Message Passing Interface

Part - I

Dheeraj Bhardwaj
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi – 110016 India
http://www.cse.iitd.ac.in/~dheerajb

Message Passing Interface

Outlines

- **Basics of MPI**
- **How to compile and execute MPI programs?**
- **MPI library calls used in Example program**
- **MPI point-to-point communication**
- **MPI advanced point-to-point communication**
- **MPI Collective Communication and Computations**
- **MPI Datatypes**
- **MPI Communication Modes**
- **MPI special features**

What is MPI?

- A message-passing library specification
  - Message-passing model
  - Not a compiler specification
  - Not a specific product
- Used for parallel computers, clusters, and heterogeneous networks as a message passing library.
- Designed to permit the development of parallel software libraries

Information about MPI

<u>Where to use MPI</u> ?

- You need a portable parallel program
- You are writing a parallel Library
- You have irregular data relationships that do not fit a data parallel model

<u>Why learn MPI</u>?

- Portable & Expressive
- Good way to learn about subtle issues in parallel computing
- Universal acceptance

Information about MPI

MPI Resources

- **The MPI Standard : http://www.mcs.anl.gov/mpi**
- **Using MPI by William Gropp, Ewing Lusk and Anthony Skjellum**
- **Pacheco S. Peter, *Parallel Programming with MPI*, Morgan Kaufman Publishers, Inc., Sanfrancisco, California (1997).**
- **Implementations : MPICH (ftp.mcs.anl.gov), LAM (tbg.osc.edu) ,Vendor specific**

SPMD Program

What is SPMD ?

- **Single Program, Multiple Data**
- **Same program runs everywhere**
- **Restriction on the general message-passing model**
- **Some vendors only support SPMD parallel programs**
- **General message-passing model can be emulated**

SPMD Program

<u>Evaluating General Message Passing with SPMD :C program</u>

```
main (int args, char **argv)
{
  if (process is to become a controller process)
  {
     Controller (/* Arguments /*);
  }
  else
  {
     Worker (/* Arguments /*);
  }
}
```


SPMD Program

<u>Evaluating General Message Passing with SPMD</u> : Fortran

```
PROGRAM

IF (process is to become  a controller process) THEN
    CALL CONTROLLER (/* Arguments /*)
ELSE
    CALL WORKER (/* Arguments /*)
ENDIF

END
```

MPMD Program

What is MPMD (Non-SPMD)?

- **Different programs run on different nodes.**
- **If one program controls the others then the controlling program is called the *Master* and the others are called the *slaves*.**

Dheeraj Bhardwaj <dheerajb@cse.iitd.ac.in>

Compile and Execute MPI programs

How to compile and execute MPI program?

- **Parallel Panther uses mpich-1.2.0 installed the path /usr/local/mpich-1.2.0**
- **mpich has been built and installed on the parallel systems knowing the architecture and the device**
  - **architecture – the kind of processor (example** LINUX**)**
  - **device – how mpich performs communication between processes (example** ch_p4**)**

Dheeraj Bhardwaj <dheerajb@cse.iitd.ac.in>

Compile and Execute MPI programs

<u>How to compile and execute MPI program</u>?

Compiling

- ❖ **On some machines, there is a special command to insure that the program links the proper MPI libraries.**

  mpif77 program.f **mpicc program.c**

- ❖ **Compiling a code : Using Makefile**
  - **Include all files for program, appropriate paths to link MPI libraries**
  - **Used for SPMD and Non-SPMD programs**

(Note that this will differ with different MPI libraries).

Message Passing Interface

Compile and Execute MPI programs

<u>How to compile and execute MPI program</u>?

- ❖ **Execution : mpirun -np 4 a.out**

  (To run a program across multiple machines; np *is the number of processes)*

- ❖ **Execution**
  - **Create ch_p4 procgroup file (File contains users account name, access to the executable of MPI program, number of processes used, for example run.pg)**
  - **Execute the command make (Makefile generates executable (say run)**
  - **Type run on command line**

MPI Basics

Basic steps in an MPI program :

- **Initialize for communications**
- **Communicate between processors**
- **Exit in a "clean" fashion from the message-passing system when done communicating.**

MPI Basics

Format of MPI Calls

C Language Bindings

Return_integer = MPI_Xxxxx(parameter, ...);

