Parallel Programming with MPI (Message Passing Interface)

Distributed Memory Programming Model

Index

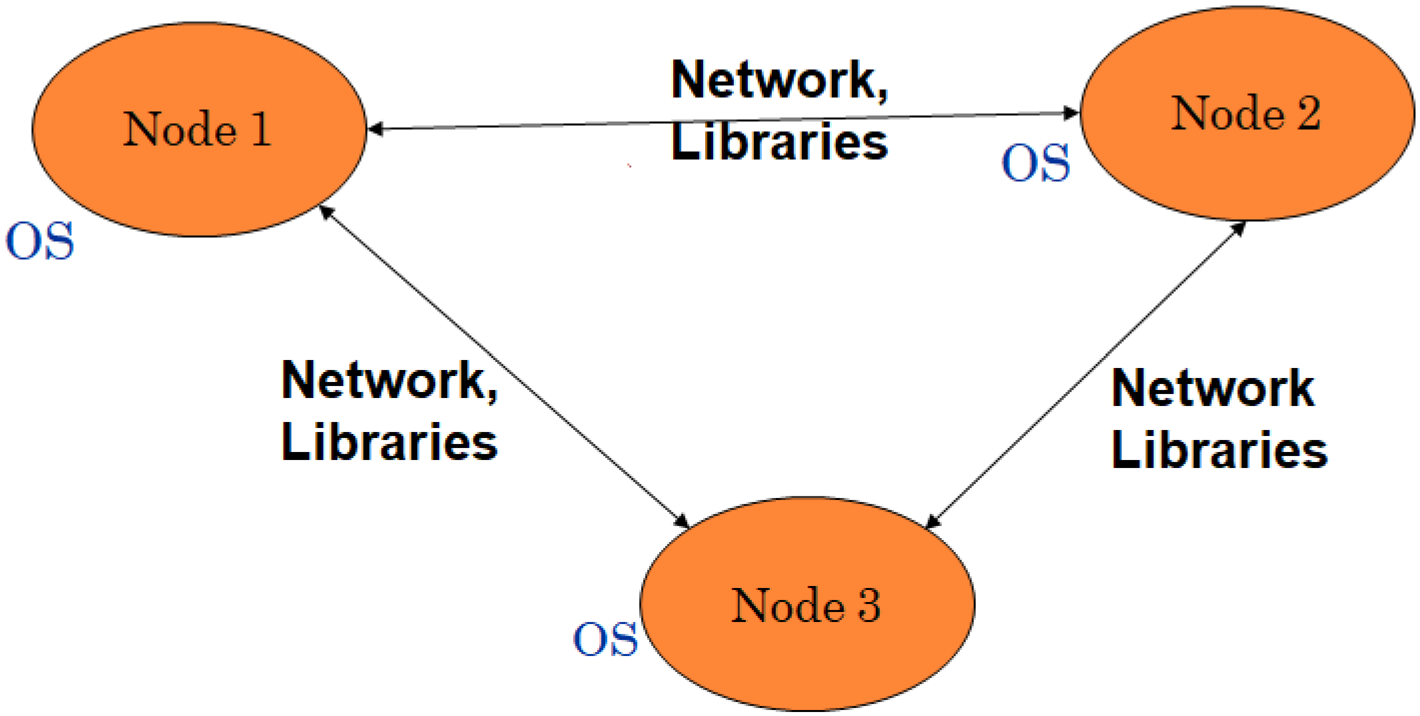
- Parallel Programming models
- Introduction to MPI
- Structure of MPI and Simple MPI program
- Point to point communication calls
- Collective Communication calls

Sample Parallel Programming Models

- Shared Memory Programming
  - Processes share memory address space (threads model)
- Transparent Parallelization
  - Compiler works magic on sequential programs
- Directive-based Parallelization
  - Compiler needs help (e.g., OpenMP)
- Message Passing
  - Explicit communication between processes (like sending and receiving emails)

