Beyond MPI-1.X

Dr Graham E Fagg  
CS-594  
Spring 2003

What we are here for

- All the stuff extra stuff in the MPI-2 Spec and a few things in the MPI-1 Spec you have not thought about yet  
- MPI 2 is it really out there?  
- Do you trust your implementation?  
- Some of this is a warm up for the MetaComputing- GRID classes coming up later..

Overview

- MPI-1 what it missed out  
  - And why...  
- MPI-2 what it included  
  - Language issues  
  - Process Management  
  - Establishing Communication

What MPI-1 Missed out

- A lot of things that were hard to agree on for a standard

- Anything that might not run well on a early 90s late 80s MPP  
  - Remember who made the standard  
  - NEC, IBM, SGI, Intel, HP...
Basics of MPI-1

- **All** characteristics of message passing are contained within **communicators**
- Communicators contain:
  - Process lists or groups
  - Connection/communication structures **topologies**
  - System derived message tags/envelopes to separate messages from each other

MPI-1 Internals

**Communicators**

- All processes in an MPI-1 application belong to a global communicator called **MPI_COMM_WORLD**
- All other communicators are derived from this global communicator.
- Communication can only occur within a communicator.
- Safe communication

**Processes**

- All process groups are derived from the membership of the **MPI_COMM_WORLD** communicator.
- i.e., no external processes.
- MPI-1 process membership is **static** not dynamic like PVM or LAM 6.X
  - simplified consistency reasoning
  - fast communication (fixed addressing) even across complex topologies.
  - interfaces well to simple run-time systems as found on many MPPs.

MPI-1 Application

**MPI_COMM_WORLD**

Derived Communicator

Disadvantages of MPI-1

- Static process model
  - If a process fails, all communicators it belongs to become invalid. i.e. No fault tolerance.
  - Dynamic resources either cause applications to fail due to loss of nodes or make applications inefficient as they cannot take advantage of new nodes by starting/spawning additional processes.
  - When using a dedicated MPP MPI implementation you cannot usually use off-machine or even off-partition nodes.

**Multi-language operation is not supported**

- MPI specifies both ANSI C and F77 binding
- No agreed standard for data type conversions between languages even upon the same architecture
- Communicators cannot be passed between different language modules due to their representation
  - In F77 a communicator is an INTEGER
  - In C it is usually a pointer to a structure
  - non standard methods exist in different implementations
MPI-2

Problem areas and needed additional features identified in MPI-1 are being addressed by the MPI-2 forum.

These include:
- inter-language operation
- dynamic process control / management
- parallel I/O
- extended collective operations
- Support for inter-implementation communication/control is not being considered.
- See the MetaComputing lectures for more...

MPI-2: Language Issues

Think back to MPI-1
- What is a communicator in ‘C’
- What is a communicator in ‘Fortran’
- Does it depend on the implementation?
- How could I write a program in both ‘C’ and ‘Fortran’ and have them pass communicators?

MPI-2: Language Issues

MPI-2 Handle conversion
- Fortran to C/C++
  - C Wrappers convert Fortran handles
  - C/C++ to Fortran
  - Not supported
  - Why do you think not??
- C to C++
  - Overloading C++ operators called on the C++ side
  - C++ to C
  - Not supported
  - Want to write a new C++ compiler??

MPI-2: Language Issues

All wrappers of the form:
- MPI_<C_CLASS> MPI_<CLASS>_f2c (MPI_Fint handle)
  - All handles in fortran are integers (INT*4)

MPI-2: Language Issues

FORTRAN PROCEDURE
SUBROUTINE MPI_SEND_ONEINT (COMM, I, IERR)
INTEGER COMM, I, IERR
CALL C_ROUTINE (COMM, I, IERR)
RETURN
END

void c_routine (MPI_Fint *fh, MPI_Fint *i, MPI_Fint *ierr)
{
    MPI_Comm fcom;
    mycom = MPI_Comm_f2c (fh);
    MPI_Send (i, 1, MPI_INT, 1, 1, mycom);
}
MPI-2: Language Issues

- Added a C++ binding
  - Constants and names the same as C except when they have been made into an object or special class

C++ binding

- C++ has types C does not such as
  - MPI::COMPLEX, MPI::BOOL etc
- Uses a namespace called MPI that contains all object types such as:
  - Class comm; class Cartcomm, class Datatype.

MPI-2: Process Management

- In MPI-1
  - You get a set of process after you call MPI_Init()
  - You keep them until you call MPI_Finalize
  - Or you halt...

How do you add more nodes to an already running MPI-1 application??

MPI-2: Process Management

- How would we handle a node failure??

How could we couple two or more applications??
MPI-2: Process Management

- How could we couple two or more applications??
- Running using two or more different MPI implementations, such as one on an IBM SP and the other on a SGI/Cray??

