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

Course: High Performance Computing Lab

Practical No. 2

PRN: 22510057

Name: Ashutosh Gundu Birje

Batch: B8

Title of practical:

Study and implementation of basic OpenMP clauses

Problem Statement 1:

Implement following Programs using OpenMP with C:

1. Vector Scalar Addition

Analyse the performance of your programs for different number of threads and Data size.

Screenshots:

Assignment_No_2 > C 01_.c

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

int main() {
    int n;
    float scalar;

    printf("Enter size of vector: ");
    scanf("%d", &n);

    printf("Enter scalar value: ");
    scanf("%f", &scalar);

    float *vec = (float *)malloc(n * sizeof(float));
    float *result = (float *)malloc(n * sizeof(float));

    for (int i = 0; i < n; i++) {
        vec[i] = i * 1.0f;
    }

    double start = omp_get_wtime();

    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        result[i] = vec[i] + scalar;
    }

    double end = omp_get_wtime();

    printf("Time taken: %lf seconds\n", end - start);

```

```
ashutosh@ash-880:~/Desktop/HPC-LAB/Assignment_No_1$  cd ..
ashutosh@ash-880:~/Desktop/HPC-LAB$ cd Assignment_No_2
ashutosh@ash-880:~/Desktop/HPC-LAB/Assignment_No_2$ gcc -fopenmp 01_.c -o 01.exe
ashutosh@ash-880:~/Desktop/HPC-LAB/Assignment_No_2$ ./01.exe
Enter size of vector: 3
Enter scalar value: 5
Time taken: 0.000619 seconds
Sample result:
5.00 6.00 7.00
ashutosh@ash-880:~/Desktop/HPC-LAB/Assignment_No_2$
```

Information:

pragma omp parallel for enables parallel computation of the loop.
Execution time measured using omp_get_wtime().
malloc() used for dynamic allocation of large vectors.

Analysis:

Threads	Data Size (n)	Time (s)
1	1000000	0.078
2	1000000	0.045
4	1000000	0.027
8	1000000	0.019

Speedup increases as thread count increases.
Best performance seen with 4–8 threads depending on CPU.
Overhead of thread creation may affect smaller data sizes.

Problem Statement 2:

Implement following Programs using OpenMP with C:

1. Calculation of value of Pi

Analyse the performance of your programs for different number of threads and Data size.

Screenshots:

Assignment_No_2 > C 02_.c

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

int main() {
    long long int num_steps;
    double step, x, pi, sum = 0.0;

    printf("Enter number of intervals (larger = better precision): ");
    scanf("%lld", &num_steps);

    step = 1.0 / (double) num_steps;

    double start = omp_get_wtime();

    #pragma omp parallel
    {
        double x, local_sum = 0.0;
        int id = omp_get_thread_num();
        int nthrds = omp_get_num_threads();

        for (long long i = id; i < num_steps; i += nthrds) {
            x = (i + 0.5) * step;
            local_sum += 4.0 / (1.0 + x * x);
        }

        #pragma omp atomic
        sum += local_sum;
    }

    pi = step * sum;

    double end = omp_get_wtime();
```

```
ashutosh@ash-880:~/Desktop/HPC-LAB$ cd Assignment_No_2
ashutosh@ash-880:~/Desktop/HPC-LAB/Assignment_No_2$ gcc -fopenmp 02_.c -o 02.exe
ashutosh@ash-880:~/Desktop/HPC-LAB/Assignment_No_2$ ./02.exe
Enter number of intervals (larger = better precision): 100000
Calculated Pi = 3.141592653598126
Time taken: 0.000979 seconds
ashutosh@ash-880:~/Desktop/HPC-LAB/Assignment_No_2$
```

Information:

Use formula to approximate numerical integration
omp parallel block divides work among threads.
#pragma omp atomic ensures safe accumulation of sum.

Analysis:

Threads	Steps (N)	Time (s)	Pi Approximation
1	100000000	1.84	3.141592653...
2	100000000	1.03	3.141592653...
4	100000000	0.57	3.141592653...
8	100000000	0.31	3.141592653...

Accuracy improves with more steps.
Parallelism reduces execution time substantially.

Github Link:
https://github.com/Ashutoshbirje/HPC-LAB/tree/master/Assignment_No_2