Distributed Architecture

- What differentiates a Supercomputer from a normal computer network



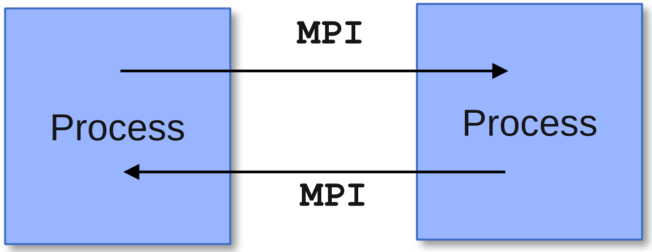
- Multiple Nodes connected together to form a Computer Cluster

Distributed Architecture…

- Each node may have many processor cores
- High Speed Network
- Parallel Programming Libraries
- Simplified Operating System – mostly Unix flavors

Introduction to MPI

- A message passing library specification
- Message passing among processes in parallel computing
- Meant for clusters and network of workstations
- Message Passing Standard for Parallel Programs
  - MPI Implementation left to individual vendors.
  - Most commonly supports C, Fortran, C++ and Python programs
- Inter-process communication consists of
  - Synchronization
  - movement of data from one process's address space to another's.



MPI – Characteristics

- Separate memory address spaces Message passing among processes in parallel computing

- Explicit data and work allocation by the user

- Asynchronous parallelism

- Explicit interaction

MPI – How it works?

- A parallel computation consists of no. of processes
- Each process has its local variables
- No mechanism for any process to directly access the memory of another
- Sharing the data among the process through message passing
- Data transfer requires cooperative operations by each process
- Different processes need not be running on different processors

Requisites for MPI Implementations

The Following features are needed in MPI Implementations

- Allow efficient Communication between processes.

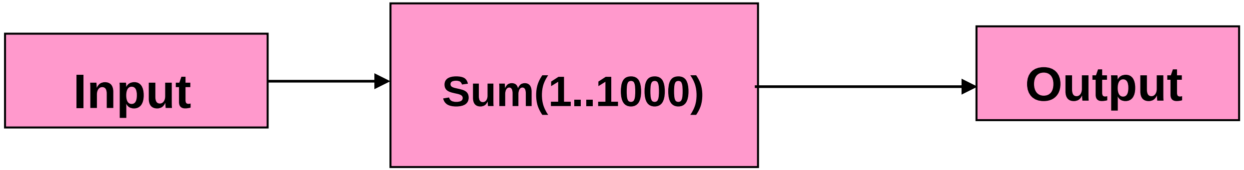
- Allow overlap of computation and communication

Important considerations while using MPI

- All parallelism is explicit: the programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs

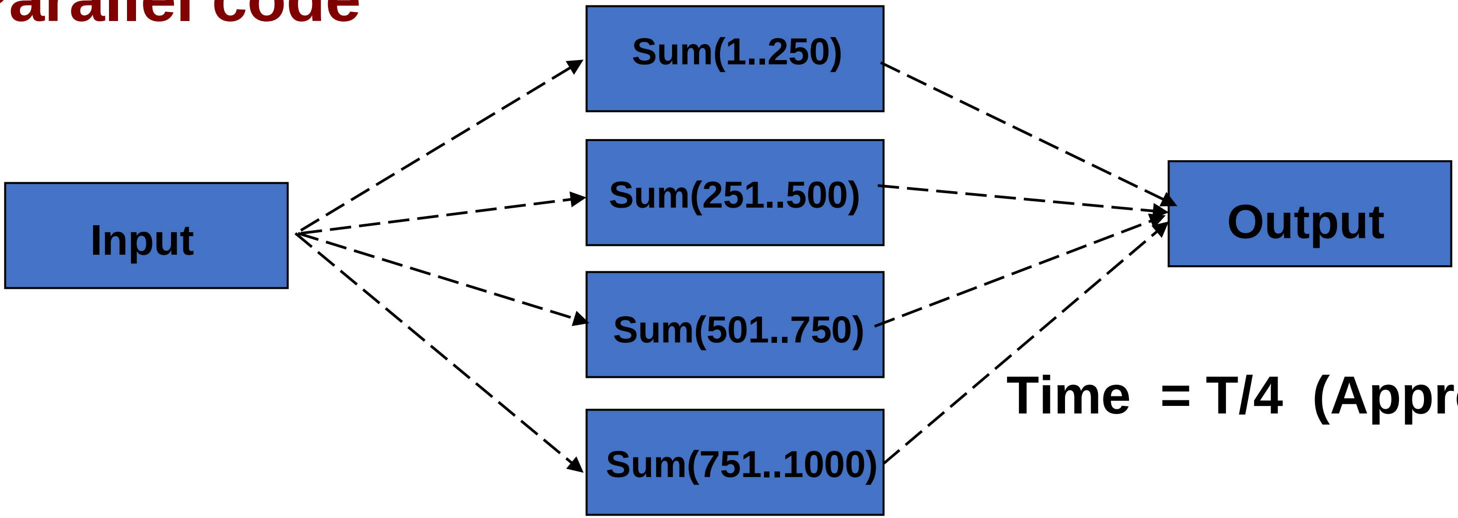
Why Parallelization of codes?

Sequential code



Time taken = T

Parallel code



Time = T/4 (Approx.)

Structure of MPI Program

MPI include file

.
.
.

Initialize MPI environment

.
.
.

Do work and make message passing calls

.
.
.

Terminate MPI Environment

BASIC MPI Calls

- Many parallel programs can be written using just these six mpi calls,
  - **MPI_INIT** – initialize the MPI library
  - **MPI_COMM_SIZE** - get the size of a communicator
  - **MPI_COMM_RANK** – get the rank of the calling process in the communicator
  - **MPI_SEND** – send a message to another process
  - **MPI_RECV** – receive a message to another process
  - **MPI_FINALIZE** – clean up all MPI state (must be the last MPI function called by a process)

Simple MPI Program

```
#include <stdio.h>
#include <mpi.h>
main(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("Process:%d out of %d\n", rank, size);
        MPI_Finalize();
}
```

Compiling and Running MPI Programs

C program compilation

- **$ mpicc filename.c**

Fortran program compilation

- **$ mpiifort filename.f90   (Fortran)**

Running C and Fortran mpi applications

- **$ mpirun –np <np> <exe-name>**
- **$ mpirun –np <np> –machinefile <hostfilesname> <exe-name>**

Where
-np: **No. of processes to run on**
-machinefile : **List of possible machines to run the executable**

Only 5 things to know before learning MPI function calls

- Communications
  - Point to point communication
  - Collective communication
- MPI Data types
- Format of MPI Calls
- Communicator
- Rank

MPI Communications

Point-to-Point Communication

- Blocking
- Non Blocking

Collective Communication

- Synchronization
- Collective computation
- Data Movement

MPI Basic Data types- C Programs

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 Basic Data types – Fortran programs

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	

Format of MPI Calls - C Language

Format	**Rc = MPI_Xxxxx(parameter, ...)**
Ex	**Rc = MPI_Send(&buf,count,type,dest,tag,comm)**

- All MPI Functions are case sensitive.
- All MPI calls begin with MPI_ followed by actual function name.
- C programs should include the file mpi.h
- Return_integer Rc is of type integer. It is set to MPI_SUCCESS upon success.

