Message Passing Interface

SHWETA DAS

shwetad@cdac.in

AGENDA

- ✓ What is MPI ?
- ✓ What does MPI offer ?
- ✓ Principle of Message Passing
- ✓ What is Message Passing ?
- ✓ Message Passing Specification
- ✓ Point to Point Communication
- ✓ Communication types
- ✓ MPI Programming

What is MPI?

- MPI stands for Message Passing Interface.

- Message Passing Interface (MPI), is a message-passing library standard, that was finalized was published in May 1994 by the University of Tennessee. Contains specifications of functions and macros that can be used in C, FORTRAN, and C++ programs.

- However, programs written in message-passing style can run on any architecture that supports such model such as:

  Distributed or shared-memory multi-processors

  Networks of workstations

  Single processor systems

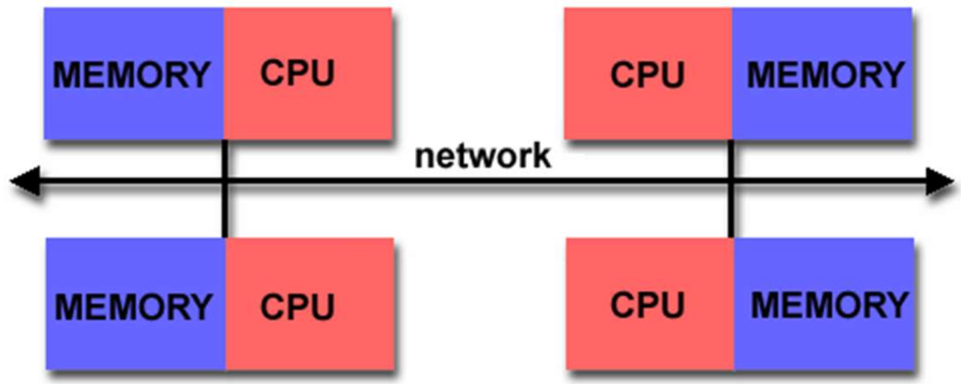
What does MPI offer?

- **Standardization : MPI is the first standardized vendor independent message passing library.**
- **Portability : Programs using MPI runs on any platform, which has a MPI implementation without any need to modify the codes. The programs are independent of machine architecture and type of network employed to transfer data from one processor to another.**
- **Availability : MPICH is free. Every major vendor of UNIX workstations have their own implementation of MPI.**

Principles of Message Passing

There are two key attributes that characterized the Message Passing Programming Paradigm :

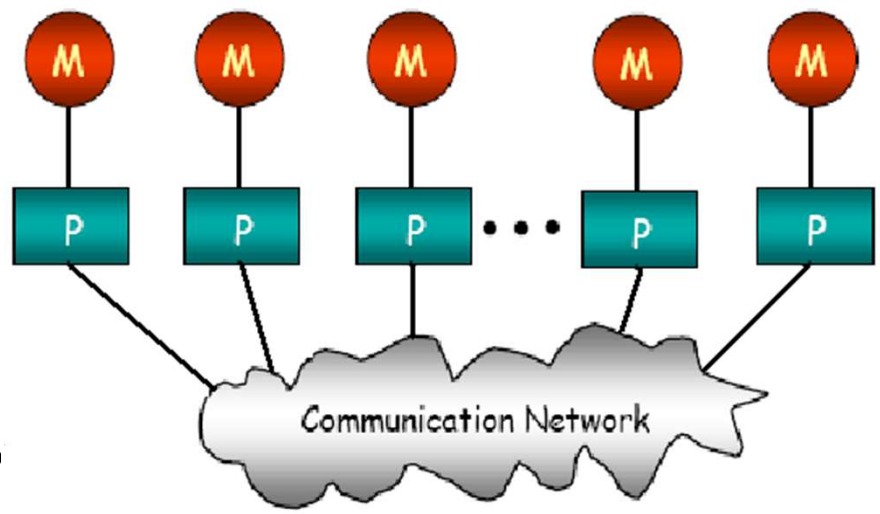
1.Partitioned Address Space



2. Explicit Programming

What is Message Passing?