MPI-2: Process Management

- MPI-2 provides both a Spawn (or remote start) call
- Depending on the implementation you have LAM 6.X+ does MPICH 1.3.1 does not
- Different vendor versions do
- Such as NEC/SUN

MPI-2: Process Management

- Two flavors
  - MPI_Comm_spawn ()
    - Starts new processes from a single binary and returns an intercommunicator to them
  - MPI_Comm_spawn_multiple ()
    - Starts new processes from more than one binary

MPI-2: Process Management

- MPI_Comm_spawn (other..)
MPI-2: Process Management

MPI_Init ( )

Intercommunicator

The creating processes get an intercommunicator to the child group as soon as the spawn call returns.
They can send to it before the children have called MPI_Init ( ).

Is the Intercommunicator functionality enough to allow for a full dynamic process model with efficient communications?

Hints
- What are the limits of an intercommunicator
- How are multiple child groups
Can we do this??

Can the two children groups create a direct connection? Using an MPI communicator create/merge call of some kind?

We do not have any overlapping communicators between the three groups...

Can not do collectives directly

How do we do communications between MPI applications that have not been started by each other?
MPI-2: Establishing communications

- MPI-2 has a TCP socket style abstraction
- Process can wait and accept connections from other processes
- Server - Client interface

How does the client find the server?

MPI does not mandate a nameservice but they did make an interface for one.

Server Side
- Three routines
  - MPI_Open_port (info, port_name)
  - Creates a port that other can connect to
  - MPI_Close_port (port_name)
  - MPI_Comm_accept (port_name, info, root, comm, newcomm)
    - comm is an intracommmunicator
    - newcomm is an intercommunicator

Client side
- MPI_Comm_connect (port_name, info, root, comm, newcomm)
  - This call does have a timeout dependant on the implementation
MPI-2: Establishing communications

Naming service is build on a simple third party publish and lookup nameservice

3 calls exist

MPI_Publish_name (servicename, info, port_name)
MPI_Unpublish_name (servicename, info, port_name)
MPI_Lookup_name (servicename, info, port_name)

Parent
Child 1
Child 2

How do they communicate?
MPI-2: Establishing communications

- Does this interface solve all of our problems?
- Can we create an intracommunicator between the two child groups?
- Can we now do collectives between children?
- What kind of operations/types of computing is it good for?

MPI-2: Establishing communications

- Once the communication is finished
  - MPI_Comm_disconnect()

- And lastly
  - MPI_Comm_join(fd, intercomm)

  Yes fd is a TCP socket descriptor used in cases when your applications can join via other means outside the MPI library

MPI-2: Single sided communications

- Normal message passing operation needs at least two parties
  - A sender who performs a send call
  - A receiver who performs a receive call

  Why is this? And what does it have to do with... memory management / protection?

MPI-2: Single sided communications

- Earlier Cray MPP systems allowed processes to remotely access other processes memory via shmget and shmput system function calls.
  - This is known as Remote Memory Access (RMA)

- Remote Memory Access (RMA)
  - Is fast
  - Can allow for simple program design
  - An operation specifies all the send and receive arguments together
MPI-2: Single sided communications

- Data (memory) in a fixed range (a window) is made available with a `MPI_Win_create()` call.
  - Freed with `MPI_Win_free()`
  - Data can then be accessed via
    - `MPI_Put()`
    - `MPI_Get()`
    - `MPI_Accumulate()`

MPI-2: Single sided communications

- The communication calls (put/get/accumulate) are non-blocking
  - The operation occurs sometime after the call BUT before a synchronization point

MPI-2: Single sided communications

- RMA communication is in two classes
  - Active
    - Memory is moved from one process to another
    - One process calls the move
    - Both must call the synchronization (including the owner of the target memory)
  - Like message passing

MPI-2: Single sided communications

- Passive
  - Memory is copied from a target to two other processes
  - Both processes call the copy
  - Both must synchronize (complete) their move, expect the target does not need to synchronize
  - Like shared memory

MPI-2: Single sided communications

- Semantics of an active target operation cycle are:
  - Sync on a win (RMA target window)
  - Start of an epoch
  - Perform zero or more transfer operations
    - Such as put/get/accumulate
  - End of an epoch
  - Sync on a win
  - Passive target operations require no sync operations

MPI-2: Single sided communications

- `MPI_Win_fence()`
  - A collective that can be used across the whole application to control operations so that they occur at the correct time such as exchanging data at the end of a time step
MPI-2: Single sided communications

- `MPI_Win_fence()` is a collective that allows groups to sync their memory operations.
- Think of a set of `MPI_Isends/MPI_Irecvs` followed by a `MPI_Waitall` and a `MPI_Barrier`.

