

Title goes here


CS 594 Fall 2005

Message Passing, history, methods and MPI-1&2

G E Fagg




INNOVATIVE COMPUTING LABORATORY
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE




Overview

- History of message passing
- Parallel Computing and MPI
- MPI-1 & MPI-2

2/10/2005 7:17 PM G.Fagg Cluster Computing 2



2/10/2005 7:17 PM G.Fagg Cluster Computing 3




Message Passing

What is message passing?
Its where tasks or processes communicate via explicit send and receive operations on fixed items of data...

As opposed to?
Shared memory where normal read/write operations can be used to **SHARE** data.

2/10/2005 7:17 PM G.Fagg Cluster Computing 4




Message Passing

Data can be passed between processes on the same machine or between processes on different machines.

They might not even exist at the same time
I.e. no need for temporal coupling but this is unusual

2/10/2005 7:17 PM G.Fagg Cluster Computing 5




Message Passing

Why message pass?
So we can break problems into smaller pieces for faster performance (DMMP / MIMD)
For fault tolerance (multiple servers)
Explicit operations can be reasoned about in a formal way, I.e. CSP
Message Passing can be standardized to allow for highly portable applications

Was not always true
Is now happening to shared memory, see OpenMP

2/10/2005 7:17 PM G.Fagg Cluster Computing 6


Title goes here

 **Message Passing**

Many computer systems cannot share resources in a transparent way. Even we can, we can implement message passing on top of that shared resource.

This sometimes yield great performance increases
Message passing allows systems that avoid contention on resources
Compare the IBM SP-2 to the SGI Origin2000.


2/10/2005 7:17 PM G.Fagg Cluster Computing 7

 **Message Passing**

How long has it been around?
Since before networks!
Copy data (known as a message onto a removable media, such as a removable disk)
Pass the message (I.e move the disk across the room/building/country)
Read the message (mount the disk and read it)

Not that silly an example, until recently some universities in NYCity passed data via motorbike couriers on Exbyte tapes
Until ATM has finally caught up in BandWidth..

2/10/2005 7:17 PM G.Fagg Cluster Computing 8

 **Basics**

We need two or more entities
A sender
A receiver or receivers


Some data, the *message*

Some means of passing the data
network, shared resource...

The two basic operations
Send and Receive

And a means of identifying each entity, and the message... *maybe*

2/10/2005 7:17 PM G.Fagg Cluster Computing 9

 **Basics**


The Sender passed the message by:

sending it:
send (data, somewhere, args)

The receiver gets it by asking for it:
receiving the message:
receive (some buffer, some other args)

The variations will be examined in a few slide time.

2/10/2005 7:17 PM G.Fagg Cluster Computing 10


 **The history of message passing as we know it**

ARPA net users passed message via protocols such as IP

E-mail is a message passing system..

But, we are interested in the 'true' parallel computing versions..

2/10/2005 7:17 PM G.Fagg Cluster Computing 11


 **Past message passing systems**

How the facilities and functionality of modern message passing systems evolved has been influenced by a vast number of research projects and commercial implementations by vendors of MPP machines.

We will briefly discuss some of these systems in terms of which features (we now think of as standard) that they introduced and what they omitted. This will also help you understand how implementers of such systems have learned to insulate users more effectively from the increasing complexity of the underlying systems.

2/10/2005 7:17 PM G.Fagg Cluster Computing 12

Title goes here


 **Vendor Message Passing Systems and Machines**

First we will cover the hardware systems and their message passing systems that led to today's range of systems

Then we will cover some portable message passing systems

Not an exhaustive list, but long enough to show just how varied it can be.

2/10/2005 7:17 PM G.Fagg Cluster Computing 13


 **Caltech Hypercube**

The Caltech Hypercube (circa 1984) was a d-dimensional hypercube structured system with a computational node at the end of each vertex, and a single host to control the machine (known as an Intermediate Host).

The system was programmed in either C or Fortran77 and communication was based on a subroutine library known as the Crystalline Operating System (CROS).

The communications library assigned addresses to tasks depending on which node they were physically located, processes could only communicate to neighbors or the intermediate host (a total of d-1 links). The CROS terminology for a link between two nodes was that of a channel through which 8 byte message packets could be sent.

2/10/2005 7:17 PM G.Fagg Cluster Computing 14

 **Caltech Hypercube**


The system only supported collective operations (broadcast) to/from the intermediate host and the overall communication pattern was SIMD in nature.

I.e. all processes had to call the same communications routine at the same time. This lead the machine to appear halfway between the earlier SIMD machines such as the ICL DAP and Thinking Machines CM-1/2 and the later Intel iPSC and XPS Paragon machines. The former where program execution and communication occurs fully in lockstep and the latter where ordering was completely independent.

On the Hypercube under CROS, the program was free to run independently to each other but the hardware forced all the communication into lockstep.

For solving very regular problems in physics such as partial derivatives for large numbers of grid points, the structure imposed by the programming environment was an aid for producing low-level very efficient implementations.

2/10/2005 7:17 PM G.Fagg Cluster Computing 15


 **Caltech Hypercube**

Went from a fixed broadcast (I.e. no addresses specified in the send operation) to an addressed based system.

Making everybody do a send and then receive the same time... not nice.

Note the small message size of 8 bytes
If you wanted more, you had to write your own message passing layer... nasty.

2/10/2005 7:17 PM G.Fagg Cluster Computing 16

 **Meiko CS-1 and Occam**


Transputer based multiprocessor

The transputer was a 32 bit microprocessor that had communication hardware built in.

4 high speed links

The transputer could context switch in a single cycle
I.e. multitasking-multithreading very quickly
more than one process per processor

2/10/2005 7:17 PM G.Fagg Cluster Computing 17

 **Meiko CS-1 and Occam**


Occam was a language based on the CSP specification language

CSP - Communicating Sequential Processes
CSP could be formally reasoned about
Popular target for S/W Eng projects in the early days of ESPRIT.
Many CS-1 machines where supplied under the ESPRIT ALPHA Project

Occam was a parallel language

2/10/2005 7:17 PM G.Fagg Cluster Computing 18

Title goes here

 **Occam**

Instructions were executed in order of blocks


blocks could be executed internally as parallel (par), sequential (seq) or alternately (alt) which meant non-deterministically (but only one)

Seq

```
do A
do B
```

A happens before B always

2/10/2005 7:17 PM G.Fagg Cluster Computing 19

 **Occam**

Par

```
do A
do B
```


I.e. they execute together or after one another

alt

```
do A
do B
```

We don't know which one executes in which order.

2/10/2005 7:17 PM G.Fagg Cluster Computing 20

 **Occam and message passing**

Message passing was built in

Based on channels between processes

a channel is a bi-direction pipe much like unix sockets


Sending a message

```
chan1 ! Data
```

Receiving a message

```
chan1 ? buffer
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 21

 **Occam and message passing**


Message passing was synchronous and blocking

I.e. both parties had to work together to message pass, and the send and receiver would wait until their operation had completed completely before the program would continue

Was easy for programmers to write code that deadlocked

lucky we had lots of ESPRIT analysis tools to help

2/10/2005 7:17 PM G.Fagg Cluster Computing 22

 **Occam and message passing**


Deadlocking code

```
Proc1
chan1 ! Data
chan1 ? Buffer

Proc 2
chan1 ! Data
chan1 ? Buffer
```

This problem happens in many systems without the MP system buffering for you.

2/10/2005 7:17 PM G.Fagg Cluster Computing 23

 **Occam lives on**

The Occam project still lives on where formally proved code is needed.

Newer versions allow recursion which was only possible by loop contracts previously (see Occam 2 1/2)


Compilers are available for systems other than transputers such as Sparc, PowerPC etc

Kent retargetable Occam compiler project (Welsh96) from the University of Kent at Canterbury.

SPOC - The Southampton Portable Occam Compiler from ECS, University of Southampton.

2/10/2005 7:17 PM G.Fagg Cluster Computing 24

Title goes here

 **NX and the Intel iPSC1 and the Intel Paragon**


The original iPSC1 (circa 86) was a seven-dimensional hypercube structured system much like the Caltech Hypercube in terms of hardware design although its software environment known as NX1 was less like CROS and more like the `` Distributed Processes environment.

iPSC? The Intel Personal Super Computer for those who could afford a personal supercomputer....
The comp.parallel Usenet news page was originally aimed at iPSC users.
One of the largest Paragons XPS machines ever installed is at ORNL

The system was very well balanced, i.e. its computational power was proportional to its message passing performance.

Far quicker than even more modern systems like IBM SP2s (like the new ASCI Blue Pacific machine) or even many Cray T3D/E machines.

2/10/2005 7:17 PM G.Fagg Cluster Computing 25


 **NX and the Intel iPSC1 and the Intel Paragon**

The NX1 operating system was based on the Caltech Reactive Kernel which provided hiding of the underlying communication topology (processes were identified by a simple integer from 0 to P-1, where P was the number of processes per partition), multiple processes per node, and any to any message passing, non-synchronous messaging.

i.e. both sender and receiver do not have to be active at the same time for communication to complete) and non-blocking (i.e. no need to wait for completion)

what is now thought of as a typical set of features that define a message passing environment.

2/10/2005 7:17 PM G.Fagg Cluster Computing 26

 **NX and the Intel iPSC1 and the Intel Paragon**

This additional flexibility also added an increase in the complexity of the message passing library. Users now needed additional routines to inquire location information, and messages needed to be addressed correctly

Additional arguments in the subroutine calls.

Messages needed to be identified on an individual bases as there was neither a fixed order of transmission and therefore receipt, but the pattern of communications was no longer dictated by topology.


To assist developers, messages could be tagged or typed.

Like a subject like in E-mail!

a user assigned integer could be associated with a message which would be used by the receiver to distinguish messages.

Unfortunately the original system did not permit the filtering of messages by sender identity and type at the same time, although the type could be set to the senders ID to allow for receive from sender semantics.

2/10/2005 7:17 PM G.Fagg Cluster Computing 27

 **NX and the Intel iPSC1 and the Intel Paragon**


An additional new feature was that different length messages could be received prior to the receiving process knowing which was which and hence knowing how big a buffer memory to allocate, or even how much data was received.

The user had to indicate for each message how large the receive buffer was. Only after the receive completed could the user find out how much data was received, up to the maximum allowed of 16Kbytes

a vast improvement over the Caltechs Hypercube 8 bytes

If the user offered a buffer that was too small the the excess message data was discarded (truncated).

2/10/2005 7:17 PM G.Fagg Cluster Computing 28


 **NX and the Intel iPSC1 and the Intel Paragon**

The later Intel machines implemented an improved version of NX, known as NX2. In the case of the Paragon, this was implemented upon an OSF/1 micro Unix kernel.

A number of improvements were added such as interrupt driven communication which allowed an application to perform computation and be woken up when a message arrived instead of having to poll for them intermittently (leading to decreased cache performance and possible page faults as the OS calls are invoked to check for messages).

i.e. Asynchronous and non-blocking messaging
Be careful many users call non-blocking asynchronous and vice versa

2/10/2005 7:17 PM G.Fagg Cluster Computing 29

 **NX and the Intel iPSC1 and the Intel Paragon**


Other changes included the inclusion of message identifiers (mids) that allowed simple identification of non-blocking operations. When a non blocking operation is initiated, a mid would be issued and the user could check for completion of this mid later, thus allowing the underlying hardware communications processors to overlap communication while the compute processors continued. Semantic changes included allowing the use of wild cards (typically negative integers) to denoted groups or processes.

receiving a message of type -1 would denote receive from anybody of any type, i.e. whatever was received next.

Sending to a node of type -1 would denote sending to all processes on a partition.


2/10/2005 7:17 PM G.Fagg Cluster Computing 30

Title goes here

 **NX and the Intel iPSC1 and the Intel Paragon**

Up to this point communication had been point to point, i.e. from a single sender to a single receiver. New broadcast functions allowed the construction of global operations such as global synchronizations (barriers) and some arithmetic reduction operations.

2/10/2005 7:17 PM G.Fagg Cluster Computing 31


 **NX and the Intel iPSC1 and the Intel Paragon**

Although NXs design inspired many of the features of current message passing systems, it also had a number of shortcomings:

- such as lack of more comprehensive group communication functions (to assist certain types of calculations)
- lack of message identification and filtering at the receivers end
- only one type compared to up to three used on later systems, and
- initially a high software overhead compared to simpler protocols such as active messages which had direct access to hardware.

But, it did give us the semantics of the most common message passing systems in current use

2/10/2005 7:17 PM G.Fagg Cluster Computing 32

 **IBM Scalable Power series and EUI**

The IBM Scalable Power (SP) Series of systems starting with the IBM 9076 SP1 and the later SP2 machines consisted of tightly coupled distributed memory set of RS/6000 RISC processors interconnected by a high-speed switch.


The systems were based on experience gained from the Vulcan hardware project and the Viper operating environment.

The system designed to program these systems were known as the IBM external user interface (EUI) and consisted of four main components:

- task management, message passing (point-to-point), task groups and collective operations.

The last two items being of particular importance to later systems such as MPI.

2/10/2005 7:17 PM G.Fagg Cluster Computing 33

 **IBM Scalable Power series and EUI**

Point to point communication under EUI was performed by sending messages to tasks directly in the same style as on the Intel iPSC systems


- addresses from 0 to N-1, where N was the number of tasks making up a parallel job.

The point to point system supported typed messages for both blocking and non-blocking messages.

Unlike NX, messages could be selected by the receiver on both message type and source (sender) including the use of wild cards.

To assist in handling non-blocking messages the user could check the status of a particular transfer as well as wait for completion of a named operation, any of a range of operations or all pending outstanding transfers.

2/10/2005 7:17 PM G.Fagg Cluster Computing 34

 **IBM Scalable Power series and EUI**

EUI allowed the construction of conceptual collections of tasks into logical groups that could be addressed by a single group ID.

See PVM groups latter.

This allowed users to avoid having to list (sometimes large numbers of) tasks explicitly when passing messages in structured ways repeatedly.


The use of collective operations on these groups avoided the use of many point to point calls and allowed the system to perform these as efficiently as possible on the given hardware.

The range of operations included, barriers, data shifts, broadcasts, gather, scatter, a generalized combine and an associative reduction.

All the collective operations required all members of each group to partake in a blocking fashion.

The term 'collective' used in MPI comes from the IBM system.

2/10/2005 7:17 PM G.Fagg Cluster Computing 35

 **IBM Scalable Power series and EUI**

Asynchronous returns from non-blocking functions required a two part status lookup. If the user interface to a non-blocking receive was as follows:

```
void recvf (&data, sizeofbuffer, &size-received)
```


The size-received variable could not be filled in by the system until the non-blocking operation had completed, which might be while the thread that made this call was in a different program module. Thus the need to get a status handle which could be queried after the operation had completed and who's memory storage was handled by the messaging system.

