**Class:** Final Year (Computer Science and Engineering)

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

**Course:** High Performance Computing Lab

**Practical No. 8**

**Exam Seat No:** 2019BTECS00070

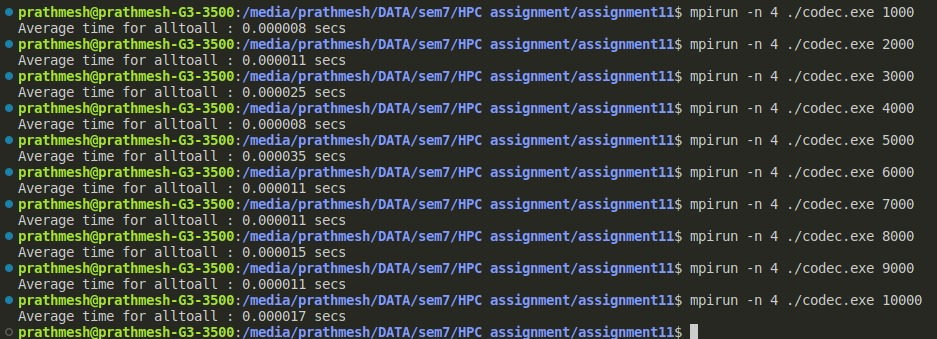
**Name:** Killedar Prathmesh

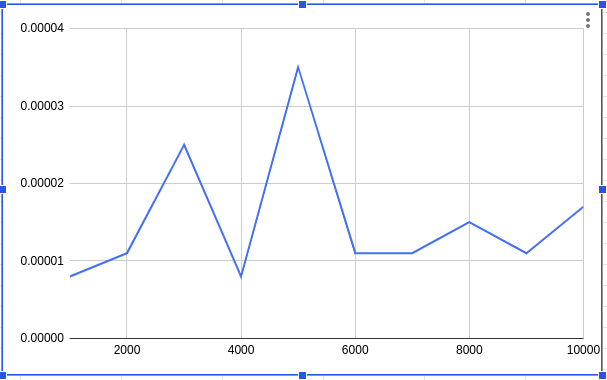
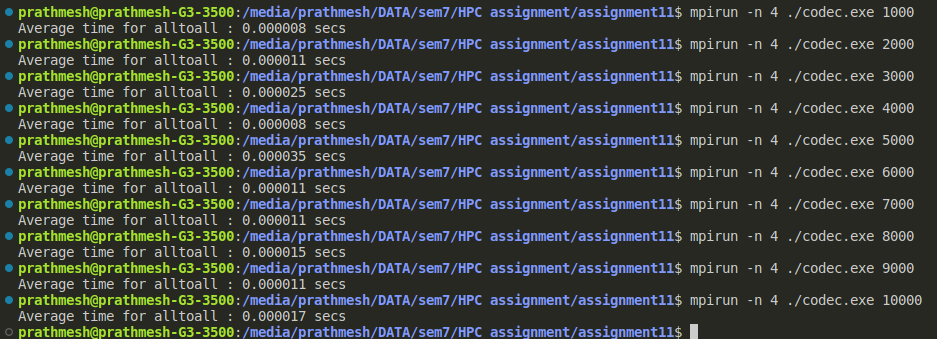
1. Study and implement 2D Convolution using MPI. Use different number of processes and analyze the performance.

**Code:**



**Output:**

****



2. Implement dot product using MPI. Use different number of processes and analyze the performance.

**Code:**

#include <stdio.h>

#include <mpi.h>

#include <unistd.h>

#include <math.h>

#include <time.h>

#include <stdlib.h>

#define NELMS 100000

#define MASTER 0

#define MAXPROCS 16

*int* dot\_product(*int* *s*,*int* *e*, *int* *x*[], *int* *y*[], *int* *n*){

*int* i,prod=0;

for (i = *s*; i < *e*; i++)

prod = prod + *x*[i] \* *y*[i];

return prod;

}

*void* init\_lst(*int* \**l*,*int* *n*){

*int* i;

for (i=0; i<*n*; i++) \**l*++ = i;

}

*void* print\_lst(*int* *l*[],*int* *n*){

*int* i;

for (i=0; i<*n*; i++) {

printf("%d ", *l*[i]);

}

printf("\n");

}

*int* main() {

*int* i,n,vector\_x[NELMS],vector\_y[NELMS];

*int* prod,sidx,eidx,size;

*int* pid,nprocs, rank;

*double* stime,etime;

MPI\_Status status;

