MPI 3.0 And Beyond

Richard Graham

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™

Contributing Organizations

- Argonne National Laboratory
- Bull
- Cisco Systems, Inc
- Cray Inc.
- CSCS
- ETH Zurich
- Fujitsu Ltd.
- German Research School for Simulation Sciences
- The HDF Group
- Hewlett-Packard
- International Business Machines
- IBM India Private Ltd
- Indiana University
- Institut National de Recherche en Informatique et Automatique (INRIA)
- Institute for Advanced Science & Engineering Corporation
- Intel Corporation
- Lawrence Berkeley National Laboratory
- Lawrence Livermore National Laboratory
- Los Alamos National Laboratory
- Mellanox Technologies, Inc.
- Microsoft Corporation
- NEC Corporation
- National Oceanic and Atmospheric Administration, Global Systems Division
- NVIDIA Corporation
- Oak Ridge National Laboratory
- The Ohio State University

Contributing Organizations

- Argonne National Laboratory
- Bull
- Cisco Systems, Inc
- Cray Inc.
- CSCS
- ETH Zurich
- Fujitsu Ltd.
- German Research School for Simulation Sciences
- The HDF Group
- Hewlett-Packard
- International Business Machines
- IBM India Private Ltd
- Indiana University
- Institut National de Recherche en Informatique et Automatique (INRIA)
- Institute for Advanced Science & Engineering Corporation
- Intel Corporation
- Lawrence Berkeley National Laboratory
- Lawrence Livermore National Laboratory
- Los Alamos National Laboratory
- Mellanox Technologies, Inc.
- Microsoft Corporation
- NEC Corporation
- National Oceanic and Atmospheric Administration, Global Systems Division
- NVIDIA Corporation
- Oak Ridge National Laboratory
- The Ohio State University

Contributing Organizations – Cont’d

- Oracle America
- Platf
- RIKEN AICS
- RunTime Computing Solutions, LLC
- Sandia National Laboratories
- Technical University of Chemnitz
- Tokyo Institute of Technology
- University of Alabama at Birmingham
- University of Chicago
- University of Houston
- University of Illinois at Urbana-Champaign
- University of Stuttgart, High Performance Computing Center Stuttgart (HLRS)
- University of Tennessee, Knoxville
- University of Tokyoorm Computing

Outline

- MPI 3.0 Goals
- MPI 3.0 major additions
  - Nonblocking collectives
  - MPI Tool Interface
  - Noncollective communicator creation
  - RMA enhancements
  - New Fortran bindings
  - Neigborhood collectives
  - Enhanced Datatype support
  - Large data counts
  - Matched probe
  - Topology Aware Communicator Creation
- What did not make it into MPI 3.0
- What was removed from MPI
- What was deprecated from MPI
- Expected Implementation Timelines
- What next ?

MPI 3.0 - Scope

Additions to the standard that are needed for better platform and application support. These are to be consistent with MPI being a library providing process group management and data exchange. This includes, but is not limited to, issues associated with scalability (performance and robustness), multi-core support, cluster support, and application support.

Backwards compatibility may be maintained - Routines may be deprecated or deleted

Nonblocking Collectives

Nonblocking Collective Operations

- Idea
  - Collective communication initiation and completion separated
  - Offers opportunity to overlap computation and communication
  - Each blocking collective operation has a corresponding nonblocking operation
  - May have multiple outstanding collective communications on the same communicator
  - Ordered initialization

Neighborhood Collectives

Sparse Collective Operations on Process Topologies

- MPI process topologies (Cartesian and (distributed) graph) usable for communication
  - MPI_NEIGHBOR_ALLGATHER(V)
  - MPI_NEIGHBOR_ALLTOALL(V,W)
  - Also nonblocking variants
- If the topology is the full graph, then neighbor routine is identical to full collective communication routine
  - Exception: s/rdispls in MPI_NEIGHBOR_ALLTOALLW are MPI_Aint
- Allow for optimized communication scheduling and scalable resource binding

MPI Tool Interface

New MPI Tools Chapter (Chapter 14)

- Replaces the existing Profiling Interface Chapter
- Two subsections:
  - MPI Profiling Interface, aka. PMPI or MPI interpositioning interface
    - Unchanged capabilities to MPI 2.2
    - Minor extensions and clarifications to work with new Fortran bindings
  - MPI Tool Information Interface, aka. the MPI_T interface
    - Access to internal, potentially implementation specific information
    - Two types of information:
      - Control: typically used for configuration information
      - Performance: typically used to report MPI internal performance data
    - "PAPI-like" interface for software counters within MPI

Overview of MPI_T Functionality

- Goal: provide tools with access to MPI internal information
  - MPI implementation agnostic: tools query available information
  - Access to configuration/control and performance variables

Examples of Performance Vars.

