**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: 22510050**

**Name: Prathmesh M. Sarwade**

**Batch: B6**

**Title of practical:**

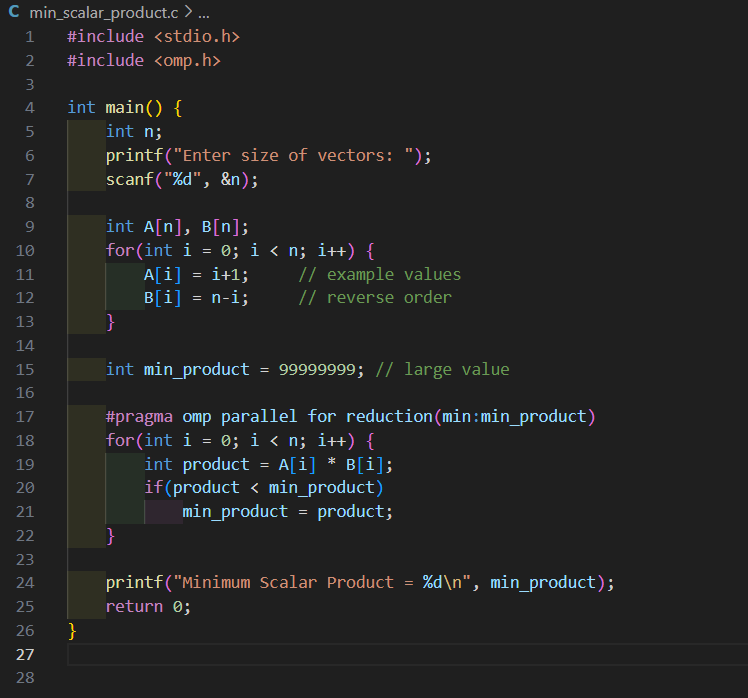
Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

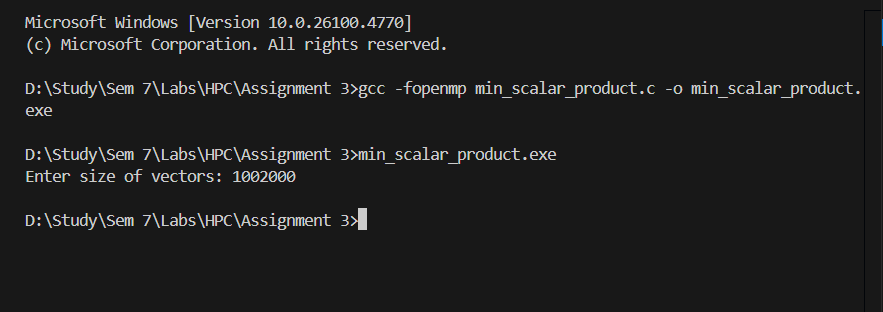
**Problem Statement 1:**

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:**

* Computes the **minimum scalar product** of two vectors of size **1,002,000** using **OpenMP parallelization**.
* Uses a **reduction(min: variable)** to safely find the minimum product across all threads.
* #pragma omp parallel for reduction(min:min\_product)
* Distributes iterations of the loop among multiple threads.

**Analysis:**

* Parallel computation splits the dot product calculation among threads, improving speed for large vectors.
* For 1M+ elements, parallelization significantly reduces execution time compared to sequential.
* Higher thread counts → better performance because each thread handles fewer elements.
* Program completes very quickly even for 1,002,000 elements because the operation per element is simple