Format of MPI Calls- Fortran

(Contd…)

	CALL MPI_XXXXX(parameter,..., ierr)
Ex	**CALL MPI_SEND(buf,count,type,dest,tag,comm,ierr)**

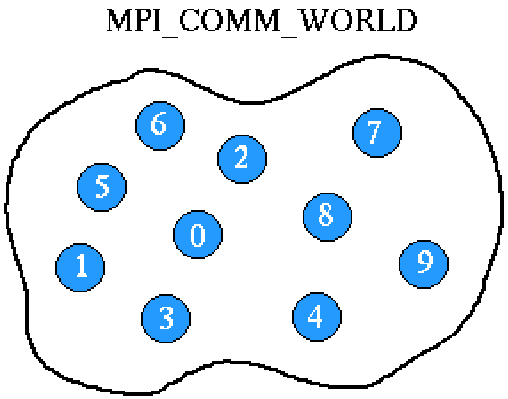
- Case insensitive.
- Fortran programs should include the file mpif.h
- Additional parameter ierr to take care of the function status

Communicator

- The communicator is Communication Universe.
  - Processes that are allowed to communicate

- Messages are sent / received within a given Universe.

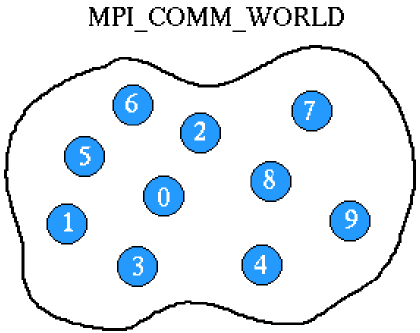
- MPI_COMM_WORLD is the default communicator.



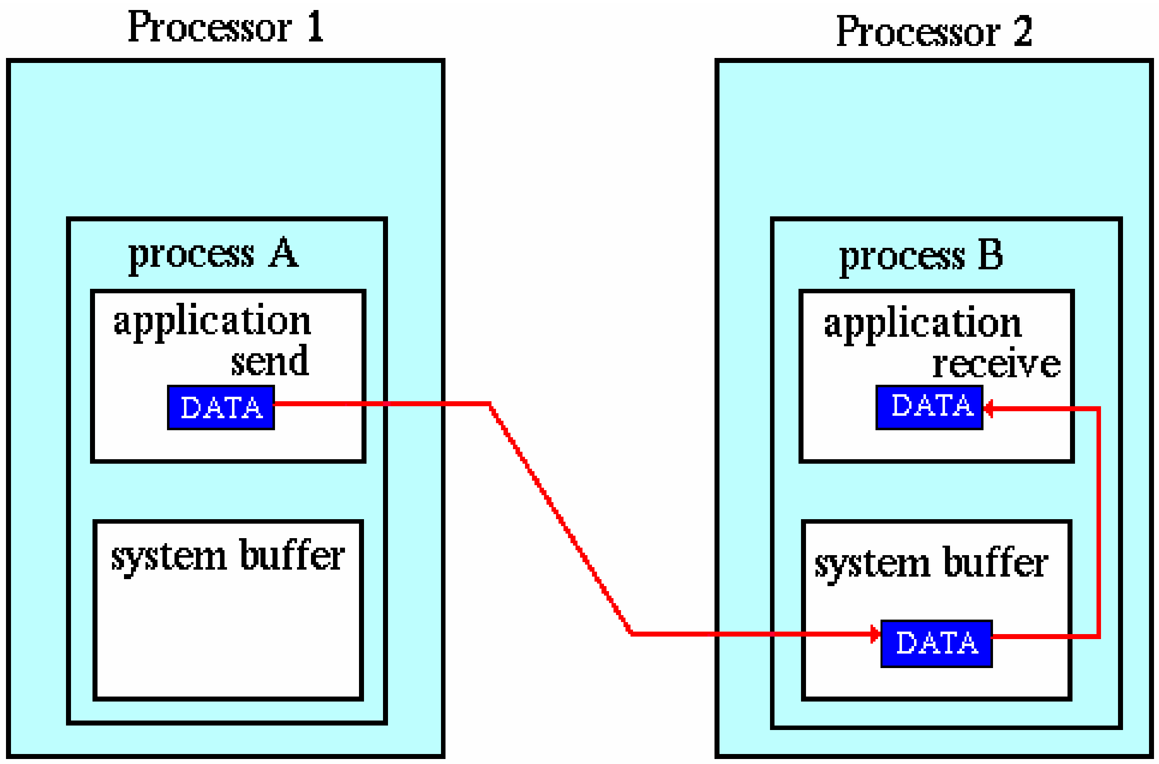
What is 'Rank' in Communicator?