MPI\_Comm world;

n = 100000;

if (n > NELMS) { printf("n=%d > N=%d\n",n,NELMS); exit(1); }

MPI\_Init(NULL, NULL);

world = MPI\_COMM\_WORLD;

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

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

*int* portion = n / nprocs;

sidx = pid \* portion;

eidx = sidx + portion;

init\_lst(vector\_x, n);

init\_lst(vector\_y, n);

*int* tmp\_prod[nprocs];

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

tmp\_prod[i] = 0;

stime = MPI\_Wtime();

if (pid == MASTER) {

prod = dot\_product(sidx, eidx, vector\_x, vector\_y, n);

for (i = 1; i < nprocs; i++)

MPI\_Recv(&tmp\_prod[i-1], 1, MPI\_INT, i, 123, MPI\_COMM\_WORLD, &status);

}

else {

prod = dot\_product(sidx, eidx, vector\_x, vector\_y, n);

MPI\_Send(&prod, 1, MPI\_INT, MASTER, 123, MPI\_COMM\_WORLD);

}

if (pid == MASTER) {

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

prod += tmp\_prod[i];

}

etime = MPI\_Wtime();

if (pid == MASTER) {

//print\_lst(vector\_x,n);

//print\_lst(vector\_y,n);

printf("pid=%d: final prod=%d\n",pid,prod);

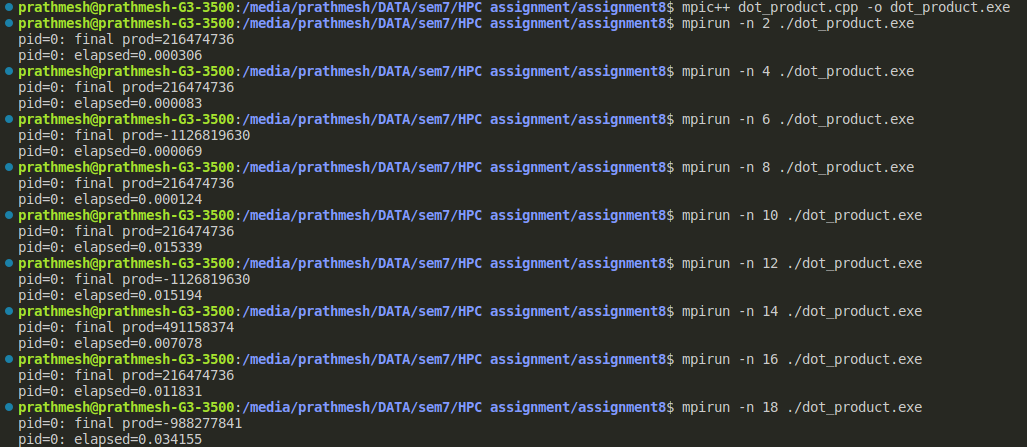
printf("pid=%d: elapsed=%f\n",pid,etime-stime);

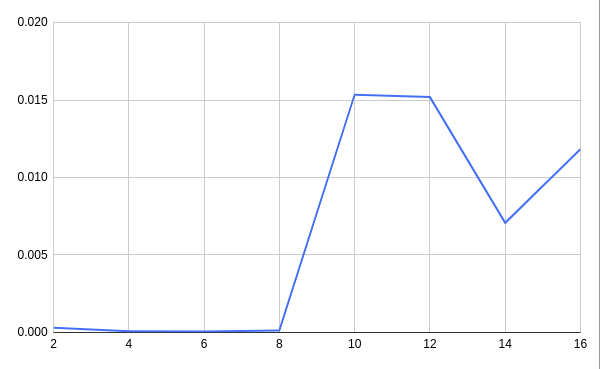
}

MPI\_Finalize();

}

**Output:**





3. Implement Prefix sum using MPI. Use different number of processes and analyze the performance.

Code:-

#include <stdio.h>

#include<stdlib.h>

#include <math.h>

#include "mpi.h"

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

*int* my\_rank; /\* rank of process \*/

*int* p; /\* number of processes \*/

MPI\_Status status ; /\* return status for receive \*/

*int* value;

/\* start up MPI \*/

MPI\_Init(&*argc*, &*argv*);

/\* find out process rank \*/

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

/\* find out number of processes \*/

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

*int* prefix\_arr[p];

/\* getting input and scatter values \*/

if(my\_rank == 0){

*int* i;

for(i = 0; i < p; ++i){

prefix\_arr[i] = i + 1;

}

}

*double* start = MPI\_Wtime();

//all call scatter

MPI\_Scatter(prefix\_arr, 1, MPI\_INT, &value, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

