Introduction to MPI
Background on MPI

- MPI - Message Passing Interface
  - Library standard defined by a committee of vendors, implementers, and parallel programmers
  - Used to create parallel programs based on message passing
- 100% portable: one standard, many implementations
- Available on almost all parallel machines in C and Fortran
Where Did MPI Come From?

• Early vendor systems (Intel’s NX, IBM’s EUI, TMC’s CMMD) were not portable (or very capable)

• Early portable systems (PVM, p4, TCGMSG, Chameleon) were mainly research efforts
  – Lacked vendor support
  – Were not implemented at the most efficient level

• MPI Forum organized in 1992 with broad participation by:
  – vendors: IBM, Intel, TMC, SGI, Convex, Meiko
  – portability library writers: PVM, p4
  – users: application scientists and library writers
Novel Features of MPI

- Communicators encapsulate communication spaces for library safety
- Datatypes reduce copying costs and permit heterogeneity
- Multiple communication modes allow precise buffer management
- Extensive collective operations for scalable global communication
- Process topologies permit efficient process placement, user views of process layout
Documentation

• MPI home page: www.mcs.anl.gov/mpi
MPI Implementations

• Most parallel supercomputer vendors provide optimized implementations

• Others:
  – www.lam-mpi.org
  – www-unix.mcs.anl.gov/mpi/mpich
  – GLOBUS:
    www.globus.org/mpi/
Example of MPI program - Hello (C)

#include "mpi.h"
#include <stdio.h>

int main( argc, argv )
int argc;
char *argv[];
{

int rank, size;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
printf( "I am %d of %d\n", rank, size );
MPI_Finalize();
return 0;
}
Example of MPI program- Hello (Fortran)

program main
include 'mpif.h'
integer ierr, rank, size

call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierr )
print *, 'I am ', rank, ' of ', size
end
Some Basic Concepts

• Processes can be collected into groups

• Each message is sent in a context, and must be received in the same context
  – Provides necessary support for libraries

• A group and context together form a communicator

• A process is identified by its rank in the group associated with a communicator

• There is a default communicator whose group contains all initial processes, called `MPI_COMM_WORLD`
MPI Datatypes

• The data in a message to send or receive is described by a triple (address, count, datatype), where

• An MPI datatype is recursively defined as:
  – predefined, corresponding to a data type from the language (e.g., MPI_INT, MPI_DOUBLE)
  – a contiguous array of MPI datatypes
  – an indexed array of blocks of datatypes
  – an arbitrary structure of datatypes

• There are MPI functions to construct custom datatypes, in particular ones for subarrays
MPI Tags

• Messages are sent with an accompanying user-defined integer tag, to assist the receiving process in identifying the message.

• Messages can be screened at the receiving end by specifying a specific tag, or not screened by specifying `MPI_ANY_TAG` as the tag in a receive.

• Some non-MPI message-passing systems have called tags “message types”.

• MPI calls them tags to avoid confusion with datatypes.
Key Concepts of MPI

- Used to create parallel programs based on message passing
  - Normally the same program is running on several different processors
  - Processors communicate using message passing

- Typical methodology:

```
start job on n processors
do i=1 to j
    each processor does some calculation
    pass messages between processor
end do
end job
```
Messages

• Simplest message: an array of data of one type.

• Predefined types correspond to commonly used types in a given language
  
  – MPI_REAL (Fortran), MPI_FLOAT (C)
  – MPI_DOUBLE_PRECISION (Fortran), MPI_DOUBLE (C)
  – MPI_INTEGER (Fortran), MPI_INT (C)

• User can define more complex types and send packages.
Communicators

• Communicator
  – A collection of processors working on some part of a parallel job
  – Used as a parameter for most MPI calls
  – MPI_COMM_WORLD includes all of the processors in your job
  – Processors within a communicator are assigned numbers (ranks) 0 to n-1
  – Can create subsets of MPI_COMM_WORLD
Include files

• The MPI include file
  – C: mpi.h
  – Fortran: mpif.h (a f90 module is a good place for this)

• Defines many constants used within MPI programs

• In C defines the interfaces for the functions

• Compilers know where to find the include files
MPI Basic (Blocking) Send

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

- The message buffer is described by (start, count, datatype).

- The target process is specified by dest, which is the rank of the target process in the communicator specified by comm.

- When this function returns, the data has been delivered to the system and the buffer can be reused.

