Assignment No :A3

Roll No.4431

1 Title:

Message Passing Interface (MPI) for coverage.

2 Problem Definition

Write a MPI program for calculating a quantity called coverage from data files. Hint :- Program distributes computation efficiently across the cluster. The program should be able to work with any number of nodes and should yield the same results as the serial code.

3 Learning Objectives

1. To understand the concept Message Passing Interface in parallel programming.
2. To understand applications of MPI in real world.
3. To be able to solve the coverage problem.

4 Learning Outcomes

1. Ability to analyze problems and induce MPI based parallelism in them properly.
2. Understanding of parallel/concurrent programming.

5 Related Mathematics

Let S be the solution perspective of the given problem.

The set S is defined as:

$S = \{\ s, e, X, Y, F, DD, NDD, S_c, F_c | \varnothing_s\}$

Where,

s= Start state, Such that $Y = \{\varnothing\}$

e= End state

X= Input Set.

X = { seq(x) | x ∈ Natural numbers }

Y=Output set.

$Y = \{CoverageOfTheSequence\}$

F= Set of functions used.

F=$\{master(), slave(), compute()\}$

master()= function for master purposes.

slave()= function for slave purposes.

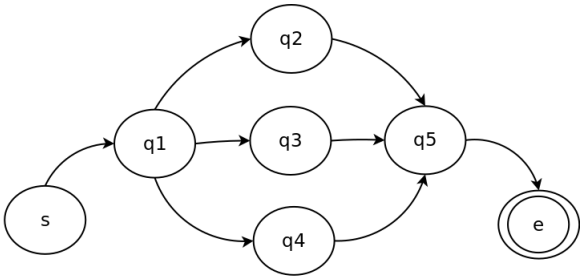
compute()= function for computing the coverage of the input sequence.

DD=Deterministic data.
DD=

1. sequence follows proper constraints.
2. coverage is defined and exists.
3. sequence terminates.

NDD= Non-deterministic data.
NDD= ∪ - DD

6 State Transition diagram



s = start state
q1 = master thread
q2 = slave thread
q3 = slave thread
q4 = slave thread
q5 = reduce the coverage
e = end state

7 Concepts related theory

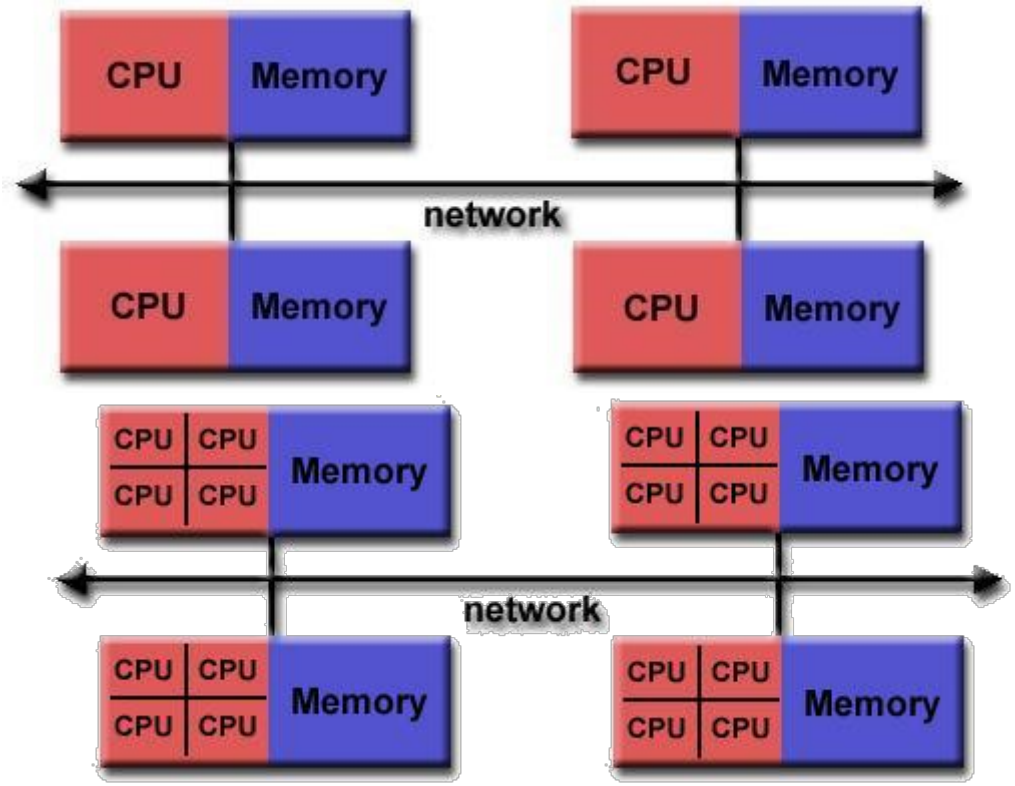
7.1 Concept of MPI

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.