For fine grain control between pairs of processes only there is a more complex interface:

- `MPI_Win_start`, `MPI_Win_complete`, `MPI_Win_post`, `MPI_Win_wait`, `MPI_Win_lock`, `MPI_Win_unlock`.
- Close to some of the Posix threads interface.

MPI-2: Intercommunicator collectives

- Collective operations require an intracommunicator.
- Sometimes it would be useful to perform a collective (such as a broadcast) across an intracommunicator.
- As in the case of when we cannot create an intracommunicator.
MPI-2: Intercommunicator collectives

- Semantics of these (MPIX*) calls
  - We need to know which is the root group and which is the leaf (target) group
  - How do we know the root group?
    - All processes in that group other than the real root call the operation with MPI_PROC_NULL as their root
    - The root calls the operation with the root set to a special constant MPI_ROOT
  - MPIX was a project at MSU that first proposed these extensions to MPI (hence MPIX)

MPI-2: MPI I/O

- Will be covered in the next lecture
  - When you get to compare it against distributed file systems

MPI-1: Profiling

- Tool writers like to tell you what is happening when running an MPI application so you can improve things
  - Three ways of doing this
    - Instrument the MPI library
    - Common way of doing it
    - Instrument your application before it links to the MPI library
    - Useful for very complex debuggers
    - Catch the runtime calls
    - Pardym (do it without the source available)

MPI-1: Profiling

- MPI-1 provides a built-in method of doing this
  - Known as the profiling interface
  - Allows instrumentation or replacing of only the required calls at run time
  - Relies on weak linking in most implementations

MPI-1

- Back to MPI-1 stuff
  - Some things you may have missed before
  - Many of them to do with using MPI under the covers
    - (like reading a book under the covers with the lights turned off)
MPI-1: Profiling

How it works
- All MPI calls have a non-profiled version called:
  - PMPI_{CALL}
- The standard MPI_{CALL} links to this PMPI_{CALL} unless a replacement MPI_{CALL} is provided.

Mylib has MPI_Send() implemented

The profile library does have full access to the real MPI library, by calling the PMPI_{CALL} calls.
- The tool writer only needs to implement the calls they require
  - Rather than 248+

An application can pass information to the profile library
- MPI_Pcontrol (level)
  - Indicated to the profile library what level of debugging you might want
  - Can also use the MPI key attributes as well.

MPICH comes with multiple profile libraries
- We are interested in the mpilog library
- Mpicc myfile.c -o myfile -mpilog -lmpe ...

G.E. Fagg UTK-CS594 Spring 2003
MPI-1: Profiling

- When executed will produce a `<exename>.clog` file
- This needs to be converted to an slog (scalable log file) before viewing etc
  - `/usr/local/mpich-1.2.1/clog2slog myfile.clog`

Then run the logview on it

- `/usr/local/mpich-1.2.1/bin/logviewer myfile.slog`
- Choose the `<MPI Process>` button and hit `<display>`

MPI-1: How do collectives work

- If we look back at the logfile there is no bcast marker
  - Or reduce, scatter, etc
  - It shows collectives as being built on point to point operations
  - When is this less efficient?

Collectives can be built using a number of different topologies, such as rings, trees, meshes etc

Some of these topologies are better than others in terms of latency or obtaining peak bi-sectional bandwidth.
MPI-1: How do collectives work

- Flat tree (or sequential broadcast)

MPI-1: How do collectives work

- Binary tree (fan out of 2)

MPI-1: How do collectives work

- Binomial tree

MPI-1: How do collectives work

- Pipeline / ring

MPI-1: How do collectives work

- What are the properties of these topologies?
  - Number of steps required to complete an operation (maximum latency)
  - How many concurrent operations?

MPI-1: How do collectives work

- Say we send a message of length X in one step down a pipeline...
MPI-1: How do collectives work

Step 1

---

MPI-1: How do collectives work

Step 2

---

MPI-1: How do collectives work

Step 3

---

MPI-1: How do collectives work

Step 4

---

MPI-1: How do collectives work

Can we do better?

Well we could segment the message into smaller parts...

End to end latency (to get the first part), would it be different?

Overall end to end bandwidth would it change?
MPI-1: How do collectives work

Step 2

What happens with a tree? Is it even better?
Comm systems have peak performances and small message sizes are usually less efficient

MPI-1: Communicators for distributed computing

If an application is spread across multiple systems but acts as a single MPI application, performance problems can occur.

MPI-1: Communicators for distributed computing

Say the collective is a ring

But the nodes are distributed...
MPI-1: Communicators for distributed computing

- But the nodes are distributed...

- What you really want...

- You can create multiple new communicators
  - One for each site
  - A new one that is made by merging the new communicators in the right order

- Works well if you know how your MPI implementation works
- Or you have to build your own collectives