Overview of MPI

Shirley Browne

Computer Science Department
University of Tennessee

browne@cs.utk.edu

Outline

- Motivation for MPI
- The process that produced MPI
- The MPI "Standard"
- Features of MPI
  - Point-to-point communication
  - Collection communication
  - User-defined datatypes
  - Process groups
  - Communication domains
  - Process topologies
  - Interaction with execution environment
  - Profiling interface
- Comparison with alternatives
- Implementation status
- MPI resources and tools
- MPI-2

Motivation for MPI

- Message passing had become a widely used parallel programming paradigm.
- Each vendor had implemented its own variant of a message passing library.
- Several freely available systems had demonstrated that a portable message passing system could be implemented efficiently.
- Existing message passing systems had some critical shortcomings.

There was a need to develop a practical, portable, efficient, and flexible standard for message passing that would

- include the best features from existing systems
- include new features to deal with shortcomings of existing systems
- allow efficient implementations across a wide variety of parallel computing systems.

Goals of MPI

- Design an application programmer interface not too different from current practice
- Achieve portability across different machines – i.e., the same message passing source code can be executed as long as the MPI library is available (although some tuning might be needed to take best advantage of the features of each system)
  - Distributed memory computers
  - Shared memory computers
  - Networks of workstations
- Allow efficient implementations
- Allow for implementations for heterogeneous environments
- Allow convenient language bindings (initially C and Fortran 77)
- Provide a reliable communication interface (i.e., the user need not cope with communication failures)
- Allow for thread safety

Process that Produced MPI

- CRPC Workshop, April 1992, Williamsburg VA
  - Discussed basic features essential to standard message passing interface
  - Formed working group
- Preliminary draft proposal (MPI1)
  - Dongarra, Hempel, Hey, and Walker
  - November 1992
  - Revised February 1993
- Message Passing Interface Forum
  - Formed in November 1992 in Minneapolis
  - Subcommittees for major components of standard with email discussion lists
  - Over 80 people from 40 organizations
  - Vendors, industrial users, researchers
  - Met every six weeks for two days
  - Draft MPI standard presented at Supercomputing '93

Geneology

MPI influenced by

- Express
- Intel NX/2
- nCUBE Vertex
- p4
- PARMACS
- Zipcode
- CHIMP
- PVM
- Chameleon
- PICL
- IBM External User Interface

The MPI Standard

- Community accepted "standard"
- Version 1.0 released in June 1994
- Version 1.1 (correction and clarifications) released in June 1995
- Syntax and semantics of a standard core of message passing library routines
- 125 functions (6 basic)
- Not in MPI-1
  - Explicit shared-memory operations
  - Operations requiring non-standard operating system support (e.g., interrupt-driven receives, remote execution, active messages)
  - Program construction tools
  - Debugging tools
  - Explicit support for threads
  - Task management
  - I/O functions

Types of MPI Calls

- **local** – if the completion of the procedure depends only on the local executing process
- **non-local** – if completion of the procedure may require execution of some MPI procedure on another process
- **blocking** – if return from the procedure indicates the user is free to reuse resources specified in the call
- **nonblocking** – if the procedure may return before the operation initiated by the call completes and before the user is allowed to reuse resources
- **collective** – if all processes in a process group need to invoke the procedure

Point-to-Point Communication

SEND

Mode	Blocking	Nonblocking
Standard	`mpi_send`	`mpi_isend`
Buffered	`mpi_bsend`	`mpi_ibsend`
Synchronous	`mpi_ssend`	`mpi_issend`
Ready	`mpi_rsend`	`mpi_irsend`

RECEIVE

Blocking	Nonblocking
`mpi_recv`	`mpi_irecv`

A **blocking** operation completes when the call returns.
A **nonblocking** operation completes when the matching Wait or Test call returns successfully.

Message Ordering

- Successive messages sent by one process to another process are received in order
  - Messages sent by concurrent threads may be received in any order.
  - A consistent, total order on communication events in the system is not implied.
- MPI makes no guarantee of fairness – i.e., starvation is possible.

