Recent Efforts of the MPI Forum for MPI-4 and Future MPI Standards

Pavan Balaji
Argonne National Laboratory

Howard Pritchard
Los Alamos National Laboratory

LA-UR-19-20284

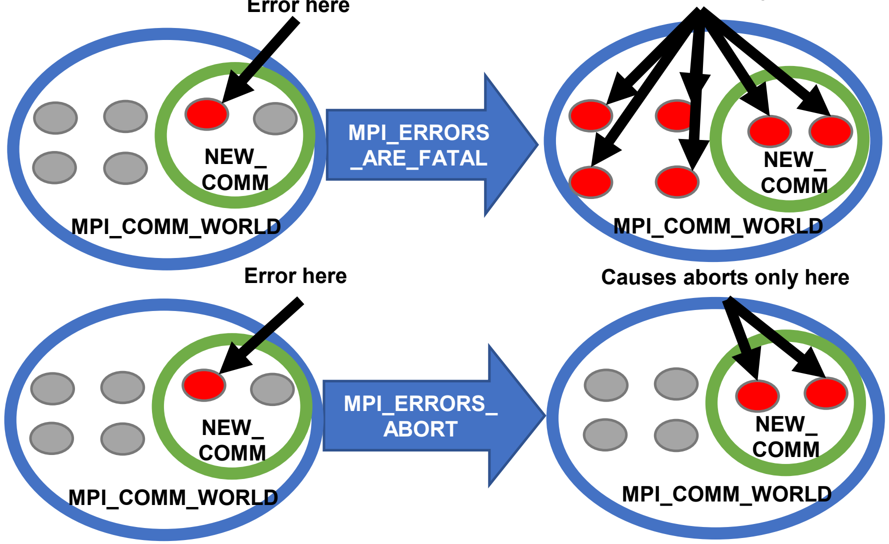
MPI-4 Standardization Process

- Multiple working groups, each working on individual topics
  - Fault Tolerance, Persistence, One-sided Communication, Sessions, …
- Beginning in 2018, started holding monthly *virtual* forums to speed up standardization process
- MPI-4 2018 draft standard released in Nov. 2018
  - Some features have already been voted in
  - The standard as a whole has not been voted in yet (it’s not yet “MPI-4”)
- Final MPI-4 is expected to be out in around 2 years

Fault Tolerance

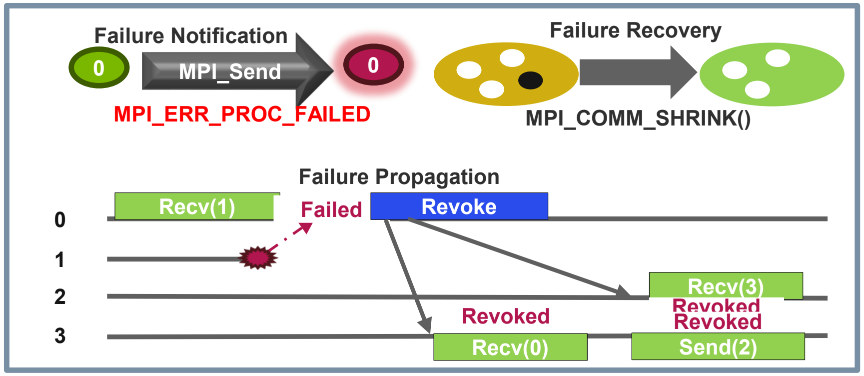
Errors_Abort

- MPI_ERRORS_ABORT: Added new predefined error handler which only causes processes in the affected communicator to abort



User Level Failure Mitigation

- Enable application-level recovery by providing minimal FT API to prevent deadlock and enable recovery
- Don't do recovery for the application, but let the application (or a library) do what is best
- Currently focused on process failure (not data errors or protection)



Non-Catastrophic Errors

- ~~After an error is detected, the state of MPI is undefined.~~
- MPI should return as much information as possible about errors.
  - Gives users more control over how to handle errors.
  - If you *really* want to, you *could* construct a resilient point-to-point-only application on top of this change.
- This small change (along with the previous one), should actually provide enough error handling improvements to avoid application aborts during simple errors like resource exhaustion.

