# ASSIGNMENT NO.:

**Title:** Write a MPI Program for calculating the value of Pi on single and multiple machine clusters.

**Aim:** Write a MPI Program for calculating the value of Pi on single and multiple machine in clusters.

## Objective:

1. To understand concept of **Message Passing Interface (MPI)**
2. To execute computation of Pi on single and multiple machine clusters.

## Theory:

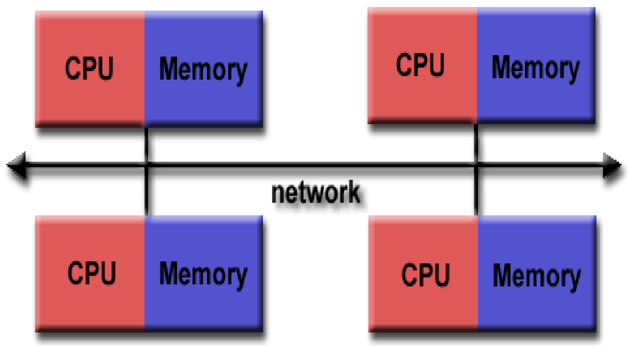
### Concept of MPI:

MPI is a specification for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be.

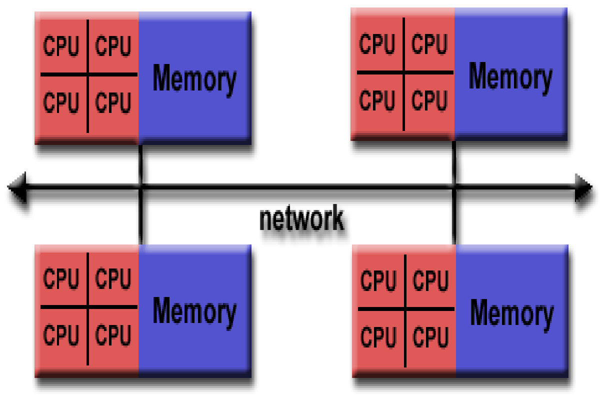
MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process.
Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs.

The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum, which has over 40 participating organizations, including vendors, researchers, software library developers, and users.
The goal of the Message Passing Interface is to establish a portable, efficient, and flexible standard for message passing that will be widely used for writing message passing programs.
MPI is not an IEEE or ISO standard, but has in fact, become the "industry standard" for writing message passing programs on HPC platforms.

### MPI Programming Model:

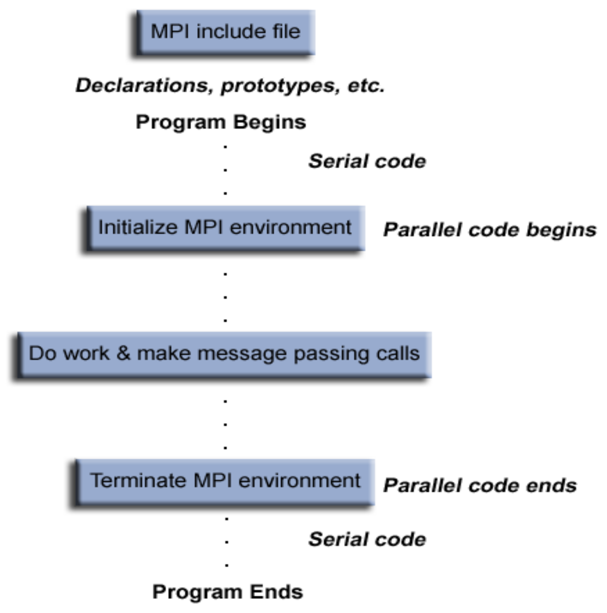


**Simple P4 machines with single core**



**Multicore machines like dual core, i3,i5,i7**

## General MPI Program Structure

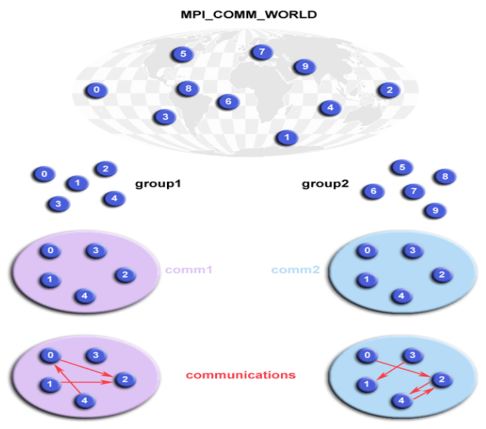

MPI include file
Declarations, prototypes, etc.
Program Begins
Serial code
Initialize MPI environment
Parallel code begins
Do work & make message passing calls
Terminate MPI environment
Parallel code ends
Serial code
Program Ends

## 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.

**MPI_COMM_WORLD** whenever a communicator is required - it is the predefined communicator that includes all of your MPI processes.



