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

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

**Course:** High Performance Computing Lab

**Practical No. 3**

**Exam Seat No:22510090**

**Title of practical:**

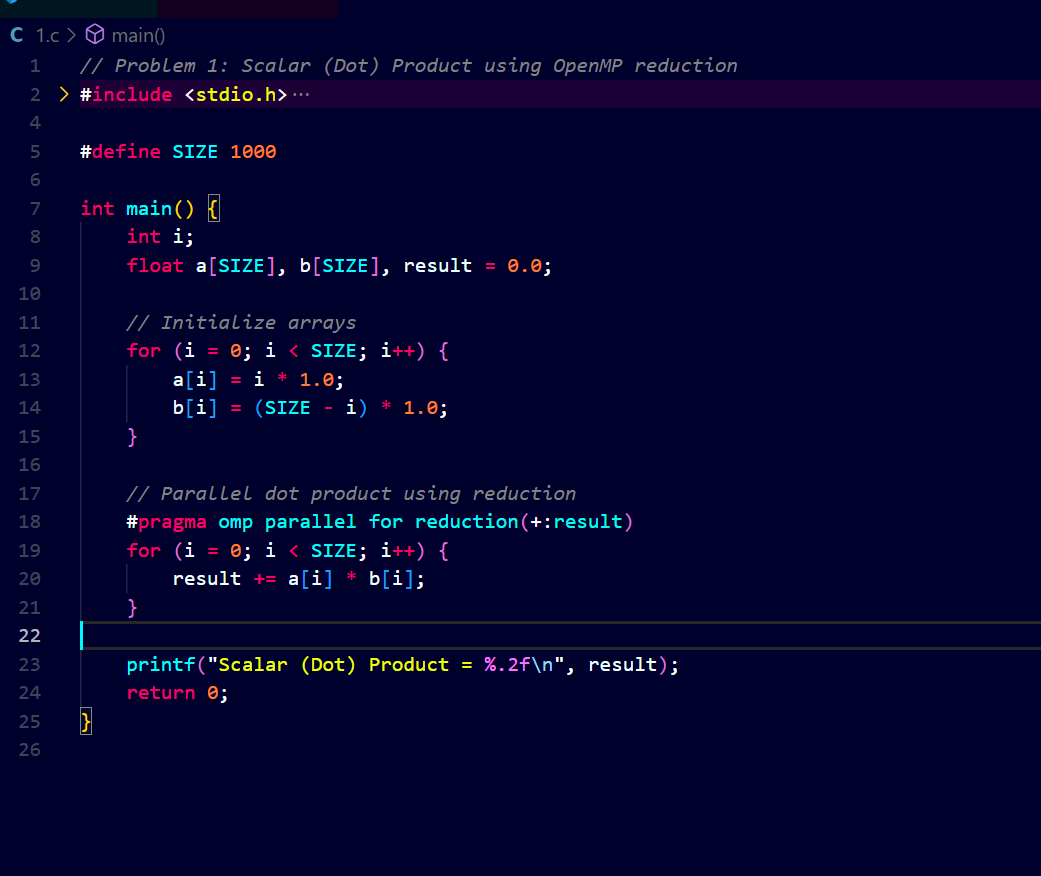
Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

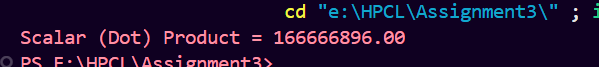
**Problem Statement 1:**

Analyse and implement a Parallel code for below program using OpenMP.

// C Program to find the minimum scalar product of two vectors (dot product)

**Screenshots:**





**Information and analysis:**

1. The program calculates the dot product of two vectors.
2. Each pair of elements from the two vectors is multiplied and summed.
3. OpenMP’s reduction(+:result) clause was used to avoid race conditions.
4. The program gave the correct result after parallel execution.
5. Using OpenMP improved performance, especially for larger vector sizes.