```
recvf (&data, ... &status)
...
wait (for above recv to complete)
```

<- status is now safe to examine, as the wait operation has completed any status data structures.

2/10/2005 7:17 PM G.Fagg Cluster Computing 36


Title goes here

 **IBM Scalable Power series and EUI**

The EUI project was not the first to introduce this two step strategy, this also occurred between NX1 and NX2, but the overall design used by IBM was the bases of that used by the MPI forum.

Also much of the EUI collective operations design also directly effected the design of MPI.

2/10/2005 7:17 PM G.Fagg Cluster Computing 37


 **Thinking Machines CM5 and the CMMD Active Message Layer (AML)**

The CMMD message passing system was unique in that it offered users access to routines from the lowest level, Active Message Layer (AML), point to point, channels and a cooperative functions library.

The AML system was the lowest level of operation and manipulated the communications hardware directly. The layer provided three basic operations:

- active messages, which are like a lightweight RPC call. This is where the sender sends the address of a function to be invoked at a remote node together with its arguments in a single 72 byte packet.
- The second operation was that of a data transport mechanism which only wrote data into a remote memory at a set location (much like Crays shmput operations).
- The final operation was a receive port data structure which assisted in handling multiple data packets.

2/10/2005 7:17 PM G.Fagg Cluster Computing 38

 **Thinking Machines CM5 and the CMMD Active Message Layer (AML)**

The CMMD point to point library was built on top of the AML, and provided the common list of operations including:

- blocking, non-blocking, synchronous and asynchronous point to point operations with selection upon source, message type or wild cards.

Interestingly the blocking calls were quicker than the non-blocking calls as they avoided system level copying of message data.


Another interesting point was the inclusion of a "send and then receive" operation.

This allowed for simpler coding of stencil operations and boundary exchanges in domain decomposition problems.

The system implementation of the joint send and receive operation being quicker than the two separate operations.

The send-receive operation was so useful that it was also provided for in MPI.

2/10/2005 7:17 PM G.Fagg Cluster Computing 39

 **Thinking Machines CM5 and the CMMD Active Message Layer (AML)**

For example previously to exchange values two operations would be required:


```
Task A      Task B
Send (B, data) make copy of data edge into data'
Recv (B, data) Recv (A, data)
Send (A, data')
```

As opposed to:

```
Task A      CMMD_send_and_receive (B, data)
Task B      CMMD_send_and_receive (A, data)
```

Note: the two operations version is made more complex by the need to copy data values to prevent them being over written before they are copied into the message layer, a common complication found in many wavefront calculations.

2/10/2005 7:17 PM G.Fagg Cluster Computing 40

 **So what did we get?**

From a very restrictive system (Caltech Hypercube) to systems with multiple ways of sending a message that is addressed in a flexible way and tagged.

Receivers have multiple ways of filtering messages (using addresses, tags, channels) and can start a receive and get back to it when finished.

2/10/2005 7:17 PM G.Fagg Cluster Computing 41

 **m4 and p4 macros**

P4 was a system, that grow out of a set of fortran macros that was developed at Argonne National Laboratory (ANL) for use on a HEP shared memory computer.


The original Macro's were called MonMac's that were processed at compile time by the Unix m4 preprocessing utility, and offered the user a set of monitor functions used to provide locks on critical sections of code that accessed shared data.

The use of macros avoided an additional set of stack operations if the monitors had be based on function calls.

The authors of the system co-wrote the book "Parallel Programs for Parallel Processors" which give the system its final name of p4.

2/10/2005 7:17 PM G.Fagg Cluster Computing 42

Title goes here

 **m4 and p4 macros**


The system was used as a bases for several specialized versions

- TCGMSG for Chemistry problems
- GMD macros for solving problems on regular grids
- GMD macros was the basis for the very successful PARMACS system developed by Rolf Hempel at GMD in Germany.

Used originally on the Suprenum machines.

He was at NEC, Germany. And NEC has one of the fastest MPI implementations in existence for their SX series computers.

2/10/2005 7:17 PM G.Fagg Cluster Computing 43

 **m4 and p4 macros**


The final p4 system was based on procedure calls and supported C as well as Fortran on a very wide range of systems including both distributed memory as well as shared memory systems.

The programming paradigm was that of procs (processes) that formed administrative clusters, that intercommunicated by either locks or explicit message passing.

The system provided user access to buffers so that they could avoid additional buffering by the system if they were knowledgeable enough.

There was no non-locally blocking or asynchronous calls. I.e. the calls returned when the data was sent, and the user did not have to probe, test or wait for completion before reusing buffers.

2/10/2005 7:17 PM G.Fagg Cluster Computing 44

 **m4 and p4 macros**


One globally blocking point to point call was also included `p4sendr()` which waited for an explicit acknowledgment from the receiver before returning, hence the `r` on the end of the call name to signify a rendezvous.

The system also included a wide range of collective operations, with the ability to use the `p4_global_op()` call to construct user defined operations.

Although the p4 system was very efficient, it was not as popular as other message passing system and was later used as a layering scheme to support other projects such as:

- Chameleon {Gropp93}
- BlockComm
- MPI in the form of MPICH {Gropp94,Gropp96}.

2/10/2005 7:17 PM G.Fagg Cluster Computing 45

 **What have we got this far?**


Non-vendor versions are more portable

Some are more efficient than others
some are almost as fast as the vendor systems

The number of features has increased

- group communications better supported
- first portable high level libraries appearing
- layering approach appearing
- MPI based on the CH ADI on top of p4.... Or even PVM

2/10/2005 7:17 PM G.Fagg Cluster Computing 46


 **Whats next ?**

MPI standard and API
You will need to look at those handouts!

What we will cover

- Safe communications – communicators, groups
- Message passing semantics – blocking, non blocking, local, global operations
- Collectives, Buffering and data types

2/10/2005 7:17 PM G.Fagg Cluster Computing 47



2/10/2005 7:17 PM G.Fagg Cluster Computing 48

Title goes here

Message Passing



INNOVATIVE COMPUTING LABORATORY
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE

Notes

This talk is a combination of lots of different material from a host of sources including:

- David Cronk & David Walker
- EPCC
- NCSA
- FT-MPI, Open MPI, LAM and MPICH teams

2/10/2005 7:17 PM G.Fagg Cluster Computing 50

Introduction to MPI

What is MPI?

- MPI stands for "Message Passing Interface"
- In ancient times (late 1980's early 1990's) each vendor had its own message passing library
- Non-portable code
- Not enough people doing parallel computing due to lack of standards

2/10/2005 7:17 PM G.Fagg Cluster Computing 51

What is MPI?

- April 1992 was the beginning of the MPI forum
- Organized at SC92
- Consisted of hardware vendors, software vendors, academicians, and end users
- Held 2 day meetings every 6 weeks
- Created drafts of the MPI standard
- This standard was to include all the functionality believed to be needed to make the message passing model a success
- Final version released may, 1994

2/10/2005 7:17 PM G.Fagg Cluster Computing 52

What is MPI?

- A standard library specification!
- Defines syntax and semantics of an extended message passing model
- It is not a language or compiler specification
- It is not a specific implementation
- It does not give implementation specifics**
- Hints are offered, but implementers are free to do things however they want
- Different implementations may do the same thing in a very different manner
- <http://www.mpi-forum.org>

2/10/2005 7:17 PM G.Fagg Cluster Computing 53

What is MPI?

- A library specification designed to support parallel computing in a distributed memory environment
- Routines for cooperative message passing
 - There is a sender and a receiver
 - Point-to-point communication
 - Collective communication
- Routines for synchronization
- Derived data types for non-contiguous data access patterns
- Ability to create sub-sets of processors
- Ability to create process topologies


2/10/2005 7:17 PM G.Fagg Cluster Computing 54

Title goes here

 **What is MPI?**


Continuing to grow!
New routines have been added to replace old routines
New functionality has been added
Dynamic process management
One sided communication
Parallel I/O

2/10/2005 7:17 PM G.Fagg Cluster Computing 55

 **Getting Started with MPI**


Outline
Introduction
6 basic functions
Basic program structure
Groups and communicators
A very simple program
Message passing
A simple message passing example
Types of programs
Traditional
Master/Slave
Examples
Unsafe communication

2/10/2005 7:17 PM G.Fagg Cluster Computing 56

 **Getting Started with MPI**


MPI contains 128 routines (more with the extensions)!
Many programs can be written with just 6 MPI routines!
Upon startup, all processes can be identified by their *rank*, which goes from 0 to N-1 where there are N processes

2/10/2005 7:17 PM G.Fagg Cluster Computing 57

 **6 Basic Functions**

MPI_INIT: Initialize MPI
MPI_Finalize: Finalize MPI
MPI_COMM_SIZE: How many processes are running?
MPI_COMM_RANK: What is my process number?
MPI_SEND: Send a message
MPI_RECV: Receive a message

2/10/2005 7:17 PM G.Fagg Cluster Computing 58

 **MPI_INIT (ierr)**

ierr: Integer error return value. 0 on success, non-zero on failure.
This **MUST** be the first MPI routine called in any program.
Except for MPI_Initialized () can be called to check if MPI_Init has been called!!
Can only be called once
Sets up the environment to enable message passing

2/10/2005 7:17 PM G.Fagg Cluster Computing 59

 **MPI_FINALIZE (ierr)**

ierr: Integer error return value. 0 on success, non-zero on failure.
This routine must be called by each process before it exits
This call cleans up all MPI state
No other MPI routines may be called after MPI_FINALIZE
All pending communication must be completed (locally) before a call to MPI_FINALIZE

2/10/2005 7:17 PM G.Fagg Cluster Computing 60

Title goes here

Basic Program Structure

```
program main                               #include "mpi.h"
include 'mpi.h'
integer ierr

call MPI_INIT (ierr)                       int main ()
.....                                     {
Do some work                               MPI_Init ()
.....                                     .....
call MPI_FINALIZE (ierr)                   Do some work
Maybe do some additional                  .....
Local computation                         MPI_Finalize ()
.....                                     Maybe do some additional
end                                         Local computation
.....                                     .....
}

2/10/2005 7:17 PM G.Fagg Cluster Computing 61
```

Groups and communicators

Communicators are containers that hold messages and groups of processes together with additional meta-data

All messages are passed only within communicators

Upon startup, there is a single set of processes associated with the communicator `MPI_COMM_WORLD`

Groups can be created which are sub-sets of this original group, also associated with communicators

2/10/2005 7:17 PM G.Fagg Cluster Computing 62

Groups and communicators

Why do communicators exist

To keep different message passing libraries from interfering with each other

Allows the building of multiple layers of message passing code

2/10/2005 7:17 PM G.Fagg Cluster Computing 63

Groups and communicators

2/10/2005 7:17 PM G.Fagg Cluster Computing 64

Groups and communicators

Nothing to stop message passing to the wrong layer.....


2/10/2005 7:17 PM G.Fagg Cluster Computing 65

Groups and communicators

Library used by application that uses Message passing

2/10/2005 7:17 PM G.Fagg Cluster Computing 66


Title goes here

 **MPI_COMM_RANK (comm, rank, ierr)**

comm: Integer communicator.
rank: Returned rank of calling process
ierr: Integer error return code

This routine returns the relative rank of the calling process, within the group associated with comm.


2/10/2005 7:17 PM G.Fagg Cluster Computing 67

 **MPI_COMM_SIZE (comm, size, ierr)**

Comm: Integer communicator identifier
Size: Upon return, the number of processes in the group associated with comm. For our purposes, always the total number of processes

This routine returns the number of processes in the group associated with comm


2/10/2005 7:17 PM G.Fagg Cluster Computing 68

 **A Very Simple Program
Hello World**

```
program main
include 'mpi.h'
integer ierr, size, rank


call MPI_INIT (ierr)
call MPI_COMM_RANK (MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, size, ierr)
print *, 'Hello World from process', rank, 'of', size
call MPI_FINALIZE (ierr)
end
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 69