**Level of Thread Support:**
**MPI libraries vary in their level of thread support:**

- MPI_THREAD_SINGLE - Level 0: Only one thread will execute.
- MPI_THREAD_FUNNELED - Level 1: The process may be multi-threaded, but only the main thread will make MPI calls - all MPI calls are funneled to the main thread.
- MPI_THREAD_SERIALIZED - Level 2: The process may be multi-threaded, and multiple threads may make MPI calls, but only one at a time. That is, calls are not made concurrently from two distinct threads as all MPI calls are serialized.
- MPI_THREAD_MULTIPLE - Level 3: Multiple threads may call MPI with no restrictions.

**Pros of MPI:**

- Runs on either shared or distributed memory architectures
- Can be used on a wider range of problems than OpenMP
- Each process has its own local variables
- Distributed memory computers are less expensive than large shared memory computers

**Cons of MPI:**

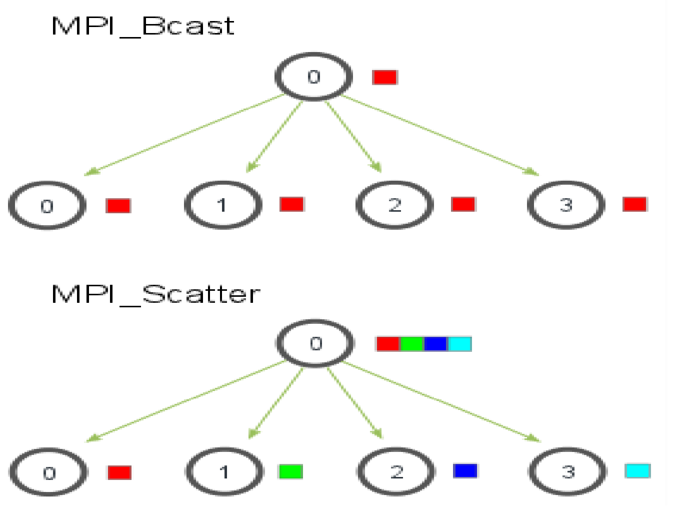
- requires more programming changes to go from serial to parallel version
- can be harder to debug
- performance is limited by the communication network between the nodes

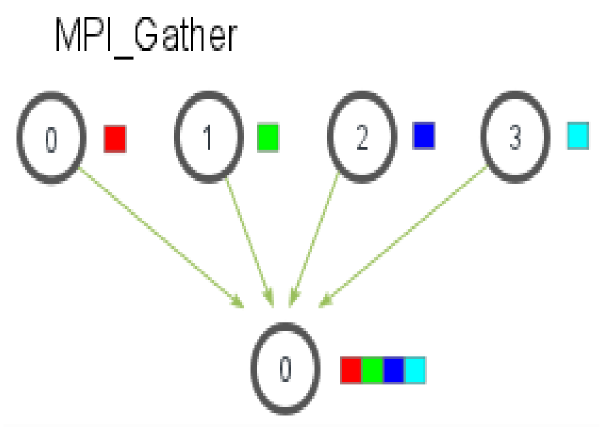
**Elementary MPI Data types:**

| MPI data type | C equivalent |
|---|---|
| MPI_SHORT | short int |
| MPI_INT | int |
| MPI_LONG | long int |
| MPI_LONG_LONG | long long int |
| MPI_UNSIGNED_CHAR | unsigned char |
| MPI_UNSIGNED_SHORT | unsigned short int |
| MPI_UNSIGNED | unsigned int |
| MPI_UNSIGNED_LONG | unsigned long int |
| MPI_UNSIGNED_LONG_LONG | unsigned long long int |
| MPI_FLOAT | float |
| MPI_DOUBLE | double |
| MPI_LONG_DOUBLE | long double |
| MPI_BYTE | char |

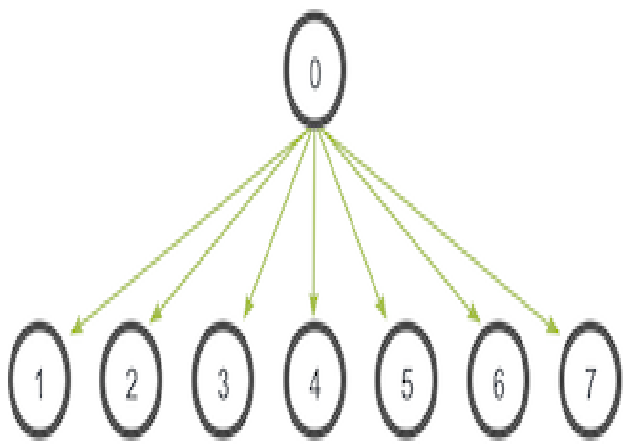
**MPI Scatter, Gather:**

**MPI_Gather(**
**void* send_data**
**int send_count,**
**MPI_Datatype send_datatype,**
**void* recv_data,**
**int recv_count,**
**MPI_Datatype recv_datatype,**
**int root,**
**MPI_Comm communicator)**