- **Number of packets sent**
- **Time spent blocking**
- **Memory allocated**

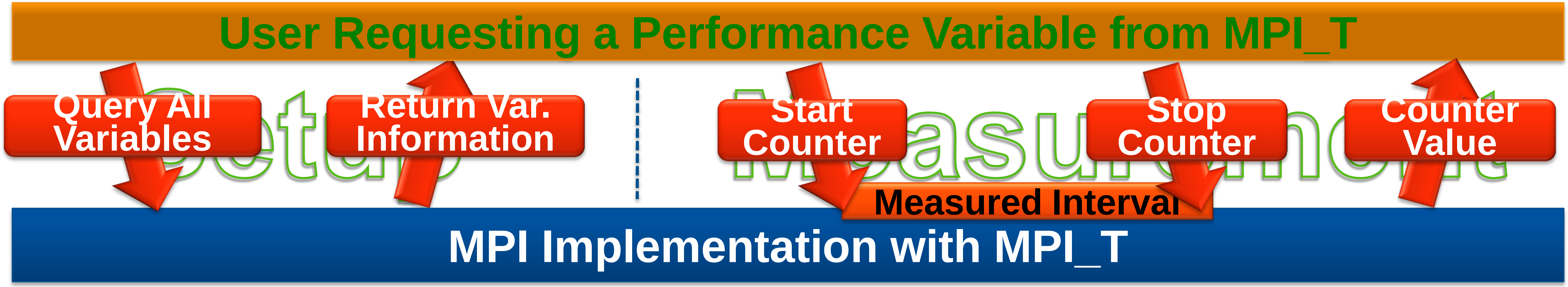
Examples for Control Vars.

- **Parameters like Eager Limit**
- **Startup control**
- **Buffer sizes and management**

- Two phase approach
  - Tool/Users queries all existing variables by name
  - Once variable has been found, allocate handle for access
  - With handle, variable contents can be read (and possibly written)
- Additional features/properties:
  - MPI_T can be used before MPI_Init / after MPI_Finalize
  - Optional variable grouping and access to semantic information

Some of MPI_T's Concepts

- Query API for all MPI_T variables / 2 phase approach
  - Setup: Query all variables and select from them
  - Measurement: allocate handles and read variables



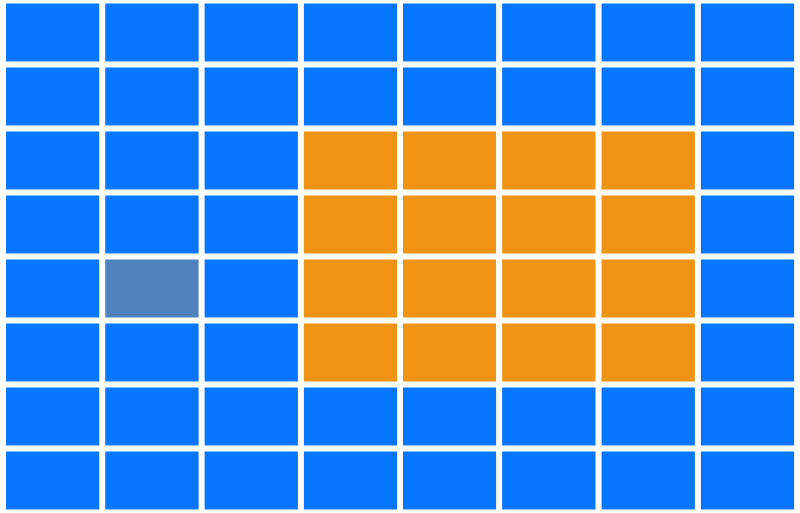
- Other features and properties
  - Ability to access variables before MPI_Init and after MPI_Finalize
  - Optional scoping of variables to individual MPI objects, e.g., communicator
  - Optional categorization of variables

Noncollective Communicator Creation

Group-Collective Communicator Creation

- MPI-2: Comm. creation is collective
- MPI-3: New group-collective creation
  - Collective only on members of new comm.

