Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

Year: 2022-23 **Semester:** 1

Course: High Performance Computing Lab

Practical No. 6

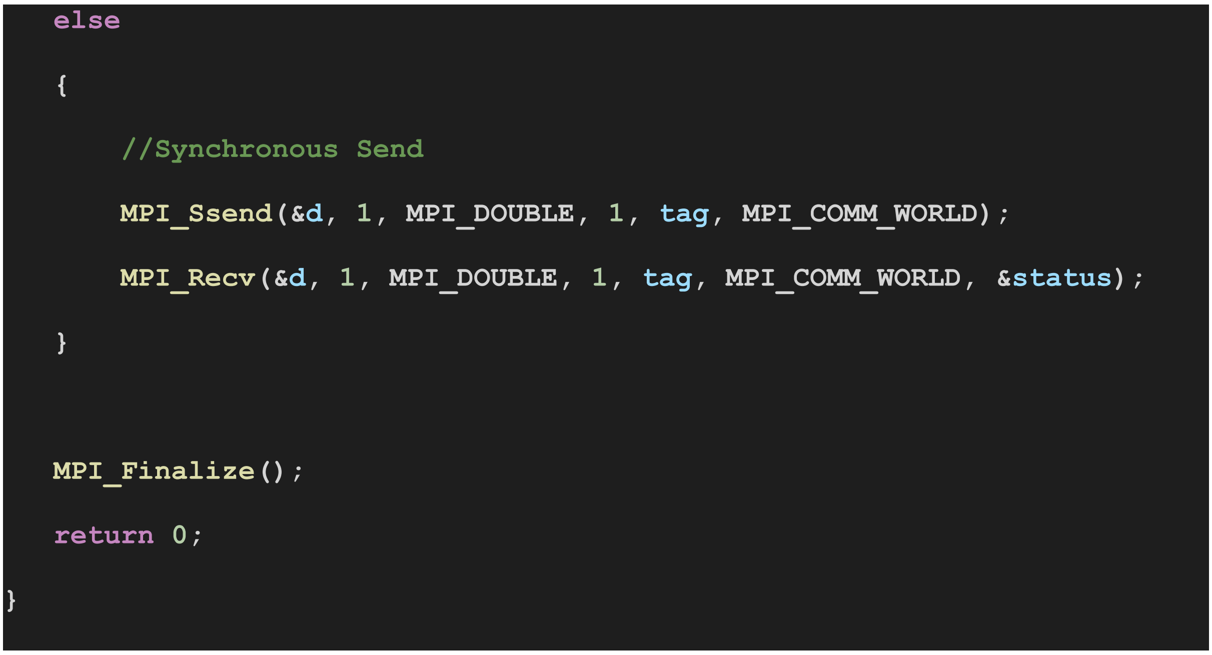
PRN No : 2019BTECS00070

Name : Killedar Prathmesh

Q1: Implement a MPI program to give an example of Deadlock.

