Outline

• Goal
• Forum Structure
• Meeting Schedule
• Scope
• Voting Rules
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
Scope: Small changes to the standard. A small change is defined as one that does not break existing user code - either by interface changes or semantic changes - and does not require large implementation changes.

Lead: Bill Gropp
• Released Sept 4th, 2009 in Helsinki, Finland

• Highlights
  – Modernize C and Fortran language support
  – Deprecate C++ bindings
  – Fix graph interface scalability issues
  – Allowing concurrent read access to user send buffers
  – Many miscellaneous corrections
We are a member of the Message Passing Interface (MPI) Forum

The new MPI-2.2 Standard (Sep. 2009) is now available: www.mpi-forum.org

Service for all MPI users:

Hardcover book (647 pp.) is sold at cost on HLRS booth #2245 and at MPI-3 BOF

Wednesday Nov.18, 5:30 – 7pm, D135-136

25 $ special price at SC09 in Portland

In USA: Without shipping costs only at SC09
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.

Lead: Rich Graham

Backwards compatibility maybe maintained - Routines may be deprecated
• Target release date: Still being release
  – Considering Sept, 2011, with incremental draft standard releases

• Comments on plan are solicited:
  http://mpi-forum.questionpro.com/
  Password: mpi3

Mailing list: mpi-comments@mpi-forum.org
Subscribe at: http://lists.mpi-forum.org/
Current Active Working Groups

- Collective Operations and Topologies: Torsten Hoefler 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
- Tools support: Martin Schulz and Bronis de Supinski, Lawrence Livermore National Laboratory
- Hybrid Programming: Pavan Balaji, Argonne National Laboratory
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.
Backward Compatibility Premises

- MPI-2 code should run on MPI-3 implementations without substantial source code changes
  - substantial == ? not easily automated

- 3.0 document must not require indefinite support for multiple versions of the standard.
  - a transition period may be acceptable
Backward Compatibility Current Idea

- Use symbol-specific version numbering, with macro (or weak symbol?) mapping the “best” name to most current name, by default.

- Use a global preprocessor macro to map all versioned symbols to the version provided by a particular version of MPI standard.

- Use symbol-specific macro to override version mapping for a particular symbol.
Backward Compatibility - Examples

• Size of the count argument in interface functions
  - `int MPI_Isend( void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request )`
  - Maybe add MPI_Count handle
  - Do we add a 2\textsuperscript{nd} set of interface functions?
    - `int MPI_Isend_ex( void *buf, MPI_Count count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request )`
  - Do we break backward compatibility?
    - `int MPI_Isend( void *buf, MPI_Count count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request )`
  - Do we just leave this as is?
Collective Operations

Goals:
- update the collective communication functions based on our experience since MPI-2.1
- enable more scalable design and more flexible specification of application communication patterns
- enable intelligent mapping and optimization strategies for application communications
- explore new ways to express application communication (beyond point-to-point)
- discuss possible scalability issues (communicator and group management)
- collective communication support for higher-level libraries
Collective Operations

• Assumption:
  − the scale of systems increases steadily
  − hierarchical (e.g., multi-core) systems will become more common
  − capabilities of network interfaces increase
  − future network might be sparse and with lower effective bisection bandwidth
  − higher-level languages become more important in parallel programming
Collective Operations

Done:

- Nonblocking Collectives: part of MPI-3 draft standard
  - MPI_Ibcast(&buf, 1, MPI_INT, 0, comm, &req)
  - /* compute */
  - MPI_Wait(&req, MPI_STATUS_IGNORE);
  - reference/preview implementation: LibNBC
Collective Operations

Under consideration:

• Topological Collectives
  – MPI_Neighbor_reduce(), MPI_Neighbor_alltoall(), MPI_Neighbor_gather()
  – Hoefler, Traeff: “Sparse Collective Operations for MPI”

• Streaming Collectives
  – react to data as it comes in
  – not decided yet, is there a need for this?

