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

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

**Course:** High Performance Computing Lab

**Practical No. 5**

**Exam Seat No: 22510050**

**Name – Prathmesh M. Sarwade**

**Batch - B6**

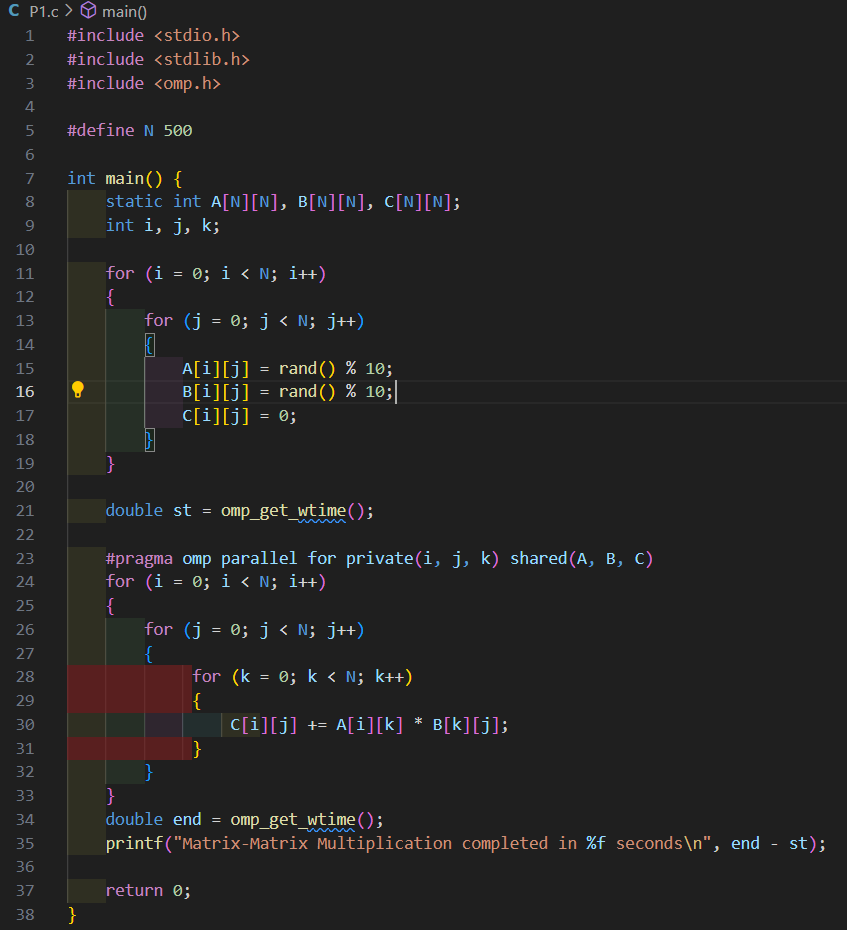
**Title of practical: Implementation of OpenMP programs.**

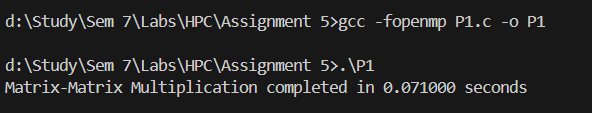
Implement following Programs using OpenMP with C:

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:** Implementation of Matrix-Matrix Multiplication.

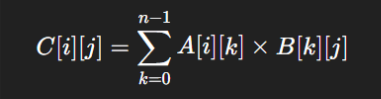
**Screenshots:**





**Information:**

Matrix–Matrix multiplication is a fundamental operation in scientific computing, computer graphics, and data analysis.  
If we have two matrices A of size (m × n) and B of size (n × p), their multiplication results in a matrix C of size (m × p).