 **Hello World**


> mpirun -np 4 a.out	> mpirun -np 4 a.out
>	>
> Hello World from 2 of 4	> Hello World from 3 of 4
> Hello World from 0 of 4	> Hello World from 1 of 4
> Hello World from 3 of 4	> Hello World from 2 of 4
> Hello World from 1 of 4	> Hello World from 0 of 4

2/10/2005 7:17 PM G.Fagg Cluster Computing 70

 **Message Passing**


Message passing is the transfer of data from one process to another
This transfer requires cooperation of the sender and the receiver, but is initiated by the sender
There must be a way to "describe" the data
There must be a way to identify specific processes
There must be a way to identify messages

2/10/2005 7:17 PM G.Fagg Cluster Computing 71

 **Message Passing**


Data is described by a triple
Address: Where is the data stored
Count: How many elements make up the message
Datatype: What is the type of the data
Basic types (integers, reals, etc)
Derived types (good for non-contiguous data access)

2/10/2005 7:17 PM G.Fagg Cluster Computing 72

 **Message Passing**

Processes are specified by a double
Communicator: safe space to pass message
Rank: The relative rank of the specified process within the group associated with the communicator
Messages are identified by a single tag
This can be used to differentiate between different types of messages
Max tag can be looked up but must be atleast 32k

2/10/2005 7:17 PM G.Fagg Cluster Computing 73

 **MPI_SEND(buf, cnt, dtype, dest, tag, comm, ierr)**


buf: The address of the beginning of the data to be sent
cnt: The number of elements to be sent
dtype: datatype of each element
dest: The rank of the destination
tag: The message tag
comm: The communicator

2/10/2005 7:17 PM G.Fagg Cluster Computing 74

 **MPI_RECV**


Once this routine returns, the message has been copied out of the user buffer and the buffer can be reused
This may require the use of system buffers. If there are insufficient system buffers, this routine will block until a corresponding receive call has been posted
Completion of this routine indicates nothing about the designated receiver

2/10/2005 7:17 PM G.Fagg Cluster Computing 75

 **MPI_RECV(buf, cnt, dtype, source, tag, comm, status, ierr)**


buf: Starting address of receive buffer
cnt: Max number of elements to receive
dtype: Datatype of each element
source: Rank of sender (may use MPI_ANY_SOURCE)
tag: The message tag (may use MPI_ANY_TAG)
comm: Communicator
status: Status information on the received message

2/10/2005 7:17 PM G.Fagg Cluster Computing 76

 **MPI_RECV**

When this call returns, the data has been copied into the user buffer
Receiving fewer than *cnt* elements is ok, but receiving more is an error
Status is a structure in C (MPI_Status) and an array in Fortran (integer status(MPI_STATUS_SIZE))

2/10/2005 7:17 PM G.Fagg Cluster Computing 77

 **MPI_STATUS**

The status parameter is used to retrieve information about a completed receive
In C, status is a structure consisting of at least 3 fields: MPI_SOURCE, MPI_TAG, MPI_ERROR
status.MPI_SOURCE, status.MPI_TAG, and status.MPI_ERROR contain the source, tag, and error code, respectively
In Fortran, status must be an integer array of size MPI_STATUS_SIZE
status(MPI_SOURCE), status(MPI_TAG), and status(MPI_ERROR) contain the source, tag, and error code

2/10/2005 7:17 PM G.Fagg Cluster Computing 78

Send/Recv Example

```

program main
include 'mpi.h'
CHARACTER*20 msg
integer ierr, rank, tag, status (MPI_STATUS_SIZE)

tag = 99
call MPI_INIT (ierr)
call MPI_COMM_RANK (MPI_COMM_WORLD, rank, ierr)
if (myrank .eq. 0) then
msg = "Hello there"
call MPI_SEND (msg, 11, MPI_CHARACTER, 1, tag, &
MPI_COMM_WORLD, ierr)
else if (myrank .eq. 1) then
call MPI_RECV(msg, 20, MPI_CHARACTER, 0, tag, &
MPI_COMM_WORLD, status, ierr)
endif
call MPI_FINALIZE (ierr)
end

```

2/10/2005 7:17 PM G.Fagg Cluster Computing 79

Types of MPI Programs

Traditional
Break the problem up into about even sized parts and distribute across all processors
What if problem is such that you cannot tell how much work must be done on each part?

Master/Slave
Break the problem up into many more parts than there are processors
Master sends work to slaves
Parts may be all the same size or the size may vary

2/10/2005 7:17 PM G.Fagg Cluster Computing 80

Traditional Example

Compute the sum of a large array of N integers

```

Comm = MPI_COMM_WORLD
DO (l = 1, npes-1)
Call MPI_COMM_RANK (comm, rank)
Call MPI_COMM_SIZE (comm, npes)
Stride = N/npes
Start = (stride * rank) + 1
Sum = 0
DO (l = start, start+stride)
sum = sum + array(l)
ENDDO
IF (rank .eq. 0) then
DO (l = 1, npes-1)
call MPI_RECV(tmp, 1, MPI_INTEGER,
& l, 2, comm, status)
sum = sum + tmp
ENDDO
ELSE
MPI_SEND (sum, 1, MPI_INTEGER, &
& 0, 2 comm)
ENDIF

```

2/10/2005 7:17 PM G.Fagg Cluster Computing 81

Unsafe Communication Patterns

Process 0 and process 1 must exchange data
Process 0 sends data to process 1 and then receives data from process 1
Process 1 sends data to process 0 and then receives data from process 0
If there is not enough system buffer space for either message, this will deadlock
Any communication pattern that relies on system buffers is unsafe
Any pattern that includes a cycle of blocking sends is unsafe

2/10/2005 7:17 PM G.Fagg Cluster Computing 82

Unsafe Communication Patterns


2/10/2005 7:17 PM G.Fagg Cluster Computing 83

Communication Modes

Outline


- Standard mode
 - Blocking
 - Non-blocking
- Non-standard mode
 - Buffered
 - Synchronous
 - Ready
- Performance issues

2/10/2005 7:17 PM G.Fagg Cluster Computing 84

 **Point-to-Point Communication Modes**


Standard Mode:
blocking:
MPI_SEND (buf, count, datatype, dest, tag, comm)
MPI_RECV (buf, count, datatype, source, tag, comm, status)
Generally **ONLY** use if you cannot call earlier **AND** there is no other work that can be done!
Standard **ONLY** states that buffers can be used once calls return. It is implementation dependant on when blocking calls return.
Blocking sends **MAY** block until a matching receive is posted. This is not required behavior, but the standard does not prohibit this behavior either. Further, a blocking send may have to wait for system resources such as system managed message buffers.
Be VERY careful of deadlock when using blocking calls!

2/10/2005 7:17 PM G.Fagg Cluster Computing 85

 **Point-to-Point Communication Modes (cont)**


Standard Mode:
Non-blocking (immediate) sends/receives:
MPI_ISEND (buf, count, datatype, dest, tag, comm, request)
MPI_IRECV (buf, count, datatype, source, tag, comm, request)
MPI_WAIT (request, status)
MPI_TEST (request, flag, status)
Allows communication calls to be posted early, which may improve performance.
Overlap computation and communication
Latency tolerance
Less (or no) buffering
*** MUST either complete these calls (with wait or test) or call MPI_REQUEST_FREE**

2/10/2005 7:17 PM G.Fagg Cluster Computing 86

 **MPI_ISEND (buf, cnt, dtype, dest, tag, comm, request)**


Same syntax as MPI_SEND with the addition of a request handle
Request is a handle (int in Fortran) used to check for completeness of the send
This call returns immediately
Data in buf may not be accessed until the user has completed the send operation
The send is completed by a successful call to MPI_TEST or a call to MPI_WAIT

2/10/2005 7:17 PM G.Fagg Cluster Computing 87

 **MPI_IRECV(buf, cnt, dtype, source, tag, comm, request)**

Same syntax as MPI_RECV except status is replaced with a request handle
Request is a handle (int in Fortran) used to check for completeness of the rcv
This call returns immediately
Data in buf may not be accessed until the user has completed the receive operation
The receive is completed by a successful call to MPI_TEST or a call to MPI_WAIT

2/10/2005 7:17 PM G.Fagg Cluster Computing 88

 **MPI_WAIT (request, status)**

Request is the handle returned by the non-blocking send or receive call
Upon return, status holds source, tag, and error code information
This call does not return until the non-blocking call referenced by *request* has completed
Upon return, the request handle is freed
If *request* was returned by a call to MPI_ISEND, return of this call indicates nothing about the destination process

2/10/2005 7:17 PM G.Fagg Cluster Computing 89

 **MPI_TEST (request, flag, status)**

Request is a handle returned by a non-blocking send or receive call
Upon return, *flag* will have been set to true if the associated non-blocking call has completed. Otherwise it is set to false
If *flag* returns true, the request handle is freed and *status* contains source, tag, and error code information
If *request* was returned by a call to MPI_ISEND, return with flag set to true indicates nothing about the destination process

2/10/2005 7:17 PM G.Fagg Cluster Computing 90

Title goes here

Non-blocking Communication

```

100 continue
if (err != Delta) goto 200
do some computation
do (l = 0, npes)
  if (l != myrank)
    set up data to send
    call MPI_SEND (data, cnt, dtype, &
      l, tag, comm, ierr)
  endif
enddo
do (l = 0, npes)
  if (l != myrank)
    set up data to recv
    call MPI_RECV (data, cnt, dtype, &
      l, tag, comm, status, ierr)
  endif
enddo
goto 100

```

Clearly unsafe

```

100 continue
if (err != Delta) goto 200
do some computation
do (l = 0, npes)
  if (l != myrank)
    set up data to send
    call MPI_ISEND (data, cnt, dtype, &
      l, tag, comm, request, ierr)
  endif
enddo
do (l = 0, npes)
  if (l != myrank)
    set up data to recv
    call MPI_IRECV (data, cnt, dtype, &
      l, tag, comm, status, ierr)
  endif
enddo
goto 100

```

May run out of handles

2/10/2005 7:17 PM G.Fagg Cluster Computing 91

Non-blocking communication

```

100 continue
.....
do (l = 0, npes)
  if (l != myrank)
    set up data to send
    call MPI_ISEND (data, cnt, dtype, &
      l, tag, comm, request, ierr)
  endif
enddo
do (l = 0, npes)
  if (l != myrank)
    set up data to recv
    call MPI_IRECV (data, cnt, dtype, &
      l, tag, comm, request, ierr)
  endif
enddo
wait for all isend requests and irecv requests
.....

```

Safe, and pretty good

2/10/2005 7:17 PM G.Fagg Cluster Computing 92

Point-to-Point Communication Modes (cont)

Non-standard mode communication

- Only used by the sender! (MPI uses the push communication model)
- Buffered mode - A buffer must be provided by the application
- Synchronous mode - Completes only after a matching receive has been posted
- Ready mode - May only be called when a matching receive has already been posted

2/10/2005 7:17 PM G.Fagg Cluster Computing 93

Point-to-Point Communication Modes: Buffered

```

MPI_BSEND (buf, count, datatype, dest, tag, comm)
MPI_IBSEND (buf, count, dtype, dest, tag, comm, req)
MPI_BUFFER_ATTACH (buff, size)
MPI_BUFFER_DETACH (buff, size)

```

Buffered sends do not rely on system buffers

- The user supplies a buffer that **MUST** be large enough for all messages
- User need not worry about calls blocking, waiting for system buffer space
- The buffer is managed by MPI
- The user **MUST** ensure there is no buffer overflow

2/10/2005 7:17 PM G.Fagg Cluster Computing 94

Buffered Sends

```

#define BUFFSIZE 1000
char *buff;
char b1[500], b2[500];
MPI_Buffer_attach (buff, BUFFSIZE);

```

Seg violation

```

#define BUFFSIZE 1000
char *buff;
char b1[500], b2[500];
buff = (char*) malloc(BUFFSIZE);
MPI_Buffer_attach(buff, BUFFSIZE);
MPI_Bsend(b1, 500, MPI_CHAR, 1, 1,
  MPI_COMM_WORLD);
MPI_Bsend(b2, 500, MPI_CHAR, 2, 1,
  MPI_COMM_WORLD);

```

Buffer overflow

```

int size;
char *buff;
char b1[500], b2[500];
MPI_Pack_size (500, MPI_CHAR,
  MPI_COMM_WORLD, &size);
size += MPI_BSEND_OVERHEAD;
buff = (char*) malloc(size);
MPI_Buffer_attach(buff, size);
MPI_Bsend(b1, 500, MPI_CHAR, 1, 1,
  MPI_COMM_WORLD);
MPI_Bsend(b2, 500, MPI_CHAR, 2, 1,
  MPI_COMM_WORLD);
MPI_Buffer_detach (&buff, &size);

```

Safe

2/10/2005 7:17 PM G.Fagg Cluster Computing 95

Point-to-Point Communication Modes: Synchronous

```

MPI_SEND (buf, count, datatype, dest, tag, comm)
MPI_ISEND (buf, count, dtype, dest, tag, comm, req)

```

Can be started (called) at any time.

Does not complete until a matching receive has been posted and the receive operation has been started

- * Does NOT mean the matching receive has completed
- Can be used in place of sending and receiving acknowledgements
- Can be more efficient when used appropriately
- buffering may be avoided

2/10/2005 7:17 PM G.Fagg Cluster Computing 96

Title goes here

Point-to-Point Communication Modes: Ready Mode

`MPI_RSEND (buf, count, datatype, dest, tag, comm)`
`MPI_IRSEND (buf, count, dtype, dest, tag, comm, req)`

- May **ONLY** be started (called) if a matching receive has already been posted.
- If a matching receive has not been posted, the results are undefined
- May be most efficient when appropriate
- Removal of handshake operation

Should only be used with **extreme** caution
Only really faster on a Paragon!

