The New & Emerging MPI Standard

presented by

Richard L. Graham- Chairman

Oak Ridge National Laboratory
U.S. Department of Energy
Outline

• Goal
• Current standard
• MPI-3 directions
• Future work
Goal

To produce new versions of the MPI standard that better serves the needs of the parallel computing user community
Structure

- Chairman and Convener: Rich Graham
- Secretary: Jeff Squyres
- Steering committee:
  - Jack Dongarra
  - Al Geist
  - Rich Graham
  - Bill Gropp
  - Andrew Lumsdaine
  - Rusty Lusk
  - Rolf Rabenseifner
Current Standard: MPI 2.2
Supported Functionality

• Point-to-Point Communication
  – Blocking/Nonblocking communications
  – Persistence

• Datatypes
  – Predefined datatypes
  – Derived Datatypes (user defined)
Supported Functionality – cont’d

• Collective Communication - blocking
  – 15 collective functions (barrier, broadcast, reduction, …)

• Groups, Contexts, Communicators

• Process Topologies
  – Perhaps the best kept secret

• Environment Management

• The Info Object
Supported Functionality – cont’d

• Process Creation and Management
  • Does not require interaction with a resource manager

• One-Sided Communication

• External Interfaces – such as thread support

• File I/O

• Profiling Interface

• Deprecated Functions
  – C++ bindings
MPI-3 Status
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 of parallel process 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 maybe maintained - Routines may be deprecated
• Target release date:
  – Considering end of 2011, with incremental draft standard releases (starting Nov, 2010)

First MPI 3.0 draft standard posted at:

http://lists.mpi-forum.org/

Support for nonblocking collectives is added

Final version of the standard may be different
Tracking Forum Activities and Commenting on them

Mailing list: mpi-comments@mpi-forum.org

Subscribe at: http://lists.mpi-forum.org/

One MUST subscribe to the list to post messages to it
Current Active Working Groups

- Collective Operations and Topologies: Torsten Hoefler – University of Illinois at Urbana-Champaign, Andrew Lumsdaine - Indiana University
- Backwards Compatibility: David Solt, HP
- Fault Tolerance: Richard Graham - Oak Ridge National Laboratory
- Fortran Bindings: Craig Rasmussen - Los Alamos National Laboratory
- Remote Memory Access: Bill Gropp, University of Illinois Champaign/Urbana - Rajeev Thakur, Argonne National Laboratory
Current Active Working Groups

- Tools support: Martin Schulz and Bronis de Supinski, Lawrence Livermore National Laboratory
- Hybrid Programming: Pavan Balaji, Argonne National Laboratory
- Persistence: Anthony Skjellum, University of Alabama at Birmingham
Backward Compatibility - Charter

- Address backward compatibility issues
- The goal is to provide recommendations to MPI 3.0 proposals and introduce new proposals when appropriate to provide a reasonable transition of MPI 2.x users and the implementations that support those users to MPI 3.0 without hindering the general goals of MPI 3.0.
The Big Issue: Counts Larger Than $2^{31}$

- Counts are expressed as “int” / “INTEGER”
  - Usually limited to $2^{31}$
- Propose a new type: MPI_Count
  - Can be larger than an int / INTEGER
- “Mixed sentiments” within the Forum
  - Is it useful? Do we need it? …oy!

MPI_SEND(void *buf, int count, …)

MPI_SEND(void *buf, MPI_Count count, …)
Do we need MPI_Count?

**YES**

- Some users have asked for it
- Trivially send large msgs.
  - No need to make a new type
- POSIX went to size_t
  - Adopt MPI
- Think about the future:
  - Bigger RAM makes $2^{31}$ relevant
  - Datasets getting larger
  - Disk IO getting larger
  - Coalescing off-node msgs.

**NO**

- Very few users
- Affects many, many MPI API functions
- Potential incompatibilities
  - E.g., mixing int and MPI_Count in the same application
Ok, so how to do it? (1 of 2)

1. Use MPI_Count only for new MPI-3 routines
   - Inconsistent, confusing to users

2. Change C bindings
   - Rely on C auto-promotion
   - Bad for Fortran, bad for C OUT params

3. Only fix MPI IO functions
   - Where MPI_BYTE is used
   - Inconsistent, confusing to users

4. New, duplicate functions
   - E.g., MPI_SEND_LARGE
   - What about sizes, tags, ranks, …oy!
Ok, so how to do it? (2 of 2)