- The message may not have been received by the target process.
MPI Basic (Blocking) Receive

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

- Waits until a matching (both source and tag) message is received from the system, and the buffer can be used
- source is rank in communicator specified by comm, or MPI_ANY_SOURCE
- tag is a tag to be matched on or MPI_ANY_TAG
- receiving fewer than count occurrences of datatype is OK, but receiving more is an error
- status contains further information (e.g. size of message)
Collective Operations in MPI

- Collective operations are called by all processes in a communicator.
- `MPI_BCAST` distributes data from one process (the root) to all others in a communicator.
- `MPI_REDUCE` combines data from all processes in communicator and returns it to one process.
- In many numerical algorithms, `SEND/RECEIVE` can be replaced by `BCAST/REDUCE`, improving both simplicity and efficiency.
Alternative Set of 6 Functions

• Using collectives:
  - MPI_INIT
  - MPI_FINALIZE
  - MPI_COMM_SIZE
  - MPI_COMM_RANK
  - MPI_BCAST
  - MPI_REDUCE
MPI Basic Send/Receive

• Things that need specifying:
  – How will “data” be described?
  – How will processes be identified?
  – How will the receiver recognize/screen messages?
  – What will it mean for these operations to complete?
Some Basic Concepts

- Processes can be collected into groups
- Each message is sent in a context, and must be received in the same context
  - Provides necessary support for libraries
- A group and context together form a communicator
- A process is identified by its rank in the group associated with a communicator
- There is a default communicator whose group contains all initial processes, called `MPI_COMM_WORLD`
MPI Datatypes

- The data in a message to send or receive is described by a triple (address, count, datatype), where

- An MPI datatype is recursively defined as:
  - predefined, corresponding to a data type from the language (e.g., MPI_INT, MPI_DOUBLE)
  - a contiguous array of MPI datatypes
  - a strided block of datatypes
  - an indexed array of blocks of datatypes
  - an arbitrary structure of datatypes

- There are MPI functions to construct custom datatypes, in particular ones for subarrays
MPI Tags

• Messages are sent with an accompanying user-defined integer tag, to assist the receiving process in identifying the message.

• Messages can be screened at the receiving end by specifying a specific tag, or not screened by specifying MPI_ANY_TAG as the tag in a receive.

• Some non-MPI message-passing systems have called tags “message types”.

• MPI calls them tags to avoid confusion with datatypes.
MPI Basic (Blocking) Send

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

• The message buffer is described by (start, count, datatype).

• The target process is specified by dest, which is the rank of the target process in the communicator specified by comm.

• When this function returns, the data has been delivered to the system and the buffer can be reused.

• The message may not have been received by the target process.
MPI Basic (Blocking) Receive

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

- Waits until a matching (both source and tag) message is received from the system, and the buffer can be used
- source is rank in communicator specified by comm, or MPI_ANY_SOURCE
- tag is a tag to be matched on or MPI_ANY_TAG
- receiving fewer than count occurrences of datatype is OK, but receiving more is an error
- status contains further information (e.g. size of message)
MPI is Simple

- Many parallel programs can be written using just these six functions, only two of which are non-trivial:
  - MPI_INIT
  - MPI_FINALIZE
  - MPI_COMM_SIZE
  - MPI_COMM_RANK
  - MPI_SEND
  - MPI_RECV
Collective Operations in MPI

- Collective operations are called by all processes in a communicator.
- `MPI_BCAST` distributes data from one process (the root) to all others in a communicator.
- `MPI_REDUCE` combines data from all processes in communicator and returns it to one process.
- In many numerical algorithms, `SEND/RECEIVE` can be replaced by `BCAST/REDUCE`, improving both simplicity and efficiency.
6 Major Functions of MPI

• Using collectives:
  - MPI_INIT
  - MPI_FINALIZE
  - MPI_COMM_SIZE
  - MPI_COMM_RANK
  - MPI_BCAST
  - MPI_REDUCE
Blocking and Non-blocking Communication

• So far we have been using blocking communication:
  
  – MPI_Recv does not complete until the buffer is full (available for use).
  
  – MPI_Send does not complete until the buffer is empty (available for use).

• Completion depends on size of message and amount of system buffering.
MPI’s Non-blocking Operations

- Non-blocking operations return (immediately) “request handles” that can be tested and waited on.