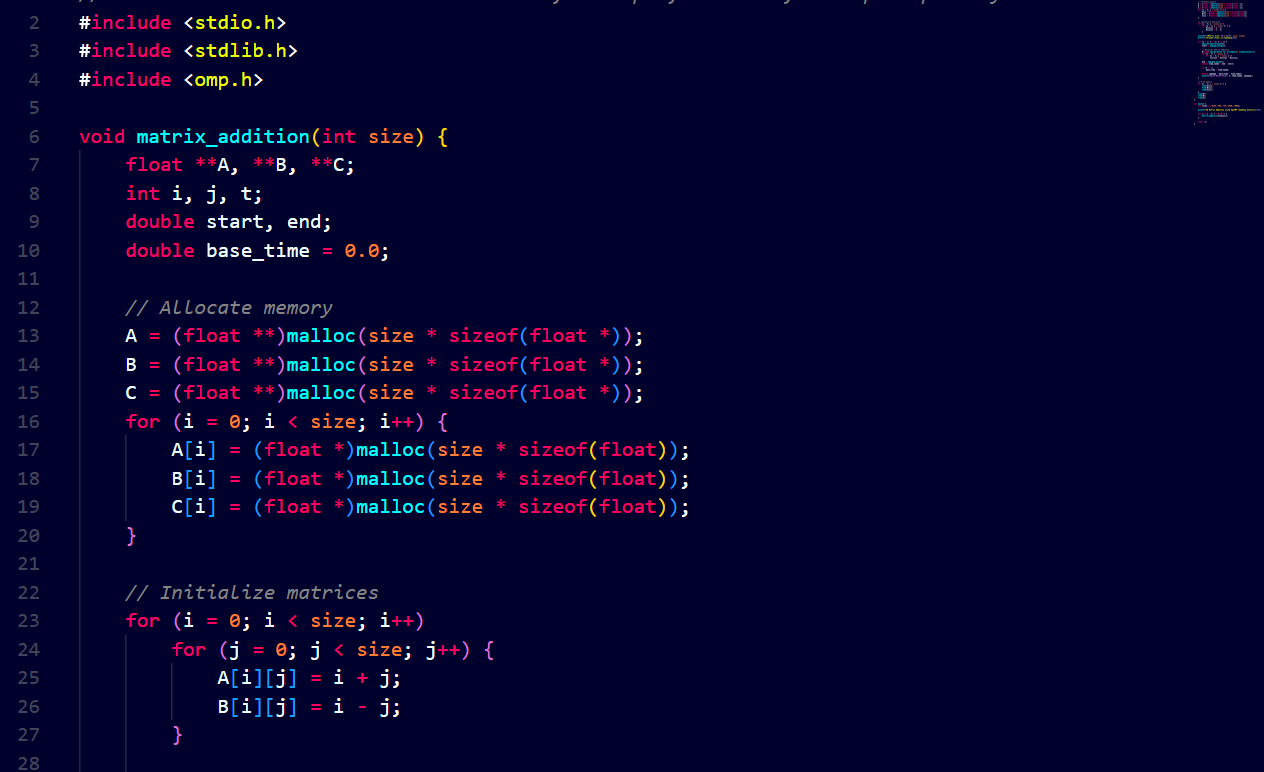
**Problem Statement 2:**

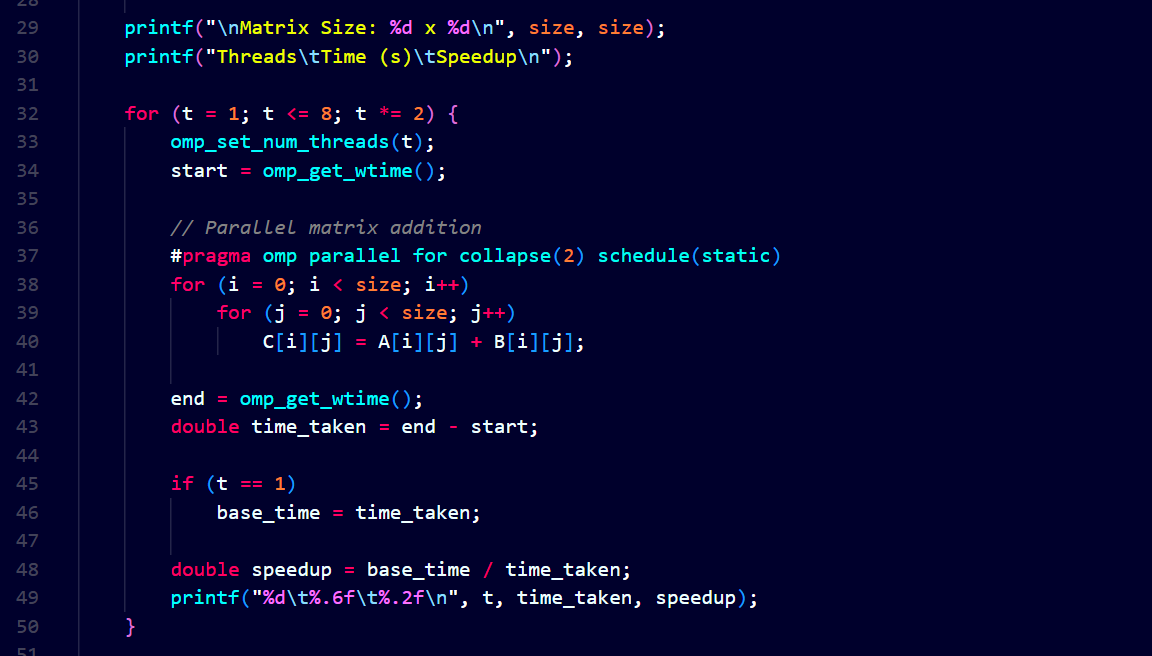
Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculate the execution time or use GPROF)

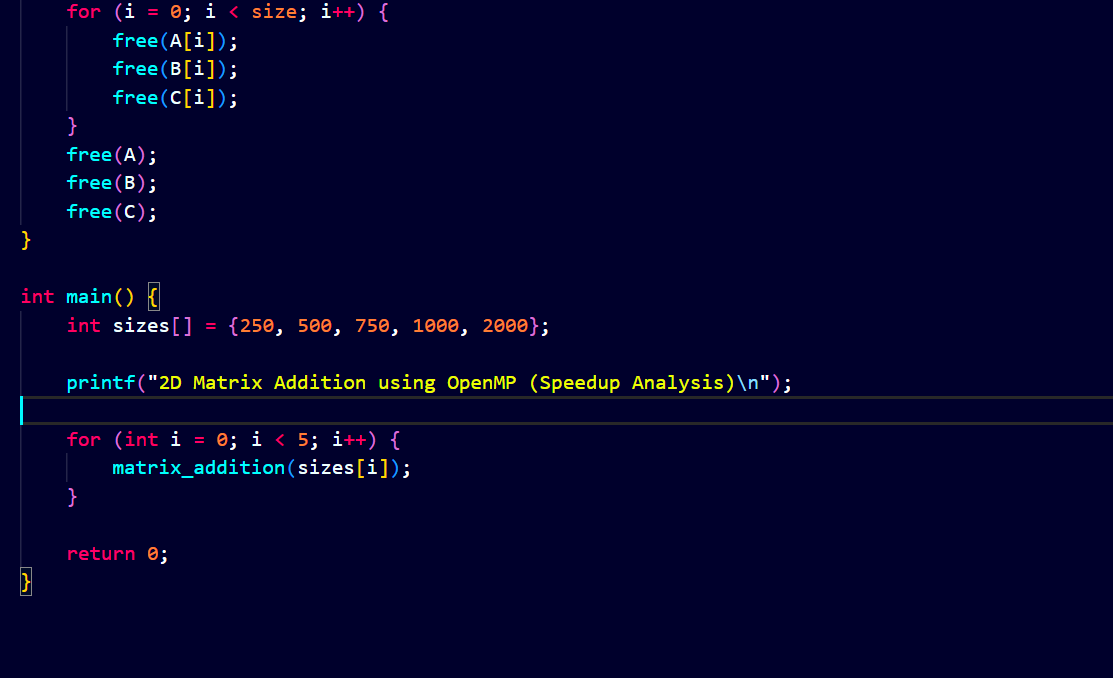
i. For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads.