Communication Modes

- Standard
  - Up to implementation to decide whether outgoing messages will be buffered
  - Non-local completion semantics
- Buffered
  - Buffer space provided by the application
  - Local completion semantics
- Synchronous
  - Completion indicates that the send buffer can be reused and that the receiver has started executing the matching receive
  - Non-local completion semantics
- Ready
  - May be started only if the matching receive has already been posted
  - May allow removal of hand-shake and improved performance
  - Otherwise, same semantics as standard mode

Blocking Send and Receive

MPI_SEND(buf, count, datatype, dest, tag, comm)

IN	`buf`	initial address of send buffer
IN	`count`	number of entries to send
IN	`datatype`	datatype of each entry
IN	`source`	rank of destination
IN	`tag`	message tag
IN	`comm`	communicator

MPI_RECV(buf, count, datatype, source, tag, comm, status)

OUT	`buf`	initial address of receive buffer
IN	`count`	max number of entries to receive
IN	`datatype`	datatype of each entry
IN	`source`	rank of source
IN	`tag`	message tag
IN	`comm`	communicator
OUT	`status`	return status

Example 1a

```
CALL MPI_COMM_RANK(comm, rank, ierr)
IF (rank .EQ. 0) THEN
    CALL MPI_RECV(recvbuf, count, MPI_REAL,
                  1, tag, comm, status, ierr)
    CALL MPI_SEND(sendbuf, count, MPI_REAL,
                  1, tag, comm, ierr)
ELSE IF (rank .EQ. 1) THEN
    CALL MPI_RECV(recvbuf, count, MPI_REAL,
                  0, tag, comm, status, ierr)
    CALL MPI_SEND(sendbug, count, MPI_REAL,
                  0, tag, comm, ierr)
END IF
```

Example 1b

```
CALL MPI_COMM_RANK(comm, rank, ierr)
IF (rank .EQ. 0) THEN
    CALL MPI_SEND(sendbuf, count, MPI_REAL,
                  1, tag, comm, ierr)
    CALL MPI_RECV(recvbuf, count, MPI_REAL,
                  1, tag, comm, status, ierr)
ELSE IF (rank .EQ. 1) THEN
    CALL MPI_SEND(sendbug, count, MPI_REAL,
                  0, tag, comm, ierr)
    CALL MPI_RECV(recvbuf, count, MPI_REAL,
                  0, tag, comm, status, ierr)
END IF
```

Example 1c

```
CALL MPI_COMM_RANK(comm, rank, ierr)
IF (rank .EQ. 0) THEN
    CALL MPI_SEND(sendbuf, count, MPI_REAL,
                  1, tag, comm, ierr)
    CALL MPI_RECV(recvbuf, count, MPI_REAL,
                  1, tag, comm, status, ierr)
ELSE IF (rank .EQ. 1) THEN
    CALL MPI_RECV(recvbuf, count, MPI_REAL,
                  0, tag, comm, status, ierr)
    CALL MPI_SEND(sendbug, count, MPI_REAL,
                  0, tag, comm, ierr)
END IF
```

Example 1d

```
CALL MPI_COMM_RANK(comm, rank, ierr)
IF (rank .EQ. 0) THEN
    CALL MPI_SENDRECV(sendbuf, count, MPI_REAL, 1, tag,
    recvbuf, count, MPI_REAL, 1, tag, comm, status, ierr)
ELSE IF (rank .EQ. 1) THEN
    CALL MPI_SENDRECV(sendbuf, count, MPI_REAL, 0, tag,
    recvbuf, count, MPI_REAL, 0, tag, comm, status, ierr)
END IF
```

Collective Communication

- Transmits data among all processes in a group
- All collective operations are blocking
- No tags – collective calls matched according to order of execution
- MPI provides the following collective operations:
  - Barrier synchronization
  - Global communication functions
    - Broadcast
    - Gather
    - Scatter
    - Allgather – variation on Gather where all members of the group receive the result)
    - Alltoall – Scatter/Gather data from all members to all members
  - Global reduction operations such as sum, max, min, or user-defined
    - Reduction
      - Result returned to all group members
      - Result returned to only one member
    - Combined reduction and scatter
    - Scan (also called prefix)

Datatypes

- A message contains elements of some particular datatype.
- MPI datatypes
  - Basic types
  - Derived types
- Derived types are constructed from basic types or previously constructed derived types.
- C types are different from Fortran types.

