**Abhishek N N**

**20BCE1025**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| Programme | : | **B.Tech.(CSE)** | Semester | : | **Fall ’22-23** | |
| Course | : | **Parallel and Distributed Computing** | Code | : | **CSE4001** | |
| Faculty | : | **R. Kumar** | Slot | : | **L9+L10** | |

1. Write a openMP program using section constructs

Function 1

Generate 100000 random numbers in an array X and find out the min value.

Function 2

Generate 1000 prime numbers using Sieve of Sundaram algorithm

Record your run times using omp\_get\_wtime() routine for Function 1 & Function 2.

**Code:**

#include <omp.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void function1() {

int arr[100000];

for (int i = 0; i < 100000; i++) arr[i] = rand() % 100000;

int min = arr[0];

for (int i = 1; i < 100000; i++)

if (arr[i] < min) min = arr[i];

printf("\nminimum in 100000 random elements is %d\n", min);

}

void SieveOfSundaram(int n) {

int nNew = (n - 1) / 2;

bool marked[nNew + 1];

memset(marked, false, sizeof(marked));

for (int i = 1; i <= nNew; i++)

for (int j = i; (i + j + 2 \* i \* j) <= nNew; j++)

marked[i + j + 2 \* i \* j] = true;

int primeArr[1000], p = 0;

if (n > 2) primeArr[p++] = 2;

for (int i = 1; i <= nNew; i++)

if (marked[i] == false)

primeArr[p++] = 2 \* i + 1;

printf("\n some prime numbers are: ");

for(int i=0;i<10;i++) printf("%d ",primeArr[i]);

printf("\n");

}

int main() {

#pragma omp parallel sections

{

#pragma omp section

{

double start = omp\_get\_wtime();

function1();

double end = omp\_get\_wtime();

printf("time taken function1 in seconds: %f", end - start);

}

#pragma omp section

{

double start = omp\_get\_wtime();

SieveOfSundaram(1000);

double end = omp\_get\_wtime();

printf("time taken function2 in seconds: %f", end - start);

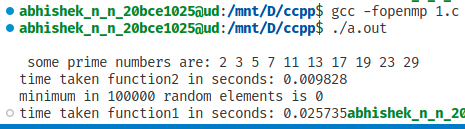
}

}

return 0;

}

**Output:**



1. Write a multithreaded program using OpenMP for computing a matrix-matrix product for a large dimension. Use the OMP\_NUM\_THREADS environment variable to control the number of threads and plot the performance with varying numbers

of threads (4,8 and 16). Consider four cases in which

i) Only the outermost loop is parallelized

ii) The outer two loops are parallelized

iii) All three loops are parallelized

iv) Use collapse clause

What is the observed result from these four cases?

Record your run times using omp\_get\_wtime() routine.

**Code:**

#include <omp.h>

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define N 1000

int A[N][N];

int B[N][N];

int C[N][N];

int main() {

int i, j, k;

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

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

A[i][j] = 2;

B[i][j] = 2;

}

int threadArr[] = {4, 8, 16};

for (int t = 0; t < 3; t++) {

omp\_set\_num\_threads(threadArr[t]);

double start = omp\_get\_wtime();

#pragma omp parallel for private(i, j, k) shared(A, B, C)

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

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

for (k = 0; k < N; ++k) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

double end = omp\_get\_wtime();

printf("%d threads, outermost: %f seconds\n", threadArr[t], end - start);

memset(C, 0, sizeof(C));

start = omp\_get\_wtime();

#pragma omp parallel for private(i, j, k) shared(A, B, C)

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

#pragma omp parallel for private(i, j, k) shared(A, B, C)

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

for (k = 0; k < N; ++k) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

end = omp\_get\_wtime();

printf("%d threads, outer two: %f seconds\n", threadArr[t], end - start);

memset(C, 0, sizeof(C));

#pragma omp parallel for private(i, j, k) shared(A, B, C)

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

#pragma omp parallel for private(i, j, k) shared(A, B, C)

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

#pragma omp parallel for private(i, j, k) shared(A, B, C)

for (k = 0; k < N; ++k) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

end = omp\_get\_wtime();

printf("%d threads, all loops : %f seconds\n", threadArr[t], end - start);

memset(C, 0, sizeof(C));

#pragma omp parallel for private(i, j, k) shared(A, B, C) collapse(3)

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

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

for (k = 0; k < N; ++k) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

end = omp\_get\_wtime();

printf("%d threads, collapse : %f seconds\n", threadArr[t], end - start);

}

return 0;

}

**Output**