1. Avoid unnecessary synchronization
   - Enable asynchronous multi-level parallelism
2. Reduce overhead
   - Lower overhead when creating small communicators
3. Recover from failures
   - Failed processes in parent communicator can't participate
4. Enable compatibility with Global Arrays
   - In the past: GA collectives implemented on top of MPI Send/Recv



RMA Enhancements

MPI-3 RMA

- Major Extension to RMA
  - New capabilities
  - Backward compatibility to MPI 2.2
- Major Extensions
  - New ways to create MPI Windows
  - New read-modify-write operations
  - New Request-based operations
  - New synchronization operations
  - Additional memory model for cache-coherent systems
  - Other extensions to simplify use

New Ways to Create MPI_Win

- MPI_Win_allocate
  - Allocate memory at creation; permits coordinated allocation (e.g., symmetric allocation for scalability)
- MPI_Win_create_dynamic
  - Attach (and detach) memory after creation; permits more dynamic use of MPI RMA
- MPI_Win_allocate_shared
  - Allocate shared memory (where supported); permits direct (load/store) use of shared memory within MPI-only programs

New Read-Modify-Write Operations

- MPI_Get_accumulate – Extends MPI_Accumulate to also return value
- MPI_Fetch_and_op, MPI_Compare_and_swap – Atomic, single word updates; intended to provide higher performance than general MPI_Get_accumulate
- Now possible to build O(1) mutex; perform mutex-free updates

New Request-Based Operations

- MPI_Rput, MPI_Rget, MPI_Raccumulate, MPI_Rget_accumulate
  - Provide MPI request; can use any MPI request test or completion operation (e.g., MPI_Waitany)
  - Only valid within passive-target epoch
    - E.g., between MPI_Win_lock/MPI_Win_unlock
  - Provides one way to complete MPI RMA operations within a passive target epoch

New Synchronization Operations

- Permitted only within passive target epoch
- Flush
  - MPI_Win_flush, MPI_Win_flush_all completes all pending RMA operations at origin and target
  - MPI_Win_flush_local, MPI_Win_flush_local_all completes all pending RMA operations at origin
- Sync
  - Synchronizes public and private copies of win (refers to MPI memory model and subtle issues of memory consistency)
- Request operations (the “R” versions) on previous slide
  - Permit completion of specific RMA operations

New “Unified” Memory Model

- MPI 2 RMA Memory model does not require cache coherence; matched fastest systems at the time. Now called the “Separate” model, reflecting the description of public and private copies
- MPI 3 adds new “Unified” Memory model, reflecting the fact that the public and private copies are the same memory
- Users can query which is supported (new MPI_WIN_MODEL attribute on an MPI window)

Other MPI RMA Extensions

- Some behavior, such as conflicting accesses, now have undefined behavior rather than erroneous
  - Behavior of correct MPI 2.2 programs unchanged; simplifies use of MPI as a target for other RMA programming models that allow conflicting accesses
- Accumulate operations ordered by default
  - No “right” choice – some algorithms much easier if RMA operations ordered; some hardware much faster if ordering not required.
  - Info key “accumulate_ordering” (on window create) can request relaxation of ordering
- New MPI_Win_lock_all/MPI_Win_unlock_all for passive target epoch for *all* processes in Win.

New Fortran Bindings

Three methods of Fortran support

- USE mpi_f08 new 26
  - This is the only Fortran support method that is consistent with the Fortran standard (Fortran 2008 + TR 29113 and later).
  - This method is highly recommended for all MPI applications.
  - Mandatory compile-time argument checking & unique MPI handle types.
  - Convenient migration path.
- USE mpi
  - This Fortran support method is **inconsistent** with the Fortran standard, and its use is therefore **not recommended**.
  - It exists only for backwards compatibility.
  - Mandatory compile-time argument checking (but all handles match with INTEGER). 39
- new INCLUDE 'mpif.h'
  - The use of the include file mpif.h is **strongly discouraged** starting with MPI-3.0. 40
  - Does not guarantee compile-time argument checking.
  - Does not solve the optimization problems with nonblocking calls,
  - and is therefore inconsistent with the Fortran standard.
  - It exists only for backwards compatibility with legacy MPI applications.

