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

Class: Final Year (CSE) Year: 2025-26 Semester: 1

Course: High Performance Computing Lab

**Practical No. 5**

**Exam Seat No: 22510058**

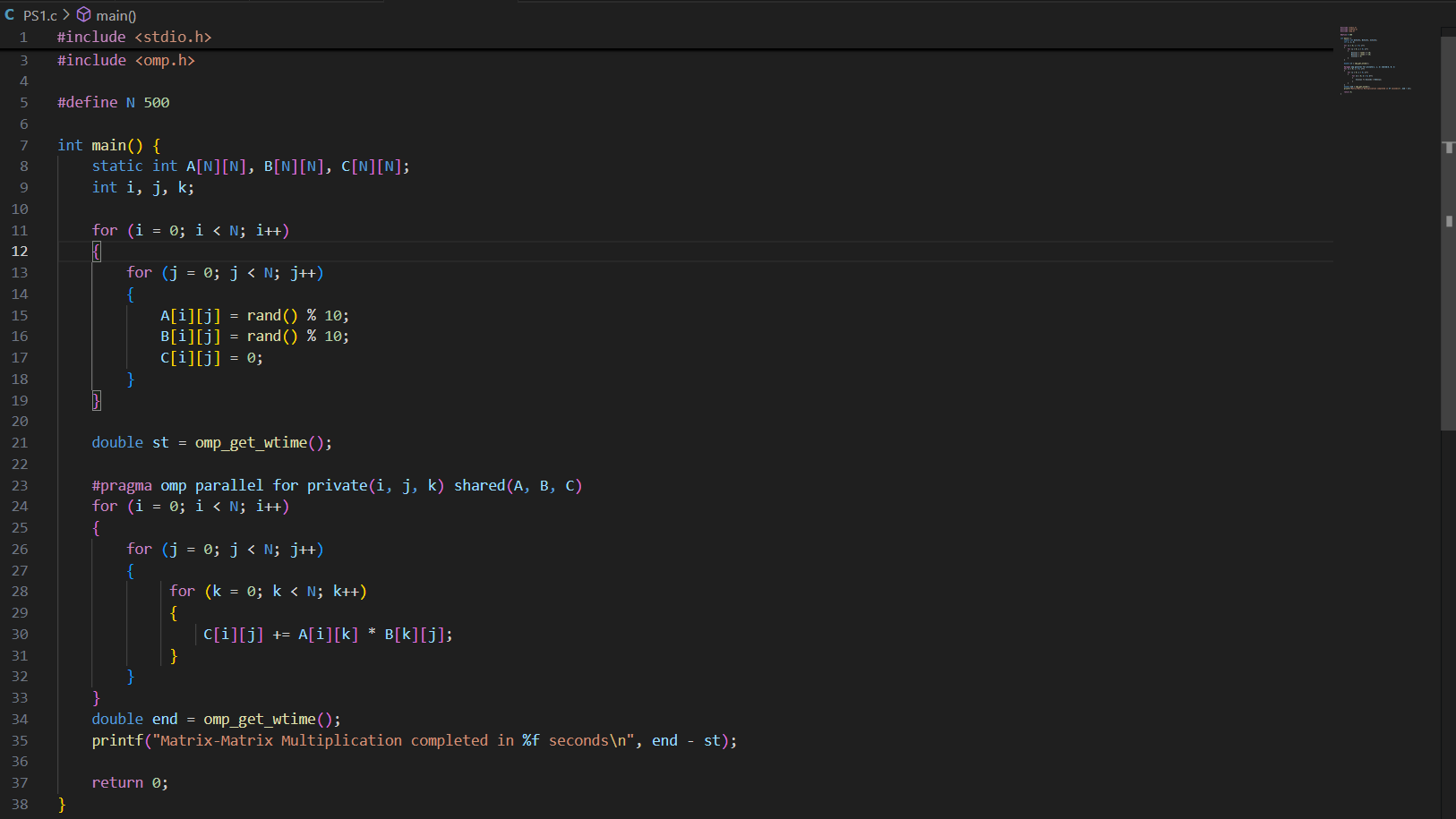
**Name : Vaishnavi Hanmant Katkar.**

**Batch : B3**

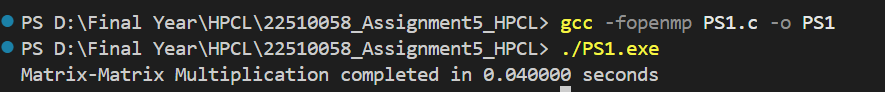
**Title of practical: Implementation of OpenMP programs.**