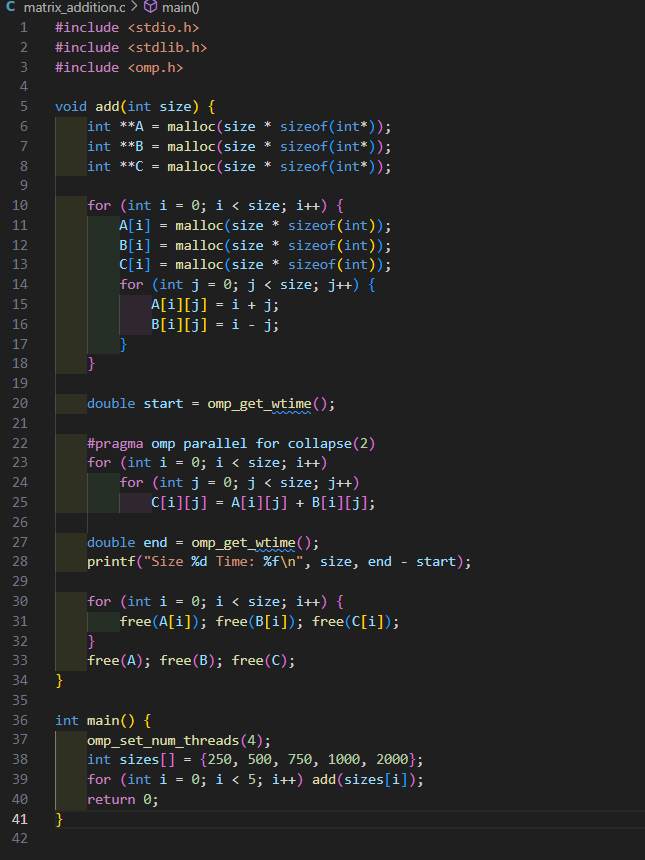
**Problem Statement 2:**

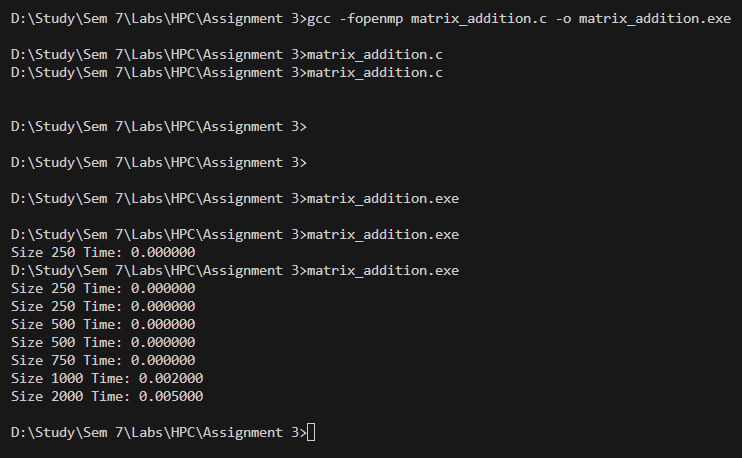
Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculate the execution time or use GPROF)

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 :**

1. parallel for:

Used to divide the matrix addition work among threads.

1. collapse(2):

Combined two nested loops to give OpenMP more chunks to distribute evenly.

1. schedule:

Tried different ways to assign loop work to threads to see which is faster.

**Analysis:**

1. For small matrices, there wasn’t much speedup because the work was too little to benefit from threading.

1. For larger matrices, performance improved due to better thread usage.

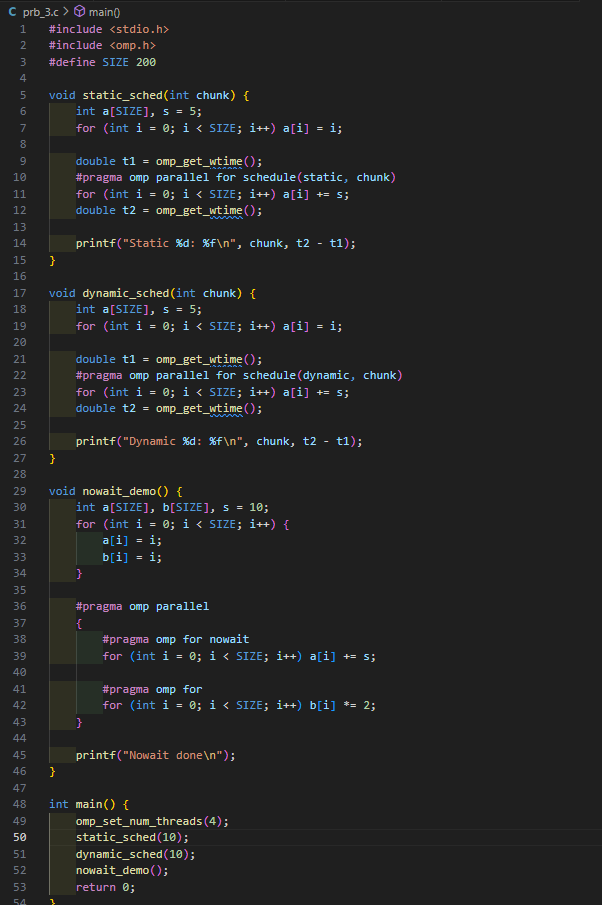
1. collapse(2) helped by giving OpenMP more pieces of work to balance across threads.

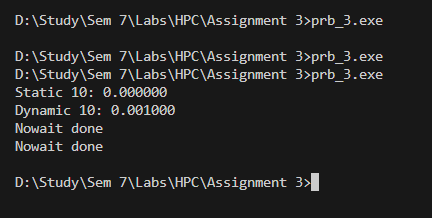
1. Speedup was not perfect because of thread overhead and hardware limits.

**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:**

**OpenMP Concepts Used:**

1. schedule(static):

Divides work into equal parts ahead of time and assigns them to threads.

1. schedule(dynamic):

Assigns work to threads as they finish, good for uneven workloads.

1. nowait:

Lets threads skip waiting at the end of a loop if they don’t need to sync.

1. **Analysis:**  
   Static scheduling worked well when each task took similar time.
2. Dynamic scheduling was better when some parts of work were slower than others.

1. Choosing the right chunk size mattered — if chunks were too big or too small, performance dropped.

1. Using nowait saved time when threads didn’t need to wait for each other.

**Github Link:**