**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: 22510012**

**Title of practical:**

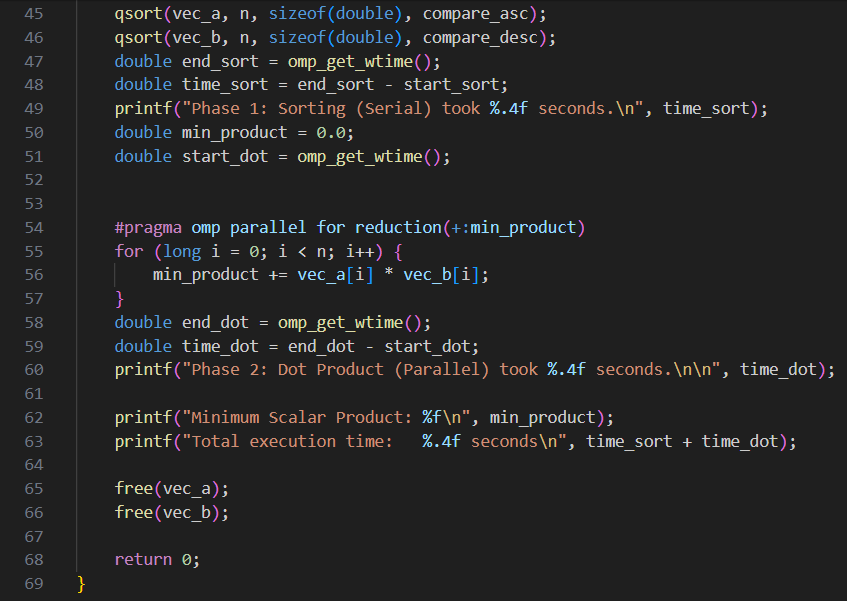
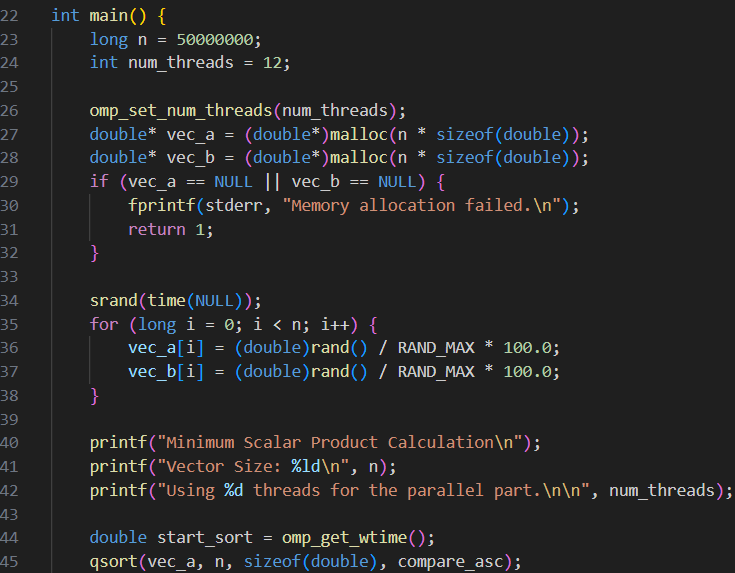
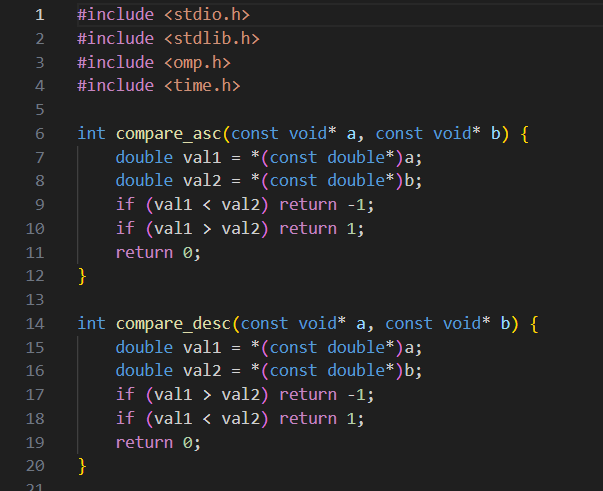
Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

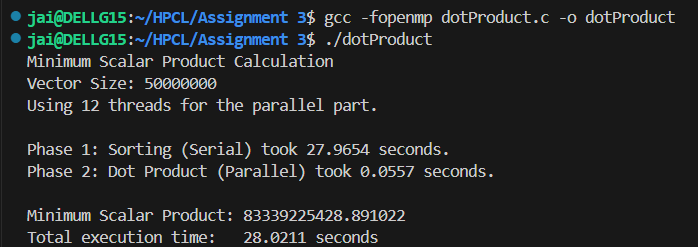
**Problem Statement 1:**

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:**

**The main computation (element-wise multiplication and summation) can be efficiently parallelized using OpenMP, since each multiplication is independent.**