- **Return_integer is a return code and is type integer. Upon success, it is set to MPI_SUCCESS.**
- **Note that case is important**
- **MPI must be capitalized as must be the first character after the underscore. Everything after that must be lower case.**
- **C programs should include the file mpi.h which contains definitions for MPI constants and functions**

Format of MPI Calls

Fortran Language Buildings

Call MPI_XXXXX(parameter,..., ierror)

or

call mpi_xxxxx(parameter,..., ierror)

- Instead of the function returning with an error code, as in C, the Fortran versions of MPI routines usually have one additional parameter in the calling list, ierror, which is the return code. Upon success, ierror is set to MPI_SUCCESS.
- Note that case is not important
- Fortran programs should include the file mpif.h which contains definitions for MPI constants and functions

Exceptions to the MPI call formats are timing routines

- Timing routines
  - MPI_WTIME and MPI_WTICK are functions for both C and Fortran
- Return double-precision real values.
- These are not subroutine calls

Fortran

Double precision MPI_WTIME()

C

double precision MPI_Wtime(void);

MPI Point-to-Point Communication

MPI Messages

- **Message :** data (3 parameters) + envelope (3 parameters)
- **Data :** startbuf, count, datatype
- **Envelope :** dest, tag, comm

Data

- **Startbuf:** address where the data starts
- **Count:** number of elements (items) of data in the message

MPI Point-to-Point Communication

Envelope

- **Destination or Source**: Sending or Receiving processes
- **Tag:** Integer to distinguish messages

Communicator

- The communicator is communication "universe."
- Messages are sent or received within a given "universe."
- The default communicator is MPI_COMM_WORLD.

MPI Point-to-Point Communication

Handles

- MPI controls its own internal data structures
- MPI releases 'handles' to allow programmers to refer to these
- C handles are of defined typedefs
- Fortran handles are INTEGERS.

Initialising MPI

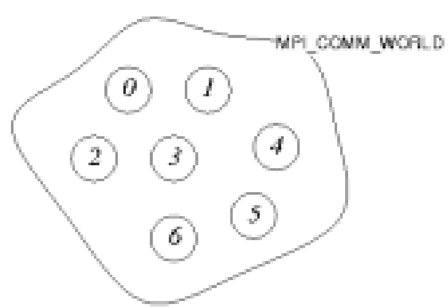
- Must be first routine called.
- C

  int MPI_Init(int *argc,char ***argv);
- Fortran

  MPI_INIT(IERROR)
  integer IERROR

MPI Point-to-Point Communication

MPI_COMM_WORLD communicator



A communicator is MPI's mechanism for establishing individual communication "universes."

MPI Point-to-Point Communication

MPI Message Passing Basics

Questions :

- **What is my processor id number ?**
  MPI_COMM_RANK (Rank starts from the integer value 0 to ….)

Fortran

```
call MPI_COMM_RANK (comm, rank, ierror)
        integer comm, rank, ierror
```

C

```
int MPI_Comm_rank (MPI_Comm comm, int *rank)
```

Message Passing Interface

MPI Point-to-Point Communication

MPI Message Passing Basics

Questions :

- **How many processes are contained within a communicator?**
- **How many processors am I using?**
  MPI_COMM_SIZE

Fortran

```
call MPI_COMM_SIZE (comm, size, ierror)
```

C

```
int MPI_Comm_size (MPI_Comm comm, int *size)
```

Message Passing Interface

MPI Basics

Exiting MPI

- **C**
  int MPI_Finalize()

- **Fortran**

  MPI_FINALIZE(IERROR)

  INTEGER IERROR

Note : Must be called last by all processes.

What makes an MPI Program ?

What makes an MPI Program ?

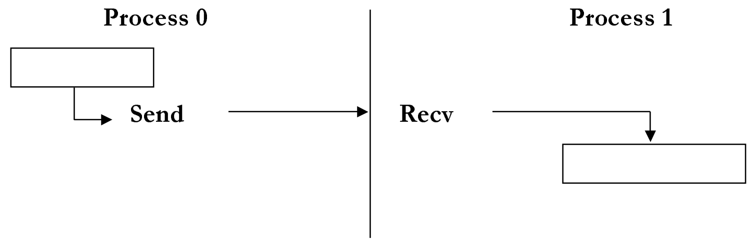
- **Include files**
  mpi.h (C)
  mpif.h (Fortran)

- **Initiation of MPI**
  MPI_INIT

- **Completion of MPI**
  MPI_FINALIZE

MPI Send and Receive

Sending and Receiving messages



Fundamental questions answered

- To whom is data sent?
- What is sent?
- How does the receiver identify it?