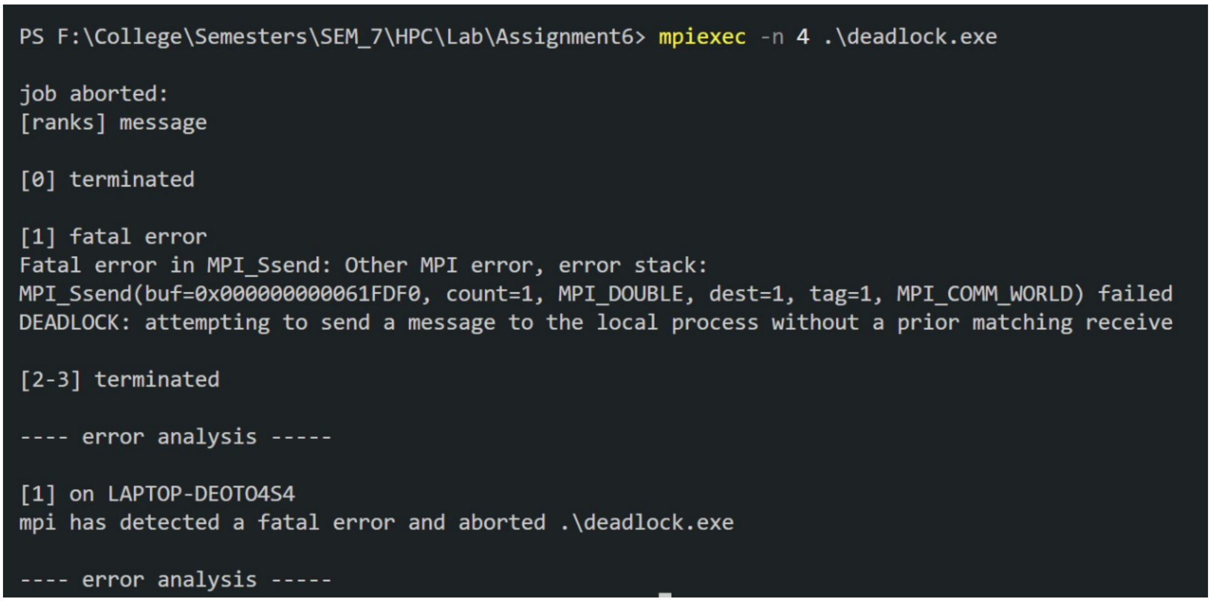
Code:

```
#include "mpi.h"
#include <math.h>
int main(int argc, char **argv)
{
   MPI_Status status;
   int num;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &num);

   double d = 100.0;
   int tag = 1;

   if (num == 0)
   {
       //synchronous Send
       MPI_Ssend(&d, 1, MPI_DOUBLE, 1, tag, MPI_COMM_WORLD);
       MPI_Recv(&d, 1, MPI_DOUBLE, 1, tag, MPI_COMM_WORLD, &status);
   }
```



```
   else

   {

       //Synchronous Send

       MPI_Ssend(&d, 1, MPI_DOUBLE, 1, tag, MPI_COMM_WORLD);

       MPI_Recv(&d, 1, MPI_DOUBLE, 1, tag, MPI_COMM_WORLD, &status);

   }


   MPI_Finalize();

   return 0;

}
```

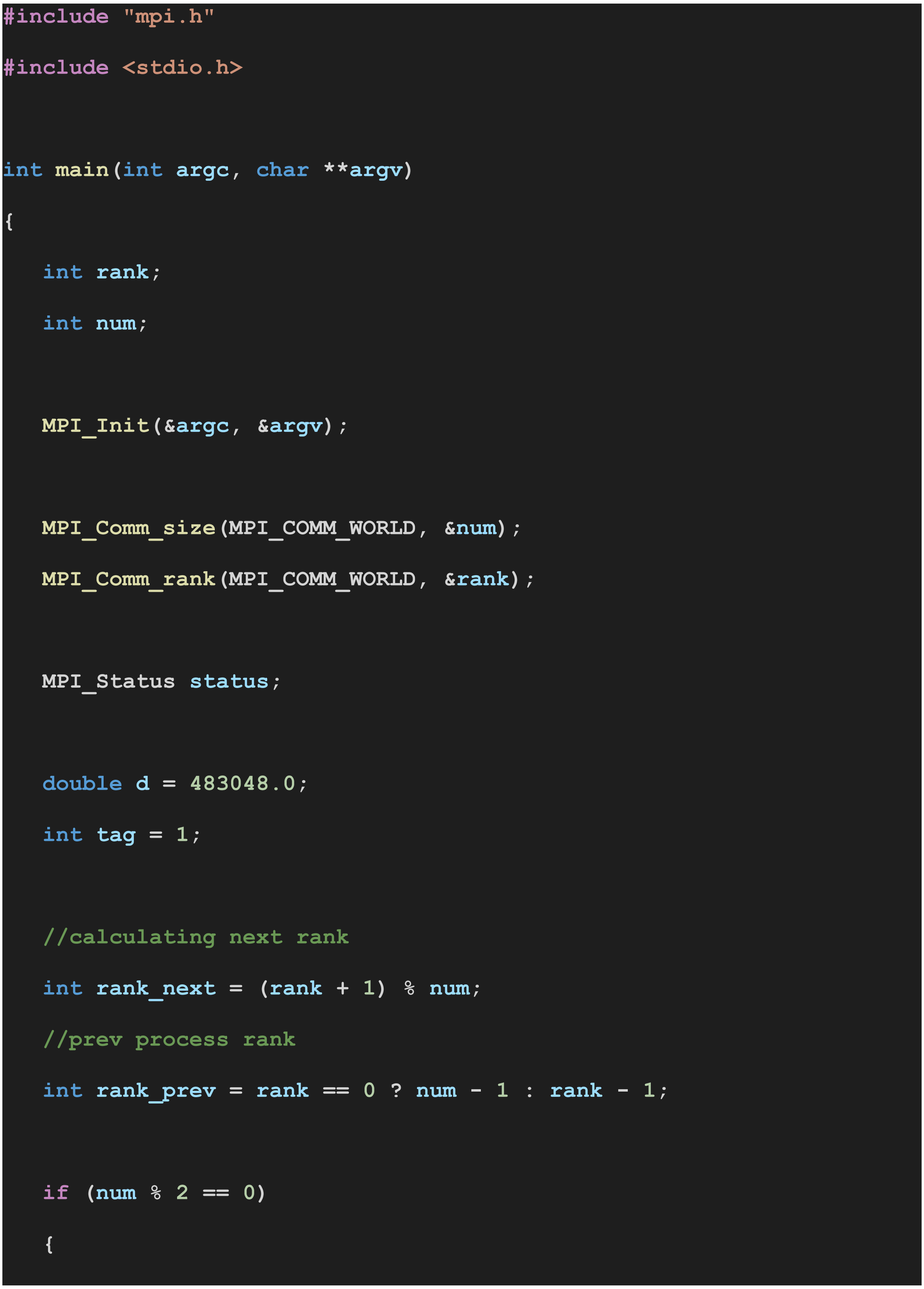
Output:



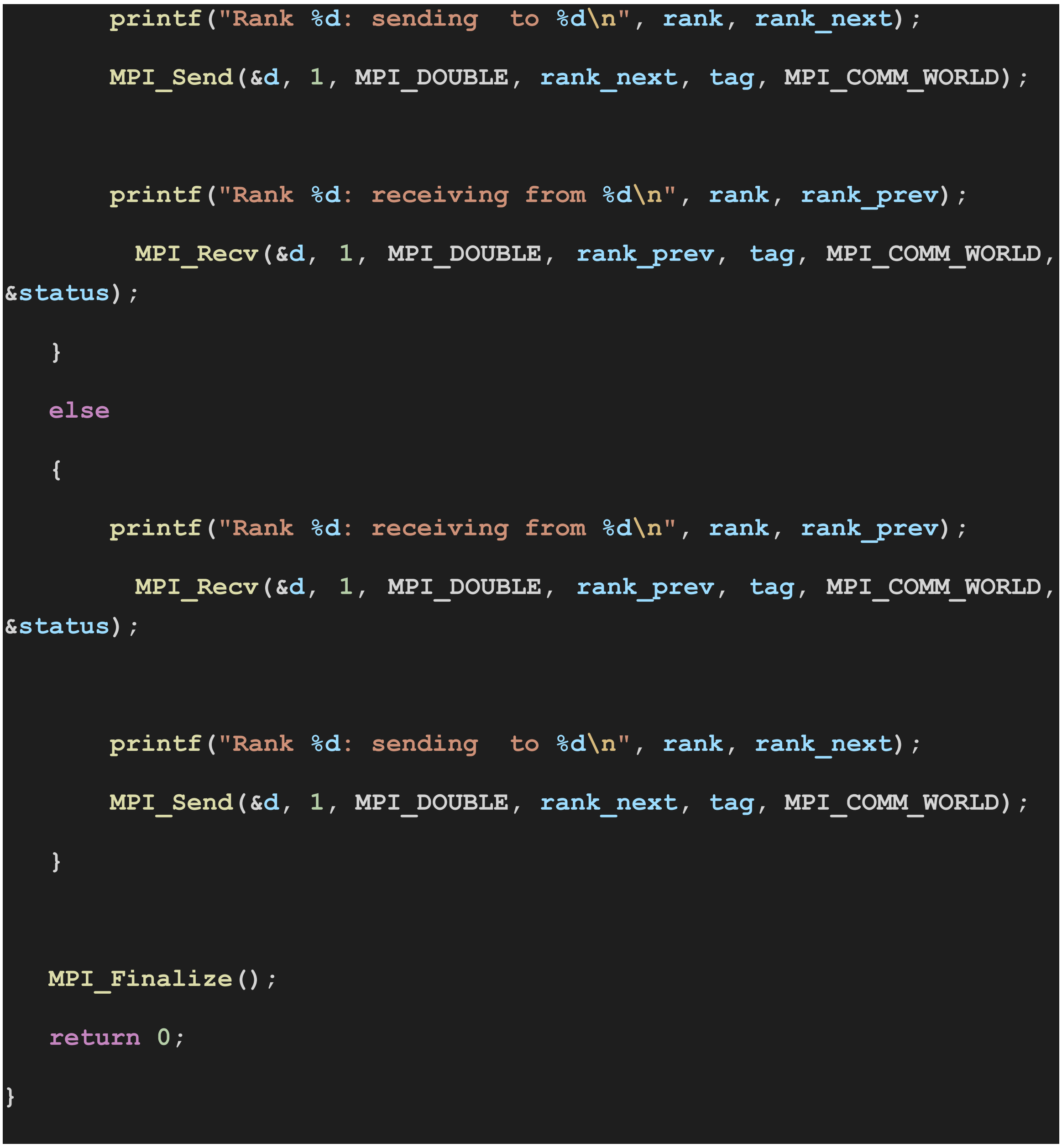
```
PS F:\College\Semesters\SEM_7\HPC\Lab\Assignment6> mpiexec -n 4 .\deadlock.exe

job aborted:
[ranks] message

[0] terminated

[1] fatal error
Fatal error in MPI_Ssend: Other MPI error, error stack:
MPI_Ssend(buf=0x000000000061FDF0, count=1, MPI_DOUBLE, dest=1, tag=1, MPI_COMM_WORLD) failed
DEADLOCK: attempting to send a message to the local process without a prior matching receive

[2-3] terminated

---- error analysis -----

[1] on LAPTOP-DEOTO4S4
mpi has detected a fatal error and aborted .\deadlock.exe

---- error analysis -----
```

