Dated : 26 -11 – 2021
Slot : L55 + L56

LAB CAT [2]

Name – Ishan Sagar Jogalekar

Reg. no – 19BCE2250

Slot – L55+l56

Lab – Parallel and distributed computing Lab CSE4001

Scenario – 1

Write a 'C' Program to initialize an array of 100 elements in order to perform the sum of the elements sharing the load among 4 processes using MPI Send and MPI Recv operation.

Source Code:

```
#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>




int main(int argc, char* argv[])

{

   //19BCE2250 - Ishan Jogalekar

   // size of array

   int  n = 101 ;


   //Array of number

   int arr[101];
```

Dated : 26 -11 – 2021
Slot : L55 + L56

LAB CAT [2]

```
// Adding 1 to 100 numbers as elements in array

for(int i = 0;i<=100;i++){

    arr[i]=i;

}


// Temporary array to store sum during each process

int a2[1000];




    int rank, np,ele_process,ele_recieved;



    MPI_Status status;


    // MPI starting

    MPI_Init(&argc, &argv);


    // find out process ID,

    // and how many processes were started

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Comm_size(MPI_COMM_WORLD, &np);
```

Dated : 26 -11 – 2021
Slot : L55 + L56

LAB CAT [2]

```
// master 0th process
if (rank == 0) {
        int index, i;
        ele_process = n / np;
printf("Number of processes = %d\n",np);

        // check if more than 1 processes are run
        if (np > 1) {
                // distributes child to sum progress

                for (i = 1; i < np - 1; i++) {
                        index = i * ele_process;

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

                // last process to add remaining
                index = i * ele_process;
                int ele_rem = n - index;

                MPI_Send(&ele_rem,1, MPI_INT,i, 0,MPI_COMM_WORLD);
```

Dated : 26 -11 – 2021
Slot : L55 + L56

LAB CAT [2]

```
                    MPI_Send(&arr[index],ele_rem,MPI_INT, i,
0,MPI_COMM_WORLD);

                }


                // 0th Master process for sum

                int sum = 0;

                for (i = 0; i < ele_process; i++)

                        sum += arr[i];


                // collects partial sums

                int tmp;

                for (i = 1; i < np; i++) {

                        MPI_Recv(&tmp, 1, MPI_INT,MPI_ANY_SOURCE,
0,MPI_COMM_WORLD,&status);

                        int sender = status.MPI_SOURCE;

         sum += tmp;

                }


                // Final sum

                printf("Sum of array is : %d\n", sum);

        }

        // other than master  processes

        else {

                MPI_Recv(&ele_recieved,1, MPI_INT, 0, 0,MPI_COMM_WORLD,&status);
```

Dated : 26 -11 – 2021
Slot : L55 + L56

LAB CAT [2]

```
        // stores the received array in temp array a2


                MPI_Recv(&a2, ele_recieved,MPI_INT, 0,
0,MPI_COMM_WORLD,&status);


                // calculates its partial sum

                int p_sum = 0;

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

                        p_sum += a2[i];


                // sends the partial sum to the root process

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

        }


        // End of MPI

        MPI_Finalize();

    return 0;

}
```

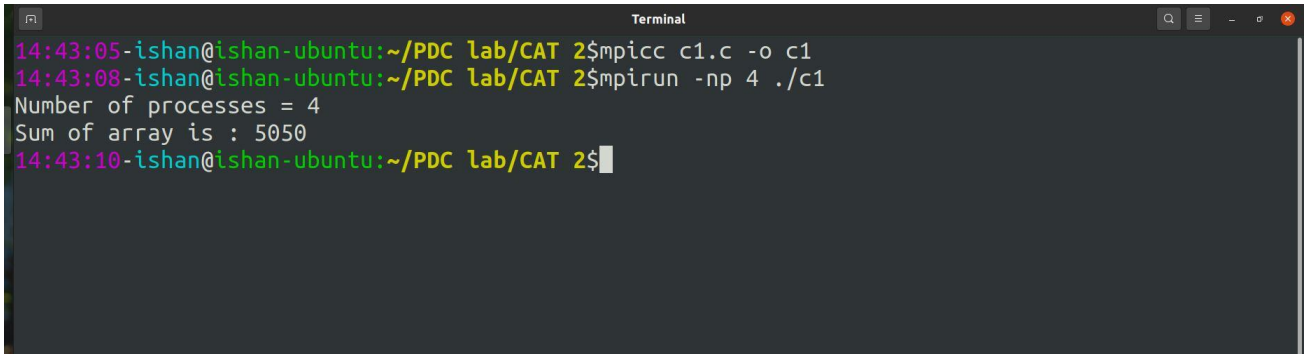
Dated : 26 -11 – 2021
Slot : L55 + L56

LAB CAT [2]

Code snippet -

```c
int main(int argc, char* argv[])
{
    //19BCE2250 - Ishan Jogalekar
    // size of array
    int  n = 101 ;

    //Array of number
    int arr[101];
    // Adding 1 to 100 numbers as elements in array
    for(int i = 0;i<=100;i++){
        arr[i]=i;
    }

    // Temporary array to store sum during each process
    int a2[1000];



    int rank, np,ele_process,ele_recieved;


    MPI_Status status;

    // MPI starting
    MPI_Init(&argc, &argv);

    // find out process ID,
    // and how many processes were started
```

Execution:



Dated : 26 -11 – 2021
Slot : L55 + L56

LAB CAT [2]

Results:

- OpenMP is a library for parallel programming in the SMP (symmetric multiprocessors or shared-memory processors) model.
- Mpi.h is header file to use all mpi functions inside program.
- MPI_COMM_SIZE : Determine number of processes.
- MPI_COMM_RANK : Determine my process identifier.
- MPI_SEND : Send a message.
- MPI_RECV : Receive a message.
- For loop is used to first create and store 1 to 100 elements then next for loop is used to calculate the sum.
- MPI_SEND is used when rank of process is 0 that is master process.
- Finally MPI_Finalize() is used to stop MPI.

Review Question:

Write the basic format of MPI_Isend and MPI_Irecv to signify the use of non-blocking.

Ans:

MPI_Isend :

int MPI_Isend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag,MPI_Comm comm, MPI_Request *request)

MPI_Irecv:

int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source,int tag, MPI_Comm comm, MPI_Request * request)

buf – initial address of send / receive buffer

count – number of elements in send / receive buffer

tag – message tag

comm – communicator

dest – rank or destination of process

datatype – datatype to send (int , char etc.)