Example Program in MPI

To write a simple parallel program in which every process with rank greater than 0 sends a message "Hello_World" to a process with rank 0. The processes with rank 0 receives the message and prints out

Example : A Sample MPI program in Fortran

```
program hello
include 'mpif.h'
integer MyRank, Numprocs, ierror, tag, status (MPI_STATUS_SIZE)
character(12) message
data/message/ 'Hello_World'

call MPI_INIT (ierror)
call MPI_COMM_SIZE (MPI_COMM_WORLD, Numprocs, ierror)
call MPI_COMM_RANK (MPI_COMM_WORLD, MyRank, ierror)
tag = 100
```

Example Program in MPI

(Contd…)

Example : A Sample MPI program in Fortran

```
if (MyRank .eq. 0) then
        do i= 1, Numprocs-1
          call MPI_RECV(message, 12, MPI_CHARACTER, i, tag,
                    MPI_COMM_WORLD, ierror)
          print *, 'node', MyRank, ':', message
        end  do
else
     MPI_SEND(message, 12, MPI_CHARACTER, 0, tag,
                MPI_COMM_WORLD, ierror)
endif
call MPI_FINALIZE (ierror)
end
```

Message Passing Interface

Example Program in MPI

Example : A Sample MPI program in C

```
# include <stdio.h>
# include "mpi.h"
main (int argc, char **argv)
{
  int MyRank, Numprocs, tag, ierror, i;
  MPI_Status status;
  char message[12];
  ierror = MPI_Init (&argc, &argv);
  ierror = MPI_Comm_size (MPI_COMM_WORLD, &Numprocs);
  ierror = MPI_Comm_rank (MPI_COMM_WORLD, &MyRank);
  tag = 100;
  strcpy (message, "Hello_World");
```

Message Passing Interface

Example Program in MPI

Example : A Sample MPI program in C (Contd…)

```
if (MyRank==0) {
    for (i=1; i<Numprocs; i++) {
      MPI_Recv ( message, 12, MPI_CHAR, i, tag, MPI_COMM_WORLD,
                  &status);
     printf ("node %d : %s \n", MyRank, message);
    }
} else
    ierror = MPI_Send( message, 12, MPI_CHAR,0, tag,
                   MPI_COMM_WORLD);
ierror = MPI_Finalize( );
}
```


About Example Program in MPI

MPI Routines used in Hello World Program :

MPI_INIT

MPI_INIT must be the first MPI routine called in each process, and it can only be called once. It establishes the necessary environment for MPI to run.

Synopsis:

C int MPI_Init(int *argc, char ***argv);

Fortran

```
integer error
call MPI_INIT(ierror)
```

About Example Program in MPI

<u>MPI Routines used in Hello World Program</u> :

MPI_COMM_SIZE

MPI_COMM_SIZE returns the number of processes within a communicator. A communicator is MPI's mechanism for establishing individual communication "universes"

MPI_COMM_WORLD - predefined communicator; contains all processes.

Synopsis:

C int MPI_Comm_size (MPI_COMM_WORLD, int *size);

Fortran

MPI_COMM_SIZE (comm, size, ierror)

integer comm, size, ierror

Message Passing Interface

About Example Program in MPI

<u>MPI Routines used in Hello World Program</u>:

MPI_COMM_RANK

MPI_COMM_RANK returns the calling process's rank in the specified communicator. Rank is an integer in the range 0 through size-1 (where size is the number of processors returned by MPI_Comm_size) and specifies a particular process.

Synopsis :

C int MPI_Comm_rank (MPI_COMM_WORLD, int *rank);

Fortran

MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierror)

integer comm, rank, ierror

Message Passing Interface

About Example Program in MPI

<u>MPI Routines used in Hello World Program:</u>
MPI_FINALIZE

MPI_FINALIZE - last call you should make in each process; ensures that MPI exists cleanly. All communication should be completed before calling MPI_FINALIZE

Synopsis :

C int MPI_Finalize( );

Fortran

int MPI_FINALIZE (ierror)
integer ierror

MPI Point-to-Point Communication

MPI Routines used in Hello_World Program :

MPI_Send/MPI_Recv

Synopsis :

