**AIM -** To Develop a distributed system, to find sum of N elements in an array by distributing

N/n elements to n number of processors MPI or OpenMP. Demonstrate by displaying the

intermediate sums calculated at different processors.

**Name –** Yogita Sunil Girigosavi

**Sum.c**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

// size of array

#define n 10

int array\_fir[] = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };

// Temporary array for slave process

int array\_sec[1000];

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

{

int pid;

int np;

int elements\_per\_process;

int n\_elements\_recieved;

// np -> no. of processes

// pid -> process id

MPI\_Status status;

// Creation of parallel processes

MPI\_Init(&argc, &argv);

// find out process ID,

// and how many processes were started

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &pid);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &np);

// master process

if (pid == 0)

{

int index, i;

elements\_per\_process = n / np;

// check if more than 1 processes are run

if (np > 1)

{

// distributes the portion of array

// to child processes to calculate

// their partial sums

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

{

index = i \* elements\_per\_process;

MPI\_Send(&elements\_per\_process,

1, MPI\_INT, i, 0,

MPI\_COMM\_WORLD);

MPI\_Send(&array\_fir[index],

elements\_per\_process,

MPI\_INT, i, 0,

MPI\_COMM\_WORLD);

}

// last process adds remaining elements

index = i \* elements\_per\_process;

int elements\_left = n - index;

MPI\_Send(&elements\_left,

1, MPI\_INT,

i, 0,

MPI\_COMM\_WORLD);

MPI\_Send(&array\_fir[index],

elements\_left,

MPI\_INT, i, 0,

MPI\_COMM\_WORLD);

}

// master process add its own sub array

int sum = 0;

for (i = 0; i < elements\_per\_process; i++)

sum += array\_fir[i];

printf("\nSum of process %d is %d\n",pid,sum);

// collects partial sums from other processes

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;

}

// prints the final sum of array

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

}

// slave processes

else

{

MPI\_Recv(&n\_elements\_recieved,

1, MPI\_INT, 0, 0,

MPI\_COMM\_WORLD,

&status);

// stores the received array segment

// in local array array\_sec

MPI\_Recv(&array\_sec, n\_elements\_recieved,

MPI\_INT, 0, 0,

MPI\_COMM\_WORLD,

&status);

// calculates its partial sum

int partial\_sum = 0;

for (int i = 0; i < n\_elements\_recieved; i++)

{

partial\_sum += array\_sec[i];

}

printf("\nSum of process %d is %d\n",pid,partial\_sum);

// sends the partial sum to the root process

MPI\_Send(&partial\_sum, 1, MPI\_INT,

0, 0, MPI\_COMM\_WORLD);

}

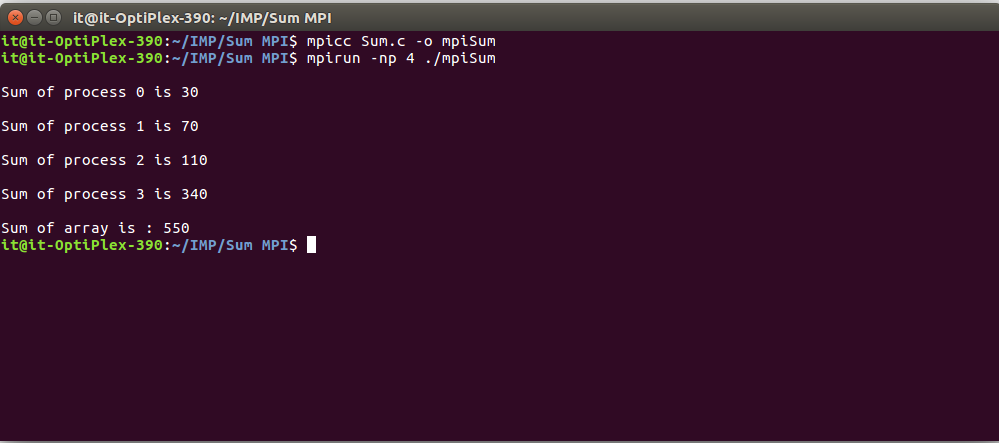
// cleans up all MPI state before exit of process

MPI\_Finalize();

return 0;

}

**Output –**

****