MPI-2: Extending the Message-Passing Interface

Bill Gropp
Rusty Lusk
Argonne National Laboratory

1

Outline

- Background
- Review of strict message-passing model
- Dynamic Process Management
  - Dynamic process startup
  - Dynamic establishment of connections
- One-sided communication
  - Put/get
  - Other operations
- Miscellaneous MPI-2 features
  - Generalized requests
  - Bindings for C++/ Fortran-90; interlanguage issues
- Parallel I/O

2

Reaction to MPI-1

- Initial public reaction:
  - It's too big!
  - It's too small!
- Implementations appeared quickly
  - Freely available (MPICH, LAM, CHIMP) helped expand the user base
  - MPP vendors (IBM, Intel, Meiko, HP-Convex, SGI, Cray) found they could get high performance from their machines with MPI.
- MPP users:
  - quickly added MPI to the set of message-passing libraries they used;
  - gradually began to take advantage of MPI capabilities.
- MPI became a requirement in procurements.

3

1995 OSC Users Poll Results

- Diverse collection of users
- All MPI functions in use, including "obscure" ones.
- Extensions requested:
  - parallel I/O
  - process management
  - connecting to running processes
  - put/get, active messages
  - interrupt-driven receive
  - non-blocking collective
  - C++ bindings
  - Threads, odds and ends

4

MPI-2 Origins

- Began meeting in March 1995, with
  - veterans of MPI-1
  - new vendor participants (especially Cray and SGI, and Japanese manufacturers)
- Goals:
  - Extend computational model beyond message-passing
  - Add new capabilities
  - Respond to user reaction to MPI-1
- MPI-1.1 released in June, 1995 with MPI-1 repairs, some bindings changes
- MPI-1.2 and MPI-2 released July, 1997
  - http://www.mpi-forum.org

5

Contents of MPI-2

- Extensions to the message-passing model
  - Dynamic process management
  - One-sided operations
  - Parallel I/O
- Making MPI more robust and convenient
  - C++ and Fortran 90 bindings
  - External interfaces, handlers
  - Extended collective operations
  - Language interoperability
  - MPI interaction with threads

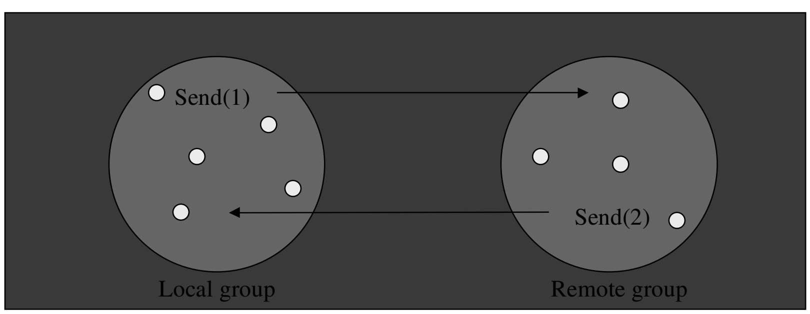
6

Intercommunicators

- Contain a *local* group and a *remote* group
- Point-to-point communication is between a process in one group and a process in the other.
- Can be merged into a normal (intra) communicator.
- Created by **`MPI_Intercomm_create`** in MPI-1.
- Play a more important role in MPI-2, created in multiple ways.

7

Intercommunicators



- In MPI-1, created out of separate intracommunicators.
- In MPI-2, created by partitioning an existing intracommunicator.
- In MPI-2, the intracommunicators may come from different MPI_COMM_WORLDs

8

Dynamic Process Management

- Issues
  - maintaining simplicity, flexibility, and correctness
  - interaction with operating system, resource manager, and process manager
  - connecting independently started processes
- Spawning new processes is *collective,* returning an intercommunicator.
  - Local group is group of spawning processes.
  - Remote group is group of new processes.
  - New processes have own **`MPI_COMM_WORLD`**.
  - **`MPI_Comm_get_parent`** lets new processes find parent communicator.