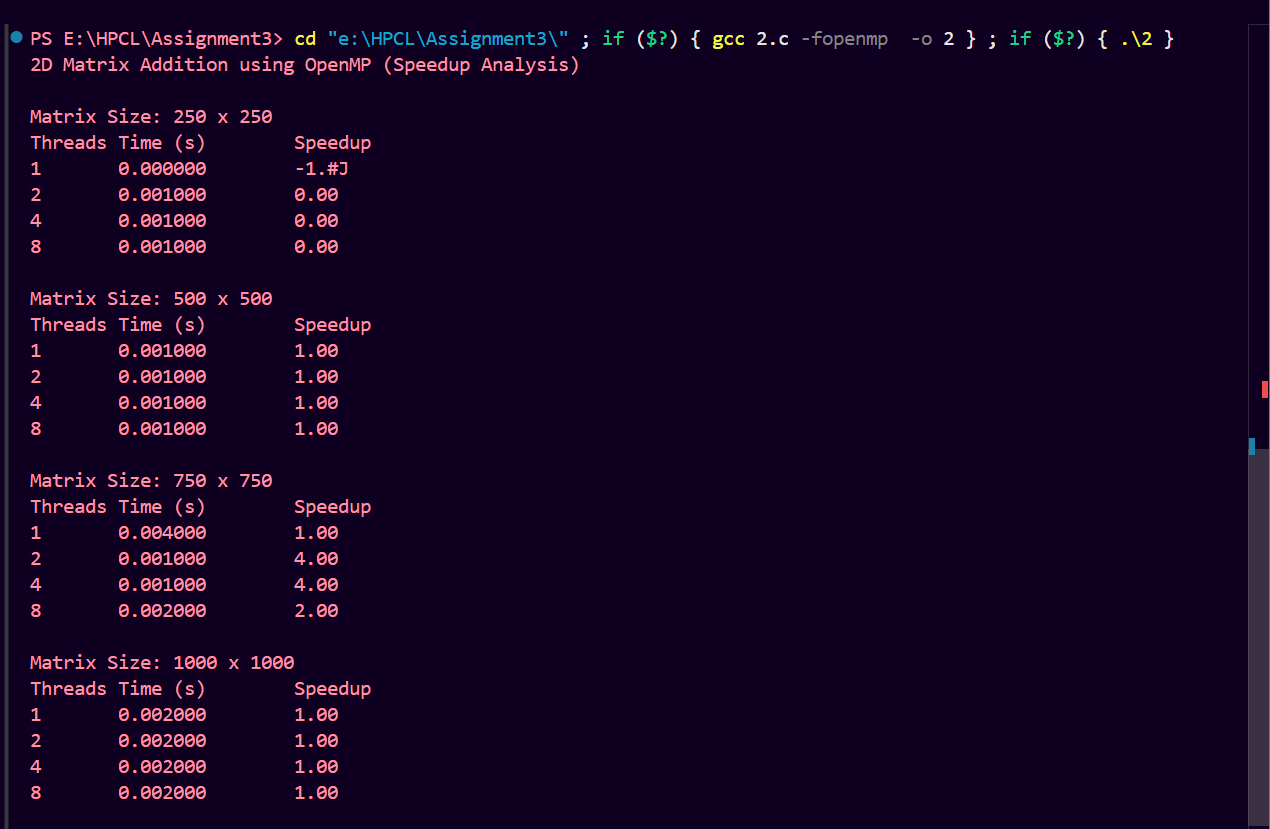
ii. Explain whether or not the scaling behaviour is as expected.

