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

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

**Course:** High Performance Computing Lab

**Practical No. 5**

**Exam Seat No:22510037**

**Name: Tejas Sanjay Chougule**

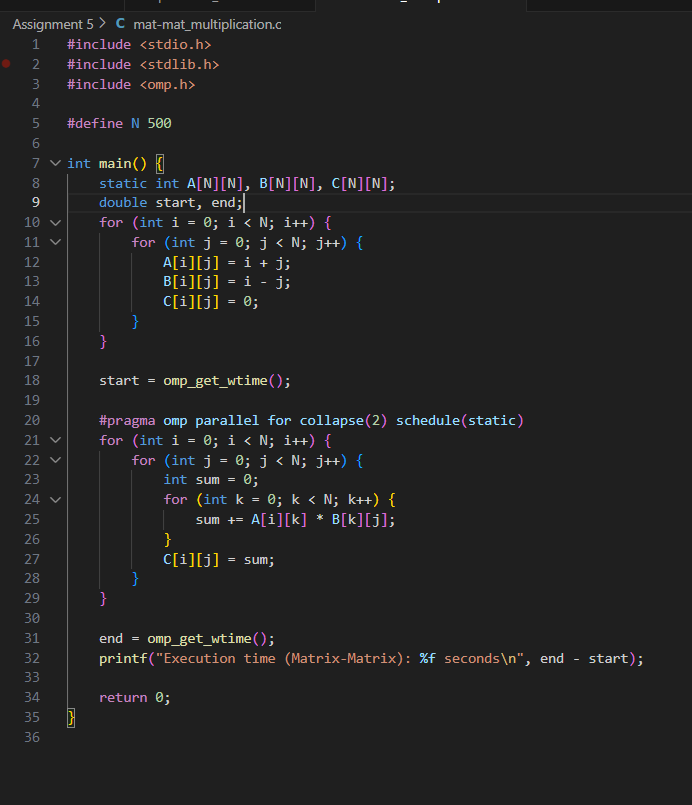
**Title of practical: Implementation of OpenMP programs.**

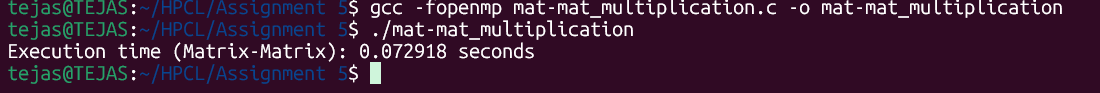
Implement following Programs using OpenMP with C:

1. Implementation of Matrix-Matrix Multiplication.
2. Implementation of Matrix-scalar Multiplication.
3. Implementation of Matrix-Vector Multiplication.
4. Implementation of Prefix sum.

**Problem Statement 1:**

**Screenshots:**

****

****

**Information:**

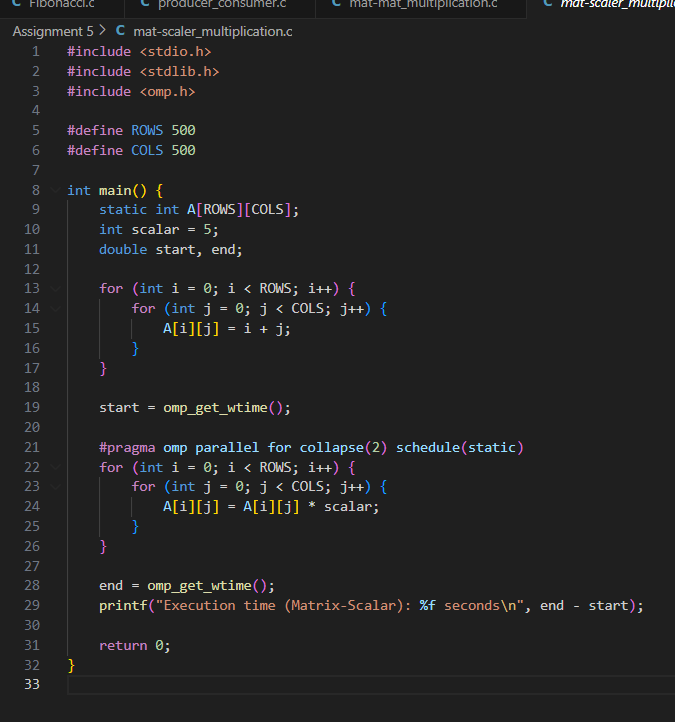
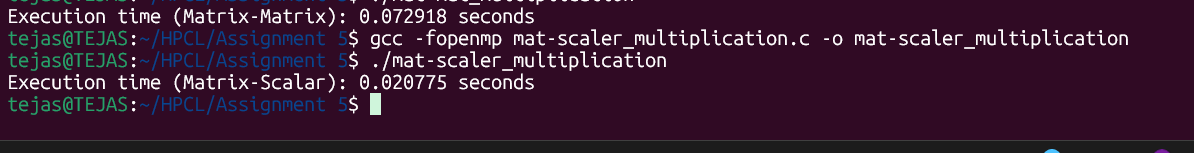
This program multiplies two matrices using OpenMP. The work is divided among threads using #pragma omp parallel for, where each thread computes a subset of rows of the result matrix.

**Analysis:**

The parallel implementation reduces computation time compared to the sequential version. Time complexity remains O(n3)O(n^3)O(n3), but with OpenMP the workload is shared, achieving near-linear speedup depending on the number of threads and system cores

**Problem Statement 2:**

**Screenshots:**

**** ****

**Information:**

The program multiplies each element of a matrix by a scalar. Parallelization is achieved using #pragma omp parallel for, allowing multiple elements to be updated concurrently

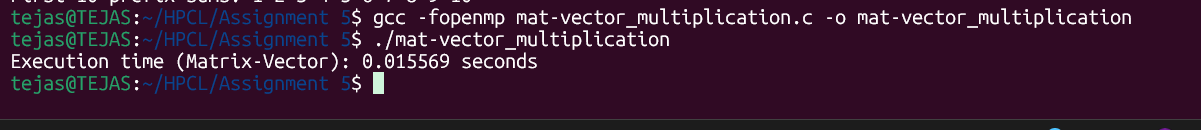
**Analysis:**

This task is embarrassingly parallel and achieves good scalability. Time complexity is O(n2)O(n^2)O(n2), with significant reduction in execution time for large matrices.

**Problem Statement 3:**

**Screenshots:**

****

****

**Information:**

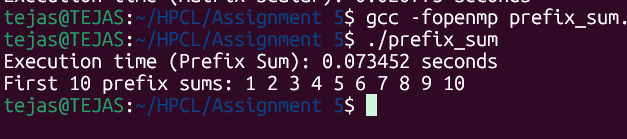
This program multiplies a matrix with a vector using OpenMP. Parallelization is applied over rows of the matrix, where each thread computes the dot product of one or more rows.

**Analysis:**

Time complexity is O(n2)O(n^2)O(n2). Parallelization reduces execution time by distributing independent dot-product computations among threads.

**Problem Statement 4:**

**Screenshots:**

**** ****

**Information:**

The prefix sum is implemented using OpenMP, where the array is divided among threads. Each thread computes partial sums, and then results are merged.

**Analysis:**

Sequential prefix sum takes O(n)O(n)O(n). Parallel prefix sum reduces time by using a tree-structured computation (O(log⁡n)O(\log n)O(logn) steps with sufficient threads). However, synchronization overhead may affect performance for small inputs.

**Github Link: https://github.com/TSC2004/Hpcl**