**MPI_Bcast(void* data, int count, MPI_Datatype datatype, int root, MPI_Comm communicator)**



**Computation of Pi : Algorithm**

This Program for Pi calculation depicts the usage of MPI_Wtime, MPI_Bcast and MPI_Reduce
This exercise presents a simple program to determine the value of pi. The algorithm suggested here is chosen for its simplicity. The method evaluates the integral of 4/(1+x*x) between 0 and 1. The method is simple: the integral is approximated by a sum of n intervals; the approximation to the integral in each interval is (1/n)*4/(1+x*x). The master process (rank 0) asks the user for the number of intervals; the master should then broadcast this number to all of the other processes. Each process then adds up every n'th interval (x = rank/n, rank/n+size/n,...). Finally, the sums computed by each process are added together using a reduction.

**Implementation of machine cluster**

1. Take 3 PC/BBB(nodes)
2. change the hostnames of the respective nodes using cmd
*root@PC#hostname* ***node1***
*root@PC #hostname* ***node2***
*root@PC #hostname* ***node3***
3. Create a user on each of the nodes
*root@node1#* ***adduser student***
*root@node2#* ***adduser student***
*root@node3#* ***adduser student***
4. check ip address of nodes using ***ifconfig*** (make sure all the PC are in the same network)
5. modify **/etc/hosts** of each node with ***IP-address hostname*** pair of every-node
Eg. 172.16.133.54 node1
172.16.132.12 node2
172.16.133.52 node3
6. now switch to student user
root@node1# su – student
root@node2# su – student
root@node3# su – student
7. Generate **ssh keys**
student@node1$ **ssh-keygen**
student@node2$ **ssh-keygen**
student@node3$ **ssh-keygen**
8. Now from node1 perform following commands
**student@node1$ ssh-copy-id node2**
**student@node1$ ssh-copy-id node3**
*similarly perform ssh-copy-if from node2 to node1&node3* ***and*** *from node3 to node1&node2*
9. Now check if you are able to ssh passwordlessly
**student@node1$ ssh node2**
**student@node2$**

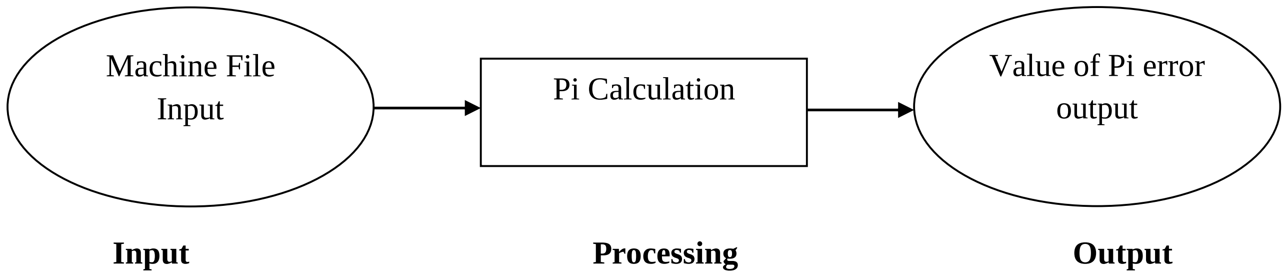
**Input :** Machine file as input to MPI

**Output:** i) Part of calculation with host name on every Processor.

ii) Value of pi.

iii) Error Calculation.

## Mathematical Model :



Let S be the system such that,

S= {I,O,Fn,Sc,Fc}

I-Input Set

Machine file & no. of processors

O-Output set

Value of Pi

Fn-Function Set

F1 – Initialize MPI

F2 – Send Machine File

F3 – Calculate value

F4 – Receive value of Pi

Sc – Success Set

Sc1 – File transfer successfully

Sc2 – Initialization of MPI is correct

Fc – Failure Cases

Fc1 – File not Transferred

Fc2 – Pi value is erroneous

**Platform:** ubuntu

**Conclusion:** Thus, we have implemented a MPI Program for calculating a quantity called coverage from data files.