2/10/2005 7:17 PM G.Fagg Cluster Computing 97

Ready Mode

MASTER

```
while (!done) {
  MPI_Recv (NULL, 0, MPI_INT, MPI_ANY_SOURCE,
            1, MPI_COMM_WORLD, &status);
  source = status.MPI_SOURCE;
  get_work (...);
  MPI_Rsend (buff, count, datatype, source, 2,
            MPI_COMM_WORLD);
  if (no more work) done = TRUE;
}
```

SLAVE

UNSAFE

```
while (!done) {
  MPI_Send (NULL, 0, MPI_INT, MASTER,
            1, MPI_COMM_WORLD);
  MPI_Recv (buff, count, datatype, MASTER,
            2, MPI_COMM_WORLD, &status);
  ...
}
```

SAFE

```
while (!done) {
  MPI_Recv (buff, count, datatype, MASTER,
            2, MPI_COMM_WORLD, &request);
  MPI_Send (NULL, 0, MPI_INT, MASTER,
            1, MPI_COMM_WORLD);
  MPI_Wait (&request, &status);
  ...
}
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 98

Point-to-Point Communication Modes: Performance Issues

- Non-blocking calls are almost always the way to go
- Communication can be carried out during blocking system calls
- Computation and communication can be overlapped if there is special purpose communication hardware
- Less likely to have errors that lead to deadlock
- Standard mode is usually sufficient - but buffered mode can offer advantages
- Particularly if there are frequent, large messages being sent
- If the user is unsure the system provides sufficient buffer space
- Synchronous mode can be more efficient if acks are needed
- Also tells the system that buffering is not required
- But, if no overlapping then non blocking is slower due to extra data structures and book keeping!
- Only way to know.. **Benchmark it!**

2/10/2005 7:17 PM G.Fagg Cluster Computing 99

Point to Point summary

Covered all basic communications
For here we can build all other communication patterns
Manually
May be slower than 'collectives' that can use special features of some MPP/SMPs.

2/10/2005 7:17 PM G.Fagg Cluster Computing 100

Point 2 point case study

Master has a large number of 'tests' that need to ran and some average result needs to be calculated.

We will consider four things

- Overall execution structure
- What this means for message passing
- Performance issue
- Improving the structure for better performance

2/10/2005 7:17 PM G.Fagg Cluster Computing 101

P2p example

work

master

Potential workers

2/10/2005 7:17 PM G.Fagg Cluster Computing 102

Title goes here

P2p case study

What to consider

- Do we have more work than workers?
- How big is the work?
- Is the work independent from each other
- Are their intermediate results?
- Do they need to get shared? Stored on disk?
- Does the master need to do some work as well?

2/10/2005 7:17 PM G.Fagg Cluster Computing 103

P2p case study

work

master

More work than worker..

Potential workers

2/10/2005 7:17 PM G.Fagg Cluster Computing 104

P2p case study

Work definition is small...

work

master

Potential workers

2/10/2005 7:17 PM G.Fagg Cluster Computing 105

P2p case study

work

master

Master CAN do some of the work

Potential workers

2/10/2005 7:17 PM G.Fagg Cluster Computing 106

P2p case study

Work result needs to be at the master

work

master

Potential workers

2/10/2005 7:17 PM G.Fagg Cluster Computing 107

P2p case study

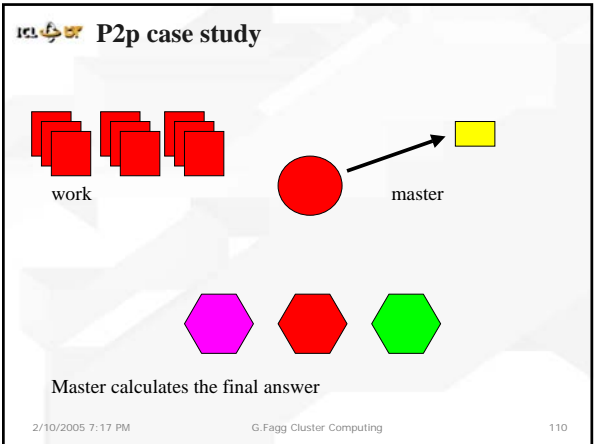
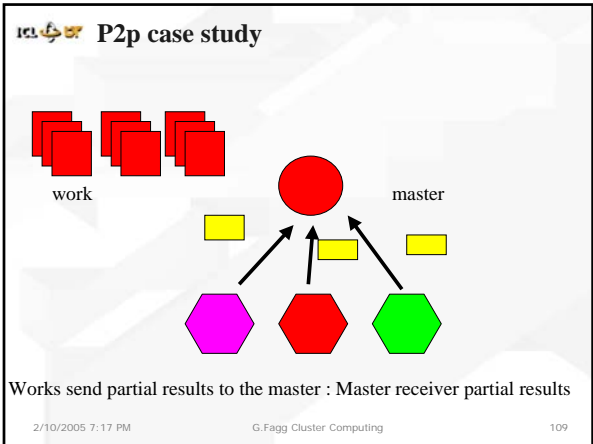
work

master

Master sends data to the workers : Slaves receive work

2/10/2005 7:17 PM G.Fagg Cluster Computing 108

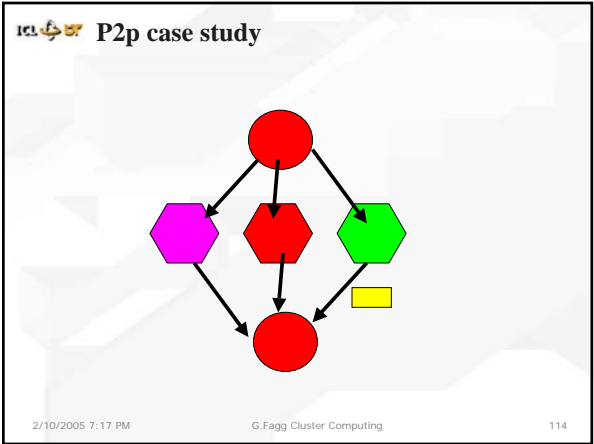
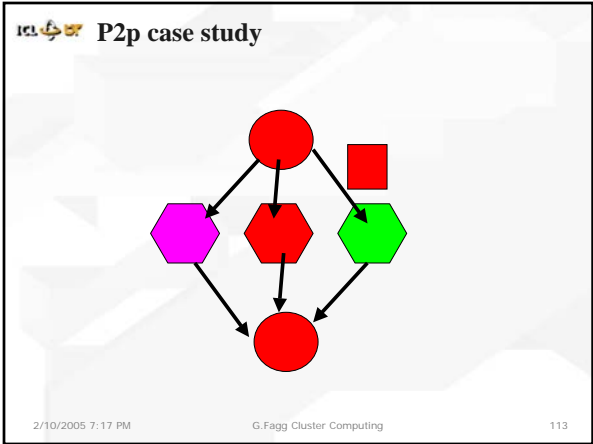
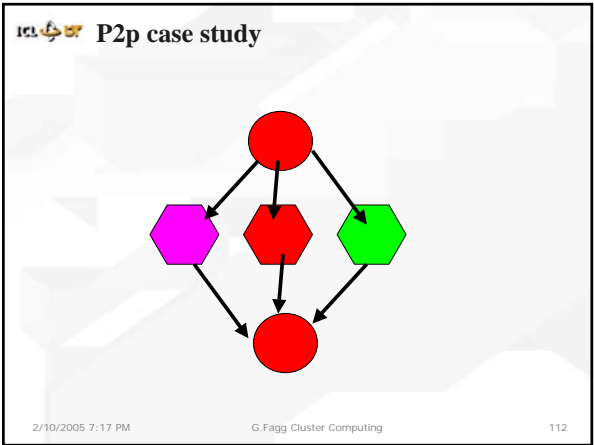
Title goes here



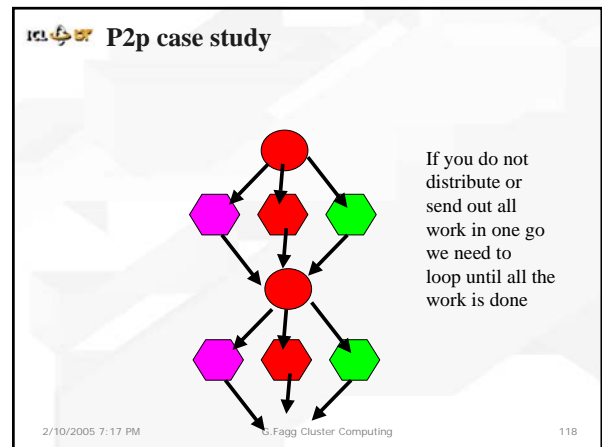
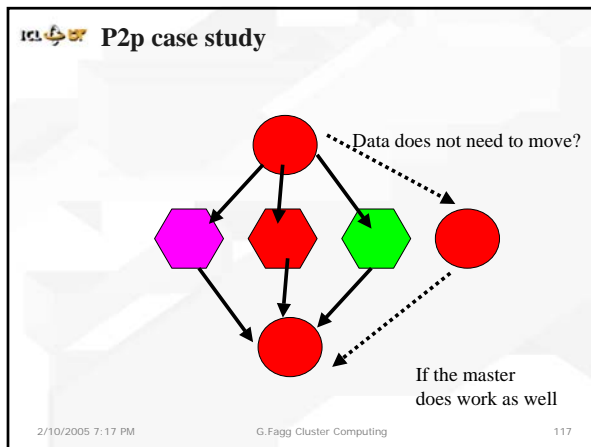
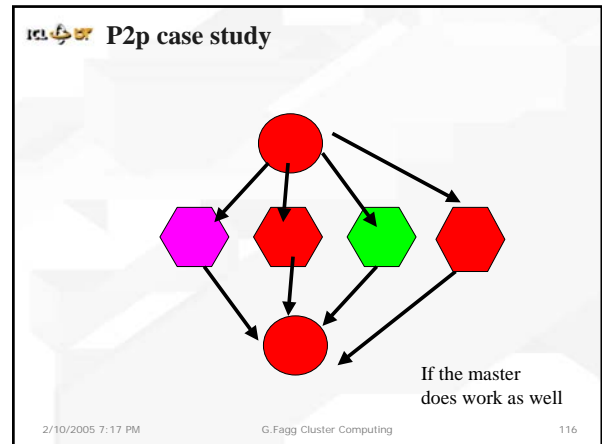
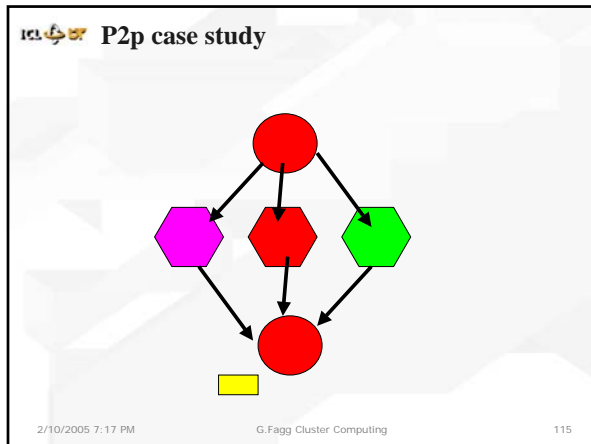
Point 2 point case study

To consider how this looks as a parallel algorithm we need to draw it as a DAG

2/10/2005 7:17 PM G.Fagg Cluster Computing 111



Title goes here



P2p case study

What does this look like in terms of code?

The job of each process is defined by who they are (master or slave)

Arcs in the graphs are data in the form of messages

2/10/2005 7:17 PM G.Fagg Cluster Computing 119

P2p case study

What does this look like in terms of code?

The job of each process is defined by who they are (master or slave)
In MPI we can use RANK to define a master
RANK can also identify who the slaves are

Arcs in the graphs are data in the form of messages
Depending on if your master or worker and which arc, we know if we are Sending to Receiving data

2/10/2005 7:17 PM G.Fagg Cluster Computing 120

Title goes here

P2p case study

```
Master
MPI_Init (...)
MPI_Comm_rank (MPI_COMM_WORLD, &rank)
if (rank==0) {
  /* I AM MASTER */
  Do_master ()
}
MPI_Finalize ()

Worker
MPI_Init (...)
MPI_Comm_rank
(MPI_COMM_WORLD, &rank):
if (rank==0) {
  /* I am Worker */
  Do_worker ( rank )
}
MPI_Finalize ()
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 121

P2p case study

```
Do_master ()
/* find out how many workers */
MPI_Comm_size (MCW, &size);
Workers=size-1;
Loop for each worker i
  Work [i] = dividework (i)
loop {
  MPI_Send (work[i], 1, worker)
}
Loop
  MPI_Recv (result[i], 1, worker...)
}
Do_calculate_result ()
Display_result ()

Do_Worker (my id)
{
  /* get work */
  MPI_Recv (work, 1, 0, status)
  PResult = Do_work (work)
  MPI_Send (work, 1, 0...)
}
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 122

P2p case study

To make the previous code work if the work does not divide up into the workers correctly you need to change the data being sent:
Special value for no-more work
you need to tell workers how much work they have they can ask for work

2/10/2005 7:17 PM G.Fagg Cluster Computing 123

P2p case study

Special value for no-more work

