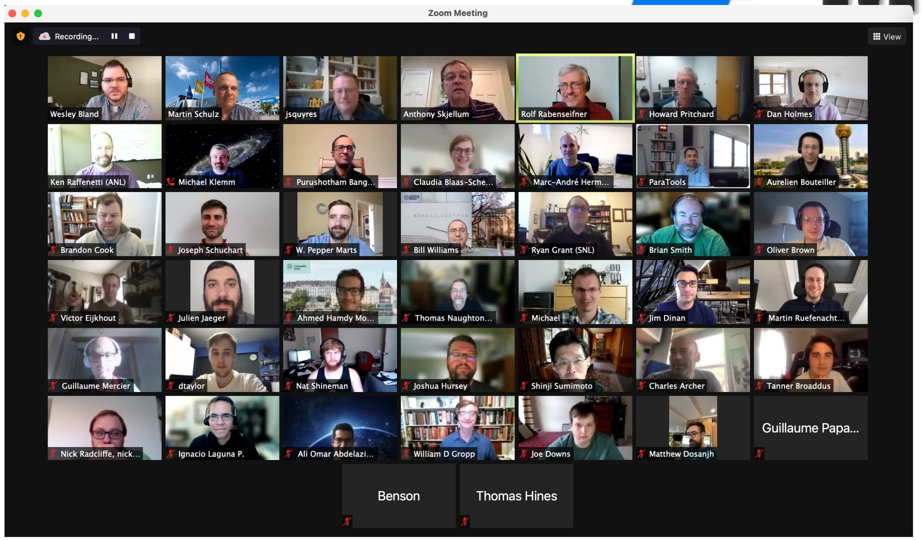
Sessions
- Dan Holmes, Howard Pritchard

Tools
- Marc-Andre Hermanns

MPI: Version 4.0 and Beyond – Q&A

Major additions for MPI 4.0

- Partitioned Communication
- New tool interface for events
- Solution for “Big Count” operations
- Persistent Collectives
- New init options via MPI Sessions
- Topology Solutions
- And much more …



MPI 4.0 Implementations in the Works

- The major implementations are already working towards MPI 4.0
- Several features already supported
- Full support across most implementations soon

The work of the MPI Forum Continues

- Next step: MPI 4.1 – minor changes/clarifications and cleanup/reorg
- Work on MPI 5.0 has begun as well
- http://www.mpi-forum.org/

Good Time to Join the MPI-Forum
The MPI-Forum is open to all interested in MPI.