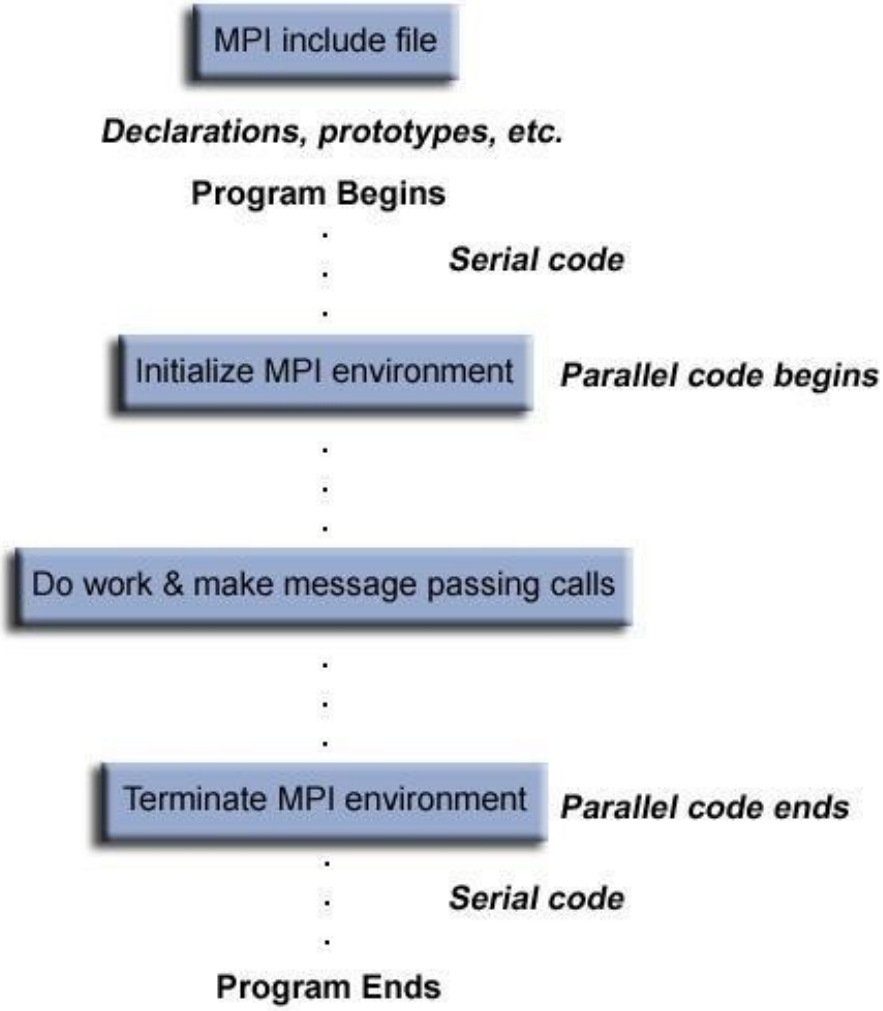
7.2 MPI Programming Model:



Both the diagrams show the general programming model of the MPI

system.