/\*

prefix sum:

repeat log n times

each time, if we are the chosen one, we receve a value from someone and add to ours

otherwise, we send to the chosen one

\*/

*int* i;

*int* logn = log2(p);

for(i = 0; i <= logn; i++){

*int* lower\_bound = pow(2,i);

*int* upper\_bound = p - lower\_bound;

if(upper\_bound < lower\_bound){

upper\_bound = lower\_bound;

}

if(my\_rank < lower\_bound){

*int* send = (*int*) (my\_rank + pow(2,i));

if(send >= p)

continue;

printf("%d sending to %d\n", my\_rank, (*int*) (my\_rank+pow(2,i)));

MPI\_Send(&value, 1, MPI\_INT, (*int*) (my\_rank+pow(2,i)), 0, MPI\_COMM\_WORLD);

}

else if(my\_rank >= upper\_bound){

*int* recv = (*int*) (my\_rank - pow(2,i));

if(recv >= p)

continue;

*int* recv\_value;

printf("%d receving..\n", my\_rank);

MPI\_Recv(&recv\_value, 1, MPI\_INT, (my\_rank - pow(2,i)), 0, MPI\_COMM\_WORLD, &status);

value += recv\_value;

}

else{

*int* send = (*int*) (my\_rank + pow(2,i));

*int* recv = (*int*) (my\_rank - pow(2,i));

if(send >= p || recv >= p)

continue;

printf("%d sending to %d\n", my\_rank, (*int*) (my\_rank+pow(2,i)));

MPI\_Send(&value, 1, MPI\_INT, (*int*) (my\_rank+pow(2,i)), 0, MPI\_COMM\_WORLD);

printf("%d receving..\n", my\_rank);

*int* recv\_value;

MPI\_Status status;

MPI\_Recv(&recv\_value, 1, MPI\_INT, (my\_rank - pow(2,i)), 0, MPI\_COMM\_WORLD, &status);

value += recv\_value;

}

}

//after algorithm, each processor hols its own prefix sum

//we gather at rank

*int* gather[p];

MPI\_Gather(&value, 1, MPI\_INT, gather, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if(my\_rank == 0){

*double* end = MPI\_Wtime();

printf("Execution Time: %f\n", end - start);

}

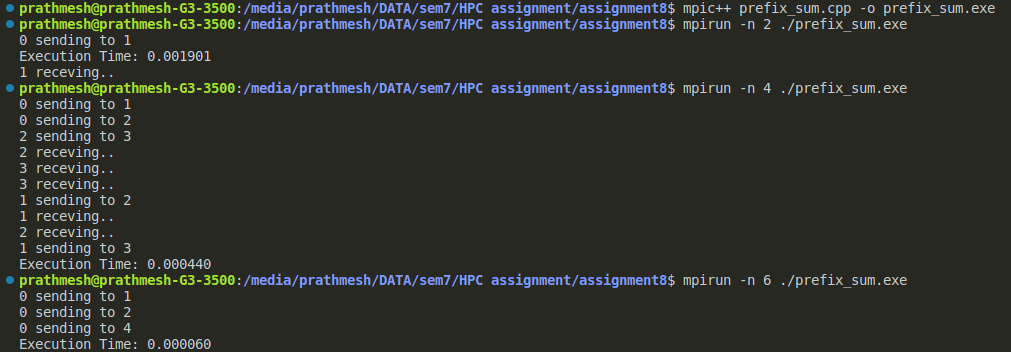
/\* shut down MPI \*/

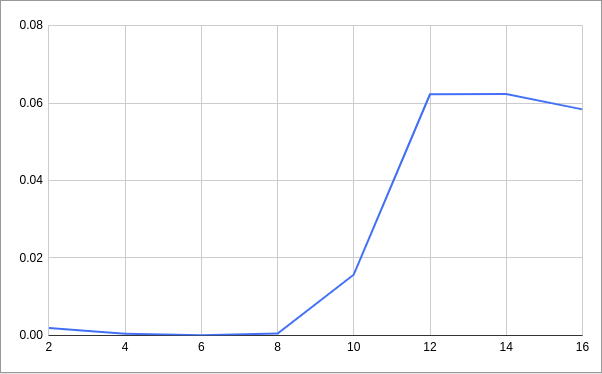
MPI\_Finalize();

return 0;

}

Output:-



**Github Link:**

[**https://github.com/killedar27/HPC-assignments/tree/main/assignment8**](https://github.com/killedar27/HPC-assignments/tree/main/assignment8)