- Many instances of sequential paradigm considered together.
- Programmer imagines several processors, each with own memory, and writes a program to run on each processor
- Processes communicate b sending each other messages



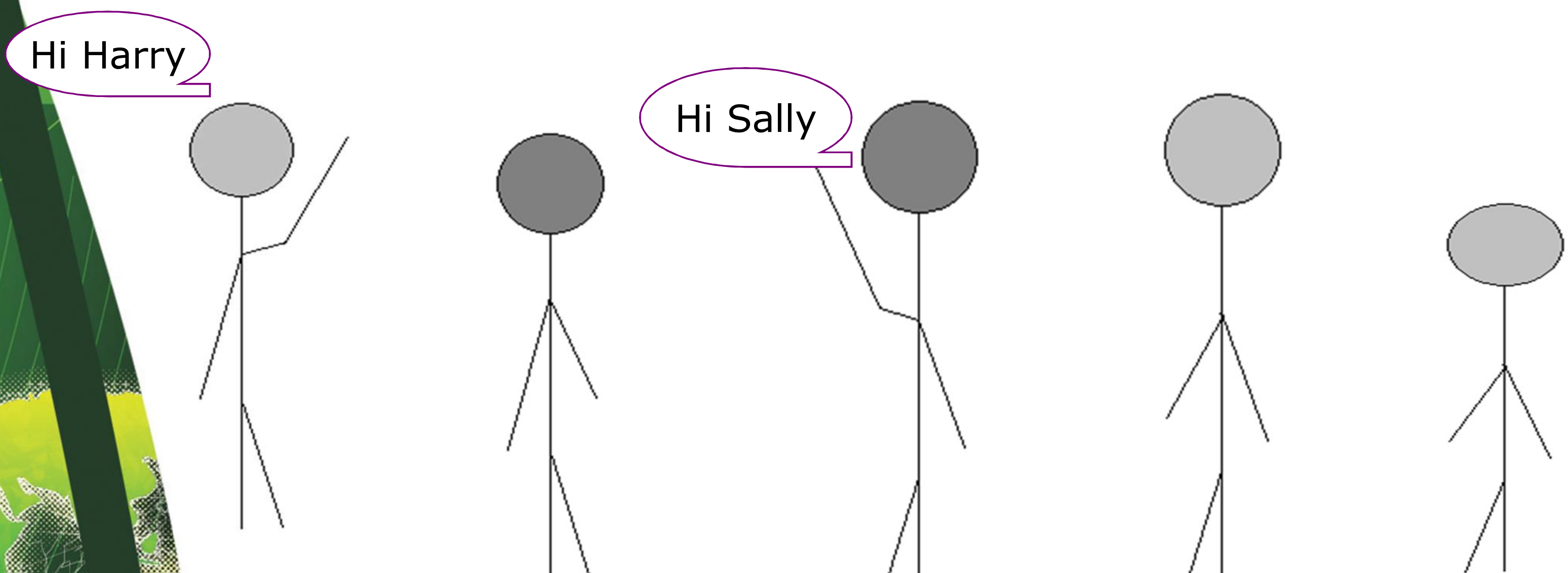
Message Passing Specification

- Message passing system provides following information to specify the message transfer :
  - Which process is sending the message.
  - Where is the data on the sending process.
  - What kind of data is being sent.
  - How much data is there.
  - Which process(s) are receiving the message.
  - Where should the data be left on the receiving process.
  - How much data is receiving process prepared to accept.

Point to Point Communication

Simplest form of message communication

- Message is sent from a sending process to a receiving process Only these two process need to know anything about the message.



Is MPI large or small ?

MPI is Large (more than 340 functions)

- Many feature requires extensive API
- Complexity of use not related to number of functions

MPI is small (6 functions)

- All that's needed to get started are only 6 functions

MPI is just right!

- Flexibility available when required
- Can start with small subset

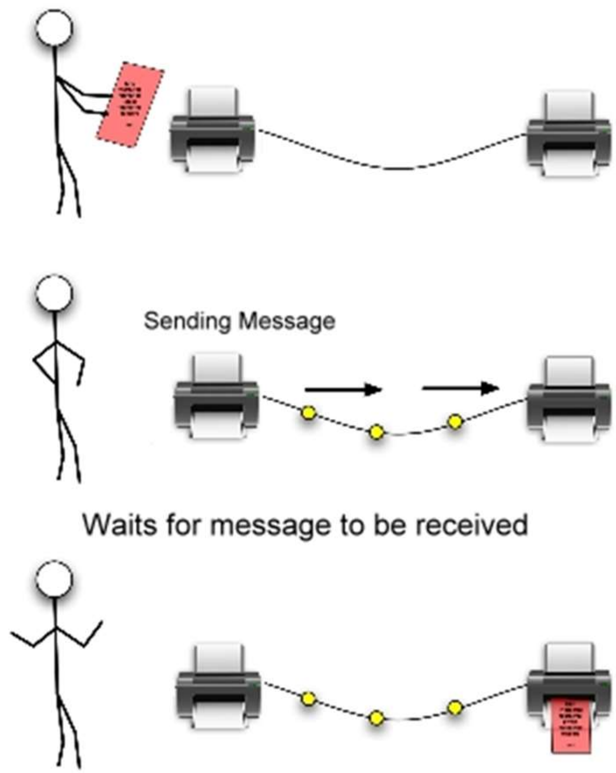
Building blocks: Send and Receive

- **Basic operations in Message-passing programming paradigm are send and receive.**

send(void *sendbuf, int noelems, int dest)

