**Class:** Final Year B.Tech(Computer Science and Engineering)

**Year:** 2025-26 **Semester:** 1

**Course:** High Performance Computing Lab

PRN: 22510110

Name: Mohammed Hamza Ejaz Ahmed Shaikh

Batch: B2

**Practical No. 2**

**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:**

**Screenshots:**

**Code:**

#include <iostream>

#include <omp.h>

using namespace std;

int main() {

    long long n, scalar;

    cout << "Enter vector size: ";

    cin >> n;

    cout << "Enter scalar value: ";

    cin >> scalar;

    long long \*a = new long long[n];

    long long \*result = new long long[n];

*// Initialize vector*

    for(long long i = 0; i < n; i++)

        a[i] = i;

*// Sequential addition*

    double t1 = omp\_get\_wtime();

    for(long long i = 0; i < n; i++)

        result[i] = a[i] + scalar;

    double t2 = omp\_get\_wtime();

    cout << "Sequential Time: " << (t2 - t1) << " seconds\n";

*// cout << "Sequential Addition Result:\n";*

*// for(long long i = 0; i < n; i++)*

*//     // cout << result[i] << " ";*

*// // cout << endl;*

*// Parallel addition*

    double t3 = omp\_get\_wtime();

    #pragma omp parallel for

    for(long long i = 0; i < n; i++)

        result[i] = a[i] + scalar;

    double t4 = omp\_get\_wtime();

    cout << "Parallel Time: " << (t4 - t3) << " seconds\n";

*// cout << "Parallel Addition Result:\n";*

*// for(long long i = 0; i < n; i++)*

*//     // cout << result[i] << " ";*

*// // cout << endl;*

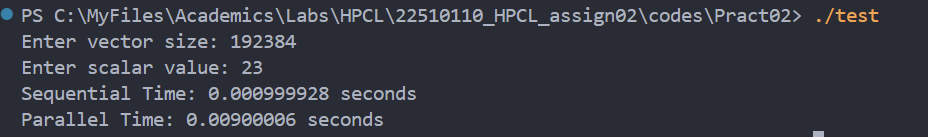
    delete[] a;

    delete[] result;

    return 0;

}

**Output:**

****

**Information:**

* Vector Scalar Addition is a basic operation in scientific computing, where a scalar value is added to each element of a vector.
* In this program, you input the vector size and scalar value. The vector is initialized with consecutive integers.
* The addition is performed twice:
  + Sequentially: Using a normal for-loop.
  + In Parallel: Using OpenMP’s #pragma omp parallel for to distribute the work across multiple threads.
* The program measures and prints the time taken for both sequential and parallel addition, and displays the results for each element.