OpenMP Tools Used

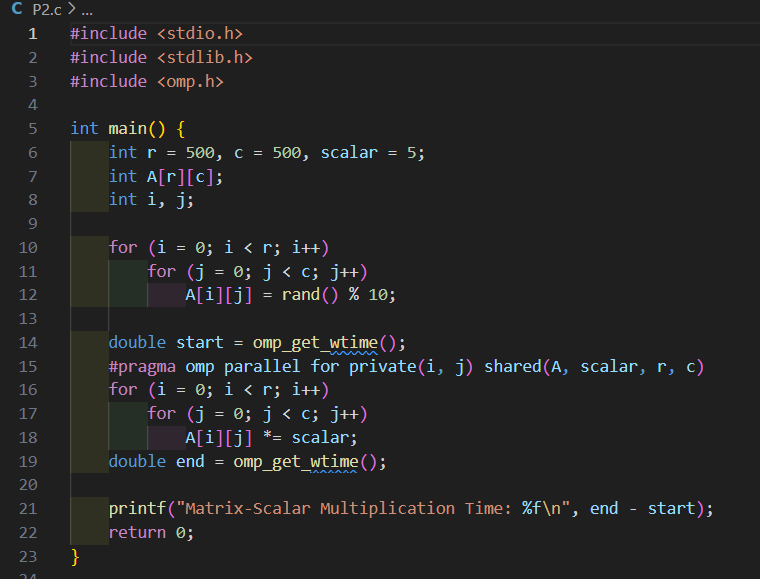
* OpenMP Tools Used #pragma omp parallel for: Splits the outer loop (i) across threads, so each thread processes different rows of matrix A.
* private(i, j): Ensures loop variables are private to each thread.
* shared(A, scalar, r, c): Shares matrix and scalar among all threads

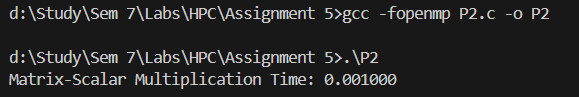
Analysis:

* Parallelization helps: Matrix multiplication elements are independent, so OpenMP parallelization distributes work efficiently across threads.
* Performance observation: For large matrices (e.g., N=500), parallel execution is significantly faster than serial due to reduced computation time.

**Problem Statement 2:** Implementation of Matrix-scalar Multiplication

**Screenshots:**

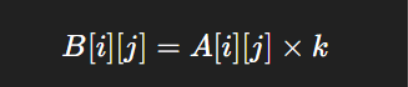




**Information:**

Matrix–Scalar multiplication is the process of multiplying each element of a matrix by a single scalar value.  
If matrix A is of size (m × n) and scalar value is k, then the result matrix B is also (m × n).

Formula:



OpenMP Tools Used

* #pragma omp parallel for: Splits the outer loop (i) across threads, so each thread processes different rows of matrix A.
* private(i, j): Ensures loop variables are private to each thread.

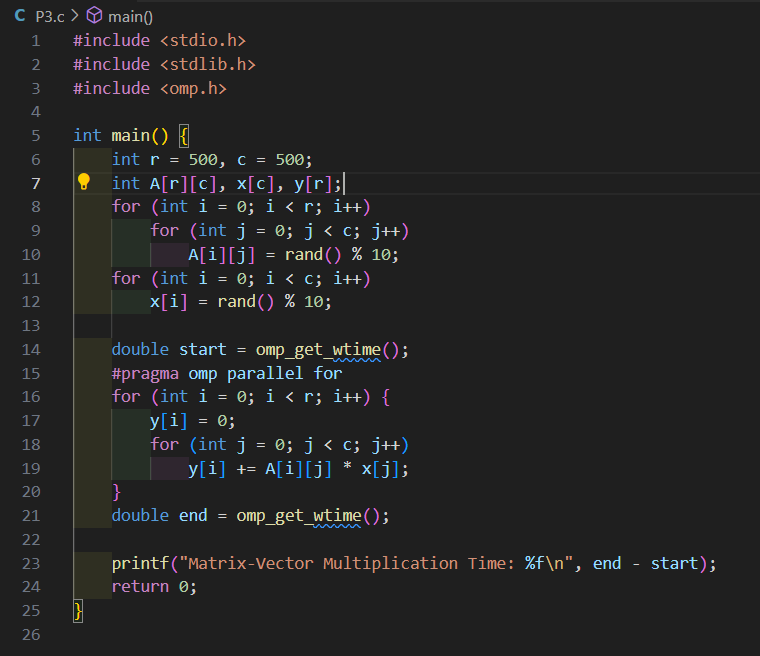
shared(A, scalar, r, c): Shares matrix and scalar among all threads

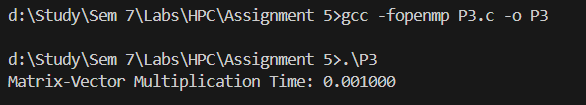
**Analysis:**

* Scalability: Parallelization scales well as matrix size increases; more threads can process more elements simultaneously.
* Parallelization helps: Each element multiplication is independent, so OpenMP efficiently distributes work across threads.