MPI Basic Datatypes – C

MPI Datatype	C Datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_BYTE	
MPI_PACKED	

MPI Basic Datatypes – Fortran

MPI Datatype	Fortran Datatype
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

Derived Datatypes

Motivation

- Send non-contiguous data
- Send one message containing mixed data types
- Allow structured access to memory based on data types

Type Maps

basic datatype 0	displacement of datatype 0
basic datatype 1	displacement of datatype 1
...	...
basic datatype n-1	displacement of datatype n-1

$$Typemap = \{(type_0, disp_0), ..., (type_{n-1}, disp_{n-1})\}$$

Examples of derived types:

- Contiguous
- Vector
- Struct

Division of Processes

Allow different (possibly overlapping) subsets of processes to perform independent work

- Same logical view for different subsets
- Avoid message conflicts between modules
- Limit collective operations to processes that need to be involved
- Dynamic – runtime callable routines

group – an ordered set of process identifiers

rank

- associated with each process in a group
- start from zero, contiguous

communicator – specifies a *communication domain*

- *intracommunicator*
  - communication within a single group
  - point-to-point
  - collective
  - predefined: MPI_COMM_WORLD, MPI_COMM_SELF
- *intercommunicator* – point-to-point communication between two disjoint groups of processes

Process Topology

- Optional attribute for intracommunicator
- Provides convenient naming mechanism that reflects logical communication pattern
- Can Assist runtime system in mapping processes onto hardware
- Machine-independent, logical, virtual
- Communication pattern represented by a graph
  - Vertices represent processes
  - Edges represent messages
- Regular structures (Cartesian)
  - Ring
  - Grid
  - Torus
  - Hypercube

Interaction with Execution Environment

Initialization and Exit

- `MPI_INIT()`
  - Must be called before any other MPI routine
  - Allows implementation-dependent setup to take place
- `MPI_FINALIZE()`
  - Must be called by every process after all other MPI calls
  - Cleans up all MPI state
- `MPI_ABORT(comm, errorcode)`
  - Makes best-effort attempt to abort all tasks in group of `comm`
  - Implementations may ignore the `comm` argument

Environmental Enquiry

- Predefined attributes for `MPI_COMM_WORLD`
  - `MPI_TAG_UB` Upper bound for tag values
  - `MPI_HOST` Host process rank, if it exists
  - `MPI_IO` Rank of a node that can provide language-standard I/O facilities
  - `MPI_WTIME_IS_GLOBAL` Whether or not clocks are synchronized
  - Vendors may provide additional attributes (e.g., memory size)
- `MPI_GET_PROCESS_NAME`

Interaction with Execution Environment (cont.)

Timer

- `MPI_WTIME()` Returns double precision floating-point number of seconds since some time in the past, local to node
- `MPI_WTICK()` Returns resolution of `MPI_WTIME()`

Interaction with Execution Environment (cont.)

Error Handling

- Possible causes of errors
  - Non-existing destination in send operation
  - Buffer too small in receive operation
  - Available system resources exceeded
- Set of errors that are handled is implementation-dependent
- When an error occurs during an MPI call
  - MPI *exception* occurs
  - MPI *error handler* is invoked
  - If error handler does not cause process to halt, MPI call returns suitable error code
- Error handler
  - Associated with a communicator
  - Inherited from parent communicator by newly formed communicator
  - Predefined error handlers
    - `MPI_ERRORS_ARE_FATAL` (default)
    - `MPI_ERRORS_RETURN` (returns error code)
  - MPI provides calls to
    - Create new error handler
    - Associate error handler with communicator
    - Get error handler associated with a communicator

Profiling Interface

- Mechanism by which implementors of profiling tools may collect performance information without access to MPI implementation source code
- May be used for other purposes – e.g., "internetworking" multiple MPI implementations
- MPI requires implementations to provide a mechanism through which all MPI functions may be accessed with a name shift "`PMPI_`"
- Link step (in C)

  ```
  cc ...  -lprof -lmpi
  ```

  or

  ```
  cc ...  -lprof -lpmpi -lmpi
  ```

Comparison with Alternatives

- Other message passing libraries
  - PVM
    - Dynamic processes
    - Fault tolerance
    - PVMPI/MPI_Glue – Allows interoperation of multiple MPI-1 applications, across diferent MPI implementations, using PVM as an intercommunication layer.
    - Unify – dual API, single programs can contain both MPI and PVM code
  - Vendor-specific (e.g., Intel NX, IBM MPL)
    - Good performance, but programs not portable
- Data parallel programming – High Performance Fortran (HPF)
  - Performance may not be as good as explicit message passing
  - Irregular problems and data distrubtions not handled (but research underway to extend HPF to handle such problems)
  - Research at Argonne on HPF/MPI
    - HPF binding for MPI
    - Expression of mixed task/data-parallel computations
    - Coupling of separately compiled data-parallel modules
- Object-oriented methods (e.g., POOMA, LAPACK++, IML++, etc.)
  - Easy to use
  - Greater software reuse
  - Reduced performance