7.3 General MPI Programming Structure

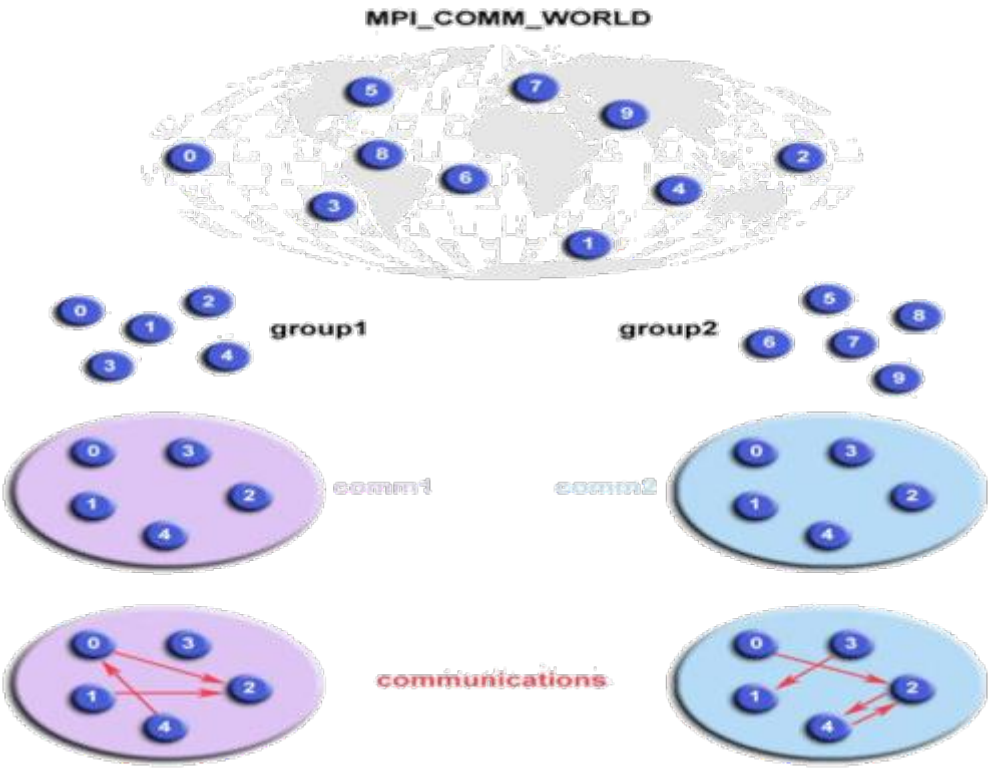


Communicators and Groups:

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 pre-defined 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

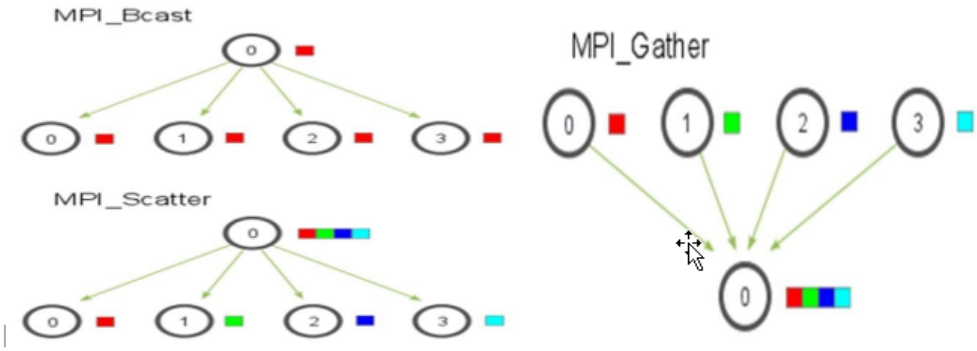
Cons of MPI:

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

8 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)



9 Program Listing

```
// Assignment No 3(Coverage)

#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define v 1 /* verbose flag, output if 1, no output if 0 */
#define tag 100 /* tag for sending a number */
int main ( int argc, char *argv[] )
{
int p,myid,i,f,*x;
double start, end;
MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&p);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
if(myid == 0) /* the manager allocates and initializes x */
{
x = (int*)calloc(p,sizeof(int));
x[0] = 50;
for (i=1; i<p; i++) x[i] = 2*x[i-1];
if(v>0)
{
printf("The data to square : ");
```

```
for (i=0; i<p; i++)
 printf(" %d",x[i]);
 printf("\n");
}
}
if(myid == 0) /* the manager copies x[0] to f */
{             /* and sends the i-th element to the i-th processor */
f = x[0];
for(i=1; i<p; i++)
MPI_Send(&x[i],1,MPI_INT,i,tag,MPI_COMM_WORLD);
}
else /* every worker receives its f from root */
{
MPI_Recv(&f,1,MPI_INT,0,tag,MPI_COMM_WORLD,&status);
if(v>0)
printf("Node %d will square %d\n",myid,f);
}
start = MPI_Wtime();
f *= f;       /* every node does the squaring */
if(myid == 0) /* the manager receives f in x[i] from processor i */
for(i=1; i<p; i++)
MPI_Recv(&x[i],1,MPI_INT,i,tag,MPI_COMM_WORLD,&status);
else      /* every worker sends f to the manager */
MPI_Send(&f,1,MPI_INT,0,tag,MPI_COMM_WORLD);
if(myid == 0) /* the manager prints results */
{
x[0] = f;
printf("The squared numbers : ");
for(i=0; i<p; i++)
printf(" %d",x[i]);
 printf("\n");
end = MPI_Wtime();
printf("Runtime = %f\n", end-start);
}

MPI_Finalize();
return 0;
}
```

10 Output

```
botman@botmatrix:~$ mpicc Square_MPI1.c -o abc
botman@botmatrix:~$ mpirun -np 2 ./abc
The data to cover :  50 100
The covered numbers :  2500 10000
Runtime = 0.000311
Node 1 will cover 100
botman@botmatrix:~$
```

11 TESTING :

Positive Test Cases:

Sr. No.	Test Condition	Steps to be executed	Expected Result	Actual Result
1.	Enter the number of process	Press Enter	Assign the numbers to n process	Same as Expected

Negative Test Cases:

Sr. No.	Test Condition	Steps to be executed	Expected Result	Actual Result
1.	Entered input data is in character or symbol	Press enter	Error message	Display result

12 CONCLUSION :

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