5. Fully support large datatypes
   - E.g., MPI_GET_COUNT_LONG  ✔️

6. Create a system for API versioning  ❌

7. Update all functions to use MPI_Count  ❌

8. Make new duplicate functions with MPI_Count, MPI_Tag, MPI_Size, …
   - E.g., MPI_SEND_EX  😊

Might be ok…?
Forum has hated every proposal
Technically makes current codes invalid
Rip the band-aid off!
Preserves backward Compatibility 😊
Nonblocking Collective Operations

• Moving forward in standardization process
  – No substantial changes since Jan. 2010
  – Reference Implementation (LibNBC) stable

• Final vote on 10/11
  – Unanimously accepted

• Has been released as Draft Standard on [put date here]
  – Ready to be implemented in MPI libraries
Sparse Collective Operations on Process Topologies

• New feature to enhance scalability and performance of MPI-3

• MPI process topologies (Cartesian and (distributed) graph) usable for communication
  - MPI_Sparse_gather(v)
  - MPI_Sparse_alltoall(v,w)
  - Also nonblocking variants

• Allow for optimized communication scheduling and scalable resource binding
Scalable Irregular Collectives

- Distribute argument lists of vector collectives
  - Simple interface extension
  - Low overhead
  - Reduce memory overhead from $O(P)$ to $O(1)$

- Proposal under discussion
  - Reference implementation on the way
  - Use-cases under investigation
Fault Tolerance Working Group
Fault Tolerance

- Goal: To define any additional support needed in the MPI standard to enable implementation of portable Fault Tolerant solutions for MPI based applications.

- Assumptions:
  - Backward compatibility is required.
  - Errors are associated with specific call sites.
  - An application may choose to be notified when an error occurs anywhere in the system.
  - An application may ignore failures that do not impact its MPI requests.
  - An MPI process may ignore failures that do not impact its MPI requests.
  - An application that does not use collective operations will not require collective recovery.
  - Byzantine failures are not dealt with.
Fault Tolerance

- Goal: To define any additional support needed in the MPI standard to enable implementation of portable Fault Tolerant solutions for MPI based applications.
  - Support restoration of consistent internal state
  - Add support to for building fault-tolerant “applications” on top of MPI (piggybacking)
Fault Tolerance

Items being discussed

- Define consistent error response and reporting across the standard
- Clearly define the failure response for current MPI dynamics - master/slave fault tolerance
- Recovery of
  - Communicators
  - File handles
  - RMA windows
- Data piggybacking
- Dynamic communicators
- Asynchronous dynamic process control
- **Current activity**: run-through process failure prototyping – AKA run through stabilization proposal
Updates to the MPI One-Sided Interface

presented by

MPI RMA Working Group

Oak Ridge National Laboratory
U.S. Department of Energy
Background of MPI-2 One Sided

• MPI-2’s One-Sided provides a programming model for put/get/update programming that can be implemented on a wide variety of systems

• The “public/private” memory model is suitable for systems without local memory coherence (e.g., special memory in the network; separate, non-coherent caches between actors working together to implement MPI One-Sided)

• However, the MPI One-Sided interface does not support other common one-sided programming models well. Good features of the MPI-2 One-sided, including the following, must be preserved
  – To allow for overlap of communication with other operations, nonblocking RMA operations are required
  – The RMA model must support non-cache-coherent and heterogeneous environments
  – Transfers of noncontiguous data, including strided (vector) and scatter/gather must be supported
  – Scalable completion (a single call for a group of processes) is required
Goals for MPI-3 One Sided

• The goal of the MPI-3 RMA Working Group is to address many of these limitations, including
  – In order to support RMA to arbitrary locations, no constraints on memory, such as symmetric allocation or collective window creation, can be required
  – RMA operations that are imprecise (such as access to overlapping storage) must be permitted, even if the behavior is undefined
  – The required level of consistency, atomicity, and completeness should be flexible
  – Read-modify-write operations and compare and swap are needed for efficient algorithms
Major New Features

- New Window Types
  - `MPI_Win_allocate` – memory allocated by routine, permits symmetric allocation
  - `MPI_Win_create_dynamic` – memory attached to window as needed by a local operation
- New Read-Modify-Write operations
  - `MPI_Get_accumulate`, `MPI_Compare_and_swap`
- New synchronization and completion calls
- Query for new mode (`MPI_RMA_UNIFIED`) to allow applications to tune for cache-coherent architectures
- Relaxed rules for certain access patterns
  - Results undefined rather than erroneous; matches other share-memory and RDMA approaches
Tool Interfaces for MPI-3

Goals of the tools working group

- Extend tool support in MPI-3 beyond the PMPI interface
- Document state of the art for de-facto standard APIs
The MPIT Performance Interface

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

Examples of Performance Vars.
  ‣ Number of packets sent
  ‣ Time spent blocking
  ‣ Memory allocated

Similar process for Control Vars.
  ‣ Parameters like Eager Limit
  ‣ Startup control
  ‣ Buffer sizes and management
The MPIT Performance Interface (cont.)