Implement following Programs using OpenMP with C:

**Implementation of Matrix-Matrix Multiplication.**

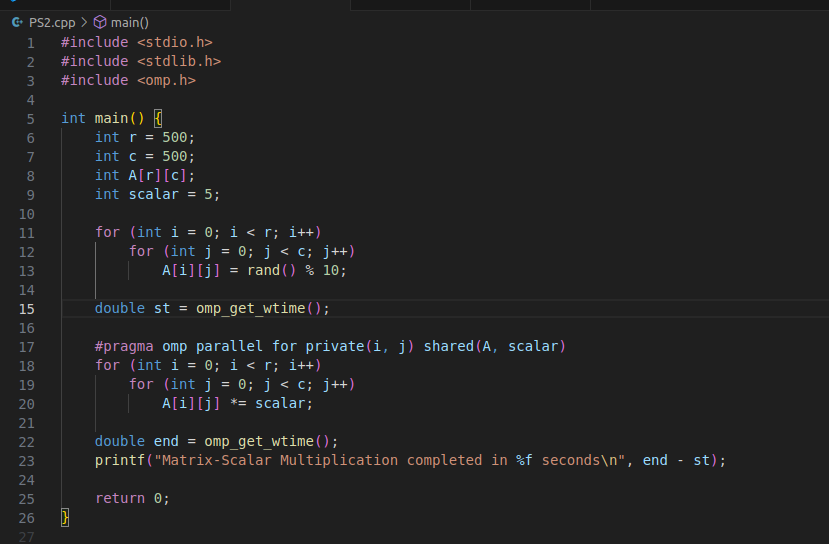
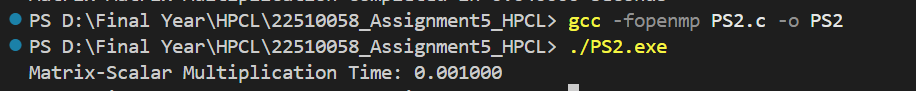


OUTPUT:



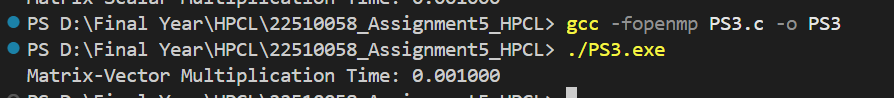
Implementation of Matrix-scalar Multiplication.

OUTPUT:



**Implementation of Matrix-Vector Multiplication.**

## OUTPUT:



Implementation of Prefix sum

## 

OUTPUT:

## 

## **Information:**

It makes parallel programming simple by using easy directives like #pragma omp parallel for.

It helps programs run faster on multi-core processors.

Only small changes are needed to convert normal C programs into parallel ones.

Parallel loops – divide work among multiple threads.

Private and shared variables – decide what data is common and what is separate for each thread.

Synchronization – needed when tasks depend on each other (like in prefix sum).

Time measurement – done using omp\_get\_wtime().

**Analysis of Programs:**

Matrix–Matrix Multiplication

Takes the most time in sequential form because of O(N³) operations.

Easy to parallelize since each element of the result can be calculated independently.

Gives the best speedup with OpenMP.

Matrix–Scalar Multiplication

Much simpler, O(N²) operations.

Each element is independent, so parallelization works very well.

Almost linear speedup.

Matrix–Vector Multiplication

Also O(N²).

Each element of the result vector can be done independently.

Speedup is good, but not as high as matrix–matrix multiplication.

Prefix Sum

Each element depends on the previous one, so it’s hard to parallelize directly.

Needs special algorithms like Blelloch scan for full parallelism.

In OpenMP, we usually get little speedup, sometimes even slower than sequential.

**Expected Performance Order**

Matrix–Matrix Multiplication → highest speedup.

Matrix–Scalar Multiplication → very good speedup.

Matrix–Vector Multiplication → moderate speedup.

Prefix Sum → lowest speedup.

**Github Link:** [**https://github.com/9022348056/HPCL\_22510058**](https://github.com/9022348056/HPCL_22510058)