**PP LAB WEEK-5**

# DSE VI-A2 Divansh Prasad 210968140

1) Write a parallel program using OpenMP to perform vector addition, subtraction, multiplication. Demonstrate task level parallelism. Analyze the speedup and efficiency of the parallelized code.

#include <stdio.h>

#include <omp.h>

#include <time.h>

#include <windows.h>

void generateRandomVector(int \*vector, int size) {

srand(time(NULL));

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

vector[i] = rand() % 100;

}

}

void VectorAddition(int \*a, int \*b, int \*c, int n) {

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

c[i] = a[i] + b[i];

}

}

void VectorSubtraction(int \*a, int \*b, int \*c, int n) {

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

c[i] = a[i] - b[i];

}

}

void VectorMultiplication(int \*a, int \*b, int \*c, int n) {

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

c[i] = a[i] \* b[i];

}

}

int main() {

printf("Vector Size\tThreads\tSequential Time (s)\tParallel Time (s)\tSpeedup\t\tEfficiency\n");

for (int size = 200; size <= 800; size += 200) {

int v1[size], v2[size], add[size], sub[size], mult[size];

generateRandomVector(v1, size);

generateRandomVector(v2, size);

for (int num\_threads = 2; num\_threads <= 8; num\_threads += 2) {

double sequential\_time = 0, parallel\_time = 0;

// Sequential operations

clock\_t start = clock();Sleep(10);

VectorAddition(v1, v2, add, size);

VectorSubtraction(v1, v2, sub, size);

VectorMultiplication(v1, v2, mult, size);

clock\_t end = clock();

sequential\_time = ((double)(end - start)) / CLOCKS\_PER\_SEC;

// Parallel operations

start = clock();Sleep(10);

#pragma omp parallel sections

{

#pragma omp section

VectorAddition(v1, v2, add, size);

#pragma omp section

VectorSubtraction(v1, v2, sub, size);

#pragma omp section

VectorMultiplication(v1, v2, mult, size);

}

end = clock();

parallel\_time = ((double)(end - start)) / CLOCKS\_PER\_SEC;

double speedup = sequential\_time / parallel\_time;

double efficiency = speedup / num\_threads;

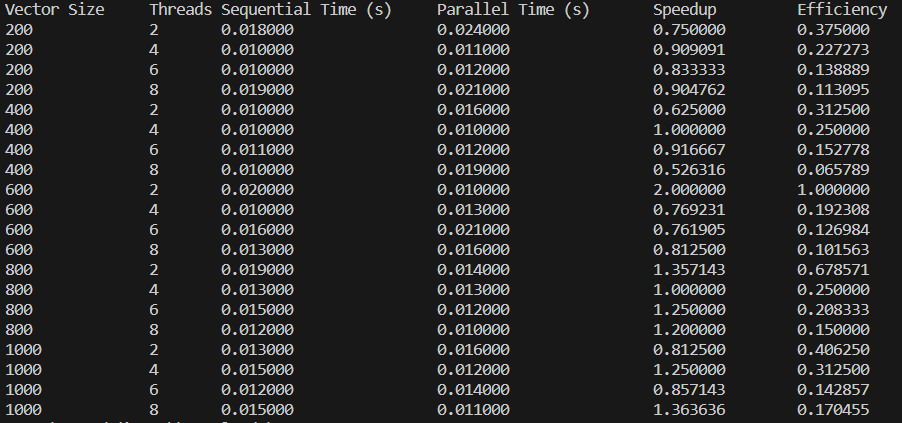
printf("%d\t\t%d\t%.6f\t\t%.6f\t\t%.6f\t%.6f\n", size, num\_threads, sequential\_time, parallel\_time, speedup, efficiency);

}

}

return 0;

}



2) Write a parallel program using OpenMP to find sum of N numbers using the following constructs/clauses.

a. Critical section

b. Atomic

c. Reduction

d. Master

e. Locks

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include<windows.h>

int main ()

{

int sum = 0;

clock\_t start, end;

double cpu\_time\_used=0;

int n;

printf("Enter N: ");

scanf("%d",&n);

int total\_threads\_used;/\*

#pragma omp master

{start = clock();Sleep(10);

omp\_lock\_t writelock;

omp\_init\_lock(&writelock);

#pragma omp parallel num\_threads(n) reduction (+:sum)

{

total\_threads\_used = n;

#pragma omp for

for(int i=1;i<=n;i++){

omp\_set\_lock(&writelock);

#pragma omp critical

#pragma omp atomic

sum = sum + i;

omp\_unset\_lock(&writelock);

}

}

omp\_destroy\_lock(&writelock);

}\*/

start = clock();

Sleep(10);

#pragma omp for reduction(+:sum)

for(int i=1;i<=n;i++){sum = sum + i;}

printf("\nThe total sum is %d\n", sum);

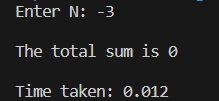
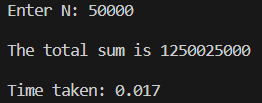
end = clock();

cpu\_time\_used=cpu\_time\_used +((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("\nTime taken: %0.3f\n",cpu\_time\_used);

return 0;

}



3) Write a parallel program using OpenMP to implement the Odd-even transposition sort. Vary the input size and analyse the program efficiency.  
  