9

Spawning Processes

```
MPI_Comm_spawn(command, argv, numprocs, info,
  root, comm, intercomm, errcodes)
```

- Tries to start **`numprocs`** process running **`command`**, passing them command-line arguments **`argv`**.
- The operation is collective over **`comm`**.
- Spawnees are in remote group of **`intercomm`**.
- Errors are reported on a per-process basis in **`errcodes.`**
- **`Info`** used to optionally specify hostname, archname, wdir, path, file, softness.

10

Spawning Multiple Executables

- **`MPI_Comm_spawn_multiple( ... )`**

- Arguments **`command, argv, numprocs, info`** all become arrays.
- Still collective

11

In the Children

- **`MPI_Init`** (only MPI programs can be spawned)
- **`MPI_COMM_WORLD`** is processes spawned with one call to **`MPI_Comm_spawn.`**
- **`MPI_Comm_get_parent`** obtains parent intercommunicator.
  - Same as intracommunicator returned by **`MPI_Comm_spawn`** in parents.
  - Remote group is spawners.
  - Local group is those spawned.

12

Manager-Worker Example

- Single manager process decides how many workers to create and which executable they should run.
- Manager spawns n workers, and addresses them as 0, 1, 2, ..., n-1 in new intercomm.
- Workers address each other as 0, 1, ... n-1 in **`MPI_COMM_WORLD`**, address manager as 0 in parent intercomm.
- One can find out how many processes can usefully be spawned.

13

Establishing Connections

- Two sets of MPI processes may wish to establish connections, e.g.,
  - Two parts of an application started separately.
  - A visualization tool wishes to attach to an application.
  - A server wishes to accept connections from multiple clients. Both server and client may be parallel programs.
- Establishing connections is collective but asymmetric ("Client"/"Server").
- Connection results in an intercommunicator.

14

Connecting Processes

- Server:
  - **`MPI_Open_port( info, port_name )`**
    - » system supplies port_name
    - » might be host:num; might be low-level switch #
  - **`MPI_Comm_accept( port_name, info, root, comm, intercomm )`**
    - » collective over comm
    - » returns intercomm; remote group is clients
- Client:
  - **`MPI_Comm_connect( port_name, info, root, comm, intercomm )`**
    - » remote group is server

15

Optional Name Service

- **`MPI_Publish_name( service_name, info, port_name )`**
- **`MPI_Lookup_name( service_name, info, port_name )`**
- allow connection between **`service_name`** known to users and system-supplied **`port_name`**

16

Bootstrapping

- **`MPI_Join( fd, intercomm )`**
- collective over two processes connected by a socket.
- **`fd`** is a file descriptor for an open, quiescent socket.
- **`intercomm`** is a new intercommunicator.
- Can be used to build up full MPI communication.
- **`fd`** is *not* used for MPI communication.

17

One-Sided Operations: Issues

- Balancing efficiency and portability across a wide class of architectures
  - shared-memory multiprocessors
  - NUMA architectures
  - distributed-memory MPP's
  - Workstation networks
- Retaining "look and feel" of MPI-1
- Dealing with subtle memory behavior issues: cache coherence, sequential consistency
- Synchronization is separate from data movement.

18

Remote Memory Access Windows

`MPI_Win_create( base, size, disp_unit, info, comm, win )`

- Exposes memory given by **`(base, size)`** to RMA operations by other processes in **`comm`**.
- **`win`** is window object used in RMA operations.
- **`Disp_unit`** scales displacements:
  - 1 (no scaling) or **`sizeof(type)`**, where window is an array of elements of type **`type`**.
  - Allows use of array indices.
  - Allows heterogeneity.

19

One-Sided Communication Calls

- **`MPI_Put`** - stores into remote memory
- **`MPI_Get`** - reads from remote memory
- **`MPI_Accumulate`** - updates remote memory
- All are non-blocking: data transfer is initiated, but may continue after call returns.
- Subsequent synchronization on window is needed to ensure operations are complete.

20

Put, Get, and Accumulate

- **`MPI_Put( origin_addr, origin_count, origin_datatype, target_addr, target_count,target_datatype, window )`**
- **`MPI_Get( ... )`**
- **`MPI_Accumulate( ..., op, ... )`**
- **`op`** is as in **`MPI_Reduce`**, but no user-defined operations are allowed.

