HPC LAB PROGRAMS

PROGRAM-1

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

int main ()

{

int n,m;

printf("Enter the size of the matrix:");

scanf("%d",&n);

printf("Enter thre size of the vector:");

scanf("%d",&m);

if(m!=n)

{

printf("Multiplication is not possible.\n");

exit(0);

}

int i= 0 , j = 0;

int \*\*arr=(int\*\*)malloc(n\*sizeof(int\*));

int\*vec = (int\*)malloc(n\*sizeof(int));

int\*res = (int\*)malloc(n\*sizeof(int));

omp\_set\_num\_threads(n);

#pragma omp parallel private(j)

{

#pragma omp for

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

{

srand(i);

arr[i]=(int\*)malloc(n\*sizeof(int));

vec[i]=rand()%100;

for(j=0;j<n;j++)

arr[i][j]=rand()%100;

}

}

#pragma omp parallel private(j)

{

#pragma omp for

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

{

res[i]=0;

for(j=0;j<n;j++)

res[i]+=arr[i][j]\*vec[j];

}

}

printf("Matrix\*Vector = Resultant Mtrix \n");

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

{

for(j=0;j<n;j++)

printf("%3d",arr[i][j]);

if(i ==n/2)

printf("\* %3d =%6d\n", vec[i],res[i]);

else

printf("\* %3d =%6d\n", vec[i],res[i]);

}

}

PROGRAM-2

#include <stdio.h>

#include <omp.h>

#include<stdlib.h>

int main()

{

#define NUM\_THREADS 4

int sum = 0, sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0;

double start = omp\_get\_wtime();

double end = omp\_get\_wtime();

omp\_set\_num\_threads(4);

printf("Enter the items in clothing \n");

const int c1 = 5, c2 = 6, c3 = 7, c4 = 8;

int clo[c1];

printf("Clothing: ");

for (int i = 0; i <= c1; i++) {

clo[i] = 5;

printf("%d \n", clo[i]);

}

printf("grocery: ");

int gro[c2];

for (int i = 0; i <= c2; i++) {

gro[i] = 6;

printf("%d \n", gro[i]);

}

printf("Stationery:");

int sta[c3];

for (int i = 0; i <= c3; i++) {

sta[i] = 7;

printf("%d \n", sta[i]);

}

printf("Gaming: ");

int gam[c4];

for (int i = 0; i <= c4; i++) {

gam[i] = 8;

printf("%d \n", gam[i]);

}

#pragma omp parallel firstprivate(sum1, sum2, sum3, sum4)

{

int threadnum = omp\_get\_thread\_num();

if (threadnum == 0)

{

for (int i = 0; i <= c1; i++) {

sum1 += clo[i];

}

printf(" Thread id %d, Total sum1 = %d \n", omp\_get\_thread\_num(), sum1);

}

if (threadnum == 1)

{

for (int i = 0; i <= c2; i++) {

sum2 += gro[i];

}

printf(" Thread id %d, Total sum2 = %d \n", omp\_get\_thread\_num(), sum2);

}

if (threadnum == 2)

{

for (int i = 0; i <= c3; i++) {

sum3 += sta[i];

}

printf(" Thread id %d, Total sum3 = %d \n", omp\_get\_thread\_num(), sum3);

}

if (threadnum == 3)

{

for (int i = 0; i <= c4; i++) {

sum4 += gam[i];

}

printf(" Thread id %d, Total sum4 = %d \n", omp\_get\_thread\_num(), sum4);

}

#pragma omp critical

{

sum += sum1 + sum2 + sum3 + sum4;

}

}

printf(" Total sum = %d \n", sum);

}

PROGRAM-3

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<omp.h>

void main() {

int num, i;

printf("Enter the number of steps : ");

scanf("%d", &num);

time\_t st, et;

st = clock();

double step = 1.0 / (double)num, pi = 0.0;

omp\_set\_num\_threads(num);

#pragma omp parallel for

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

double x = (i + 0.5) \* step;

double local\_pi = (4.0 \* step) / (1 + x \* x);

#pragma omp atomic

pi += local\_pi;

}

et = clock();

printf("Time Taken : %lf\n", (double)((double)(et - st) / CLOCKS\_PER\_SEC));

printf("Value of Pi = %lf\n", pi);

}

PROGRAM-4

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<omp.h>

int main() {

int n, i;

printf("Number of terms : ");

scanf("%d",&n);

int\* a = (int\*)malloc(n \* sizeof(int));

a[0] = 0;

a[1] = 1;

time\_t st, et;

st = clock();

omp\_set\_num\_threads(2);

#pragma omp parallel

{

#pragma omp single

{

printf("id of thread involved in the computation of fibonacci numbers = %d\n", omp\_get\_thread\_num());

for (i = 2; i < n; i++)

a[i] = a[i - 2] + a[i - 1];

}

#pragma omp single

{

printf("id of thread involved in the displaying of fibonacci numbers = %d\n", omp\_get\_thread\_num());

printf("Fibonacci numbers : ");

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

printf("%d ", a[i]);

printf("\n");

}

}

et = clock();

printf("Time Taken : %lfms\n", ((double)(et - st)\*1000 / CLOCKS\_PER\_SEC));

return 0;

}

PROGRAM-5

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<omp.h>

int main() {

int n, i;

time\_t st, et;

st = clock();

printf("Enter the number of students : ");

scanf("%d", &n);

double\* arr = (double\*)malloc(n \* sizeof(double));

double arr\_max = 0;

#pragma omp parallel for

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

srand(i);

arr[i] = (double)(rand() % 100)/10 ;

}

printf("CGPA of students : ");

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

printf("%.2lf ", arr[i]);

printf("\n");

#pragma omp parallel for

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

#pragma omp critical

if (arr\_max < arr[i])

arr\_max = arr[i];

}

et = clock();

printf("Student with highest CGPA = %.2lf\n", arr\_max);

printf("Time Taken : %.2lfms\n", ((double)(et - st) \* 1000 / CLOCKS\_PER\_SEC));

}

PROGRAM-6

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<omp.h>

void main() {

int n;

printf("Enter the dimension of square matrices : ");

scanf("%d", &n);

int i = 0, j = 0, k = 0;

int\*\* arr1 = (int\*\*)malloc(n \* sizeof(int\*));

int\*\* arr2 = (int\*\*)malloc(n \* sizeof(int\*));

int\*\* res = (int\*\*)malloc(n \* sizeof(int\*));

omp\_set\_num\_threads(64);

#pragma omp parallel private(j)

{

#pragma omp for

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

srand(i);

arr1[i] = (int\*)malloc(n \* sizeof(int));

arr2[i] = (int\*)malloc(n \* sizeof(int));

res[i] = (int\*)malloc(n \* sizeof(int));

for (j = 0; j < n; j++) {

arr1[i][j] = rand() % 100;

arr2[i][j] = rand() % 100;

}

}

}

time\_t st, et;

st = clock();

#pragma omp parallel private(j,k)

{

#pragma omp for

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

for (j = 0; j < n; j++) {

res[i][j] = 0;

for (k = 0; k < n; k++)

res[i][j] += arr1[i][k] \* arr2[k][j];

}

}

}

et = clock();

printf("Time taken by parallel algorithm : %lf\n", (double)(et - st) / CLOCKS\_PER\_SEC);

st = clock();

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

for (j = 0; j < n; j++) {

res[i][j] = 0;

for (k = 0; k < n; k++)

res[i][j] += arr1[i][k] \* arr2[k][j];

}

}

et = clock();

printf("Time taken by Sequential algorithm : %lf\n", (double)(et - st) / CLOCKS\_PER\_SEC);

}