The mpi_f08 Module

- Example:

new

```
MPI_Irecv(buf, count, datatype, source, tag, comm, request, ierror) BIND(C)
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf
    INTEGER, INTENT(IN) :: count, source, tag
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Mainly for implementer's reasons.
Not relevant for users.

Fortran compatible buffer declaration allows correct compiler optimizations

Unique handle types allow best compile-time argument checking

INTENT → Compiler-based optimizations & checking

```
MPI_Wait(request, status, ierror) BIND(C)
    TYPE(MPI_Request), INTENT(INOUT) :: request
    TYPE(MPI_Status) :: status
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Status is now a Fortran structure, i.e., a Fortran derived type

OPTIONAL ierror:
MPI routine can be called without ierror argument

Other enhancements

- Unused ierror

  INCLUDE ‘mpif.h’

  ! wrong call:

  CALL MPI_SEND(…., MPI_COMM_WORLD)

  ! → terrible implications because ierror=0 is written somewhere to the memory

- With the new module

  29

  USE mpi_f08

  new

  ! Correct call, because ierror is **optional**:

  CALL MPI_SEND(…., MPI_COMM_WORLD)

Other enhancements, continued

- With the mpi & mpi_f08 module:

new

  - Positional and **keyword-based** argument lists 33
    - CALL MPI_SEND(sndbuf, 5, MPI_REAL, right, 33, MPI_COMM_WORLD)
    - CALL MPI_SEND(**buf**=sndbuf, **count**=5, **datatype**=MPI_REAL,
      dest=right, **tag**=33, **comm**=MPI_COMM_WORLD)

The keywords are defined in the language bindings.
Same keywords for both modules.

  - Remark: Some keywords are changed since MPI-2.2 33
    - For consistency reasons, or
    - To prohibit conflicts with Fortran keywords, e.g., type, function.

Major enhancement with a full MPI-3.0 implementation

- The following features require Fortran 2003 + TR 29113
  - Subarrays may be passed to nonblocking routines 28 new
    - This feature is available if the LOGICAL compile-time constant MPI_SUBARRAYS_SUPPORTED == .TRUE.
  - Correct handling of buffers passed to nonblocking routines37
    - new if the application has declared the buffer as ASYNCHRONOUS within the scope from which the nonblocking MPI routine and its MPI_Wait/Test is called,
    - and the LOGICAL compile-time constant MPI_ASYNC_PROTECTS_NONBLOCKING == .TRUE.
  - These features **must** be available in MPI-3.0 if the target compiler is Fortran 2003+TR 29113 compliant.
    - For the mpi module and mpif.h, it is a question of the quality of the MPI library.

Status of reference implementation

- An initial implementation of the MPI 3.0 Fortran bindings are available in Open MPI
- A full implementation will not be available until compilers implement new Fortran syntax added specifically to support MPI
  - need ASYNCHRONOUS attribute for nonblocking routines
  - need TYPE(*), DIMENSION(..) syntax to support subarrays
    - e.g. MPI_Irecv(Array(3:13:2), ...)

Enhanced Datatype Support

Datatype Chapter

- Full support for MPI_Aint, MPI_Offset and MPI_Count. These types are now allowed in reduction operations (ticket #187).
- Support for large counts. New versions of MPI_Get_elements, MPI_Get_count, MPI_Set_elements, MPI_Type_size that take an MPI_Count type instead of an int for the count parameter (postfixed by _X) (ticket #265).
- Full support for C++ types in both Fortran and C)(ticket #340).
- New datatype creating function MPI_Type_create_hindexed_block similar to MPI_Type_create_indexed_block introduced in 2.2 (ticket #280).

Large Counts

Large Counts

- MPI-2.2
  - All counts are int / INTEGER
  - Producing longer messages through derived datatypes may cause problems
- MPI-3.0
  - New type to store long counts:
    - MPI_Count / INTEGER(KIND=MPI_COUNT_KIND)
  - Additional routines to handle "long" derived datatypes:
    - MPI_Type_size_x, MPI_Type_get_extent_x, MPI_Type_get_true_extent_x
  - "long" count information within a status:
    - MPI_Get_elements_x, MPI_Status_set_elements_x
  - Communication routines are not changed !!!
  - Well-defined overflow-behavior in existing MPI-2.2 query routines:
    - count in MPI_GET_COUNT, MPI_GET_ELEMENTS, and
      size in MPI_PACK_SIZE and MPI_TYPE_SIZE
      is set to MPI_UNDEFINED when that argument would overflow.