  MPI_Isend(start, count, datatype, dest, tag, comm, request)

  MPI_Irecv(start, count, datatype, dest, tag, comm, request)

  MPI_Wait(&request, &status)

- One can also test without waiting:

  MPI_Test(&request, &flag, status)
MPI’s Non-blocking Operations (Fortran)

• Non-blocking operations return (immediately) “request handles” that can be tested and waited on.
  
  Call MPI_Isend(start, count, datatype, dest, tag, comm, request, ierr)
  call MPI_Irecv(start, count, datatype, dest, tag, comm, request, ierr)
  call MPI_Wait(request, status, ierr)

• One can also test without waiting:
  call MPI_Test(request, flag, status, ierr)

Slides prepared by: Farzana Rahman
Multiple Completions

• It is sometimes desirable to wait on multiple requests:

```c
MPI_Waitall(count, array_of_requests, array_of_statuses)

MPI_Waitany(count, array_of_requests, &index, &status)

MPI_Waitsome(count, array_of_requests, array_of_indices, array_of_statuses)
```

• There are corresponding versions of test for each of these.
Multiple Completions (Fortran)

• It is sometimes desirable to wait on multiple requests:
  call MPI_Waitall(count, array_of_requests, array_of_statuses, ierr)
  call MPI_Waitany(count, array_of_requests, index, status, ierr)
  call MPI_Waitsome(count, array_of_requests, array_of_indices, array_of_statuses, ierr)

• There are corresponding versions of test for each of these.
Communication Modes

- MPI provides multiple *modes* for sending messages:
  - **Synchronous mode** (**MPI_Ssend**): the send does not complete until a matching receive has begun. (Unsafe programs deadlock.)
  - **Buffered mode** (**MPI_Bsend**): the user supplies a buffer to the system for its use. (User allocates enough memory to make an unsafe program safe.)
  - **Ready mode** (**MPI_Rsend**): user guarantees that a matching receive has been posted.
    - Allows access to fast protocols
    - undefined behavior if matching receive not posted
- **Non-blocking versions** (**MPI_Issend**, etc.)
- **MPI_Recv** receives messages sent in any mode.
Other Point-to Point Features

- MPI_Sendrecv
- MPI_Sendrecv_replace
- MPI_Cancel
MPI_Sendrecv

• Allows simultaneous send and receive

• Everything else is general.
  
  – Send and receive datatypes (even type signatures) may be different

  – Can use Sendrecv with plain Send or Recv (or Irecv or Ssend_init, …)
Collective Operations in MPI

• Collective operations must be called by all processes in a communicator.

• `MPI_BCAST` distributes data from one process (the root) to all others in a communicator.

• `MPI_REDUCE` combines data from all processes in communicator and returns it to one process.

• In many numerical algorithms, `SEND/RECEIVE` can be replaced by `BCAST/REDUCE`, improving both simplicity and efficiency.
MPI Collective Communication

• Communication and computation is coordinated among a group of processes in a communicator.

• Groups and communicators can be constructed “by hand” or using topology routines.

• Tags are not used; different communicators deliver similar functionality.

• No non-blocking collective operations.

• Three classes of operations: synchronization, data movement, collective computation.
Synchronization

• `MPI_Barrier( comm )`
• Blocks until all processes in the group of the communicator `comm` call it.
Synchronization

• `MPI_Barrier( comm, ierr )`
• Blocks until all processes in the group of the communicator `comm` call it.
MPI Collective Routines

• **Many Routines:** `Allgather`, `Allgatherv`, `Allreduce`, `Alltoall`, `Alltoallv`, `Bcast`, `Gather`, `Gatherv`, `Reduce`, `Reduce_scatter`, `Scan`, `Scatter`, `Scatterv`

• **All versions deliver results to all participating processes.**

• **V versions allow the hunks to have different sizes.**

• **Allreduce, Reduce, Reduce_scatter, and Scan take both built-in and user-defined combiner functions.**
MPI Built-in Collective Computation Operations

- MPI_Max: Maximum
- MPI_Min: Minimum
- MPI_Prod: Product
- MPI_Sum: Sum
- MPI_Land: Logical and
- MPI_Lor: Logical or
- MPI_Lxor: Logical exclusive or
- MPI_Band: Binary and
- MPI_Bor: Binary or
- MPI_Bxor: Binary exclusive or
- MPI_Maxloc: Maximum and location
- MPI_Minloc: Minimum and location
Thank You