Dated : 10 – 12 – 2021 19BCE2250
Assessment No. : FINAL EXAM

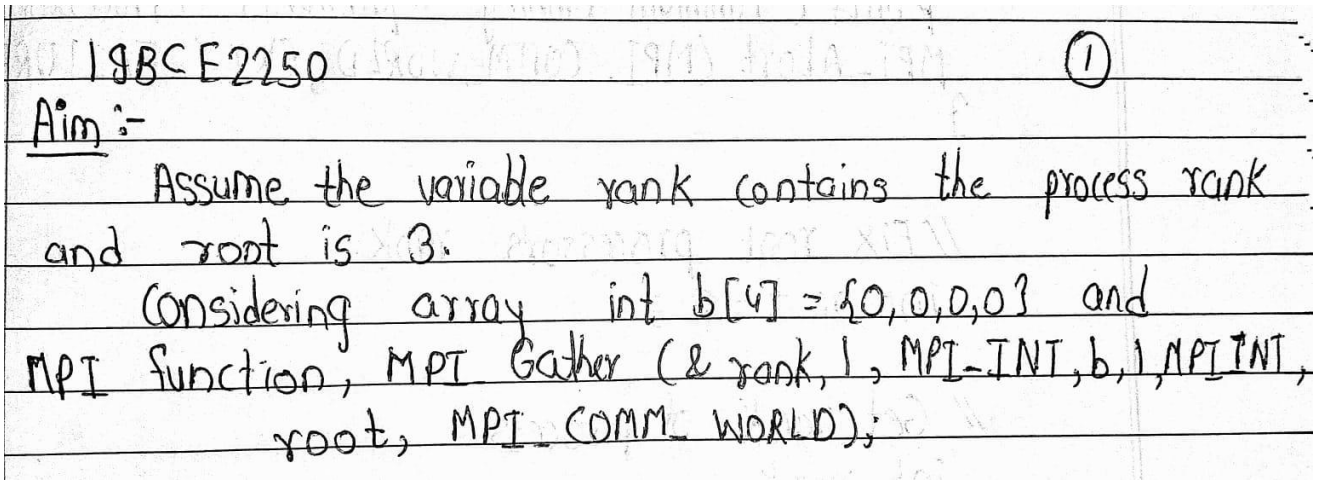
School of Computer Science and Engineering
FALL 2021 – 2022
CSE4001: Parallel and Distributed Computing
SLOT: L55+L56

Name: Ishan Sagar Jogalekar

Reg No: 19BCE22500 Slot: L55+L56 Course: CSE4001

Group: A

Aim :

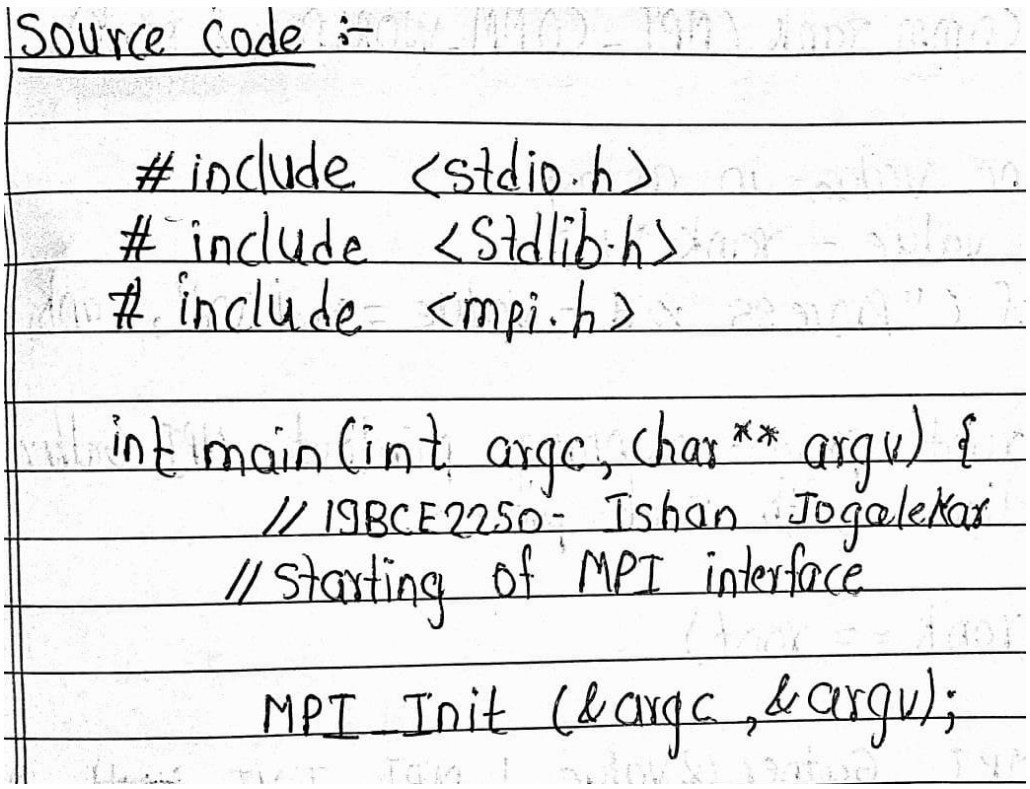

19BCE2250 (1)

Aim :-

Assume the variable rank contains the process rank and root is 3.