Synchronization

Multiple methods for synchronizing on window:

- **`MPI_Win_fence`** - like barrier, supports BSP model
- **`MPI_Win_{start, complete, post, wait}`** - for closer control, involves groups of processes
- **`MPI_Win_{lock, unlock}`** - provides shared-memory model.

Extended Collective Operations

- In MPI-1, collective operations are restricted to ordinary (intra) communicators.
- In MPI-2, most collective operations apply also to intercommunicators, with appropriately different semantics.
- E.g, Bcast/Reduce in the intercommunicator resulting from spawning new processes goes from/to root in spawning processes to/from the spawned processes.
- In-place extensions

23

External Interfaces

- Purpose: to ease extending MPI by layering new functionality portably and efficiently
- Aids integrated tools (debuggers, performance analyzers)
- In general, provides portable access to parts of MPI implementation internals.
- Already being used in layering I/O part of MPI on multiple MPI implementations.

24

Components of MPI External Interface Specification

- Generalized requests
  - Users can create custom non-blocking operations with an interface similar to MPI's.
  - **`MPI_Waitall`** can wait on combination of built-in and user-defined operations.
- Naming objects
  - Set/Get name on communicators, datatypes, windows.
- Adding error classes and codes
- Datatype decoding
- Specification for thread-compliant MPI

25

C++ Bindings

- C++ binding alternatives:
  - use C bindings
  - Class library (e.g., OOMPI)
  - "minimal" binding
- Chose "minimal" approach
- Most MPI functions are member functions of MPI classes:
  - example: **`MPI::COMM_WORLD.send( ... )`**
- Others are in MPI namespace
- C++ bindings for both MPI-1 and MPI-2

26

Fortran Issues

- "Fortran" now means Fortran-90.
- MPI can't take advantage of some new Fortran (-90) features, e.g., array sections.
- Some MPI features are incompatible with Fortran-90.
  - e.g., communication operations with different types for first argument, assumptions about argument copying.
- MPI-2 provides "basic" and "extended" Fortran support.

27

Fortran

- Basic support:
  - **`mpif.h`** must be valid in both fixed- and free-from format.
- Extended support:
  - mpi module
  - some new functions using parameterized types

28

Language Interoperability

- Single `MPI_Init`
- Passing MPI objects between languages
- Constant values, error handlers
- Sending in one language; receiving in another
- Addresses
- Datatypes
- Reduce operations

29

Why MPI is a Good Setting for Parallel I/O

- Writing is like sending and reading is like receiving.
- Any parallel I/O system will need:
  - collective operations
  - user-defined datatypes to describe both memory and file layout
  - communicators to separate application-level message passing from I/O-related message passing
  - non-blocking operations
- I.e., lots of MPI-like machinery

30

Informal Status Assessment

- Released July, 1997
- All MPP vendors now have MPI-1. (1.0, 1.1, or 1.2)
- Free implementations (MPICH, LAM, CHIMP) support heterogeneous workstation networks.
- MPI-2 implementations are being undertaken now by all vendors.
  - Fujitsu has a complete MPI-2 implementation
- MPI-2 is harder to implement than MPI-1 was.
- MPI-2 implementations will appear piecemeal, with I/O first.

31

Performance Issues in MPI-2

- Threads
  - Use of threads requires thread-safe code; may require expensive locks inside the implementation
  - MPI_INIT_THREAD options
- Dynamic Processes
  - Adding processes with MPI_Spawn may not provide peak performance
- One-sided
  - New ways to defer synchronization
  - Efficient implementation on non-shared memory hardware is challenging.
- Parallel I/O
  - Implementations exist...

32

Summary

- MPI-2 provides major extensions to the original message-passing model targeted by MPI-1.
- MPI-2 can deliver to libraries and applications portability across a diverse set of environments.
- Implementations are under way.
- The MPI standard documents are available at
  `http://www.mpi-forum.org`

33