Implementation Status

- MPICH, Argonne National Lab/Mississippi State University
- CHIMP/MPI, Edinburgh Parallel Computing Centre
- LAM MPI, Ohio Supercomputing Center
- MPICH/NT for Windows NT Workstations, Mississippi State
- MPI-FM, high performance port of MPICH for SPARCstation clusters interconnected by Myrinet, UIUC
- SGI MPI for the Power Challenge
- CRI/EPCC MPI for the Cray T3D
- IBM MPI Library for RS/6000, SP
- HP-MPI for HP S-series (Exemplar)
- Paragon OS R1.4

See `http://www.osc.edu/mpi/` for current information.

MPICH – A Freely-available Portable MPI Implementation

- Complete MPI implementation
- MPP's: IBM SP1 and SP2, Intel IPSC860 and Paragon, TMC CM-5, Meiko CS-2, NCube, Cray T3D
- SMP's: SGI, Convex Exemplar
- On workstation networks: Sun, Dec, HP, IBM, SGI, Linux, FreeBSD, NetBSD
- Includes multiple profiling libraries for timing, event logging, and animation of programs.
- Includes trace upshot visualization program, graphics library
- Efficiently implemented for shared-memory, high-speed switches, and network environments
- Man pages
- Source included
- Available at ftp.mcs.anl.gov in pub/mpi/mpich.tar.Z

MPI Resources

- Web pages
  - `http://www.netlib.org/mpi/`
  - `http://www.erc.msstate.edu/mpi/`
  - `http://www.mcs.anl.gov/mpi/`
- Books
  - *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, by Gropp, Lusk, and Skjellum, MIT Press, 1994
  - *MPI Annotated Reference Manual*, by Otto, Dongarra, Lederman, Snir, and Walker, MIT Press, 1995
- Newsgroup
  - `comp.parallel.mpi`
- Mailing list
  - `mpi-comm@mcs.anl.gov`: the MPI Forum discussion list.
- Test suites
  - `http://www.mcs.anl.gov/Projects/mpi/mpi-test/tsuite.html`
- MPI-2 draft document
  - http://www.cs.wisc.edu/ lederman/mpi2/mpi2-report.ps.Z

MPI Tools and Libraries

- Debuggers and Performance Analyzers
  - MPE, upshot for MPICH (Argonne)
  - Pablo 5.0 (UIUC)
  - XMPI for LAM MPI
  - VAMPIR (Pallas)
  - p2d2 (NASA Ames)
- Libraries
  - ScaLAPACK
  - Aztec
  - PETSc
  - PIM
  - PGAPack
- More information available from the National HPCC Software Exchange – `http://www.nhse.org/`

MPI-2

- Process creation and management
- One-sided communications
- Extended collective operations
  - Extension of collective operations to intercommunicators
  - Non-blocking collective communication
  - Persistent collection communication
  - New collective operations (in-place, exclusive scan)
- External interfaces
  - User-defined requests
  - Association of names with communicators
  - User-defined error classes and error codes
  - Access to datatypes
  - Interaction with threads
- C++ and Fortran 90 language bindings
- Parallel I/O

MPI-2: Dynamic Processes

- Important classes of applications require process control
  - Task farms
  - Serial applications with parallel modules
  - Run-time determination of number and type of processes
- MPI-2 proposes a minimal interface between the application and external resource and process managers
- Starting processes
  - `MPI_SPAWN`
  - `MPI_SPAWN_MULTIPLE`
  - `MPI_ISPAWN`
  - `MPI_ISPAWN_MULTIPLE`
- Establishing communication
  - Client/server connections via ports

MPI-2: One-Sided Communications

- Remote Memory Access (RMA)
  - Allows one process to specify communication parameters for both sending and receiving side
  - Allows MPI implementors to take advantage of fast communication mechanisms provided by various platforms (e.g., coherent or non-coherent shared memory, DMA engines, communication co-processors)
- New communication calls
  - `MPI_PUT` (remote write)
  - `MPI_GET` (remote read)
  - `MPI_ACCUMULATE` (remote update)
- Three synchronization barrier synchronization
  - Barrier synchronization
  - "Window" synchronization
  - Shared and exclusive locks