Considering array int b[4] = {0,0,0,0} and MPI function, MPI_Gather (&rank, 1, MPI_INT, b, 1, MPI_INT, root, MPI_COMM_WORLD);

Source Code :


Source Code :-

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char** argv) {
    // 19BCE2250 - Ishan Jogalekar
    // Starting of MPI interface

    MPI_Init (&argc, &argv);
```

Dated : 10 – 12 – 2021 19BCE2250
Assessment No. : FINAL EXAM

School of Computer Science and Engineering
FALL 2021 – 2022
CSE4001: Parallel and Distributed Computing
SLOT: L55+L56

(2)

```
// size of processes.
int size;
MPI_Comm_Size (MPI_COMM_WORLD, &size);

// Array initialization
int b[4] = {0,0,0,0};
if (size != 4)
{
  Printf ("Minimum running requirements: 4 processes.\n");
  MPI_Abort (MPI_COMM_WORLD, EXIT FAILURE);
}

// Fix root processor's rank
int root = 3;

// Get rank of process
int rank;
MPI_Comm_rank (MPI_COMM_WORLD, &rank);

// Define value in array
int value = rank * 1;
printf ("Process %d - value = %d \n", rank, value);

// if root process running printout MPI_Gather
function with root process.

If (rank == root)
{
  MPI_Gather (&value, 1, MPI_INT, b, 1,
  MPI_INT, root, MPI_COMM_WORLD);
```

Dated : 10 – 12 – 2021 19BCE2250
Assessment No. : FINAL EXAM

School of Computer Science and Engineering
FALL 2021 – 2022
CSE4001: Parallel and Distributed Computing
SLOT: L55+L56

```
printf("Array on (root) process %d : [%d, %d, %d, %d]\n",
        rank, b[0], b[1], b[2], b[3]);
}
else{
MPI_Gather (&value, 1, MPI_INT, NULL, 0, MPI_INT,
        root, MPI_COMM_WORLD);

printf(" Array elements collected on process %d :
  [%d, %d, %d, %d]\n", rank, b[0], b[1], b[2],
  b[3]);
}
// END of MPI interface

MPI_Finalize();
return EXIT_SUCCESS;
}
```

Dated : 10 – 12 – 2021 19BCE2250
Assessment No. : FINAL EXAM

School of Computer Science and Engineering
FALL 2021 – 2022
CSE4001: Parallel and Distributed Computing
SLOT: L55+L56

Conceptual Discussion :

Conceptual discussion: (4)

- MPI.h header file to use all mpi function inside program.
- MPI_COMM_Rank is process identifier, to get rank of processes or return the id of each processes.
- Here root rank consider to be 3 (given) hence root variable is initialized and set to 3.
- MPI_COMM_Size is used to determine size of processes pool that here considering maximum 4 is allowded or size is set to 4 only.
- Using array b of integers adding each element = element^1 , here element is reffered to value variable. It is going to add while running through each processes.
- MPI_Gather used to take elements from many processes & gathers them to one single processes.
- Using if statement, at root rank i.e 3rd process MPI_Gather is used and MPI_Finalize() function used to end the MPI interface.

Dated : 10 – 12 – 2021 19BCE2250
Assessment No. : FINAL EXAM

School of Computer Science and Engineering
FALL 2021 – 2022
CSE4001: Parallel and Distributed Computing
SLOT: L55+L56

Execution Output :

```
Terminal
14:22:54-ishan@ishan-ubuntu:~/PDC lab/FAT$mpicc c1.c -o c1
14:22:56-ishan@ishan-ubuntu:~/PDC lab/FAT$mpirun -n 4 ./c1
Process 0 - value = 0
Array elements collected on process 0: [0,0,0,0]
Process 1 - value = 1
Array elements collected on process 1: [0,0,0,0]
Process 2 - value = 2
Array elements collected on process 2: [0,0,0,0]
Process 3 - value = 3
Array on process(root) 3: [0,1,2,3]
14:22:57-ishan@ishan-ubuntu:~/PDC lab/FAT$
```

Results :

Results : (5)

- mpicc c1.c -o c1 command used to compile the MPI program and mpirun -n 4 ./c1 command is used to run program with 4 process.
- At process 0, value of array is 0 so collected element will be 0 only.
- At process 1 same thing ~~will~~ happened but this time value incremented to 1. value=2 and for process 2 it is value=2.
- Now at process 3, value=3. This is root process as mentioned in question. MPI_Gather function ~~will~~ is initialize over array b with value and root rank hence, output is Array at rank 3 : [0,1,2,3] here all values of each processes are gathered in single processes.

Dated : 10 – 12 – 2021 19BCE2250
Assessment No. : FINAL EXAM

School of Computer Science and Engineering
FALL 2021 – 2022
CSE4001: Parallel and Distributed Computing
SLOT: L55+L56
Review Question : [Viva-Voce]

Review question :-

1. Write importance of shared and distributed memory to signify inter process communication

→ Shared memory :-

- Shared memory is memory system in which memory is accessed by multiple program running on that system simultaneously.
- Shared memory provides communication without redundant copies.
- It is efficient means of passing data between several programs.
- In case of IPS models, shared memory systems are easy to implement using simple programming.
- It is fastest way to communicate directly in inter process communication.
- But the disadvantage is it can create problem in synchronization & memory protection.

Eg: Solaris 2.0 is based on shared memory system.

Dated : 10 – 12 – 2021 19BCE2250
Assessment No. : FINAL EXAM

School of Computer Science and Engineering
FALL 2021 – 2022
CSE4001: Parallel and Distributed Computing
SLOT: L55+L56

Distributed memory:-

- Memory in which each processor is having own private memory apart from common memory pool is distributed memory system.
- Distributed memory offers unified address space for each processor.
- Distributed memory excludes race condition in shared memory.
- Distributed shared memory is easier to design machine (hardware) rather than performed through algorithm.
- Distributed memory eleminate data exchange phase between processors that will increase the throughput of each processors.
- Distributed memory provides large virtual memory space than shared memory.
- Distributed memory provides scalability that is scales are very good even in large numbe of process nodes.