**Screenshots:**











**Information and analysis:**

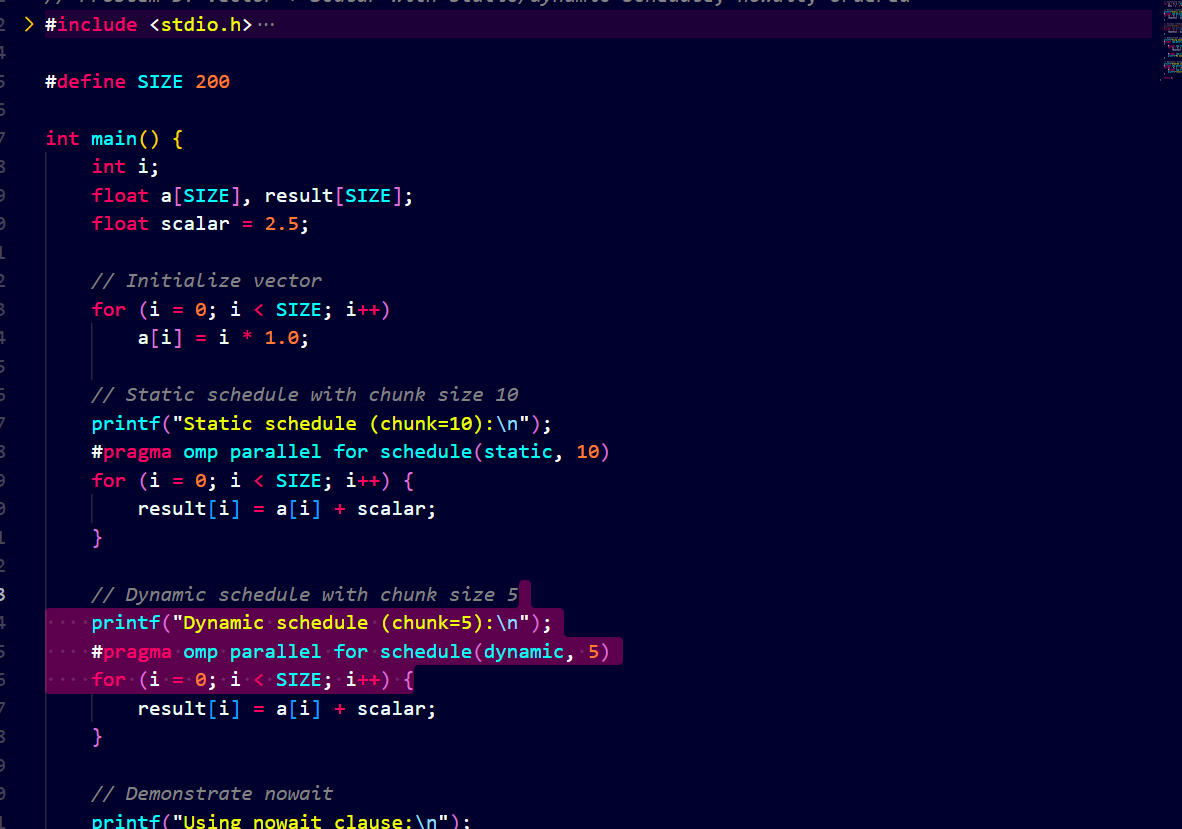
1. The program adds two 2D matrices using OpenMP with collapse(2) and schedule(static) clauses.
2. Matrices are dynamically allocated using malloc to support large sizes (up to 2000×2000).
3. Execution time is measured using omp\_get\_wtime() for 1, 2, 4, and 8 threads.
4. **Speedup formula used:** ​