**Problem Statement 3:** Implementation of Matrix-Vector Multiplication

**Screenshots:**

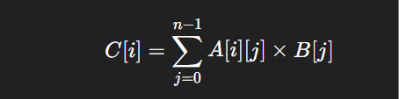




**Information:**

Matrix–Vector multiplication multiplies an (m × n) matrix with a vector of size (n × 1), producing a result vector of size (m × 1).

Formula:



OpenMP tools used:

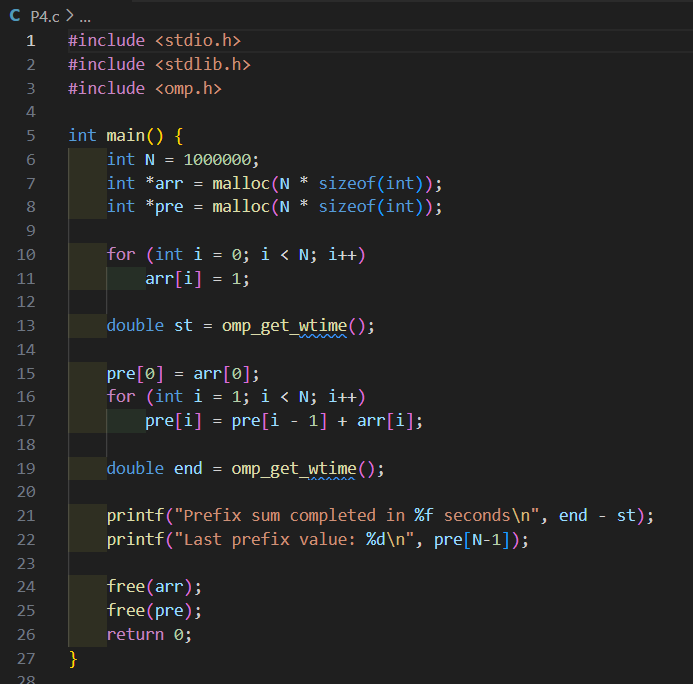
* #pragma omp parallel for: Splits the outer loop (i) across threads, so each thread computes a different row of the result vector y.

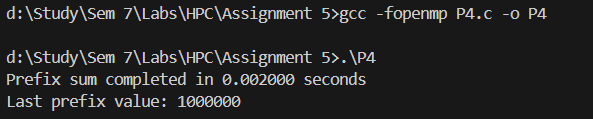
**Analysis:**

* Parallelization helps: Each row computation for y[i] is independent, allowing efficient parallelization.
* Performance: For large matrices, parallel execution is much faster than serial due to reduced computation time.
* Synchronization: Not needed, as each thread writes to a unique y[i]

**Problem Statement 4:** Implementation of Prefix sum

**Screenshots:**





**Information:**

The Prefix Sum (also called scan) of an array generates a new array where each element is the sum of all previous elements.

Formula:  
For an array A = [a₀, a₁, a₂, …, an−1],

Prefix[i]=a0+a1+a2+…+aiPrefix[i] = a\_0 + a\_1 + a\_2 + … + a\_iPrefix[i]=a0 +a1 +a2 +…+ai

Example:  
A = [1, 2, 3, 4] → Prefix = [1, 3, 6, 10]

OpenMP tools used :

* #pragma omp parallel for: Distributes the outer loop (i) across threads, so each thread computes a different row of the output vector y.

**Analysis:**

* Parallelization: Each row calculation is independent, enabling efficient parallelization.
* Performance: Parallel execution is significantly faster for large matrices due to reduced computation time.

**Github Link:**

**https://github.com/prathmesh967/HPC-Assignment/tree/main/Assignment%205**