



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Computer Science & Engineering

CSE4001

Parallel and Distributed Computing

LAB ASSIGNMENT 9

Submitted to **Prof. DEEBAK B.D.**

TOPIC: PROBLEMS USING MPI

NAME: PUNIT MIDDHA

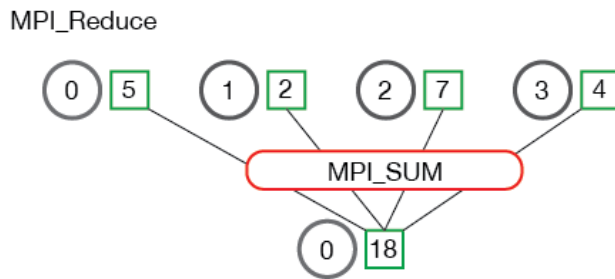
REG.NO: 19BCE2060

SLOT: L55+L56

DATE: 30/11/2021

QUESTION – I

Write a C program to use MPI_Reduce that divides the processors into the group to find the addition independently.



Hint. The function prototype is as follows:

```
MPI_Reduce(  
void* send_data,  
void* recv_data,      int  
count,  
                    MPI_Datatype datatype,  
                    MPI_Op op,  
int root,  
                    MPI_Comm communicator)
```

SOURCE CODE:

```
#include<stdio.h>  
#include<stdlib.h>  
#include<math.h>  
#include<mpi.h>  
int main(int argc, char* argv[]) {  
    int rank, num_of_procs, i;  
    //Intializing MPI  
    MPI_Init(&argc, &argv);  
  
    //Rank and Size of process  
    MPI_Comm_size(MPI_COMM_WORLD, &num_of_procs);  
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
  
    //Variables Declarations  
    int local_sum = 0;  
    int global_sum = 0;  
    int a[] = { 5, 2, 7, 4 };  
    char o;  
  
    if (rank == 0) {  
        printf("Addition of the array using MPI_Reduce \n");  
    }  
    //Addition of elements in an array
```

```

for (i = rank; i < 4; i += num_of_procs) {
printf("array[%d]: %d\n",i, a[i]);
local_sum = local_sum + a[i];
}
//MPI Reduce function
MPI_Reduce(&local_sum, &global_sum, 3, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
//Print final sum
if (rank == 0) {
printf("Global Sum: %d\n", global_sum);
}
//Ending MPI
MPI_Finalize();
return 0;
}

```

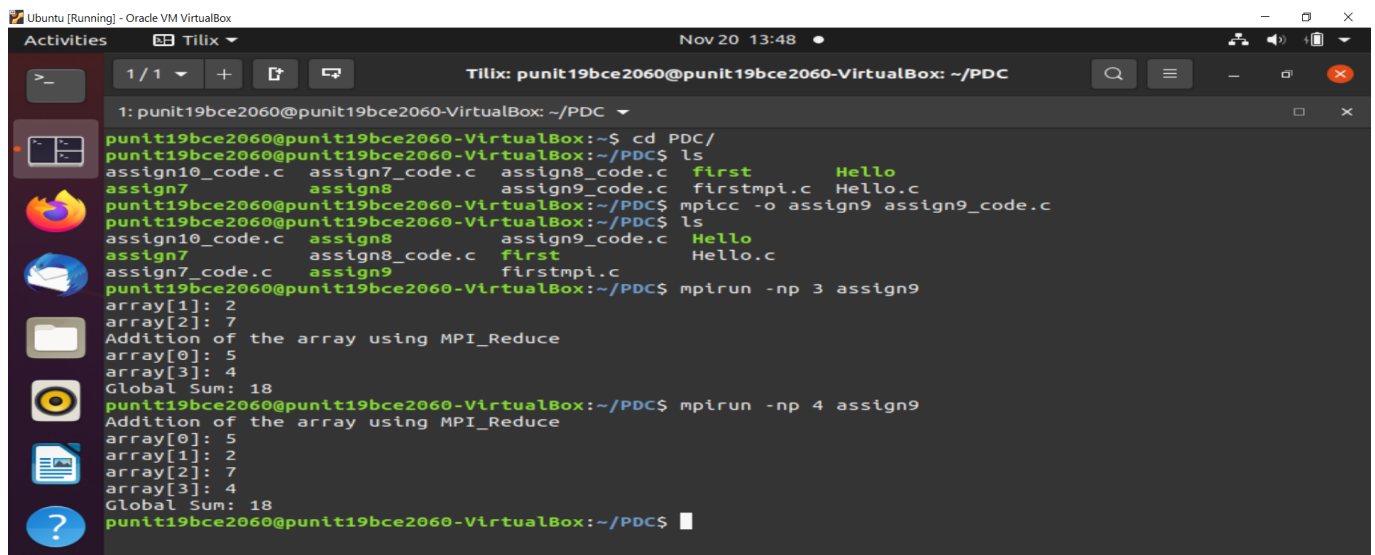


```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<math.h>
4 #include<mpi.h>
5 int main(int argc, char* argv[]) {
6     int rank, num_of_procs, i;
7     //Initializing MPI
8     MPI_Init(&argc, &argv);
9
10    //Rank and Size of process
11    MPI_Comm_size(MPI_COMM_WORLD, &num_of_procs);
12    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
13
14    //Variables Declarations
15    int local_sum = 0;
16    int global_sum = 0;
17    int a[] = { 5, 2, 7, 4 };
18    char o;
19
20    if (rank == 0) {
21        printf("Addition of the array using MPI_Reduce \n");
22    }
23    //Addition of elements in an array
24    for (i = rank; i < 4; i += num_of_procs) {
25        printf("array[%d]: %d\n",i, a[i]);
26        local_sum = local_sum + a[i];
27    }
28    //MPI Reduce function
29    MPI_Reduce(&local_sum, &global_sum, 3, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
30    //Print final sum
31    if (rank == 0) {
32        printf("Global Sum: %d\n", global_sum);
33    }
34    //Ending MPI
35    MPI_Finalize();
36    return 0;
37 }

```

EXECUTION:



The screenshot shows a terminal window titled 'Tilix: punit19bce2060@punit19bce2060-VirtualBox: ~/PDC'. The user is in the directory ~/PDC. The terminal shows the following commands and output:

```
punit19bce2060@punit19bce2060-VirtualBox:~$ cd PDC/
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ ls
assign10_code.c  assign7_code.c  assign8_code.c  first      Hello
assign7          assign8         assign9_code.c  firstmpi.c  Hello.c
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ mpicc -o assign9 assign9_code.c
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ ls
assign10_code.c  assign8         assign9_code.c  Hello
assign7          assign8_code.c  first          firstmpi.c
assign7_code.c   assign9         firstmpi.c
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ mpirun -np 3 assign9
array[1]: 2
array[2]: 7
Addition of the array using MPI_Reduce
array[0]: 5
array[3]: 4
Global Sum: 18
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ mpirun -np 4 assign9
Addition of the array using MPI_Reduce
array[0]: 5
array[1]: 2
array[2]: 7
array[3]: 4
Global Sum: 18
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$
```

REMARKS:

- ✓ The purpose of this experiment was to learn about the MPI Reduce() function.
- ✓ The reduce function speeds up the calculation for all processes in a collection.
- ✓ The reduction operation is used to compute the items sent by the MPI, and the result is saved in the root.
- ✓ In the above implementation, the root process computes the sum of each process's array items, which is subsequently shown in the Global sum variable.