Fall Sem 2021-22

Assignment: 10

Date: 3/12/21

Name: Ishan Sagar Jogalekar

Reg no: 19BCE2250

Course: Parallel and distributed computing LAB - CSE4001

Slot: L55+L56

Aim: Assume the variable rank contains the process rank and root is 3. What will be stored in array b [ ] on each of four processes if each executes the following code fragment? int b [4] = {0 , 0 , 0 , 0};

Ans –

SOURCE CODE:

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

int main(int argc, char** argv) {
   //19BCE2250 - Ishan Jogalekar
   //Starting MPI
   MPI_Init(&argc, &argv);

   // Size of processes
   int size;
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   // Array initialization
   int b[8] ={0,0,0,0,0,0,0,0};
   if(size != 8)
   {
      printf("Minimum running requirements : 8 MPI processes.\n");
      MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
   }
   // Fix root'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 MPI gather and print array
   if(rank == root)
   {
      MPI_Gather(&value, 1, MPI_INT, b, 1, MPI_INT, root, MPI_COMM_WORLD);
      printf("Array on process(root) %d: [%d,%d,%d,%d,%d,%d,%d,%d] \n", rank, b[0],
b[1], b[2],b[3],b[4],b[5],b[6],b[7]);
   }
   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,%d,%d,%d,%d] \n",
rank, b[0], b[1], b[2],b[3],b[4],b[5],b[6],b[7] );
   }
   // End MPI
   MPI_Finalize();
   return EXIT_SUCCESS;
}
```

EXECUTION:

1. OpenMP is a library for parallel programming in the SMP (symmetric multi-processors or shared-memory processors) model.
2. Mpi.h is header file to use all mpi functions inside program.
3. MPI_COMM_RANK is used to determine process identifier, that processes id number.
4. MPI_COMM_SIZE is used to determine total size of process pool inside that particular MPI program. Declaring root process as 3 given in the question.
5. Using array b of integers adding value*1 as elements in arrays while processing through each process.

6. MPI_Gather takes elements from many processes and gathers them to one single process. Here at root process MPI_Gather function will gather all elements value and print all elements in array at root process at 3.
7. Using if statement to check whether rank of current process is at root level, to print array otherwise in else condition to print without passing value in MPI_Gather function will execute.
8. Using MPI_Finalize function ending MPI interface within program.

RESULTS:

Output:

1. Running program with 4 processes –

```
11:58:55-ishan@ishan-ubuntu:~/PDC lab/lab10$mpicc c1.c -o c1
11:58:57-ishan@ishan-ubuntu:~/PDC lab/lab10$mpirun -n 4 ./c1
Process 1 - value = 1
Process 0 - value = 0
Array elements collected on process 0: [0,0,0,0]
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]
11:58:59-ishan@ishan-ubuntu:~/PDC lab/lab10$
```

2. Running program with 8 processes –

```
12:04:22-ishan@ishan-ubuntu:~/PDC lab/lab10$mpicc c1.c -o c1
12:04:23-ishan@ishan-ubuntu:~/PDC lab/lab10$mpirun -n 8 ./c1
Process 4 - value = 4
Array elements collected on process 4: [0,0,0,0,0,0,0,0]
Process 7 - value = 7
Process 3 - value = 3
Process 1 - value = 1
Process 2 - value = 2
Array elements collected on process 2: [0,0,0,0,0,0,0,0]
Array elements collected on process 1: [0,0,0,0,0,0,0,0]
Process 6 - value = 6
Array elements collected on process 6: [0,0,0,0,0,0,0,0]
Process 5 - value = 5
Array elements collected on process 5: [0,0,0,0,0,0,0,0]
Process 0 - value = 0
Array elements collected on process 0: [0,0,0,0,0,0,0,0]
Array elements collected on process 7: [0,0,0,0,0,0,0,0]
Array on process(root) 3: [0,1,2,3,4,5,6,7]
12:04:27-ishan@ishan-ubuntu:~/PDC lab/lab10$
```

Code:

```
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 MPI gather and print array
if(rank == root)
{
    MPI_Gather(&value, 1, MPI_INT, b, 1, MPI_INT, root, MPI_COMM_WORLD);
    printf("Array on process(root) %d: %d,%d,%d,%d,%d,%d,%d,%d.\n", rank, b[0], b[1
}
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,%d,%d,%d,%d \n", ra
}
// End MPI
MPI_Finalize();
return EXIT_SUCCESS;
```