Q2. Implement blocking MPI send & receive to demonstrate Nearest neighbor exchange of data in a ring topology.

Code:

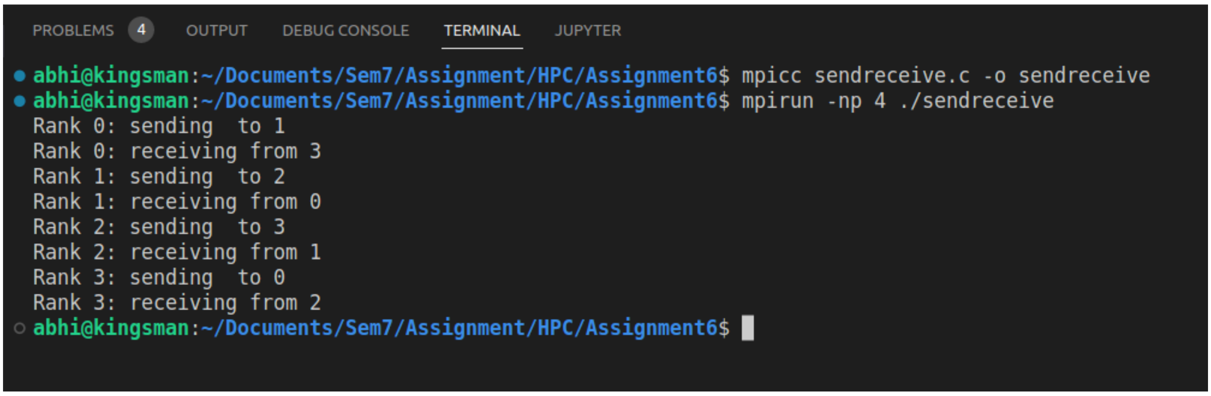


```
#include "mpi.h"
#include <stdio.h>

int main(int argc, char **argv)
{
    int rank;
    int num;

    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &num);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Status status;

    double d = 483048.0;
    int tag = 1;

    //calculating next rank
    int rank_next = (rank + 1) % num;
    //prev process rank
    int rank_prev = rank == 0 ? num - 1 : rank - 1;

    if (num % 2 == 0)
    {
```



```
        printf("Rank %d: sending  to %d\n", rank, rank_next);
        MPI_Send(&d, 1, MPI_DOUBLE, rank_next, tag, MPI_COMM_WORLD);

        printf("Rank %d: receiving from %d\n", rank, rank_prev);
         MPI_Recv(&d, 1, MPI_DOUBLE, rank_prev, tag, MPI_COMM_WORLD,
&status);
    }
    else
    {
        printf("Rank %d: receiving from %d\n", rank, rank_prev);
         MPI_Recv(&d, 1, MPI_DOUBLE, rank_prev, tag, MPI_COMM_WORLD,
&status);

        printf("Rank %d: sending  to %d\n", rank, rank_next);
        MPI_Send(&d, 1, MPI_DOUBLE, rank_next, tag, MPI_COMM_WORLD);
    }

    MPI_Finalize();
    return 0;
}
```