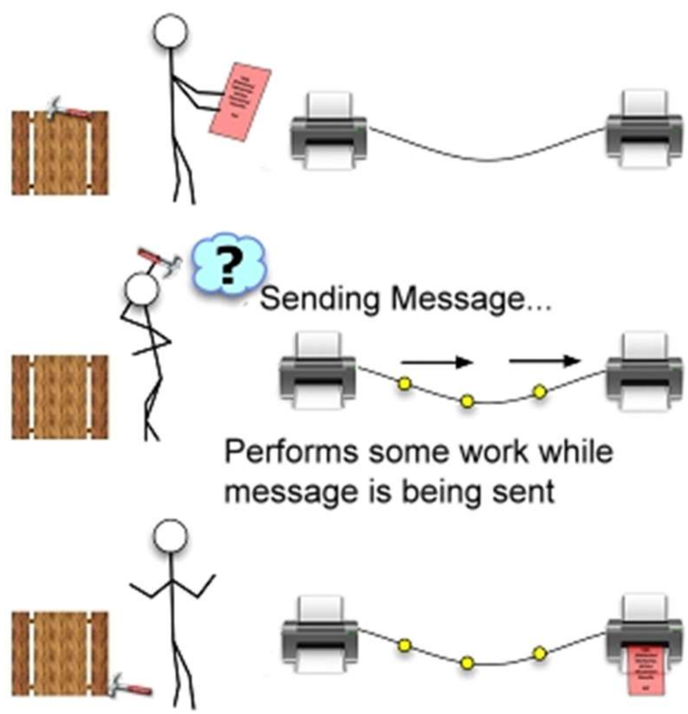
receive(void *recvbuf, int noelems, int source)

Blocking Operation



Non-Blocking Operation

An operation, such as sending or receiving a message, that returns immediately whether or not the operation was completed.



Send and Receive: A simple example

```
P0                              P1

a = 100;                        receive(&a, 1, 0);
send(&a, 1, 1);                 printf("%d\n", a);
a = 0;
```

What gets printed at P1?

How to ensure semantic consistency?

Getting Started With MPI Programming

Header File:

Required for all programs/routines which make MPI library calls.

C Include File:

```
# include "mpi.h"
```

Fortran Include File:

```
include "mpif.h"
```


14

Format of MPI Calls

C Binding	
Format:	`rc = MPI_Xxxxx(parameter, ... )`
Example:	`rc = MPI_Bsend(&buf,count,type,dest,tag,comm)`
Error code:	Returned as "rc". MPI_SUCCESS if successful

Fortran Binding	
Format:	`CALL MPI_XXXXX(parameter,..., ierr)` `call mpi_xxxxx(parameter,..., ierr)`
Example:	`CALL MPI_BSEND(buf,count,type,dest,tag,comm,ierr)`
Error code:	Returned as "ierr" parameter. MPI_SUCCESS if successful

General MPI Program Structure

MPI include file

.
.
.

Initialize MPI environment

.
.
.

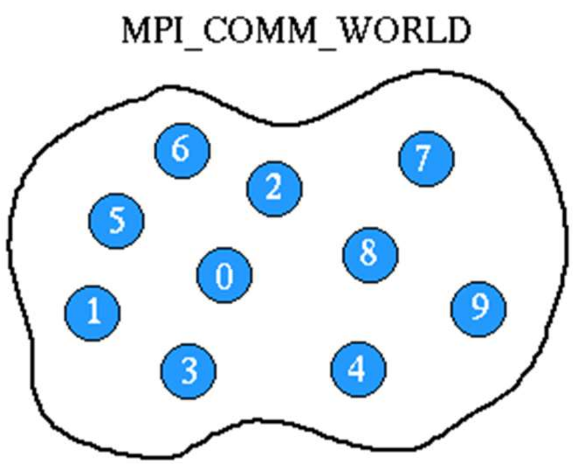
Do work and make message passing calls

.
.
.

Terminate MPI Environment

Communicators and Groups

➢ MPI uses objects called communicators and groups to define which collection of processes may communicate with each other. Most MPI routines require you to specify a communicator as an argument.