Matched Probe

Thread-safe probe: MPI_(I)MPROBE & MPI_(I)MRECV

- MPI_PROBE & MPI_RECV together are not thread-safe:
  - Within one MPI process, thread A may call MPI_PROBE
  - Another tread B may steal the probed message
  - Thread A calls MPI_RECV, but may not receive the probed message
- New thread-safe interface:
  - MPI_IMPROBE(source, tag, comm, flag, message, status) or
  - MPI_MPROBE(source, tag, comm, message, status)

  together with

  - MPI_MRECV(buf, count, datatype, message, status) or
  - MPI_IMRECV(buf, count, datatype, message, request)

**Message handle,
e.g., stored in a thread-local variable**

Topology Aware Communicator Creation

Topology-aware communicator creation

- Allows you to create a communicator whose processes can create a shared memory region
  - MPI_Comm_split_type(comm, comm_type, key, info, split_comm)
  - More generally: it splits a communicator into subcommunicators split_comm of a certain type:
    - MPI_COMM_TYPE_SHARED: shared memory capability
    - Other implementation specific types are possible: rack, switch, etc.

Removed Functionality

Removed Functionality

- Current state
  - Deprecated in MPI 2.2
  - Technical aspects
    - Supports MPI namespace
    - Support for exception handling
    - Not what most C++ programmers expect
  - Special C++ types are supported through additional MPI predefined datatypes

MPI_CXX_BOOL	bool
MPI_CXX_FLOAT_COMPLEX	std::complex<float>
MPI_CXX_DOUBLE_COMPLEX	std::complex<double>
MPI_CXX_LONG_DOUBLE_COMPLEX	std::complex<long double>

- Removed MPI-1.1 functionality (deprecated since MPI-2.0):
  - Routines: MPI_ADDRESS, MPI_ERRHANDLER_CREATE / GET / SET, MPI_TYPE_EXTENT / HINDEXED / HVECTOR / STRUCT / LB / UB
  - Datatypes: MPI_LB / UB
  - Constants MPI_COMBINER_ HINDEXED/HVECTOR/STRUCT _INTEGER
  - Removing deprecated functions from the examples and definition of MPI_TYPE_GET_EXTENT

Deprecated Functionality

Did Not Make It In

Major Functionality

- Immediate versions of nonblocking file I/O operations
- Fault Tolerance
- Helper Threads
- Clarification on multiple MPI processes in same address space

Expected Implementation Timelines
What next ?

Status of MPI-3 Implementations

	MPICH	MVAPICH	Cray	TH-MPI	IBM	Open MPI	Fujitsu	SGI-MPT
NB collectives	✔	✔	✔	✔	Spring 2014	✔	Open MPI + rel Delta	✔
Neighborhood collectives	✔	✔	✔	✔	Spring 2014		Open MPI + rel Delta	Spring 2014
RMA	✔	✔	✔	✔	Spring 2014		Open MPI + rel Delta	Fall 2013
MPI shared memory	✔	✔	✔	✔	Spring 2014			Fall 2013
Tools Interface	SC '13				Spring 2014	✔		Fall 2013
Non-collective comm. create	✔	✔	✔	✔	Spring 2014			✔
F08 Bindings (Needs fixes to MPI-3)	(Spring 2014)		(Sep. 2014)	(Spring 2014)	(Spring 2014)	(✔)		(Spring 2014)
New Datatypes	✔	✔	✔	✔	Spring 2014	✔		✔
Large Counts	✔	✔	✔	✔	Spring 2014	✔		Spring 2014
Matched Probe	✔	✔	✔	✔	Spring 2014	✔		✔

Current MPI-Forum Activities – MPI next (3.1/4.0/?)

- Fault tolerance
- Better threading support
- Cleanup from MPI-3: As implementations are maturing, small (and not so small) items are showing up that need addressing in the standard

Thank You

Mellanox® TECHNOLOGIES

Connect. Accelerate. Outperform.™