Persistence Collective Operations

- Mirror regular nonblocking collective operations
- For each nonblocking MPI collective, add a persistent variant

- For every MPI_I<coll>, add MPI_<coll>_init
- Parameters are identical to the corresponding nonblocking variant – plus additional MPI_INFO parameter

- All arguments “fixed” for subsequent uses
- Persistent collective operations cannot be matched with blocking or nonblocking collective calls
- Has been voted in to Standard by the Forum

Nonblocking collectives API

```
for (i = 0; i < MAXITER; i++) {
    compute(bufA);
    MPI_Ibcast(bufA, …, rowcomm, &req[0]);
    compute(bufB);
    MPI_Ireduce(bufB, …, colcomm, &req[1]);
    MPI_Waitall(2, req, …);
}
```

Persistent collectives API

```
MPI_Bcast_init(bufA, …, rowcomm, &req[0]);
MPI_Reduce_init(bufB, …, colcomm, &req[1]);
for (i = 0; i < MAXITER; i++) {
    compute(bufA);
    MPI_Start(req[0]);
    compute(bufB);
    MPI_Start(req[1]);
    MPI_Waitall(2, req, …);
}
```

Remote Memory Access (RMA)

MPI Generalized Atomics

- MPI-3 atomic operations are, in some cases, restrictive and are not precisely defined
- **Two proposals**:
  - Clarify what operations are atomic and what are not (minor change)
  - Allow for generality of atomic operations with room for performance optimization
- **Generality**: Ability for different atomic operations to be issued on the same target location
- **Performance**: Additional info hints to restrict what the user will use (e.g., only CAS, only FOP, only basic datatypes)

Neighborhood Communication in MPI RMA

- MPI-3 defined **neighborhood collectives** as a process only communicates with its neighbors
- Neighborhood RMA is a generalization of that concept to allow RMA to neighboring processes
  - Allows MPI implementations to optimize state that is internally managed
  - Primarily an optimization for memory usage (e.g., MPI does not need to store information about non-neighbor processes)
    - Can also improve performance in some rare cases

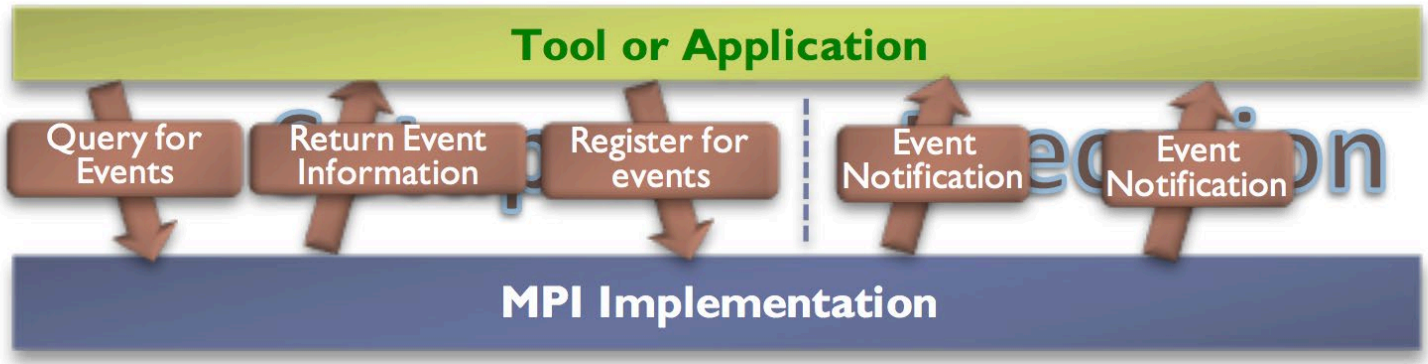
RMA Notifications

- In passive target mode, notifying the target that data has been transmitted is currently inefficient
- Two proposals for target notification: **1) Notification on PUT/GET 2) Notification on Flush**
- Idea is to notify the target when the data has been deposited into the target public memory