C int MPI_Send (void* buf, int count, MPI_Datatype datatype, int dest, int tag MPI_Comm comm) ;

int MPI_Recv(void*buf, int count, MPI_Datatype datatype, int source, int tag MPI_Comm comm, MPI_Status *status);

Fortran

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

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

<type> bufffer,

integer count, datatype, dest, source, tag, comm, ierror

MPI Point-to-Point Communication

MPI Message Passing : Send and Receive

Fortran

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

[*IN buf*] initial address of send buffer (choice)

[*IN count*] number of elements in send buffer (nonnegative integer)
[*IN datatype*] datatype of each send buffer element (handle)
[*IN dest*] rank of destination (integer)
[*IN tag*] message tag (integer)
[*IN comm*] communicator (handle)

C

int MPI_Send (datatype, int dest, int tag, MPI_Comm comm void* buf, int count, MPI_Datatype);

MPI Point-to-Point Communication

MPI Message Passing : Send and Receive

Fortran

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

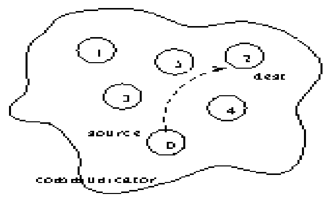
[OUT buf] initial address of receive buffer (choice)
[IN count] number of elements in receive buffer (integer)
[IN datatype] datatype of each receive buffer element (handle)
[IN source] rank of source (integer)
[IN tag] message tag (integer)
[IN comm] communicator (handle)
[OUT status] status object (Status)

C

int MPI_Recv (void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status);

MPI Point-to-Point Communication

Information on MPI Send and Recv



- **Communication between two processes**
- ***Source*** **process sends message to** ***destination*** **process**
- **Communication takes place within a** ***communicator***
- **Destination process is identified by its** ***rank*** **in the communicator**

MPI Point-to-Point Communication

MPI_Send and **MPI_Recv**

- **MPI provides for point-to-point communication between pair of processes**
- **Message selectively is by rank and message tag**
- **Rank and tag are interpreted relative to the scope of the communication**
- **The scope is specified by the communicator**
- **Rank and tag may be wildcarded**
- **The components of a communicator may not be wildcarded**

MPI Basic Datatypes

MPI Basic Datatypes - Fortran

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

MPI Basic Datatypes

MPI Basic Datatypes - C

MPI Datatype	C datatype
MPI_CHAR	Signed char
MPI_SHORT	Signed short int
MPI_INT	Signed int
MPI_LONG	Signed long int
MPI_UNSIGNED_CHAR	Unsigned char
MPI_UNSIGNED_SHORT	Unsigned short int
MPI_UNSIGNED	Unsigned int
MPI_UNSIGNED_LONG	Unsigned long int
MPI_FLOAT	Float
MPI_DOUBLE	Double
MPI_LONG_DOUBLE	Long double
MPI_BYTE	
MPI_PACKED	

Is MPI Large or Small?

Is MPI Large or Small?

- MPI is large (125 Functions)
  - MPI's extensive functionality requires many functions
  - Number of functions not necessarily a measure of complexity
- MPI is small (6 Functions)
  - Many parallel programs can be written with just 6 basic functions
- MPI is just **right** candidate for message passing
  - One can access flexibility when it is required
  - One need not master all parts of MPI to use it

Is MPI Large or Small?

The MPI Message Passing Interface Small or Large

MPI can be small.

One can begin programming with 6 MPI function calls

MPI_INIT	*Initializes MPI*
MPI_COMM_SIZE	*Determines number of processors*
MPI_COMM_RANK	*Determines the label of the calling process*
MPI_SEND	*Sends a message*
MPI_RECV	*Receives a message*
MPI_FINALIZE	*Terminates MPI*

MPI can be large

One can utilize any of 125 functions in MPI.

References

1. Gropp, W., Lusk, E. and Skjellum, A., Using MPI: Portable Parallel Programming with Message-Passing Interface, The MIT Press, 1999.

1. Pacheco, P. S., Parallel Programming with MPI, Morgan Kaufmann Publishers, Inc, California (1997).
2. Vipin Kumar, Ananth Grama, Anshul Gupta, George Karypis, Introduction to Parallel Computing, Design and Analysis of Algorithms, Redwood City, CA, Benjmann/Cummings (1994).
3. William Gropp, Rusty Lusk, Tuning MPI Applications for Peak Performance, Pittsburgh (1996)