```
Loop {
  MPI_Recv (work, 1, 0, ... Status)
  if (work==NOWORKLEFT) return ();
  Else
    Result = Do_work () ...
    MPI_Send (Result...)
}
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 124

P2p case study

you need to tell workers how much work they have

Master:

```
work has 3 pieces of work...
MPI_Send (howmuch, work...)
loop {
  MPI_Send (work[1]...)
}
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 125

P2p case study


you need to tell workers how much work they have

Slave:

```
do_work
  MPI_recv (howmuch,...)
  loop (1..howmuch)
    MPI_Recv (work, 1, 0, ... Status)
    Result = Do_work () ...
    MPI_Send (Result...)
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 126

Title goes here


 **P2p case study**

they can ask for work

Master

```
If work-left or workers-still-working{
  MPI_Recv (what&who..)
  If what=result add it to partial result
  If work-left MPI_Send (nextwork, who..)
  Else MPI_Send (NOMOREWORK, who..)
}
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 127


 **P2p case study**

they can ask for work

Worker

```
MPI_Send (Iwantsomework, 1, 0..)
Loop (
  MPI_Recv (work, 1, 0, ... Status)
  If (work==NOWORKLEFT) return ();
  Else
    Result = Do_work ( ) ...
    MPI_Send (Result..)
)
```


2/10/2005 7:17 PM G.Fagg Cluster Computing 128

 **Collective Communication**

Outline

- Introduction
- Barriers
- Broadcasts
- Gather
- Scatter
- All gather
- Alltoall
- Reduction
- Performance issues

2/10/2005 7:17 PM G.Fagg Cluster Computing 129

 **Collective Communication**

Total amount of data sent must exactly match the total amount of data received

Collective routines are collective across an entire communicator and must be called in the same order from all processors within the communicator

Collective routines are all blocking

This simply means buffers can be re-used upon return


Collective routines return as soon as the calling process' participation is complete

Does not say anything about the other processors

Collective routines may or may not be synchronizing

No mixing of collective and point-to-point communication

2/10/2005 7:17 PM G.Fagg Cluster Computing 130


 **Collective Communication**

Barrier: MPI_BARRIER (comm)

Only collective routine which provides explicit synchronization

Returns at any processor only after all processes have entered the call

2/10/2005 7:17 PM G.Fagg Cluster Computing 131

 **Collective Communication**

Collective Communication Routines:

Except broadcast, each routine has 2 variants:

- Standard variant: All messages are the same size
- Vector Variant: Each item is a vector of possibly varying length

If there is a single origin or destination, it is referred to as the *root*

Each routine (except broadcast) has distinct send and receive arguments

Send and receive buffers must be disjoint

Each can use MPI_IN_PLACE, which allows the user to specify that data contributed by the caller is already in its final location.

2/10/2005 7:17 PM G.Fagg Cluster Computing 132

Collective Communication: **Bcast**

MPI_BCAST (buffer, count, datatype, root, comm)
 Strictly in place
 MPI-1 insists on using an intra-communicator
 MPI-2 allows use of an inter-communicator
REMEMBER: A broadcast need not be synchronizing. Returning from a broadcast tells you nothing about the status of the other processes involved in a broadcast. Furthermore, though MPI does not require MPI_BCAST to be synchronizing, it neither prohibits synchronous behavior.

2/10/2005 7:17 PM G.Fagg Cluster Computing 133

BCAST

```

    If (myrank == root) {
        fp = fopen (filename, 'r');
        fscanf (fp, '%d', &iters);
        fclose (&fp);
        MPI_Bcast (&iters, 1, MPI_INT,
                  root, MPI_COMM_WORLD);
    }
    else {
        MPI_Recv (&iters, 1, MPI_INT,
                 root, tag, MPI_COMM_WORLD,
                 &status);
    }
  
```

```

    If (myrank == root) {
        fp = fopen (filename, 'r');
        fscanf (fp, '%d', &iters);
        fclose (&fp);
    }
    MPI_Bcast (&iters, 1, MPI_INT,
               root, MPI_COMM_WORLD);
    cont
  
```

THAT'S BETTER

OOPS!

2/10/2005 7:17 PM G.Fagg Cluster Computing 134

Collective Communication: **Gather**

MPI_GATHER (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
 Receive arguments are only meaningful at the root
 Each processor must send the same *amount* of data
 Root can use MPI_IN_PLACE for sendbuf:
 data is assumed to be in the correct place in the recvbuf

2/10/2005 7:17 PM G.Fagg Cluster Computing 135

Collective Communication: **Gatherv**

MPI_GATHERV (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm)
 Vector variant of MPI_GATHER
 Allows a varying amount of data from each proc
 allows root to specify where data from each proc goes
 No portion of the receive buffer may be written more than once
 MPI_IN_PLACE may be used by root.

2/10/2005 7:17 PM G.Fagg Cluster Computing 136

Collective Communication: **Gatherv (cont)**

1	2	3	4
---	---	---	---

counts

9	7	4	0
---	---	---	---

displs

2/10/2005 7:17 PM G.Fagg Cluster Computing 137

Collective Communication: **Gatherv (cont)**

```

    stride = 105;
    root = 0;
    for (i = 0; i < nprocs; i++) {
        displs[i] = i*stride;
        counts[i] = 100;
    }
  
```

MPI_Gatherv (sbuf, 100, MPI_INT,
 rbuf, counts, displs, MPI_INT,
 root, MPI_COMM_WORLD);

2/10/2005 7:17 PM G.Fagg Cluster Computing 138

Collective Communication: Scatter

MPI_SCATTER (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
 Opposite of MPI_GATHER
 Send arguments only meaningful at root
 Root can use MPI_IN_PLACE for recvbuf

2/10/2005 7:17 PM G.Fagg Cluster Computing 139

Collective Communication: Scatterv

MPI_SCATTERV (sendbuf, counts, displs, sendtype, recvbuf, recvcount, recvtype)
 Opposite of MPI_GATHERV
 Send arguments only meaningful at root
 Root can use MPI_IN_PLACE for recvbuf
 No location of the sendbuf can be read more than once

2/10/2005 7:17 PM G.Fagg Cluster Computing 140

Collective Communication: Scatterv (cont)

counts

1	2	3	4
---	---	---	---

 displs

9	7	4	0
---	---	---	---

2/10/2005 7:17 PM G.Fagg Cluster Computing 141

Collective Communication: Allgather

MPI_ALLGATHER (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
 Same as MPI_GATHER, except all processors get the result
 MPI_IN_PLACE may be used for sendbuf of all processors
 Equivalent to a gather followed by a bcast

2/10/2005 7:17 PM G.Fagg Cluster Computing 142

Collective Communication: Allgatherv

MPI_ALLGATHERV (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, comm)
 Same as MPI_GATHERV, except all processors get the result
 MPI_IN_PLACE may be used for sendbuf of all processors
 Equivalent to a gatherv followed by a bcast

2/10/2005 7:17 PM G.Fagg Cluster Computing 143

Collective Communication: Alltoall (scatter/gather)

MPI_ALLTOALL (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)

2/10/2005 7:17 PM G.Fagg Cluster Computing 144

Title goes here

Collective Communication: Alltoallv

MPI_ALLTOALLV (sendbuf, sendcounts, sdispls, sendtype, recvbuf, recvcounsts, rdispls, recvtype, comm)

Same as MPI_ALLTOALL, but the vector variant
Can specify how many blocks to send to each processor, location of blocks to send, how many blocks to receive from each processor, and where to place the received blocks

2/10/2005 7:17 PM G.Fagg Cluster Computing 145

Collective Communication: Alltoallw

MPI_ALLTOALLW (sendbuf, sendcounts, sdispls, sendtypes, recvbuf, recvcounsts, rdispls, recvtypes, comm)

Same as MPI_ALLTOALLV, except different datatypes can be specified for data scattered as well as data gathered
Can specify how many blocks to send to each processor, location of blocks to send, how many blocks to receive from each processor, and where to place the received blocks
Displacements are now in terms of bytes rather than types

2/10/2005 7:17 PM G.Fagg Cluster Computing 146

Collective Communication: Reduction


Global reduction across all members of a group
Can use predefined operations or user defined operations
Can be used on single elements or arrays of elements
Counts and types must be the same on all processors
Operations are assumed to be associative
User defined operations can be different on each processor, but not recommended

2/10/2005 7:17 PM G.Fagg Cluster Computing 147

Collective Communication: Reduction (reduce)

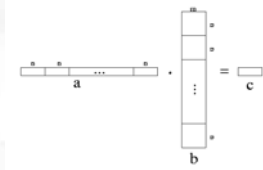
MPI_REDUCE (sendbuf, recvbuf, count, datatype, op, root, comm)

recvbuf only meaningful on root
Combines elements (on an element by element basis) in sendbuf according to op
Results of the reduction are returned to root in recvbuf
MPI_IN_PLACE can be used for sendbuf on root



2/10/2005 7:17 PM G.Fagg Cluster Computing 148

MPI_REDUCE



```
REAL a(n), b(n,m), c(m)
REAL sum(m)
DO j=1,m
  sum(j) = 0.0
  DO i=1,n
    sum(j) = sum(j) + a(i)*b(i,j)
  ENDDO
ENDDO
CALL MPI_REDUCE(sum, c, m, MPI_REAL,
MPI_SUM, 0, MPI_COMM_WORLD, ierr)
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 149

Collective Communication: Reduction (cont)

MPI_ALLREDUCE (sendbuf, recvbuf, count, datatype, op, comm)
Same as MPI_REDUCE, except all processors get the result
MPI_REDUCE_SCATTER (sendbuf, recv_buf, recvcounsts, datatype, op, comm)
Acts like it does a reduce followed by a scatterv

2/10/2005 7:17 PM G.Fagg Cluster Computing 150

Collective Communication: Reduction - user defined ops

MPI_OP_CREATE (function, commute, op)
if *commute* is true, operation is assumed to be commutative
Function is a user defined function with 4 arguments
invec: input vector
inoutvec: input and output value
len: number of elements
datatype: MPI_DATATYPE
Returns invec[i] op inoutvec[i], i = 0..len-1
MPI_OP_FREE (op)

2/10/2005 7:17 PM G.Fagg Cluster Computing 151

Collective Communication: Performance Issues

Collective operations should have much better performance than simply sending messages directly
Broadcast may make use of a broadcast tree (or other mechanism)
All collective operations can potentially make use of a tree (or other) mechanism to improve performance
Important to use the simplest collective operations which still achieve the needed results
Use MPI_IN_PLACE whenever appropriate
Reduces unnecessary memory usage and redundant data movement

2/10/2005 7:17 PM G.Fagg Cluster Computing 152

Case study again

In the previous example we sent all the work out using point to point calls
Received all the results using point to pint calls.
Could use collectives

2/10/2005 7:17 PM G.Fagg Cluster Computing 153

Case study

2/10/2005 7:17 PM G.Fagg Cluster Computing 154

Case study

Broadcast or scatter ?

2/10/2005 7:17 PM G.Fagg Cluster Computing 155

Case study

Broadcast

If broadcast all nodes get the same set of work
the workers have to understand what work they are doing

2/10/2005 7:17 PM G.Fagg Cluster Computing 156

Title goes here

Case study

scatter

If scatter then custom work per worker can be sent

2/10/2005 7:17 PM G.Fagg Cluster Computing 157

Case study

Gather

2/10/2005 7:17 PM G.Fagg Cluster Computing 158

What Else is There

- Lots of other routines
 - Derived datatypes
 - Process groups and communicators
 - Process topologies
 - Profiling
- MPI-2
 - Parallel I/O
 - Dynamic process management
 - One sided communication

2/10/2005 7:17 PM G.Fagg Cluster Computing 159

Communicators and Groups

If you need to handle lots of processes in a simple way by breaking them into relative groups that have a certain relationship

- Column communicator
- Row communicator
- Simplifying communication
 - A group just for summing a residue value

2/10/2005 7:17 PM G.Fagg Cluster Computing 160

Communicators and Groups

Many MPI users are only familiar with MPI_COMM_WORLD

A communicator can be thought of as a handle to a group

A group is an ordered set of processes

- Each process is associated with a rank
- Ranks are contiguous and start from zero

For many applications (dual level parallelism) maintaining different groups is appropriate

Groups allow collective operations to work on a subset of processes

Information can be added onto communicators to be passed into routines

2/10/2005 7:17 PM G.Fagg Cluster Computing 161

Communicators and Groups(cont)

While we think of a communicator as spanning processes, it is actually unique to a process

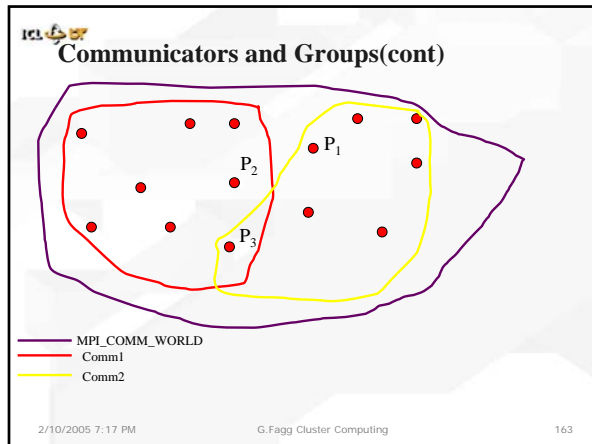
A communicator can be thought of as a handle to an object (group attribute) that describes a group of processes