• Persistent Collectives
  – persistent P2P does not seem to be used much
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
Remote Memory Access

- Goal: To provide improved support for Remote Memory Access.
  - Read-Modify-Write operations
  - Flexible RMA synchronization
  - Scalable (not global) completion
  - Registration of data for one-sided operations
  - Support for non-contiguous data, and for overlapping regions

Just getting off the ground
• Current “proposals”
  – Fix performance issues within the current standard specification
  – New interface where users can specify
    • Completion semantics
    • Synchronous/Asynchronous
    • Ordering
  – Simplified implementation
    • Restricting use support (predefined data types)
    • User responsible for data consistency
Tools

• The goal of the tools WG are interfaces to
  ‐ Ease and standardize tool deployment and control
  ‐ Enable more introspection into the internals of an MPI implementation

• Support for wide range of tools, including, but not limited to
  ‐ Performance measurements tools
  ‐ Debuggers
  ‐ Correctness checkers

• Motivation:
  ‐ Provide reliable and portable interfaces
  ‐ Ability to create cross-platform tools

• All efforts are complimentary to the existing PMPI interface
Tools

• A Process Acquisition Interface close to the MPIR pseudo standard
  – Locate all MPI tasks for external tools

• A Performance Information Interface providing low level performance details
  – Access to configuration variables and MPI internal performance counters

• Symbol Detection Interface
  – Enable the dynamic detection of debugger extensions

• The existing Message Queue Interface with extensions for Collectives
  – Introspection of the messages queues during debugging.

• An interface to query information about opaque MPI handles
  – Ability for debuggers to show context for datatypes, communicators, ...
There are Severe Problems with the Existing MPI Fortran Interfaces

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

• The MPI Fortran module is impossible to fully implement in a standards-compliant way.

• Very scary issues with compiler optimizations:
  − The Fortran compiler may employ copyin/copyout semantics, thus completely interfering with asynchronous MPI calls.
  − The Fortran compiler can legally move code statements surrounding MPI_Wait calls. This may break code in an unpredictable fashion.
Goals of the MPI-3 Fortran Effort

• Provide a Fortran standards-compliant mechanism to suppress copy-in/copy-out semantics and code motion for MPI asynchronous operations.

• Provide explicit interfaces that suppress argument checking for MPI choice buffers (C (void *) formal parameters).

• Allow vendors to take advantage of the Fortran 2003 interoperability standard with C.

• Examine the feasibility of simplifying the Fortran interfaces by making some of the arguments optional.

• Design a palatable application migration path from older MPI Fortran bindings to the new/proposed MPI-3 bindings.
Highlight of things to come

• New syntax has been added to the Fortran language, specifically for MPI interfaces using void * buffers, indicating *any type, any rank*:
  - TYPE(*), DIMENSION(..) :: buffer

• Derived types have been defined to enhance type safety:
  - MPI_Comm, MPI_Datatype, MPI_Errhandler,
    MPI_Info, MPI_Request, and MPI_Status

• The ierr argument in Fortran calls is optional.

• TARGET and ASYNCHRONOUS attributes are to be employed by users to inhibit compiler optimizations.
Hybrid Programming

• 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.)
Example issues being addressed

- Threads as first-class citizens (rank != process)
  - Lockless Communication for MPI+threads
  - Allow MPI implementations to avoid internal locks when multiple threads communicate using MPI
  - Useful to boost performance on multi- and many-core architectures

- Interoperating MPI with PGAS languages
  - Hybrid programs that can make MPI and/or PGAS calls

- Additional API to improve programmability for MPI + threads applications
  - E.g., allowing a thread to receive the data related to a request that it probed
Example Proposal: Threads with Endpoints

Current Design

Proposed Design
Each MPI Endpoint has unique rank in MPI_COMM_EWORLD
- rank in derived communicators computed using MPI rules

MPI code executed by thread(s) attached to endpoint
- Including collectives
- thread is attached to at most one endpoint
On Line Information

meetings.mpi-forum.org

Meeting Schedule
Meeting logistics
Mailing list signup
Mail archives
Wiki pages for each working group
Comments on plan are solicited:

http://mpi-forum.questionpro.com/
Password: mpi3

Mailing list: mpi-comments@mpi-forum.org
Subscribe at: http://lists.mpi-forum.org/