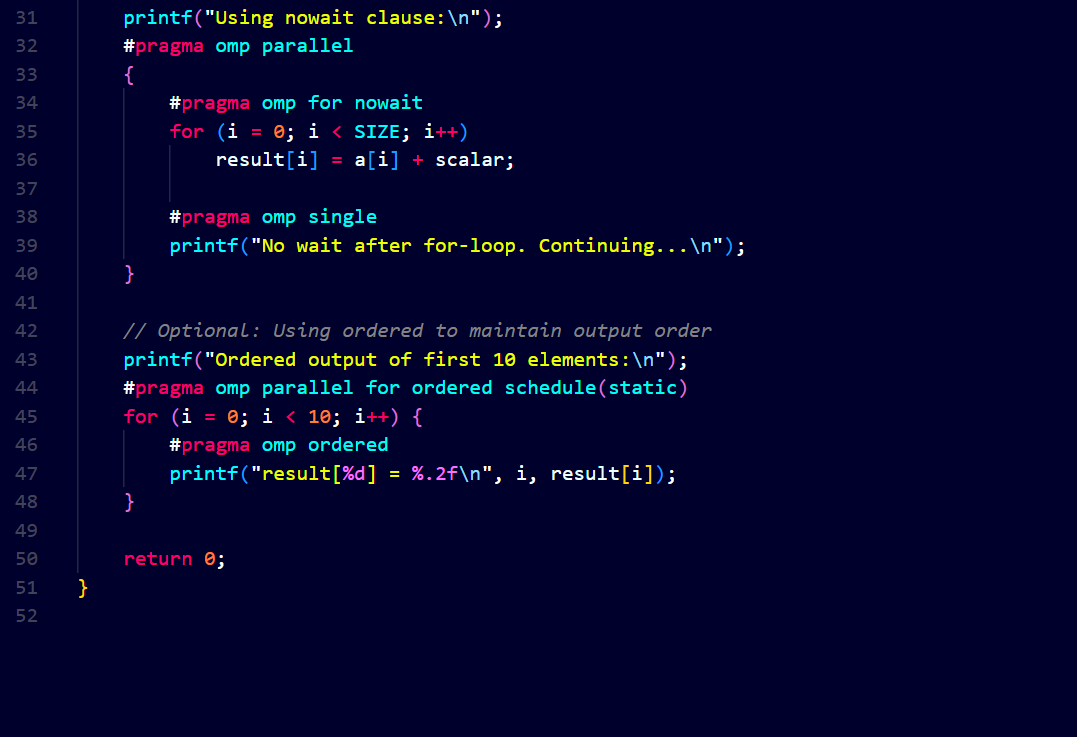


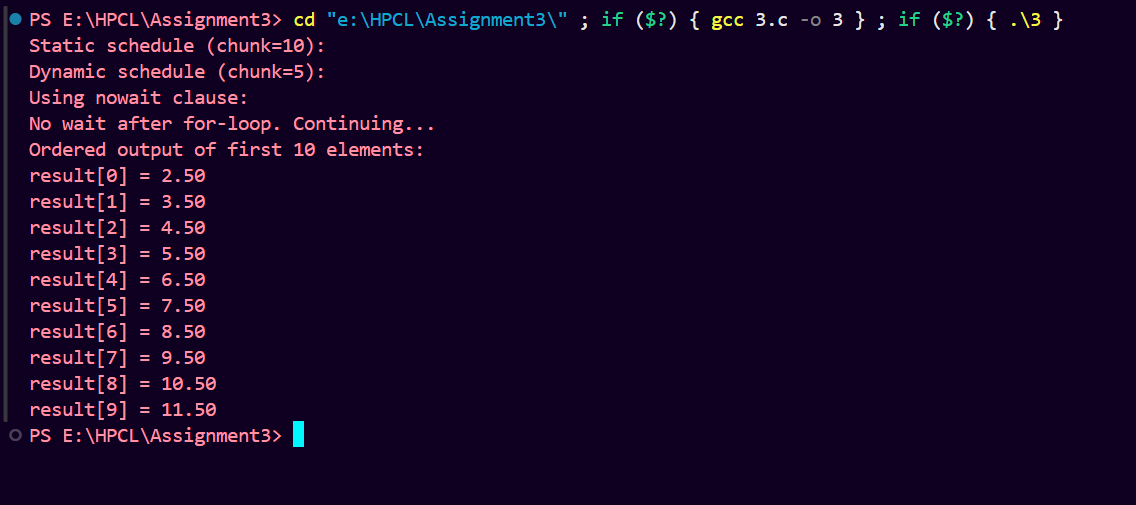
1. Speedup increases with thread count for larger matrix sizes. Scaling is mostly as expected, though not perfectly linear due to overhead.

**Problem Statement 3:**

For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following: i. Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup. ii. Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup. iii. Demonstrate the use of nowait clause.

**Screenshots:**





**Information and analysis:**

1. The output showed scalar addition using both static and dynamic schedules.
2. The message "No wait after for-loop. Continuing..." proved that nowait was used correctly.
3. The ordered output of the first 10 values (e.g., result[0] = 2.50 … result[9] = 11.50) confirmed that ordered was working.
4. Each version (static, dynamic, nowait, ordered) ran successfully and gave correct results.
5. Different scheduling methods didn’t affect the output values but are used to study speed and behavior.

**Github Link:**

<https://github.com/purvamarkam/HPCL/tree/main/Assignment3>