An intracommunicator is used for communication within a single group

An intercommunicator is used for communication between 2 disjoint groups

2/10/2005 7:17 PM G.Fagg Cluster Computing 162

Title goes here



Communicators and Groups(cont)

Refer to previous slide

There are 3 distinct groups
These are associated with MPI_COMM_WORLD, comm1, and comm2

P₃ is a member of all 3 groups and may have different ranks in each group(say 0, 3, & 4)

If P₂ wants to send a message to P₁ it can use MPI_COMM_WORLD (intracommunicator) or an intercommunicator (covered later)

If P₂ wants to send a message to P₃ it can use MPI_COMM_WORLD (send to rank 0), comm1 (send to rank 3), or and intercommunicator

2/10/2005 7:17 PM G.Fagg Cluster Computing 164

Group Management

All group operations are local
As will be clear, groups are initially not associated with communicators
Groups can only be used for message passing within a communicator
We can access groups, construct groups, and destroy groups

2/10/2005 7:17 PM G.Fagg Cluster Computing 165

Group Accessors

MPI_GROUP_SIZE(group, size)
MPI_Group group
int size
This routine returns the number of processes in the group

MPI_GROUP_RANK(group, rank)
MPI_Group group
int rank
This routine returns the rank of the calling process

2/10/2005 7:17 PM G.Fagg Cluster Computing 166

Group Accessors (cont)


MPI_GROUP_TRANSLATE_RANKS (group1, n, ranks1, group2, ranks2)
MPI_Group group1, group2
int n, *ranks1, *ranks2
This routine takes an array of n ranks (ranks1) which are ranks of processes in group1. It returns in ranks2 the corresponding ranks of the processes as they are in group2
MPI_UNDEFINED is returned for processes not in group2

2/10/2005 7:17 PM G.Fagg Cluster Computing 167

Groups Accessors (cont)

MPI_GROUP_COMPARE (group1, group2 result)
MPI_Group group1, group2
int result
This routine returns the relationship between group1 and group2
If group1 and group2 contain the same processes, ranked the same way, this routine returns MPI_IDENT
If group1 and group2 contain the same processes, but ranked differently, this routine returns MPI_SIMILAR
Otherwise this routine returns MPI_UNEQUAL

2/10/2005 7:17 PM G.Fagg Cluster Computing 168

 **Group Constructors**

Group constructors are used to create new groups from existing groups
Base group is the group associated with MPI_COMM_WORLD
Group creation is a local operation
No communication needed
Following group creation, no communicator is associated with the group
No communication possible with new group

2/10/2005 7:17 PM G.Fagg Cluster Computing 169

 **Group Constructors (cont)**

MPI_COMM_GROUP(comm, group)
MPI_Comm comm
MPI_Group group
This routine returns in group the group associated with the communicator comm

2/10/2005 7:17 PM G.Fagg Cluster Computing 170

 **Group Constructors (cont)**
Set Operations


MPI_GROUP_UNION(group1, group2, newgroup)
MPI_GROUP_INTERSECTION(group1, group2, newgroup)
MPI_GROUP_DIFFERENCE(group1, group2, newgroup)
MPI_Group group1, group2, *newgroup

2/10/2005 7:17 PM G.Fagg Cluster Computing 171

 **Group Constructors (cont)**
Set Operations


Union: Returns in newgroup a group consisting of all processes in group1 followed by all processes in group2, with no duplication
Intersection: Returns in newgroup all processes that are in both groups, ordered as in group1
Difference: Returns in newgroup all processes in group1 that are not in group2, ordered as in group1

2/10/2005 7:17 PM G.Fagg Cluster Computing 172

 **Group Constructors (cont)**
Set Operations

Let group1 = {a,b,c,d,e,f,g} and group2 = {d,g,a,c,h,i}
MPI_Group_union(group1,group2,newgroup)
Newgroup = {a,b,c,d,e,f,g,h,i}
MPI_Group_intersection(group1,group2,newgroup)
Newgroup = {a,c,d,g}
MPI_Group_difference(group1,group2,newgroup)
Newgroup = {b,e,f}

2/10/2005 7:17 PM G.Fagg Cluster Computing 173

 **Group Constructors (cont)**
Set Operations

Let group1 = {a,b,c,d,e,f,g} and group2 = {d,g,a,c,h,i}
MPI_Group_union(group2,group1,newgroup)
Newgroup = {d,g,a,c,h,i,b,e,f}
MPI_Group_intersection(group2,group1,newgroup)
Newgroup = {d,g,a,c}
MPI_Group_difference(group1,group2,newgroup)
Newgroup = {h,i}

2/10/2005 7:17 PM G.Fagg Cluster Computing 174

Title goes here

 **Group Constructors (cont)**


```
MPI_GROUP_INCL(group, n, ranks, newgroup)
MPI_Group group, *newgroup
int n, *ranks
This routine creates a new group that consists of all the n processes with
ranks ranks[0]..ranks[n-1]
The process with rank i in newgroup has rank ranks[i] in group
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 175

 **Group Constructors (cont)**


```
MPI_GROUP_EXCL(group, n, ranks, newgroup)
MPI_Group group, *newgroup
int n, *ranks
This routine creates a new group that consists of all the processes in
group after deleting processes with ranks ranks[0]..ranks[n-1]
The ordering in newgroup is identical to the ordering in group
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 176

 **Group Constructors (cont)**


```
MPI_GROUP_RANGE_INCL(group, n, ranges, newgroup)
MPI_Group group, *newgroup
int n, ranges[][3]
Ranges is an array of triplets consisting of start rank, end rank, and
stride
Each triplet in ranges specifies a sequence of ranks to be included in
newgroup
The ordering in newgroup is as specified by ranges
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 177

 **Group Constructors (cont)**


```
MPI_GROUP_RANGE_EXCL(group, n, ranges, newgroup)
MPI_Group group, *newgroup
int n, ranges[][3]
Ranges is an array of triplets consisting of start rank, end rank, and
stride
Each triplet in ranges specifies a sequence of ranks to be excluded from
newgroup
The ordering in newgroup is identical to that in group
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 178