Starting with MPI Programming

Six basic functions to start :

▶ **MPI_INIT**	**Initialize MPI Environment.**
▶ **MPI_FINALIZE**	**Finish MPI Environment.**
▶ **MPI_COMM_RANK**	**Get the processor rank.**
▶ **MPI_COMM_SIZE**	**Get the number of processors.**
▶ **MPI_Send**	**Send data to another processor.**
▶ **MPI_Recv**	**Get data from another processor.**

Hello World

```
#include "mpi.h"                                        Header File
#include <stdio.h>
int main( argc, argv)
int argc; char **argv;          Communicator
{
int rank, size;
MPI_Init( &argc, &argv );                               Initializing MPI
MPI_Comm_rank(MPI_COMM_WORLD, &rank );                  Rank
MPI_Comm_size(MPI_COMM_WORLD, &size );                  Size
/* Your code here */
printf("Hello world! I'm %d of %d\n", rank, size);
MPI_Finalize();                                         Exiting MPI
return 0;
}
```

Hello world output

To Compile

mpicc hello.c -o hello

To run with 4 processes

mpirun -np 4 hello

Output

Hello world! I'm 2 of 4

Hello world! I'm 1 of 4

Hello world! I'm 3 of 4

Hello world! I'm 0 of 4

Note - Order of output is not specified by MPI

Starting and Terminating MPI library

```
int MPI_Init(int *argc, char **argv)   - C Program
MPI_INIT (ierr)                        - Fortran Program
```

- Initializes the MPI environment
- Called prior to any calls to other MPI routines

```
int MPI_Finalize()                     - C Program
MPI_FINALIZE (ierr)                    - Fortran Program
```

- Performs various clean-ups tasks to terminate the MPI environment.
- Always called at end of the computation.

Getting communicator information

- int MPI_Comm_size(MPI_Comm comm, int *size) – C Prog

  MPI_COMM_SIZE (comm,size,ierr) – Fortran Prog

- Returns in size, the number of processes in communicator comm.

- int MPI_Comm_rank(MPI_Comm comm, int *rank) – C Prog

  MPI_COMM_RANK (comm, rank, ierr) – Fortran Prog

- Returns rank of the process in the communicator
- Rank ranges from 0 to (size of the communicator – 1)

MPI Send

```
int MPI_Send( void *buf,   // Data To be sent
              int count, // Total Data Elements to be sent
MPI_Datatype datatype,    // Datatype of the data to be sent
              int dest,  // Processor to which data is being sent
                int tag, // To distinguish from diff types of msg
      MPI_Comm comm) // Communicator
```

MPI Receive

```
int MPI_Recv(void *buf,   // Data To be Receive
             int count,  // Total Data Elements to be recv
MPI_Datatype datatype,  // Datatype of the data to be recv
             int source,  // Processor from where data is
    being sent
             int tag,  // To distinguish from diff types of
    msg
             MPI_Comm comm,  // Communicator
             MPI_Status *status)
```

Wildcards

Allow you to not necessarily specify a tag or source

Example

- MPI_ANY_SOURCE and MPI_ANY_TAG are wild cards

Status structure is used to get wildcard values

MPI Status

The status parameter returns additional information for some MPI routines

– Additional Error status information

– Additional information with wildcard parameters

C declaration : a predefined struct

– MPI_Status status;

Fortran declaration : an array is used instead

– INTEGER STATUS(MPI_STATUS_SIZE)

MPI Status

Accessing status information

- **The tag of a received message**
  - **– C : status.MPI_TAG**
  - **– Fortran : STATUS(MPI_TAG)**
- **The source of a received message**
  - **– C : status.MPI_SOURCE**
  - **– Fortran : STATUS(MPI_SOURCE)**
- **The error code of the MPI call**
  - **– C : status.MPI_ERROR**
  - **– Fortran : STATUS(MPI_ERROR)**
- **Other uses...**

MPI Data Types

MPI Types

- MPI has many different predefined datatypes.
- Can be used in any communication operation.

C Data Types		Fortran Data Types	
MPI_CHAR	signed char	MPI_CHARACTER	character(1)
MPI_SHORT	signed short int		
MPI_INT	signed int	MPI_INTEGER	integer
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_REAL	real
MPI_DOUBLE	double	MPI_DOUBLE_PRECISION	double precision
MPI_LONG_DOUBLE	long double		
		MPI_COMPLEX	complex
		MPI_DOUBLE_COMPLEX	double complex
		MPI_LOGICAL	logical
MPI_BYTE	8 binary digits	MPI_BYTE	8 binary digits
MPI_PACKED	data packed or unpacked with MPI_Pack()/ MPI_Unpack	MPI_PACKED	data packed or unpacked with MPI_Pack()/ MPI_Unpack

Non-Blocking Communication Operations

int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_request *request)

int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_request *request)

Detecting Completion

- **MPI_Test tests for the completion of a send or receive.**

int MPI_Test (MPI_Request *request, int *flag, MPI_Status *status)

- request, status as for MPI_Wait.
- does not block.
- flag indicates whether operation is complete or not.
- enables code which can repeatedly check for communication completion.

Pros

Pros of MPI

- **Runs on either shared or distributed memory architectures.**
- **Each process has its own local variables.**
- **Distributed memory computers are less expensive than large shared memory computers.**
- **Explicit programming.**

Cons

Cons of MPI

- **Requires more programming changes to go from serial to parallel version.**
- **Can be harder to debug.**
- **Performance is limited by the communication network between the nodes.**

Thank You