**Class:** Final Year (Computer Science and Engineering)

**Year:** 2024-25 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 2**

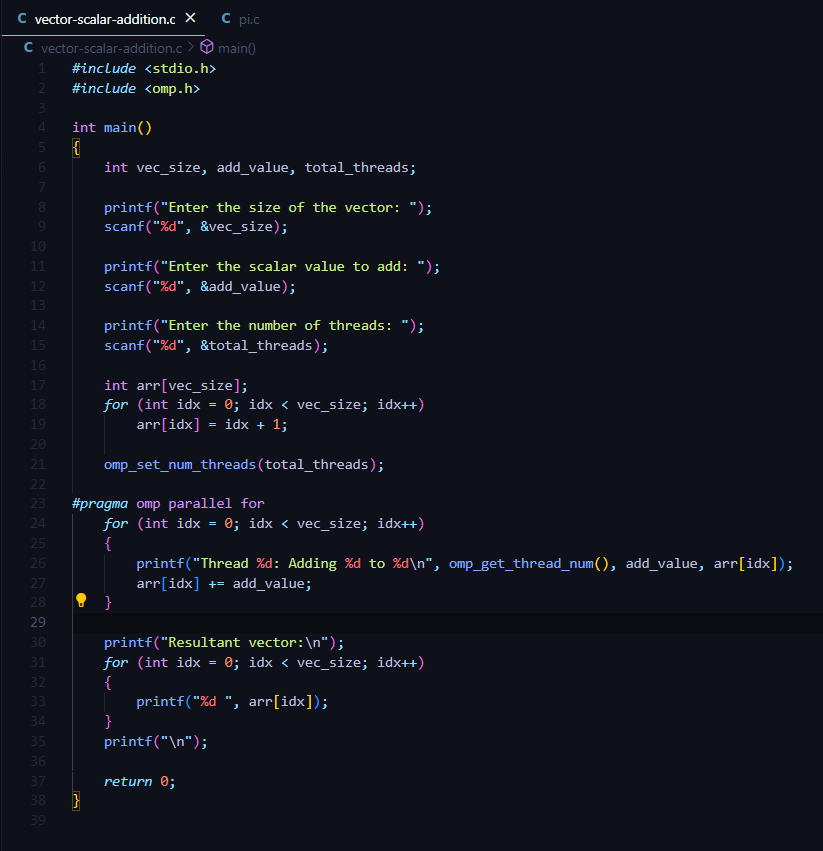
**Exam Seat No: 21510069**

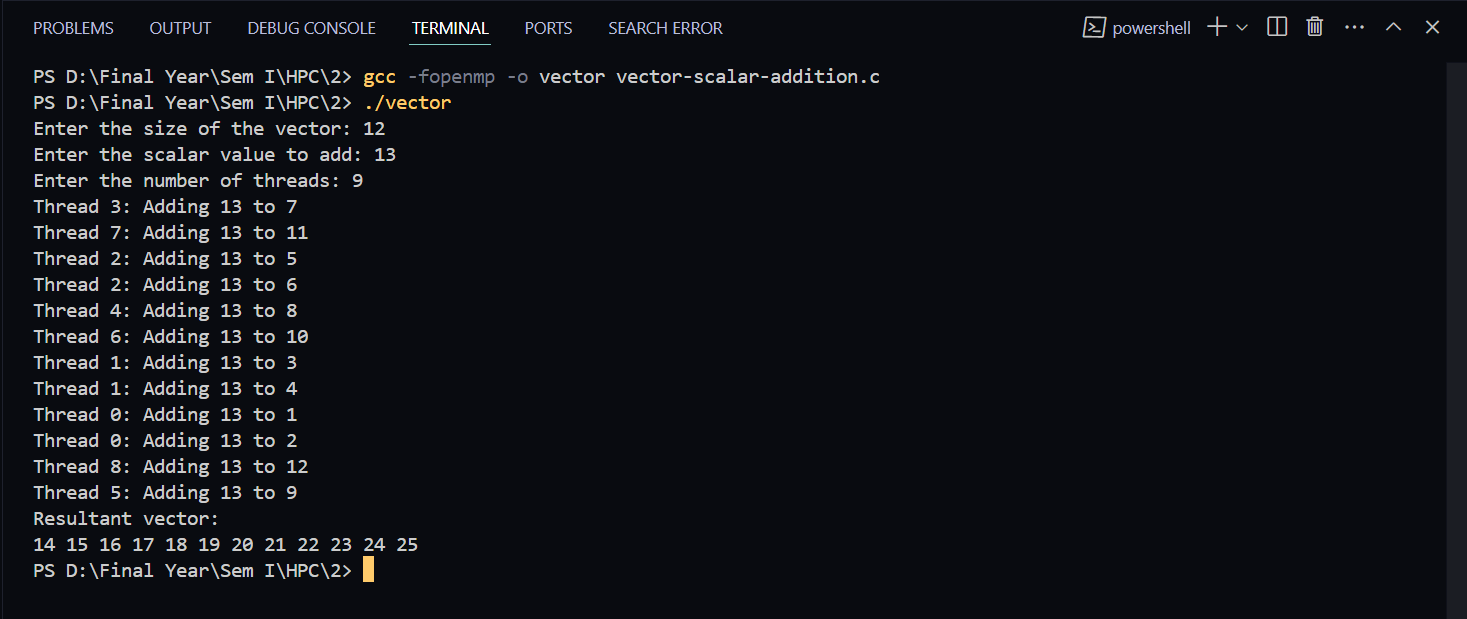
**Title of practical: Study and implementation of basic OpenMP clauses**