**The sorting step, if required, is usually done serially (efficient parallel sort is possible but not covered here).**

**The reduction clause in OpenMP safely accumulates the sum across threads**

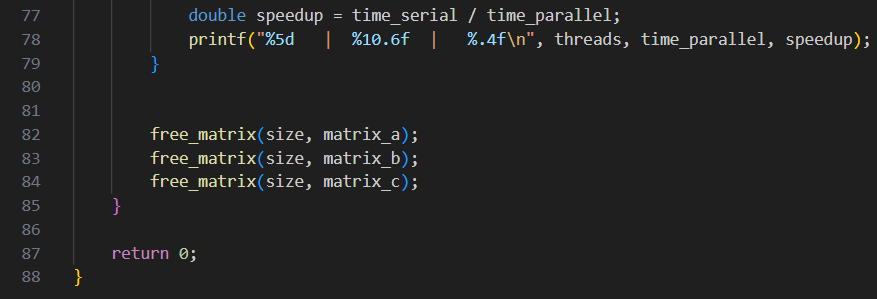
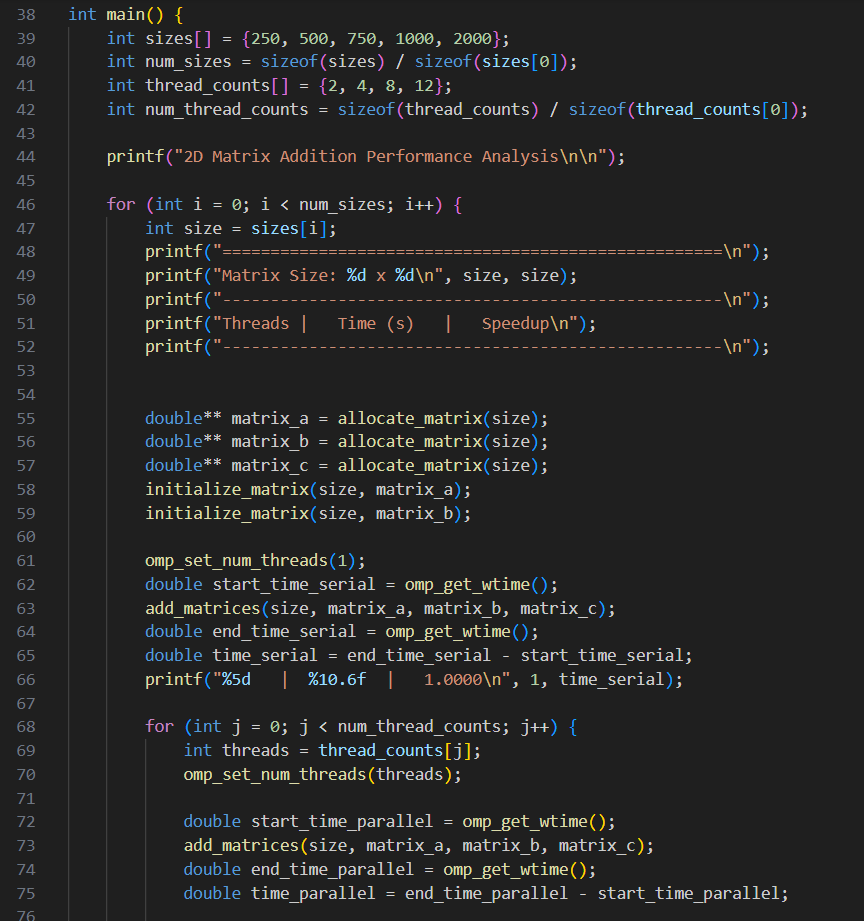
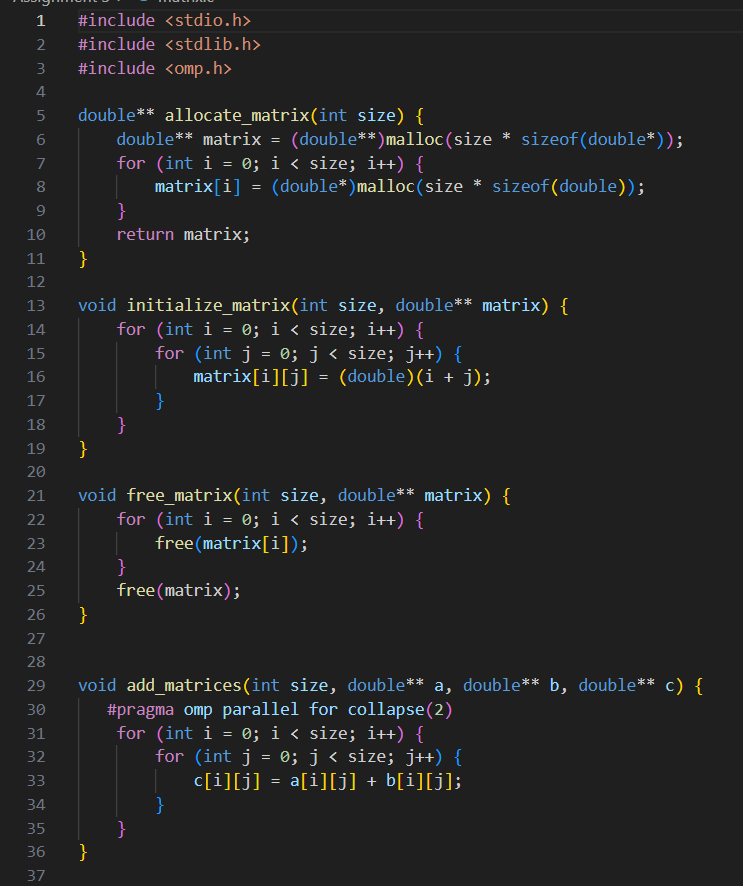
**This result is a clear example of Amdahl's Law, where the serial part limits overall speedup.**