Output:



```
● abhi@kingsman:~/Documents/Sem7/Assignment/HPC/Assignment6$ mpicc sendreceive.c -o sendreceive
● abhi@kingsman:~/Documents/Sem7/Assignment/HPC/Assignment6$ mpirun -np 4 ./sendreceive
  Rank 0: sending  to 1
  Rank 0: receiving from 3
  Rank 1: sending  to 2
  Rank 1: receiving from 0
  Rank 2: sending  to 3
  Rank 2: receiving from 1
  Rank 3: sending  to 0
  Rank 3: receiving from 2
○ abhi@kingsman:~/Documents/Sem7/Assignment/HPC/Assignment6$ 
```

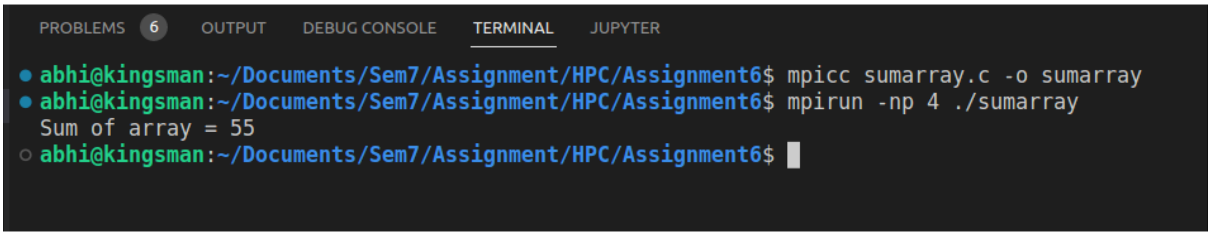
Q3. Write a MPI program to find the sum of all the elements of an array A of size n. Elements of an array can be divided into two equals groups. The first [n/2] elements are added by the first process, P0, and last [n/2] elements the by second process, P1. The two sums then are added to get the final result.

Code:

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

#define localSize 1000

int local[1000]; // to store the subarray data comming from process 0;

int main(int argc, char **argv)
{
   int rank;
   int num;

   int n = 10;
   int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

   int per_process, elements_received;

   MPI_Init(&argc, &argv);

   MPI_Comm_size(MPI_COMM_WORLD, &num);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   MPI_Status status;
```

```
    // process with rank 0 will divide data among all processes and add partial sums to get final sum
    if (rank == 0)
    {
        int index, i;

        per_process = n / num;

        if (num > 1) // if more than 1 processes available
        {
            //divide array data among processes
            for (i = 1; i < num - 1; i++)
            {
                //calculating first index of subarray that need to be send to ith process
                index = i * per_process;

                //send no of elements and subarray of that lenght to each process
                MPI_Send(&per_process, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
                MPI_Send(&arr[index], per_process, MPI_INT, i, 0, MPI_COMM_WORLD);
            }

            // for last process send all remaining elements
            index = i * per_process;
            int ele_left = n - index;

            MPI_Send(&ele_left, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
            MPI_Send(&arr[index], ele_left, MPI_INT, i, 0, MPI_COMM_WORLD);
        }

        // add numbers on process with rank 0
        int sum = 0;
        for (int i = 0; i < per_process; i++)
        {
            sum += arr[i];
        }

        // add all partial sums from all processes
        int tmp;
        for (int i = 1; i < num; i++)
        {
            MPI_Recv(&tmp, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status);

            int sender = status.MPI_SOURCE;

            sum += tmp;
        }

        printf("Sum of array = %d\n", sum);
    }
    else // if rank of process is not 0, then receive elements and calculate partial sums
    {
        // receive no of elements and elements form process 0 and store them on local array
        MPI_Recv(&elements_received, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
&status);


        MPI_Recv(&local, elements_received, MPI_INT, 0, 0,
MPI_COMM_WORLD, &status);


        // calculate partial local sum

        int partial_sum = 0;

        for (int i = 0; i < elements_received; i++)

        {

            partial_sum += local[i];

        }


        //send calculated partial sum to process with rank 0

        MPI_Send(&partial_sum, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);

    }


    MPI_Finalize();

    return 0;

}
```

Output:



```
abhi@kingsman:~/Documents/Sem7/Assignment/HPC/Assignment6$ mpicc sumarray.c -o sumarray
abhi@kingsman:~/Documents/Sem7/Assignment/HPC/Assignment6$ mpirun -np 4 ./sumarray
Sum of array = 55
abhi@kingsman:~/Documents/Sem7/Assignment/HPC/Assignment6$ 
```

Github Link: https://github.com/killedar27/HPCAssignment