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:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ gcc -fopenmp 1.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ ./a.out

 some prime numbers are: 2 3 5 7 11 13 17 19 23 29
time taken function2 in seconds: 0.009828
minimum in 100000 random elements is 0
time taken function1 in seconds: 0.025735abhishek_n_n_20
```

2. 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:

```c
#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

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ gcc -fopenmp 1.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ ./a.out
4 threads, outermost: 3.027410 seconds
4 threads, outer two: 3.151151 seconds
4 threads, all loops : 7.940060 seconds
4 threads, collapse : 10.953520 seconds
8 threads, outermost: 2.265373 seconds
8 threads, outer two: 2.380733 seconds
8 threads, all loops : 7.850639 seconds
8 threads, collapse : 10.237010 seconds
16 threads, outermost: 2.169480 seconds
16 threads, outer two: 2.170995 seconds
16 threads, all loops : 7.458266 seconds
16 threads, collapse : 9.631188 seconds
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$
```