Rank:

- Unique integer identifier for every process.
- Assigned by the system during initialization.
- Ranks are contiguous & begin at zero.
- Used to specify the source and destination of messages.
- Often used to control program execution (if rank=0 do this / if rank=1 do that).



MPI Point-to-Point Communication



Path of a message buffered at the receiving process

MPI Send and Receive: Syntax

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

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

When MPI_Recv routine returns, the received message data have been copied into the buffer; and the tag, source, and actual count of data received are available via the status argument

MPI Send and Receive: Blocking

```
strcpy(Message, "Hello India");
Destination = 0; source= 1; BUFFER_SIZE = 32;
MPI_Send (Message, BUFFER_SIZE, MPI_CHAR, Destination, tag,MPI_COMM_WORLD);

MPI_Recv(Message, BUFFER_SIZE, MPI_CHAR, source, tag,MPI_COMM_WORLD, &status);
```

Arguments of MPI_send & Receive:

- Message: Address where the data starts.
- BUFFER_SIZE: Number of elements (items) of data in the message.
- MPI_CHAR: Datatype of the message passed / received.
- Destination/source: Rank of Receiving or Sending processes
- tag: Integer to distinguish messages
- MPI_COMM_WORLD: Communicator, status: contains status

Performance Analysis of MPI Programs

- Why Performance Analysis of MPI Applications
  - To get the benefit of parallelization
  - Tuning the Parallel algorithm
  - Tuning the application
- Communication Vs Computation Time
- Function Profiling

Challenges Involved in Debugging Parallel Applications

- Complexity increases with the multiplicity of processes
- Processes may be running on different machines.
- Co-coordinating the communication between processes.
- Processes may be operating on different data sets.

Debugging Parallel Applications

- Through Command Line Interface
  - GDB
  - Valgrind
- Through Graphical User Interface (GUI)
  - TotalView debugger
  - Valgrind

Collective Communication - Overview

- Involves the sending and receiving of data among processes
- All processes need to participate in the communication.
- Usage of Point-to-point communication routines
- Customized communicator
- No message tags used

Collective Communications – Characteristics

- Communications are locally blocking
- Some routines use a root process to originate or receive all data
- Synchronization is not guaranteed (implementation dependent)
- Data amounts must exactly match
- Different types

  ** One-to-All

  ** All-to-One

  ** All-to-All

Collective Communications – Patterns

- Synchronization (Ex: Barrier synchronization)
- Data movement (Ex: Broadcast, Scatter, Gather)
- Collective computation (Ex: SUM, MAX, MIN, etc)

MPI Collective Communication - Calls

- Barrier synchronization
- Broadcast from one member to all other members
- Gather data from an array spread across processors into one array
- Scatter data from one member to all members
- All-to-all exchange of data
- Global reduction (e.g., sum, min of "common" data elements)
- Scan across all members of a communicator

Data Movement – Calls

- Broadcast
- Gather
- Scatter
- Allgather
- Alltoall

MPI-2 Overview

- Extensions to the message-passing model
  - ** Parallel I/O
  - ** One-sided operations
  - ** Dynamic process management
- Making MPI more robust and convenient
  - ** C++ and Fortran 90 bindings
  - ** External interfaces, handlers
  - ** Extended collective operations
  - ** Language interoperability
  - ** MPI interaction with threads

MPI-3 Overview

- Major update to the MPI standard.

- Includes the extension of collective operations to include Non-blocking versions

- Extensions to the one-sided operations, and a new Fortran 2008 binding.

- Deprecated C++

Thank You!!