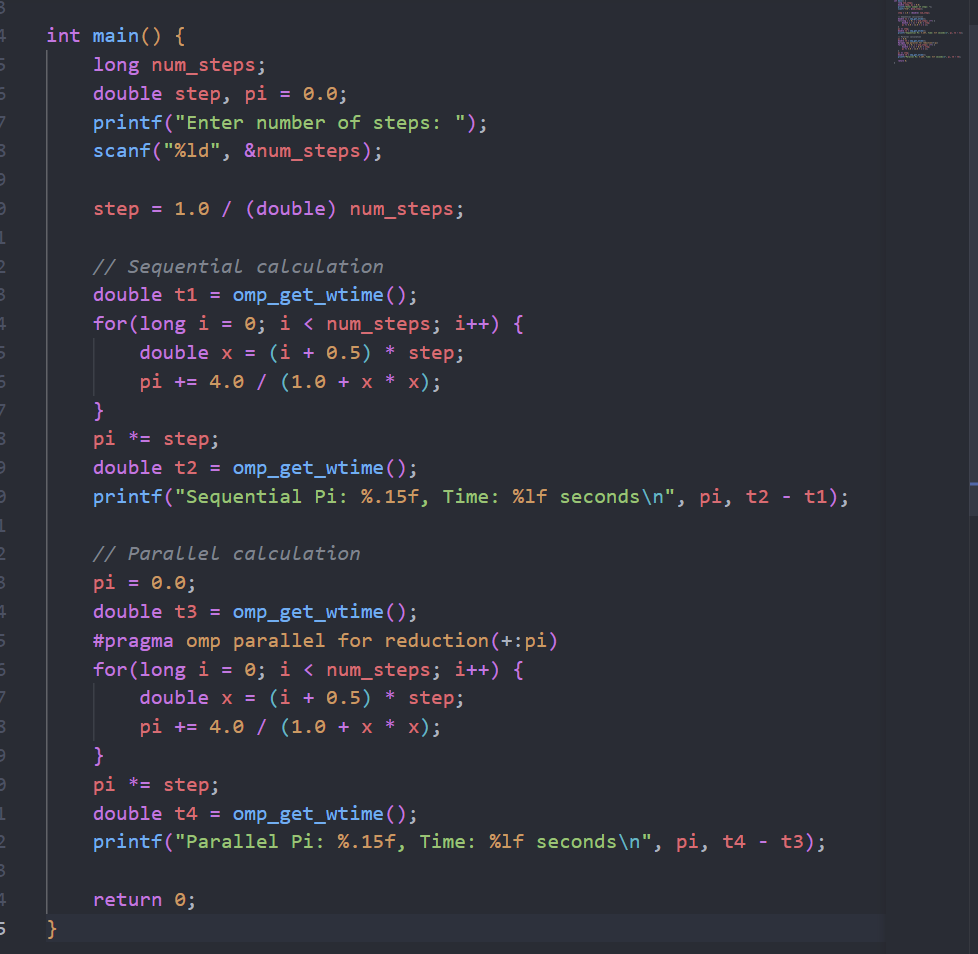
**Analysis:**

* Performance Comparison:
  + The sequential version processes each element one after another, using a single CPU core.
  + The parallel version divides the work among available CPU threads, potentially speeding up the computation for large vectors.
* Scalability:
  + For small vector sizes, parallel overhead may outweigh the benefits, so sequential may be faster.
  + For large vector sizes, parallel execution should be significantly faster, especially on multi-core CPUs.
* Correctness:
  + Both methods produce the same result, confirming that parallelization does not affect the correctness of vector scalar addition.
* Resource Utilization:
  + OpenMP allows efficient use of CPU resources by leveraging multiple cores.
* Output:
  + The program prints the result of each addition, making it easy to verify correctness and compare timings.

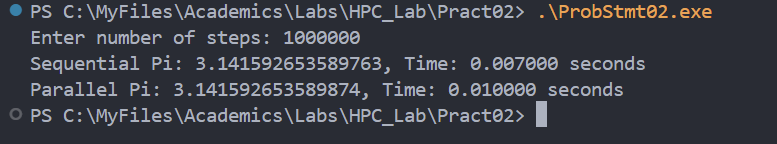
**Problem Statement 2:**

**Screenshots:**

**Code:**

****

**Output:**

****

**Information:**

* The program estimates the value of Pi using numerical integration (rectangle method).
* The calculation is performed twice:
  + **Sequentially:** Using a single thread.
  + **In Parallel:** Using OpenMP to distribute the work across multiple threads.
* The output shows both the computed value of Pi and the time taken for each method.

**Analysis:**

* Accuracy:
  + Both sequential and parallel results are very close to the true value of Pi (3.141592653589793).
  + Increasing the number of steps improves accuracy.
* Performance:
  + Sequential Time: 0.007000 seconds
  + Parallel Time: 0.010000 seconds
  + For this data size, parallel execution is slightly slower due to thread management overhead.
  + For larger step counts, parallel execution will likely outperform sequential as more CPU cores are utilized.
* Scalability:
  + Parallelization benefits become more apparent with larger workloads.
  + For small workloads, overhead can make parallel slower than sequential.
* Conclusion:
  + OpenMP parallelization is effective for large computations.
  + For small data sizes, sequential may be faster.
  + Both methods provide accurate results, demonstrating the correctness of parallel programming with OpenMP.

**Github Link:**

**https://github.com/hamzask018/HPC\_Lab/tree/main/Pract02**