MPI Tools - Events Interface

MPI_T Events

- Interface for tools to get notification of events occurring within an MPI implementation
- No events defined explicitly in the standard
- MPI implementation decides which events exposed through to MPI_T events interface and when
- Callback interface used to notifies tools of an event occurrence

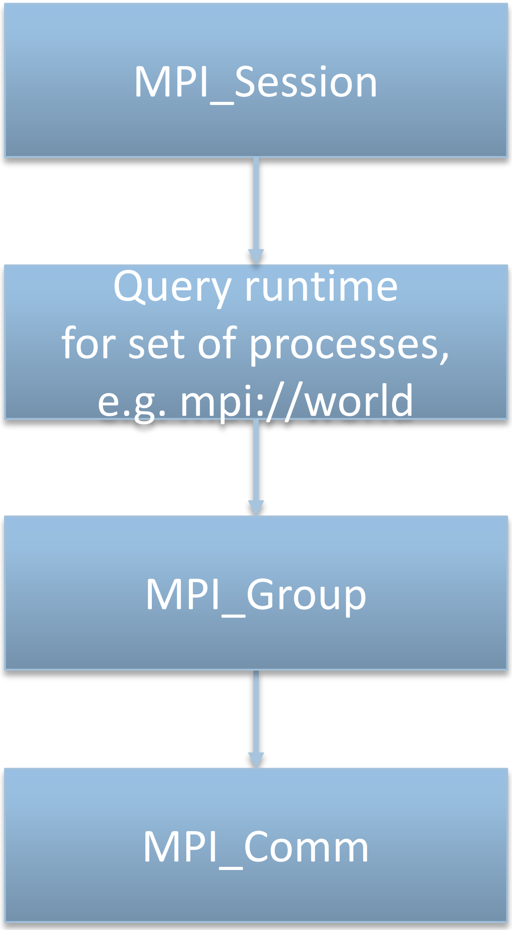


- Proto-type implemented in Open MPI
- EuroMPI 2018 paper: Hermanns et al., Enabling callback-driven runtime introspection via MPI_T

MPI Sessions -

MPI without the Init

- Different components of an application allocate MPI resources independently
- MPI resources can be allocated/deallocated multiple times within a process
- Introduces the notion of isolation of MPI resources allocated by different components of an application
- Can be used with MPI applications that currently use MPI_Init/MPI_Finalize



Proto-type Status

- Proto-type implemented in Open MPI
- Makes use of PMIx Groups API
- https://github.com/hppritcha/mpi_sessions_tests

Point-to-Point Communication

Communication Relaxation Hints

- mpi_assert_no_any_tag
  - The process will not use MPI_ANY_TAG
- mpi_assert_no_any_source
  - The process will not use MPI_ANY_SOURCE
- mpi_assert_exact_length
  - Receive buffers must be correct size for messages
- mpi_assert_overtaking_allowed
  - All messages are logically concurrent

Hybrid MPI Programming

MPI endpoints

- Idea is to have multiple addressable communication entities within a single MPI process
  - Instantiated in the form of multiple ranks per MPI process
- Each rank can be associated with one or more threads
- Reduced contention for communication on each “rank”
- In the extreme case, we could have one rank per thread (or some ranks might be used by a single thread)

Implementation phases/options

- Most common current approach
  - Single endpoint per MPI process
  - Worst case contention
- Possible optimization in MPI-3.1: multiple invisible endpoints
  - Multiple internal endpoints (BG/Q style)
  - Transparent to the user
  - E.g. one endpoint per comm, per neighbor process (regular apps)
- Endpoints proposal for MPI-4: multiple user-visible endpoints
  - Multiple endpoints managed by the user