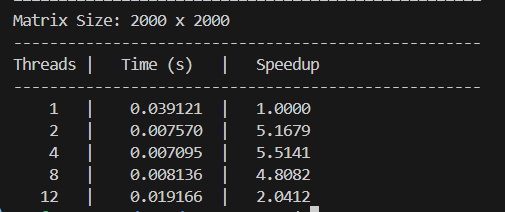
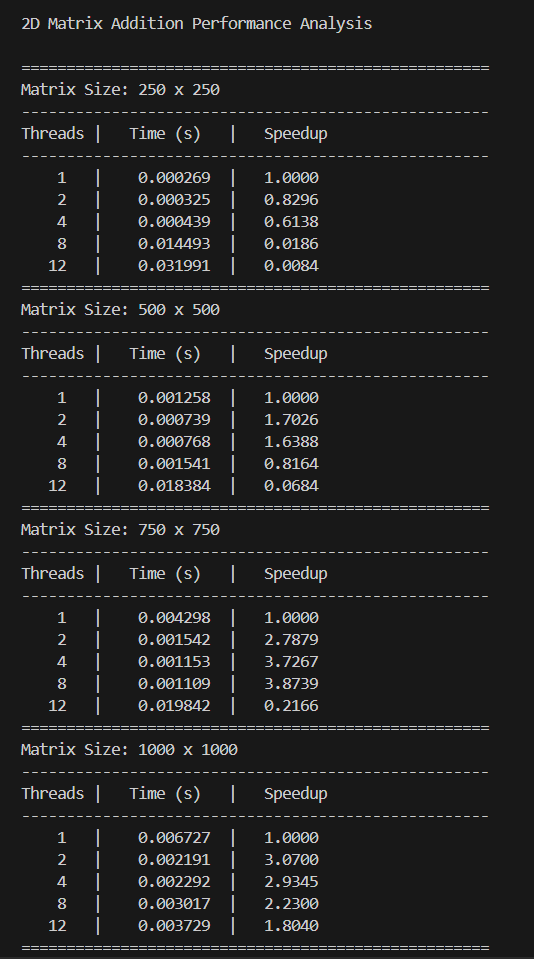
**Problem Statement 2:**