#include<stdlib.h>

#include<stdio.h>

#include<time.h>

#include<string.h>

#include <omp.h>

#include<windows.h>

#define MAX\_VALUE 1000

void odd\_even\_sort(int\* a, int n)

{

int phase, i, temp;

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

{

if(phase % 2==0) //even phase

{

#pragma omp parallel for shared(a, n) private(i, temp)

for(i = 1; i < n; i += 2)

{

if(a[i-1] > a[i])

{

temp = a[i];

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

a[i-1] = temp;

}

}

}

else //odd phase

{

#pragma omp parallel for shared(a, n) private(i, temp)

for(i = 1; i < n-1; i += 2)

{

if(a[i] > a[i+1])

{

temp = a[i];

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

a[i+1] = temp;

}

}

}

}

}

void generate\_array(int\* a, int size)

{

int i = 0;

srand(time(NULL));

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

{

a[i] = rand() % MAX\_VALUE;

}

}

int main()

{

printf("Array Size\tThreads\tSequential Time (s)\tParallel Time (s)\tSpeedup\t\tEfficiency\n");

for (int size = 200; size <= 800; size += 200) {

for (int threads = 2; threads <= size; threads \*= 2) {

double sequential\_time = 0, parallel\_time = 0;

int \*a = (int\*)calloc(size, sizeof(int));

int \*initial = (int\*)calloc(size, sizeof(int));

generate\_array(a, size);

memcpy(initial, a, size \* sizeof(int));

// Sequential operation

clock\_t start = clock();Sleep(10);

odd\_even\_sort(a, size);

clock\_t end = clock();

sequential\_time = ((double)(end - start)) / CLOCKS\_PER\_SEC;

// Parallel operation

#pragma omp parallel

start = clock();Sleep(10);

odd\_even\_sort(a, size);

end = clock();

parallel\_time = ((double)(end - start)) / CLOCKS\_PER\_SEC;

double speedup = sequential\_time / parallel\_time;

double efficiency = speedup / threads;

printf("%d\t\t%d\t%.6f\t\t%.6f\t\t%.6f\t%.6f\n", size, threads, sequential\_time, parallel\_time, speedup, efficiency);

free(a);

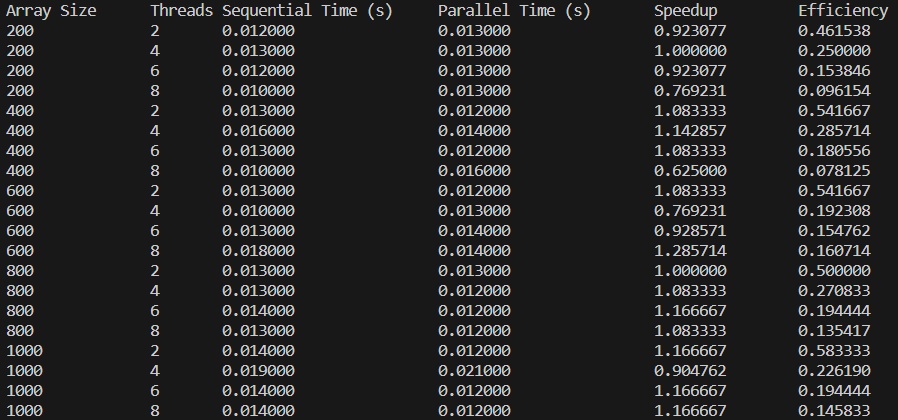
free(initial);

}

}

return 0;

}



4) Write an OpenMP program to find the Summation of integers from a given interval. Analyze the performance of various iteration scheduling strategies.

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include<windows.h>

int main ()

{

int sum = 0;

clock\_t start, end;

double cpu\_time\_used=0;

int low,high;

printf("Enter Lower Limit: ");

scanf("%d",&low);

printf("Enter Uppper Limit: ");

scanf("%d",&high);

int total\_threads\_used;

start = clock();

Sleep(10);

#pragma omp parallel num\_threads(high-low+1) reduction (+:sum)

{

total\_threads\_used = high-low+1;

#pragma omp for

for(int i=low;i<=high;i++){

sum = sum + i;

}

}

printf("\n\nThe total sum is %d\n\n\n", sum);

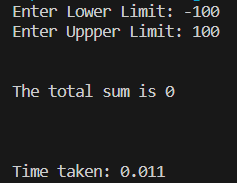
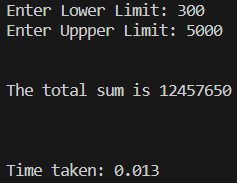
end = clock();

cpu\_time\_used=cpu\_time\_used +((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("\nTime taken: %0.3f\n",cpu\_time\_used);

return 0;

}



5) Write a parallel program using OpenMP to generate the histogram of the given array A.

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#define MAX\_VALUE 10

void generate\_array(int\* a, int size)

{

int i = 0;

srand(time(0));

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

{

a[i] = rand() % MAX\_VALUE;

}

}

int main() {

printf("Array Size\tThreads\tTime (s)\n");

int temp;

for (int size = 200; size <= 1200; size += 200) {

for (int num\_threads = 2; num\_threads <= 8; num\_threads += 2) {

double cpu\_time\_used = 0;

int \*array = (int\*)calloc(size, sizeof(int));

generate\_array(array, size);

clock\_t start = clock();Sleep(10);

#pragma omp parallel for num\_threads(num\_threads) shared(array, size) private(i, j, size, temp)

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

for (int j = 0; j < size; ++j) {

if (array[j] < temp) {

// Do nothing

} else {

// Print #

--array[j];

}

--temp;

}

}

clock\_t end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("%d\t\t%d\t%.6f\n", size, num\_threads, cpu\_time\_used);

free(array);

}

}

return 0;

}