- Main philosophy
  - MPI specifies what information is available
  - Tools can query this information (similar concept as PAPI)
  - Complementary to/will NOT replace the MPI profiling interface PMPI

- Information provided as a set of variables
  - **Performance variables**
    Provided functionality: Query internal state of the MPI library at runtime
  - **Configuration/control variables**
    Provided functionality: List, query, and (if the MPI implementation supports this) set configuration settings

- Status of MPIT
  - Current draft available on MPI-3 tools WG WiKi
  - (Hopefully) final discussions in tools WG
  - Feedback wanted!
The MPIR Companion Document

- MPIR = established process acquisition interface for MPI
  - Enables tools to query all processes involved in an MPI job
  - Implemented by most MPIs
  - Used by many tools, (Totalview, DDT, O|SS)
  - MPIR not standardized / Exists in several variants

- Goal of MPIR activity in tools WG
  - Document the current state of the art as a guide for users
  - No extensions or changes (for now)
  - Published as a companion document to MPI

- Status
  - Final draft available on MPI-3 tools WG WiKi
  - Passed first vote, Second vote scheduled for December
Next Steps for the Tools WG

• Additional areas under discussion or possible directions
  – Companion document to describe the message queue interface
  – Extensions for further third party debug interfaces
  – Standardization of a more scalable process acquisition API
  – Extended version of MPI_Pcontrol
  – Low-level tracing options in MPIT

• Other suggestions/contributions welcome!
  – MPI-3 tools working open to everyone
  – Bi-weekly phone calls: Monday 8am PT
  – Documents, Minutes, Discussion on WG Wiki:
    http://svn.mpi-forum.org/ MPI 3.0, Tools Workgroup
MPI-3 Fortran

Finally, quality MPI interfaces for Fortran
Severe Problems with the Existing MPI Fortran Interfaces

• Use of “mpif.h” provides no type checking

• The “use mpi” module is impossible to fully implement in a standards-compliant way

• Very scary issues with compiler optimizations:
  – Compiler may copy buffers used with non-blocking communication
  – Compiler can move code statements surrounding MPI_WAIT calls
Some of the Proposed Changes

- Existing “use mpi” module with full compile time argument checking

- New “use mpi_f08” module with typed MPI handles
  - MPI_Comm, MPI_Datatype, MPI_Errhandler, MPI_Info, MPI_Request, ... etc.

- Array subsections supported

- The IERROR argument in Fortran calls is optional

- Formal guidance provided to users how to use non-blocking MPI functionality

- Strong type checking ✔
- Enhanced type checking ✔
- Safety in asynchronicity ✔
- yay! ✔
- No one uses it anyway ✔
Implications

• Backwards compatibility is preserved
  – New features are available in a new module
  – You must modify your code to get the new features

• Old and new Fortran MPI features can be combined in a single MPI application

• Implementation being prototyped in Open MPI
Collective Communications and Topology Working Group
Hybrid Programming WG Goals

• Ensure that MPI has the features necessary to facilitate efficient hybrid programming

• Investigate what changes are needed in MPI to better support:
  − Traditional thread interfaces (e.g., Pthreads, OpenMP)
  − Emerging interfaces (like TBB, OpenCL, CUDA, and Ct)
  − PGAS (UPC, CAF, etc.)
  − Shared Memory

• Mailing list: mpi3-hybridpm@lists.mpi-forum.org

• Wiki: https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/MPI3Hybrid

• Biweekly telecons every Tuesday at 11am Central time
Threads with Endpoints

Current Design

Proposed Design
MPI Helper Thread Teams

• Thread teams are allowed to share MPI work
  – Group of threads join the team, and make MPI calls
    – MPI will share resources provided by all threads
      for all the MPI calls together (compute resources,
      end points)

• Useful for OpenMP applications where threads are
  forked for computational parallelism, but the MPI part
  is serialized
Shared Memory Extensions to MPI

• Allowing MPI to create and destroy SystemV style shared memory regions
  – MPI_COMM_ALLOC_SHM and MPI_COMM_FREE_SHM

• User’s responsibility to figure out what processes can create shared memory regions and what processes cannot
On Line Information

meetings mpi-forum org

Meeting Schedule
Meeting logistics
Mailing list signup
Mail archives
Wiki pages for each working group