Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

Year: 2023-24 **Semester:** 1

Course: High Performance Computing Lab

Practical No. 2

Exam Seat No: 2020BTECS00041

Title of practical: Study and implementation of basic OpenMP clauses

Implement following Programs using OpenMP with C:

1. Vector Scalar Addition
2. Calculation of value of Pi
   Analyse the performance of your programs for different number of threads and Data size.

Problem Statement 1: Vector Scalar Addition

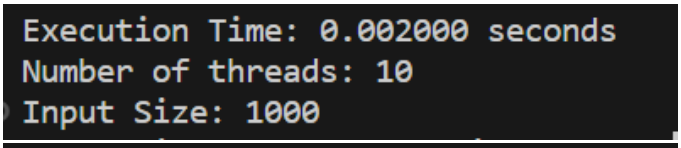
```c
#include <stdio.h>
#include <omp.h>
#include <time.h>

#define size 10
#define NUM_THREADS 5

void vectorScalarAdd(int* vector, int scalar)
{
    #pragma omp parallel for private(scalar) num_threads(NUM_THREADS)
        // #pragma omp nowait
        for (int i = 0; i < size; i++)
        {
            vector[i] += scalar;
        }
}

int main() {
    int vector[size];
    int scalar = 5;

    for (int i = 0; i < size; i++) {
        vector[i] = i+10;
```

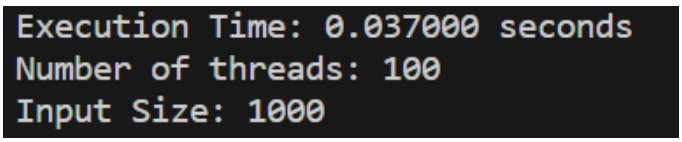
```
    }

    printf("Original Vector:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", vector[i]);
    }
    printf("\n");
    clock_t start_time = clock();
    vectorScalarAdd(vector, scalar);

    clock_t end_time = clock();
    double elapsed_time = (double)(end_time - start_time) /
CLOCKS_PER_SEC;


    printf("Vector after Scalar Addition:\n");
    for (int i = 0; i < size; i++)
    {
        printf("%d ", vector[i]);
    }
    printf("\n");
    printf("Execution Time: %f seconds\n", elapsed_time);
    printf("Number of threads: %d \n",NUM_THREADS);
    printf("Input Size: %d",size);

    return 0;
}
```

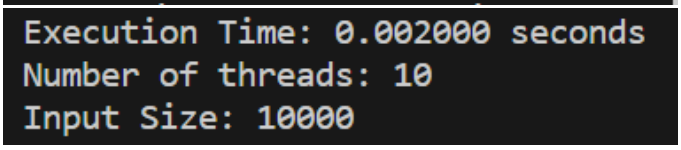
Using nowait clause for 10 data size and 5 threads

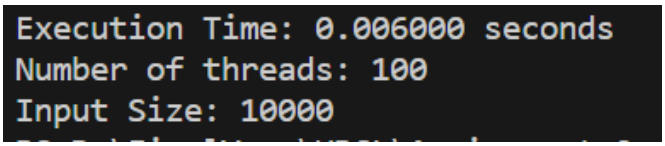
```
void vectorScalarAdd(int* vector, int scalar)
{
    #pragma omp parallel for private(scalar) num_threads(NUM_THREADS)
        #pragma omp nowait
        for (int i = 0; i < size; i++)
        {
            vector[i] += scalar;
        }
}
```

Screenshots:

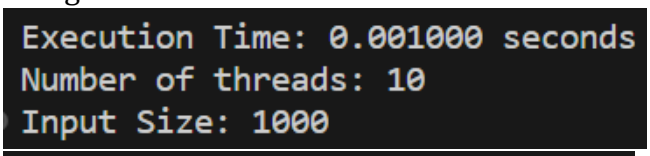


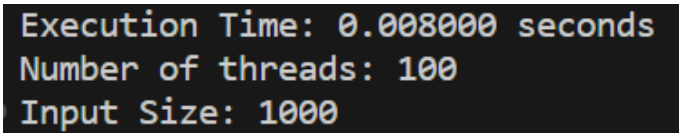


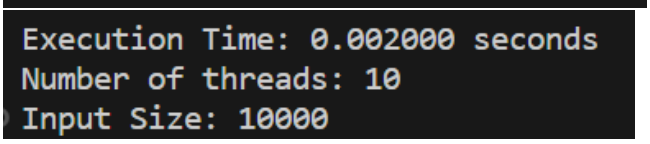


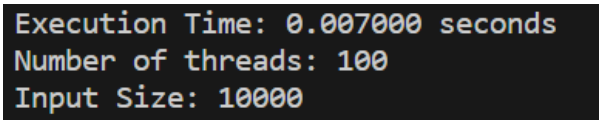


Using nowait









Information:

Code achieves parallelism by distributing the work of element-wise addition across multiple threads using OpenMP directives. This can lead to significant performance improvements in terms of execution time for large-scale computations. However, considerations for data safety, thread management, and memory access are essential for creating efficient and correct parallel code.

Analysis:

THREADS / INPUT SIZE	10	100
1000	0.002 sec	0.037 sec
10000	0.002 sec	0.006 sec

Using Private clause:

THREADS / INPUT SIZE	10	100
1000	0.001 sec	0.008 sec
10000	0.002 sec	0.007 sec

Using nowait clause for 100 input size and 10 threads: execution time is 0.001sec.

Problem Statement 2: Calculation of value of Pi

```
// Calaculation of pi
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define NUM_SAMPLES 10000000

int main()
{
    int num_threads[] = {1, 2, 4, 6, 8, 10}; // Number of threads to test
    int num_sizes[] = {1000, 900, 100000, 600}; // Diff data sizes to test
    for (int size_index = 0; size_index < sizeof(num_sizes) /
sizeof(num_sizes[0]); size_index++) {
        int size = num_sizes[size_index];
        double step = 1.0 / size;
        double pi_sum = 0.0;
        printf("Data size: %d\n", size);
        for (int thread_index = 0; thread_index < sizeof(num_threads) /
sizeof(num_threads[0]); thread_index++) {
            int num_thread = num_threads[thread_index];
            double pi_estimate = 0.0;

            omp_set_num_threads(num_thread);
            #pragma omp parallel
            {
                double local_sum = 0.0;
                #pragma omp for
                for (int i = 0; i < size; i++) {
                    double x = (i + 0.5) * step;
                    local_sum += 4.0 / (1.0 + x * x);
                }
                #pragma omp critical
                pi_sum += local_sum;
            }
            pi_estimate = step * pi_sum;
            printf("Threads: %d, Estimated Pi: %f\n", num_thread,
pi_estimate);
        }

        printf("\n");
    }
    return 0;
}
```

Screenshots:

```
PS D:\FinalYear\HPCL\Assignment 2> .\a.exe
Data size: 1000
Threads: 1, Estimated Pi: 3.141593
Threads: 2, Estimated Pi: 6.283185
Threads: 4, Estimated Pi: 9.424778
Threads: 6, Estimated Pi: 12.566371
Threads: 8, Estimated Pi: 15.707964
Threads: 10, Estimated Pi: 18.849556

Data size: 900
Threads: 1, Estimated Pi: 3.141593
Threads: 2, Estimated Pi: 6.283186
Threads: 4, Estimated Pi: 9.424778
Threads: 6, Estimated Pi: 12.566371
Threads: 8, Estimated Pi: 15.707964
Threads: 10, Estimated Pi: 18.849557

Data size: 100000
Threads: 1, Estimated Pi: 3.141593
Threads: 2, Estimated Pi: 6.283185
Threads: 4, Estimated Pi: 9.424778
Threads: 6, Estimated Pi: 12.566371
Threads: 8, Estimated Pi: 15.707963
Threads: 10, Estimated Pi: 18.849556

Data size: 600
Threads: 1, Estimated Pi: 3.141593
Threads: 2, Estimated Pi: 6.283186
Threads: 4, Estimated Pi: 9.424779
Threads: 6, Estimated Pi: 12.566372
Threads: 8, Estimated Pi: 15.707964
Threads: 10, Estimated Pi: 18.849557
```

Analysis:

Threads / DATA SIZE	1	2	4	6
600	**3.141593**	**6.283186**	**9.424779**	**12.566372**
900	**3.141593**	**6.283186**	**9.424778**	**12.566371**
1000	**3.141593**	**6.283185**	**9.424778**	**12.566371**
100000	**3.141593**	**6.283185**	**9.424778**	**12.566371**

Github Link: https://github.com/Nikita226/HPCL/tree/main/A2