**Problem Statement**

**1: Vector Scalar Addition**

**Screenshots:**

****



**Information:**

This program performs vector-scalar addition in parallel using OpenMP. The user inputs the size of the vector, the scalar value to add, and the number of threads to use. The program initializes the vector with sequential values, then uses multiple threads to add the scalar value to each element in the vector. The parallel computation is handled by the

**#pragma omp parallel for** directive.

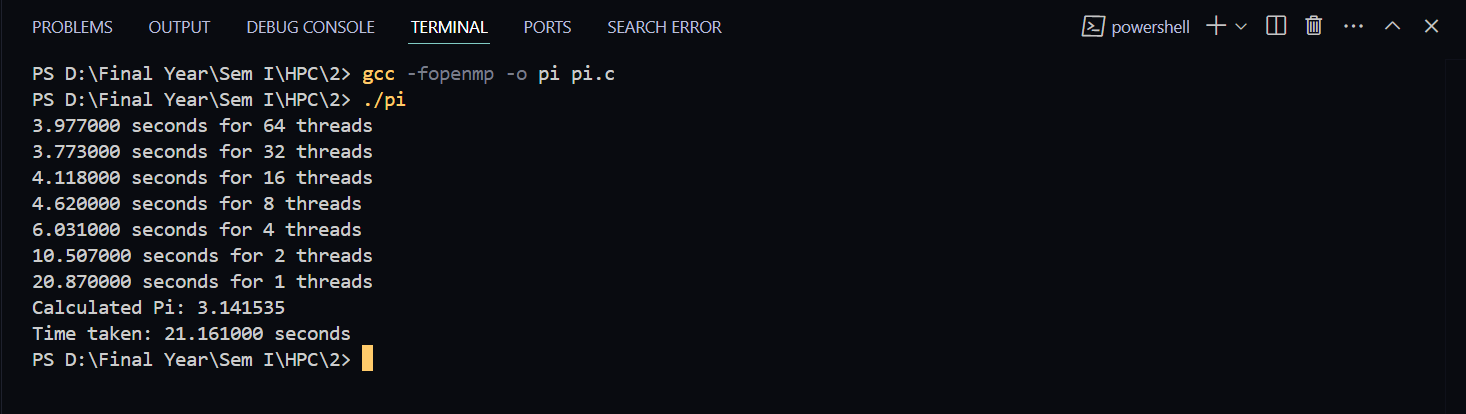
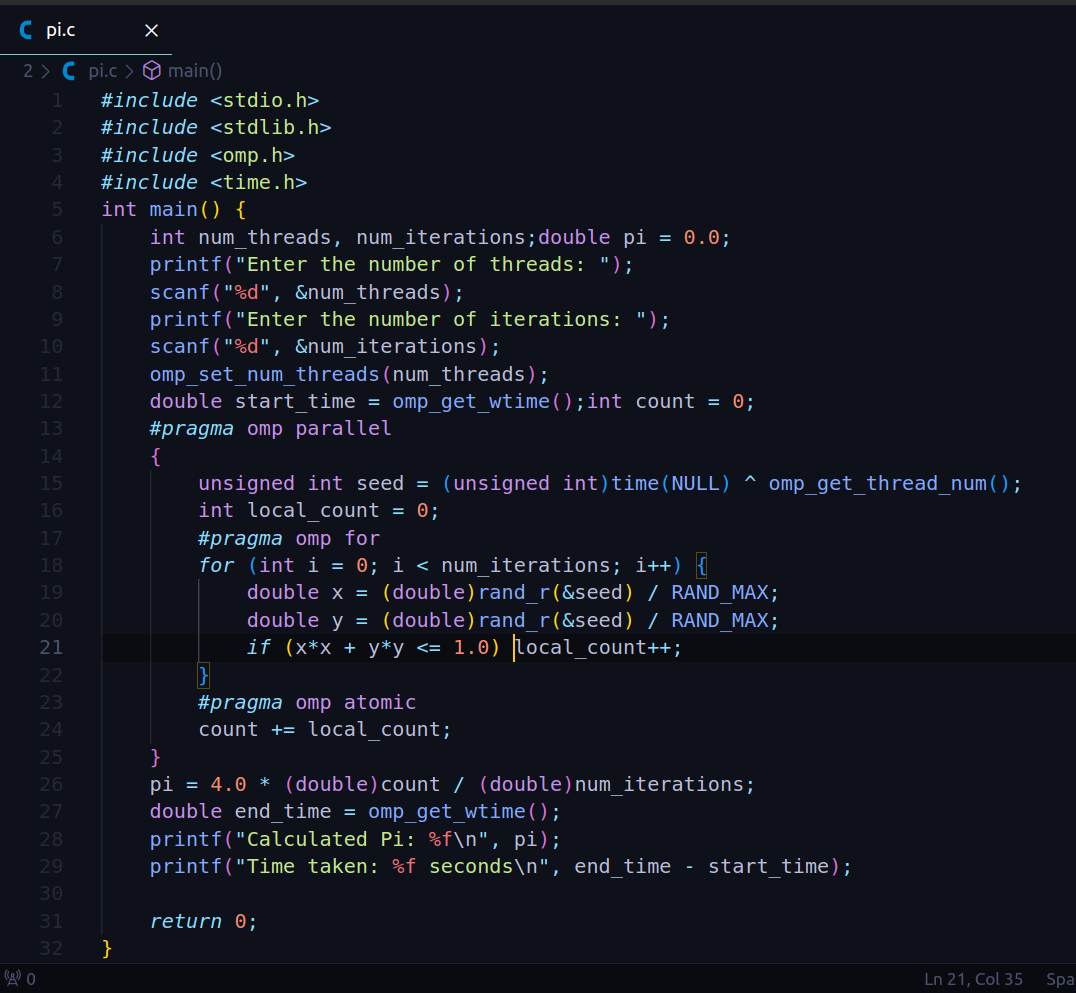
**Analysis:**

By setting the number of threads with **omp\_set\_num\_threads(num\_threads)**,the computation is distributed across multiple threads, potentially reducing execution time for large vectors. The program also showcases how OpenMP assigns different iterations of the loop to different threads, as indicated by the thread IDs in the output. This approach effectively speeds up the computation but may introduce overhead from thread management, especially for small vectors.

**Problem Statement 2:**

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

**Screenshots:**

****

**Information:**

This program estimates the value of Pi using the Monte Carlo method, implemented in C with OpenMP for parallel processing. The program prompts the user to input the number of threads and the number of iterations (data size) to perform. It then divides the work among the specified number of threads, where each thread generates random points within a square and counts how many fall inside a quarter circle. The ratio of points inside the circle to the total points is used to calculate an approximation of Pi. The program also measures and outputs the time taken to perform the computation.

Information:

### Analysis:

The program leverages OpenMP to parallelize the Monte Carlo simulation, which significantly reduces computation time as the number of threads increases. By analyzing the performance with different thread counts and data sizes, you can observe how the parallelization efficiency scales. For smaller data sizes, the overhead of thread management may not justify the parallel execution, while for larger data sizes, more threads typically lead to better performance. However, due to the random nature of the Monte Carlo method, running the simulation with a higher number of iterations is necessary for a more accurate Pi approximation. This demonstrates how computational workload and parallel processing interact to impact both performance and accuracy in high-performance computing scenarios.

**Github Link:**