 **Group Constructors (cont)**

```
Let group = {a,b,c,d,e,f,g,h,i,j}
n=5, ranks = {0,3,8,6,2}
ranges = {(4,9,2),(1,3,1),(0,9,5)}
MPI_Group_incl(group,n,ranks,newgroup)
newgroup = {a,d,l,g,c}
MPI_Group_excl(group,n,ranks,newgroup)
newgroup = {b,e,f,h,j}
MPI_Group_range_incl(group,n,ranges,newgroup)
newgroup = {e,g,l,b,c,d,a,f}
MPI_Group_range_excl(group,n,ranges,newgroup)
newgroup = {h}
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 179

 **Communicator Management**


Communicator access operations are local, thus requiring no interprocess communication

Communicator constructors are collective and may require interprocess communication

All the routines in this section are for intracommunicators, intercommunicators will be covered separately

2/10/2005 7:17 PM G.Fagg Cluster Computing 180

Title goes here


 **Communicator Accessors**

MPI_COMM_SIZE (comm, size)
Returns the number of processes in the group associated with comm

MPI_COMM_RANK (comm, rank)
Returns the rank of the calling process within the group associated with comm


MPI_COMM_COMPARE (comm1, comm2, result) returns:
MPI_IDENT if comm1 and comm2 are handles for the same object
MPI_CONGRUENT if comm1 and comm2 have the same group attribute
MPI_SIMILAR if the groups associated with comm1 and comm2 have the same members but in different rank order
MPI_UNEQUAL otherwise

2/10/2005 7:17 PM G.Fagg Cluster Computing 181

 **Communicator Constructors**


MPI_COMM_DUP (comm, newcomm)
This routine creates a duplicate of comm
newcomm has the same fixed attributes as comm
Defines a new communication domain
A call to MPI_Comm_compare (comm, newcomm, result) would return MPI_CONGRUENT
Useful to library writers and library users

2/10/2005 7:17 PM G.Fagg Cluster Computing 182

 **Communicator Constructors**


MPI_COMM_CREATE (comm, group, newcomm)
This is a collective routine, meaning it must be called by all processes in the group associated with comm
This routine creates a new communicator which is associated with group
MPI_COMM_NULL is returned to processes not in group
All group arguments must be the same on all calling processes
group must be a subset of the group associated with comm

2/10/2005 7:17 PM G.Fagg Cluster Computing 183

 **Communicator Constructors**

MPI_COMM_SPLIT(comm,color,key,newcomm)
MPI_Comm comm, newcomm
int color, key
This routine creates as many new groups and communicators as there are distinct values of color
The rankings in the new groups are determined by the value of key, ties are broken according to the ranking in the group associated with comm
MPI_UNDEFINED is used as the color for processes to not be included in any of the new groups


2/10/2005 7:17 PM G.Fagg Cluster Computing 184

 **Communication Constructors**

Rank	0	1	2	3	4	5	6	7	8	9	10
Process	a	b	c	d	e	f	g	h	i	j	k
Color	U	3	1	1	3	7	3	3	1	U	3
Key	0	1	2	3	1	9	3	8	1	0	0

Both process a and j are returned MPI_COMM_NULL
3 new groups are created
{i, c, d}
{k, b, e, g, h}
{f}

2/10/2005 7:17 PM G.Fagg Cluster Computing 185

 **Destructors**

The communicators and groups from a process' viewpoint are merely handles
Like all handles in MPI, there is a limited number available – **YOU CAN RUN OUT**
MPI_GROUP_FREE (group)
MPI_COMM_FREE (comm)

2/10/2005 7:17 PM G.Fagg Cluster Computing 186

Intercommunicators

Intercommunicators are associated with 2 groups of disjoint processes
 Intercommunicators are associated with a remote group and a local group
 A communicator is either intra or inter, never both

2/10/2005 7:17 PM G.Fagg Cluster Computing 187

Intercommunicators

Intercommunicator

2/10/2005 7:17 PM G.Fagg Cluster Computing 188

Intercommunicator Accessors

MPI_COMM_TEST_INTER(comm, flag)
 This routine returns true if comm is an intercommunicator, otherwise, false

MPI_COMM_REMOTE_SIZE(comm, size)
 This routine returns the size of the remote group associated with intercommunicator comm

MPI_COMM_REMOTE_GROUP(comm, group)
 This routine returns the remote group associated with intercommunicator comm

2/10/2005 7:17 PM G.Fagg Cluster Computing 189

Intercommunicator Constructors

The communicator constructors described previously will return an intercommunicator if they are passed intercommunicators as input

MPI_COMM_DUP: returns an intercommunicator with the same groups as the one passed in

MPI_COMM_CREATE: each process in group A must pass in group the same subset of group A (A1). Same for group B (B1). The new communicator has groups A1 and B1 and is only valid on processes in A1 and B1

MPI_COMM_SPLIT: As many new communicators as there are distinct pairs of colors are created

2/10/2005 7:17 PM G.Fagg Cluster Computing 190

Communication Constructors

Rank	0	1	2	3	4	5	6	7	8	9	10
Process	a	b	c	d	e	f	g	h	i	j	k
Color	U	3	3	1	1	7	3	3	1	U	3
Key	0	1	2	3	1	9	3	8	1	0	0

A

Rank	0	1	2	3	4	5	6	7	8	9	10
Process	l	m	n	o	p	q	r	s	t	u	v
Color	5	3	1	U	3	3	3	7	1	U	3
Key	0	1	2	3	1	9	3	8	1	0	0

B

2/10/2005 7:17 PM G.Fagg Cluster Computing 191

Intercommunicator Constructors

Processes a, j, l, o, and u would all have MPI_COMM_NULL returned in newcomm

newcomm1 would be associated with 2 groups: {e, i, d} and {t, n}

newcomm2 would be associated with 2 groups: {k, b, c, g, h} and {v, m, p, r, q}

newcomm3 would be associated with 2 groups: {f} and {s}

2/10/2005 7:17 PM G.Fagg Cluster Computing 192

Title goes here

Intercommunicator Constructors

MPI_INTERCOMM_CREATE (local_comm, local_leader, bridge_comm, remote_leader, tag, newintercomm)

This routine is called collectively by all processes in 2 disjoint groups

All processes in a particular group must provide matching local_comm and local_leader arguments

The local leaders provide a matching bridge_comm (a communicator through which they can communicate), in remote_leader the rank of the other leader within bridge_comm, and the same tag

The bridge_comm, remote_leader, and tag are significant only at the leaders

There must be no pending communication across bridge_comm that may interfere with this call

2/10/2005 7:17 PM G.Fagg Cluster Computing 193

Intercommunicators

2/10/2005 7:17 PM G.Fagg Cluster Computing 194

Intercommunicators

MPI_INTERCOMM_MERGE (intercomm, high, newintracomm)

This routine creates an intracommunicator from a union of the two groups associated with intercomm

High is used for ordering. All process within a particular group must pass the same value in for high (true or false)

The new intracommunicator is ordered with the high processes following the low processes

If both groups pass the same value for high, the ordering is arbitrary

2/10/2005 7:17 PM G.Fagg Cluster Computing 195

Attribute Caching

It is possible to *cache* attributes to be associated with a communicator

This cached information is process specific.

The same attribute can be cached with multiple communicators

Many attributes can be cached with a single communicator

This is most commonly used in libraries

2/10/2005 7:17 PM G.Fagg Cluster Computing 196

Selected References

- MPI - The Complete Reference Volume 1, The MPI Core
- MPI - The Complete Reference Volume 2, The MPI Extensions
- USING MPI: Portable Parallel Programming with the Message-Passing Interface
- Using MPI-2: Advanced Features of the Message-Passing Interface

2/10/2005 7:17 PM G.Fagg Cluster Computing 197

2/10/2005 7:17 PM G.Fagg Cluster Computing 198

Title goes here

Threads and MPI

Graham E. Fagg
University of Tennessee



INNOVATIVE COMPUTING LABORATORY
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE

ICL Threading

- Multi-threading can improve performance
 - Better CPU utilization
 - IO latency hiding
 - Simplified logic (letting threads block)
- Most useful on SMPs
 - Each thread can have its own CPU
- Overloading CPU's can be ok
 - Depends on application (e.g., latency hiding)
 - Even on uniprocessors

2/10/2005 7:17 PM G.Fagg Cluster Computing 200

ICL Threads and MPI

- Extend the threaded model to multi-level parallelism
 - Threads within an MPI process
 - Possibly spanning multiple processors
 - Allowing threads to block in communication
- Overlap communication and computation

2/10/2005 7:17 PM G.Fagg Cluster Computing 201

ICL Application Level Threading

- Freedom to use blocking MPI functions
 - Allow threads to block in MPI_SEND / MPI_RECV
 - Simplify application logic
- Separate communication and computation

2/10/2005 7:17 PM G.Fagg Cluster Computing 202

ICL Implementation Threading

- Asynchronous communication progress
 - Allow communication "in the background"
 - Even while no application threads in MPI
- Can help single-threaded user applications
 - Non-blocking communications can progress independent of application

2/10/2005 7:17 PM G.Fagg Cluster Computing 203

ICL Asynchronous Communication

		MPI implementation	
		One thread	Multiple threads
App	One thread	X	☺
	Multiple threads	X ☺	☺

2/10/2005 7:17 PM G.Fagg Cluster Computing 204

Title goes here

🔧 What About “One Big Lock”?

Put a mutex around MPI calls
Only allow one application thread in MPI at any given time
This allows a multi-threaded application to utilize MPI
Problem: can easily lead to deadlock
If multiple threads try to use MPI
Example
Thread 1 calls MPI_RECV
Thread 2 later calls matching MPI_SEND

2/10/2005 7:17 PM G.Fagg Cluster Computing 205

🔧 Why Not Use Non-Blocking?

Why not use MPI_ISEND? (and friends)
This has worked for years
MPI implementations already support it
Allows at least some degree of overlap
Threads can allow simplicity of logic
Do not have to poll for MPI completion
Concurrency within application
Let threads block in MPI_SEND / MPI_RECV

2/10/2005 7:17 PM G.Fagg Cluster Computing 206

🔧 Doesn't MPI Do This Already?

MPI_SEND: Does it progress after return?
Example: in TCP, MPI typically calls write(2)
OS buffers and sends “in the background”
Do not get “true” progress (e.g., rendezvous)
If the MPI implementation can use threads:
True asynchronous progress
Progress pending communications while application is outside of MPI

2/10/2005 7:17 PM G.Fagg Cluster Computing 207

🔧 Threads and MPI

MPI does not define if a MPI process is a thread or an OS process
Threads are not addressable
MPI_SEND(...thread_id...) is not possible
MPI-2 Specification
Does not mandate thread support
Does define what a “Thread Compliant MPI” should do
Specifies 4 levels of thread support

2/10/2005 7:17 PM G.Fagg Cluster Computing 208

🔧 Thread Compliant MPI

All MPI library calls are thread safe
Blocking calls block the calling thread only and allow progress on other threads

2/10/2005 7:17 PM G.Fagg Cluster Computing 209

🔧 MPI Threading Rules

MPI_INIT and MPI_FINALIZE should only be called once
Should only be called by a single thread
Both should be called by the same thread
Known as the **main thread**

2/10/2005 7:17 PM G.Fagg Cluster Computing 210

Title goes here

Threads and Requests

Multiple threads should not attempt to complete the same request
Erroneous example:

Thread1 `MPI_Wait (req...)`

Thread2 `MPI_Wait (req...)`

2/10/2005 7:17 PM G.Fagg Cluster Computing Time¹¹

Threads and Exceptions

Exception handlers can arise in a different thread context than the one making the MPI call

User Thread `MPI_Send (..req..)`

Internal Thread `Performing send`

Passing com request to internal send thread

Internal thread has a problem, throws exception

Error handler etc

2/10/2005 7:17 PM G.Fagg Cluster Computing Time¹²

More Thread Rules

Undefined behavior of MPI call when:

- If a thread executes an MPI call that is cancelled by another thread
- If a thread executes an MPI call and catches a signal

How to deal with signals?

2/10/2005 7:17 PM G.Fagg Cluster Computing 213

Avoiding Signal Problems

Create extra thread that waits in `sigwait()`
MPI threads mask signals

User Thread `MPI_Send / Recv / Wait / etc.`

sigmask()

Extra Thread `sigwait()`

OS signals etc

Thread catches almost all signals

2/10/2005 7:17 PM G.Fagg Cluster Computing Time¹⁴

Threads and MPI

Normally initialize MPI process with `MPI_INIT`
Threaded MPI programs use `MPI_INIT_THREAD(argv, argv, requested, provided)`
Tells MPI application threading requirements
Implementation informs application of what it can provide
If implementation cannot support a requested thread level, it returns the highest level it can provide
This is not an error!

2/10/2005 7:17 PM G.Fagg Cluster Computing 215

Threads and MPI

Available levels of thread support

- `MPI_THREAD_SINGLE`
- `MPI_THREAD_FUNNELED`
- `MPI_THREAD_SERIALIZED`
- `MPI_THREAD_MULTIPLE`

2/10/2005 7:17 PM G.Fagg Cluster Computing 216

Title goes here

MPI_THREAD_SINGLE

Application is NOT allowed to use threads
 This allows an MPI implementation to avoid potentially expensive locking
 Might cause problems / errors if the application actually does use threads
 * Specification is unclear on if the Implementation can use threads

2/10/2005 7:17 PM G.Fagg Cluster Computing 217

MPI_THREAD_FUNNELED

The user application can be multi-threaded but only the main thread calls MPI functions

2/10/2005 7:17 PM G.Fagg Cluster Computing 218

MPI_THREAD_SERIALIZED

Users application is multi-threaded any thread can make MPI calls
 But only one thread can / will be in MPI at a time

2/10/2005 7:17 PM G.Fagg Cluster Computing 219

MPI_THREAD_SERIALIZED

Application can be multi-threaded any thread can make MPI calls
 But only one thread can / will be in MPI at a time

2/10/2005 7:17 PM G.Fagg Cluster Computing 220

MPI_THREAD_MULTIPLE

Application can be multi-threaded and any thread can make an MPI call at any time
 Least restricted and most flexible programming model

2/10/2005 7:17 PM G.Fagg Cluster Computing 221

Threads and MPI

MPI_QUERY_THREAD
 Returns provided level of thread support
 Useful if MPI_INIT was invoked (vs. MPI_INIT_THREAD)
 → Thread level may be set via environment variable!

MPI_IS_THREAD_MAIN
 Returns true if this is the thread that invoked MPI_INIT / MPI_INIT_THREAD

2/10/2005 7:17 PM G.Fagg Cluster Computing 222

Title goes here

Threading Example

Use a common master / slave framework
Master sends out work
Workers receive work, do work, return work
Loop until complete
Show how threads can be beneficial in this scenario

2/10/2005 7:17 PM G.Fagg Cluster Computing 223

Method 1: Pure Master / Slave

Total of N processes
1 Master process
(N-1) Slave processes

Master
Send initial set of work
Loop receiving / sending

Worker
Loop: receive, work, send

2/10/2005 7:17 PM G.Fagg Cluster Computing 224

Pure Master / Slave

2/10/2005 7:17 PM G.Fagg Cluster Computing 225

Application main()

```
MPI_Init(...);  
MPI_Comm_rank(..., &rank);  
if (rank == 0)  
    do_master()  
else  
    do_slave()  
MPI_Finalize()
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 226

Master Main Loop

```
for (i = 0; i < n; ++i)  
    MPI_Send(work[i], ..., slaves[i], ...);  
while (i < total_work) {  
    MPI_Recv(answer, ..., MPI_ANY_SOURCE, ...);  
    process_answer(answer);  
    if (++i < total_work) {  
        MPI_Send(work[i], ..., slave[X], ...);  
    } else {  
        MPI_Send(you_are_done, ..., slave[X], ...);  
    }  
}
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 227

Slave Main Loop

```
while (1) {  
    MPI_Recv(work, ...);  
    if (work == you_are_done)  
        break;  
    answer = do_work(work);  
    MPI_Send(answer, ...);  
}
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 228

Title goes here


Summary

- Benefits
 - Easily understood paradigm
 - Robust algorithm
- Drawbacks
 - Master process cannot do any work other than calculating the final result
 - To improve: Master needs to do work and control simultaneously

2/10/2005 7:17 PM G.Fagg Cluster Computing 229

Method 2: Combined Master / Slave

- Total of N MPI processes
- N Slave processes
- Master is combined with Slave 1
- Not wasting a full process for the Master



2/10/2005 7:17 PM G.Fagg Cluster Computing 230

Combined Master / Slave

- Combined master and slave routines in 1st Slave
 - Send / receive work
 - Do work / calculate answers
- Use non-blocking receives to collect results
 - Use MPI_TEST calls to poll for results
- Master must track state of receives rather than simple outstanding work counter

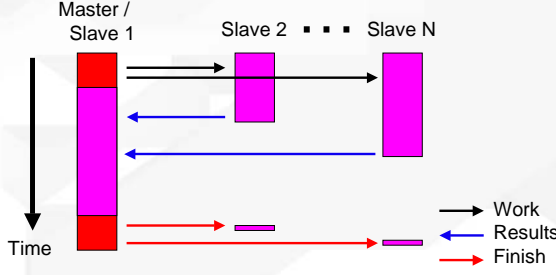
2/10/2005 7:17 PM G.Fagg Cluster Computing 231

Combined Master / Slave

- New combined master algorithm
 - Send initial work set
 - Post MPI_IRECV for each item of work sent
- Loop
 - If work available, do work locally
 - Check for completion of other slaves
 - If completion, send more work or "finish" message
- End loop when no more work to be done and all slaves finished

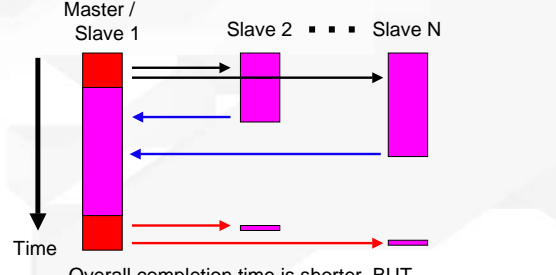
2/10/2005 7:17 PM G.Fagg Cluster Computing 232

Combined Master / Slave

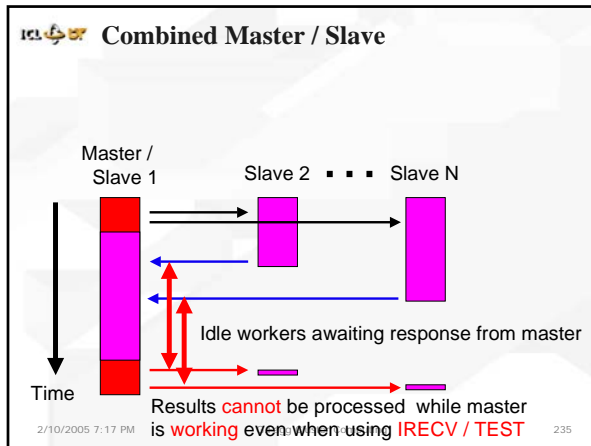


2/10/2005 7:17 PM G.Fagg Cluster Computing 233

Combined Master / Slave



2/10/2005 7:17 PM G.Fagg Cluster Computing 234



Summary

Benefits

- Does not waste a process for the Master

Drawbacks

- Complicated application code
- Master does not asynchronously process messages while working
- Not just simple overlapping of computation and communication
- Stalls the work pipeline -- idle workers

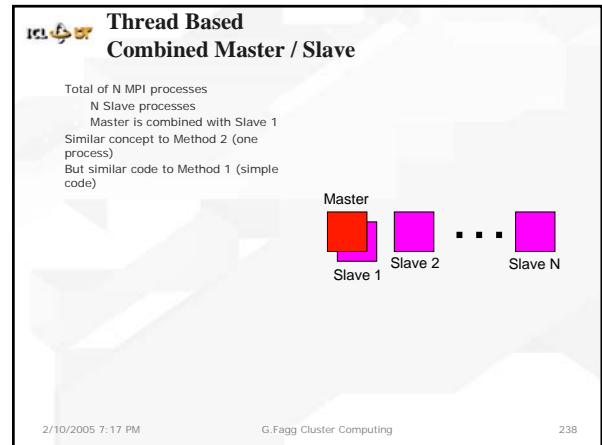
2/10/2005 7:17 PM G.Fagg Cluster Computing 236

Method 3: Thread Based Combined Master / Slave

Use threads

- Master code in one thread
- Slave code in another thread
- Independent progress
- Code now almost identical to Method 1
- Simplified code / less custom code = less errors

2/10/2005 7:17 PM G.Fagg Cluster Computing 237

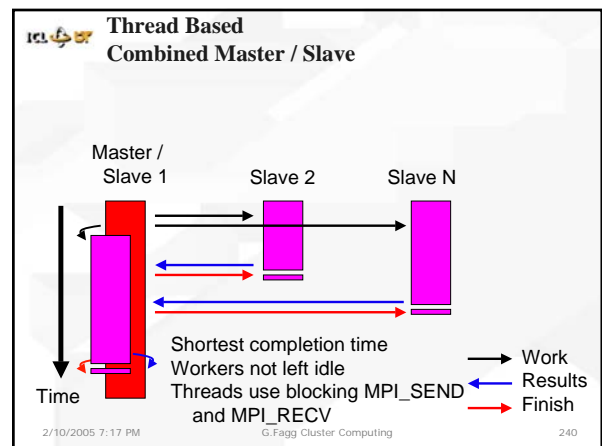


Application main()

```

MPI_Init_thread(..., MPI_THREAD_MULTIPLE, ...);
MPI_Comm_rank(..., &rank)
if (rank == 0)
    pthread_create(..., do_master, ...);
do_slave();
pthread_join(...);
MPI_Finalize();
    
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 239



Title goes here

Summary

Benefits

- Simple code -- similar to method 1
- Overlap communication and computation

Drawbacks

- 1st Slave might run somewhat slower than its peers

2/10/2005 7:17 PM G.Fagg Cluster Computing 241

Dynamic Processes: Spawn

Graham E. Fagg
University of Tennessee

ICL
INNOVATIVE COMPUTING LABORATORY
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE

Dynamic Processes

Adding processes to a running job

- As part of the algorithm i.e. branch and bound
- When additional resources become available
- Some master-slave codes where the master is started first and asks the environment how many processes it can create

Joining separately started applications

- Client-server or peer-to-peer
- Handling faults/failures

2/10/2005 7:17 PM G.Fagg Cluster Computing 243

MPI-1 Processes

All process groups are derived from the membership of the MPI_COMM_WORLD

- No external processes

Process membership static (vs. PVM)

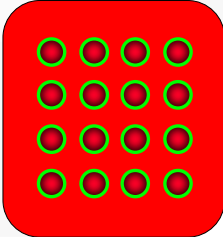
- Simplified consistency reasoning
- Fast communication (fixed addressing) even across complex topologies
- Interfaces well to many parallel run-time systems

2/10/2005 7:17 PM G.Fagg Cluster Computing 244

Static MPI-1 Job

MPI_COMM_WORLD
Contains 16 processes

MPI_COMM_WORLD

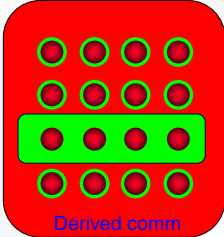


2/10/2005 7:17 PM G.Fagg Cluster Computing 245

Static MPI-1 Job

MPI_COMM_WORLD
Contains 16 processes
Can only subset the original MPI_COMM_WORLD
No external processes

MPI_COMM_WORLD



2/10/2005 7:17 PM G.Fagg Cluster Computing 246

Disadvantages of Static Model

- Cannot add processes
- Cannot remove processes
- If a process fails or otherwise disappears, all communicators it belongs to become invalid

→ Fault tolerance undefined

2/10/2005 7:17 PM G.Fagg Cluster Computing 247

Types of Communicators

- Intra**communicator
 - "Normal" communicator
 - MPI_COMM_WORLD is an intracommunicator
 - One group of processes
- Inter**communicator
 - Two groups of processes: local and remote
 - Always communicate relative to remote group
 - Both types can be used with MPI_SEND / MPI_RECV

2/10/2005 7:17 PM G.Fagg Cluster Computing 248

Continue Previous Example

MPI_COMM_WORLD and one derived communicator
Both are intracoms

2/10/2005 7:17 PM G.Fagg Cluster Computing 249

Continue Previous Example

MPI_COMM_WORLD and one derived communicator
Both are intracoms
Create another derived communicator
Now have 2 groups

2/10/2005 7:17 PM G.Fagg Cluster Computing 250

Continue Previous Example

MPI_COMM_WORLD and one derived communicator
Both are intracoms
Create another derived communicator
Now have 2 groups
Create intercomm from the two groups


2/10/2005 7:17 PM G.Fagg Cluster Computing 251

MPI-2

- Added support for dynamic processes
- Creation of new processes on the fly
- Connecting previously existing processes
- Does not standardize inter-implementation communication
- Interoperable MPI (IMPI) created for this


2/10/2005 7:17 PM G.Fagg Cluster Computing 252

Title goes here

 **Open Questions**


How do you add more processes to an already-running MPI-1 job?

2/10/2005 7:17 PM G.Fagg Cluster Computing 253

 **Open Questions**


How would you handle a process failure?

2/10/2005 7:17 PM G.Fagg Cluster Computing 254

 **Open Questions**

How could you establish MPI communication between two independently initiated, simultaneously running MPI jobs?


2/10/2005 7:17 PM G.Fagg Cluster Computing 255

 **MPI-2 Process Management**

MPI-2 provides "spawn" functionality
Launches a child MPI job from a parent MPI job
Some MPI implementations support this

- Open MPI
- LAM/MPI
- NEC MPI
- Sun MPI
- ...


2/10/2005 7:17 PM G.Fagg Cluster Computing 256

 **MPI-2 Spawn Functions**

MPI_COMM_SPAWN
Starts a set of new processes with the same command line
SPMD

MPI_COMM_SPAWN_MULTIPLE
Starts a set of new processes with potentially different command lines
Different executables and / or different arguments
MPMD

2/10/2005 7:17 PM G.Fagg Cluster Computing 257

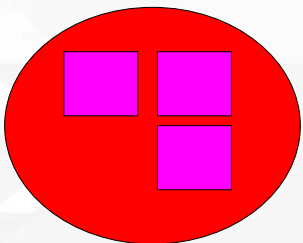
 **Spawn Semantics**

Group of parents collectively call spawn
Launches a new set of children processes
Children processes become an MPI job
An **inter**communicator is created between parents and children
Parents and children can then use MPI functions to pass messages

2/10/2005 7:17 PM G.Fagg Cluster Computing 258

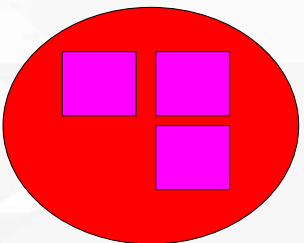
Title goes here

Spawn Example



2/10/2005 7:17 PM G.Fagg Cluster Computing 259

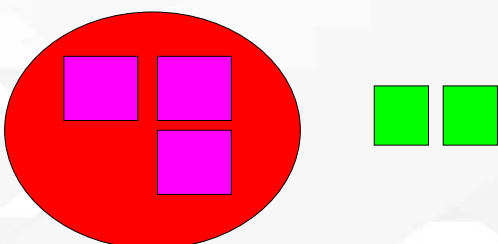
Spawn Example



Parents call `MPI_COMM_SPAWN`

2/10/2005 7:17 PM G.Fagg Cluster Computing 260

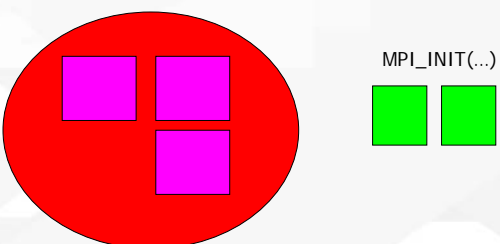
Spawn Example



Two processes are launched

2/10/2005 7:17 PM G.Fagg Cluster Computing 261

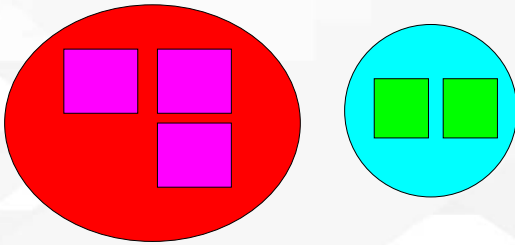
Spawn Example



Children processes call `MPI_INIT(...)`

2/10/2005 7:17 PM G.Fagg Cluster Computing 262

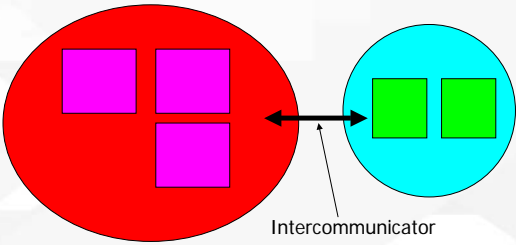
Spawn Example



Children create their own `MPI_COMM_WORLD`

2/10/2005 7:17 PM G.Fagg Cluster Computing 263

Spawn Example

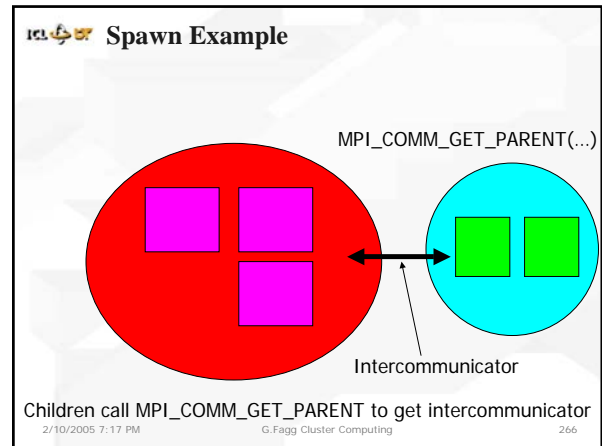
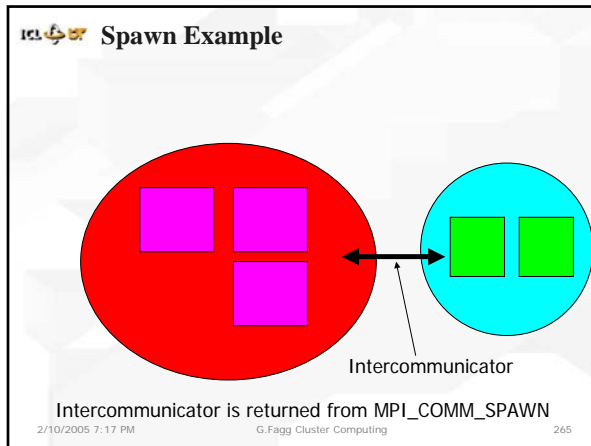


Intercommunicator

An intercommunicator is formed between parents and children

2/10/2005 7:17 PM G.Fagg Cluster Computing 264

Title goes here



Master / Slave Demonstration

Simple 'PVM' style example

- User starts singleton master process
- Master process spawns slaves
- Master and slaves exchange data, do work
- Master gathers results
- Master displays results
- All processes shut down

2/10/2005 7:17 PM G.Fagg Cluster Computing 267

Master / Slave Demonstration

```
Master program
MPI_Init(...)
MPI_Spawn(..., slave, ...);
for (i=0; i < size; i++)
  MPI_Send(work, ...i, ...);
for (i=0; i < size; i++)
  MPI_Recv(result, ...);
calc_and_display_result(...);
MPI_Finalize();

Slave program
MPI_Init(...)
MPI_Comm_get_parent (&intercomm)
MPI_Recv(work, ... intercomm)
result = do_something(work)
MPI_Send(result, ... intercomm)
MPI_Finalize()
```

2/10/2005 7:17 PM G.Fagg Cluster Computing 268

MPI "Connected"

"Two processes are connected if there is a communication path directly or indirectly between them."

- E.g., belong to the same communicator
- Parents and children from SPAWN are connected

Connectivity is transitive

- If A is connected to B, and B is connected to C
- A is connected to C

2/10/2005 7:17 PM G.Fagg Cluster Computing 269

MPI "Connected"

Why does "connected" matter?

- MPI_FINALIZE is collective over set of connected processes
- MPI_ABORT may abort all connected processes

How to disconnect?

- ...stay tuned

2/10/2005 7:17 PM G.Fagg Cluster Computing 270

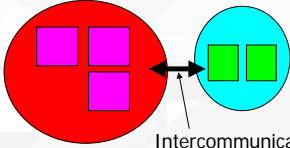
Title goes here

Multi-Stage Spawning

What about multiple spawns?
Can sibling children jobs communicate directly?
Or do they have to communicate through a common parent?
→ Is all MPI dynamic process communication hierarchical in nature?

2/10/2005 7:17 PM G.Fagg Cluster Computing 271

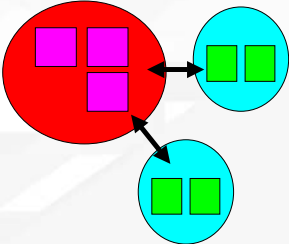
Multi-Stage Spawning



Intercommunicator

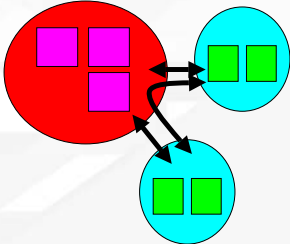
2/10/2005 7:17 PM G.Fagg Cluster Computing 272

Multi-Stage Spawning



2/10/2005 7:17 PM G.Fagg Cluster Computing 273

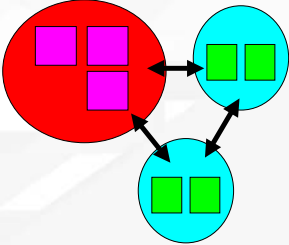
Multi-Stage Spawning



Do we have to do this?

2/10/2005 7:17 PM G.Fagg Cluster Computing 274

Multi-Stage Spawning



Or can we do this?

2/10/2005 7:17 PM G.Fagg Cluster Computing 275