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:**

**For small matrices, timings are unreliable and parallel overhead causes slowdowns (e.g., 0.05x speedup).**

**- The impossible 17.11x speedup is a clear benchmarking artifact from inconsistent CPU states or caching.**

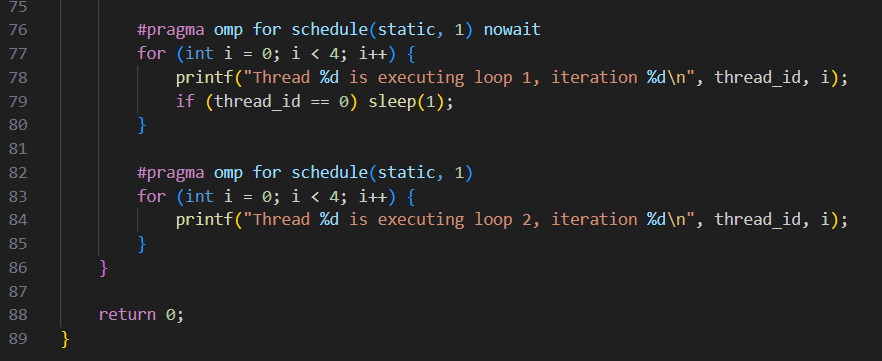
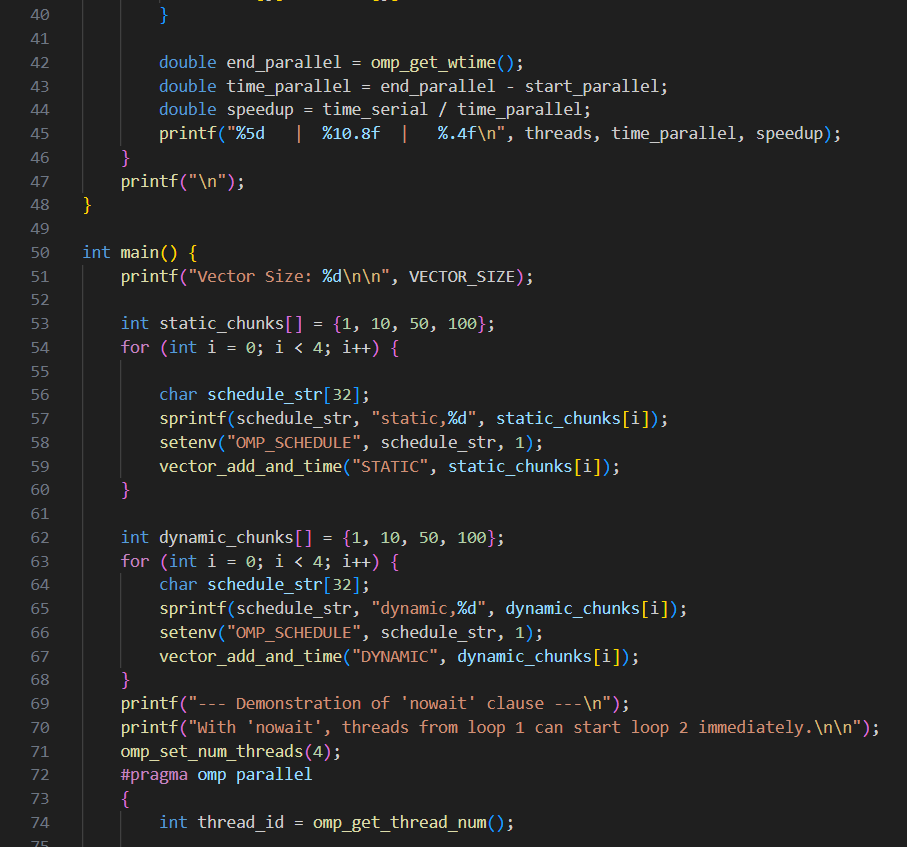
**- Larger matrices show the true trend: performance peaks early at 3.22x speedup with just 4 threads.**

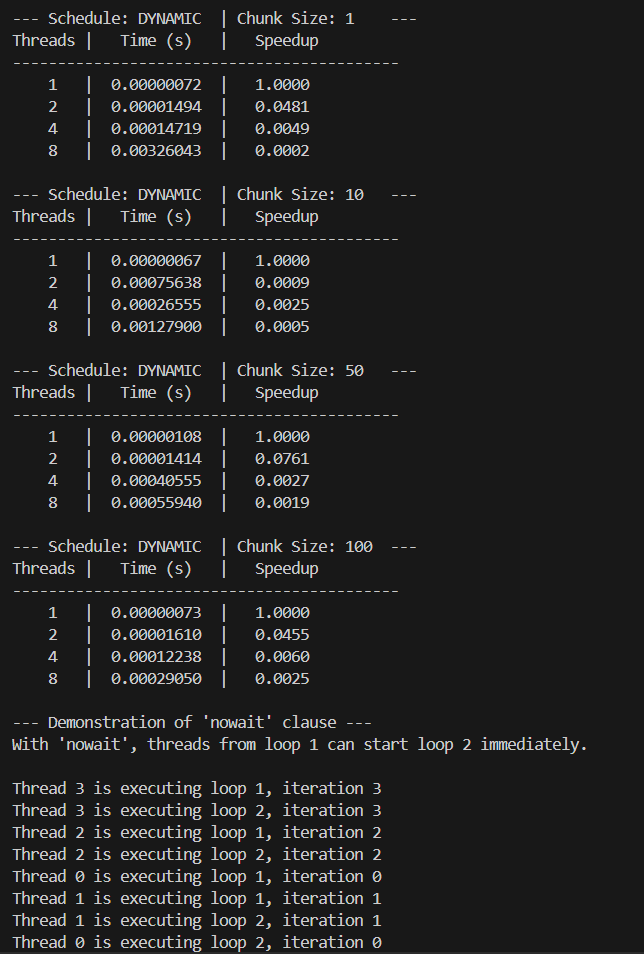
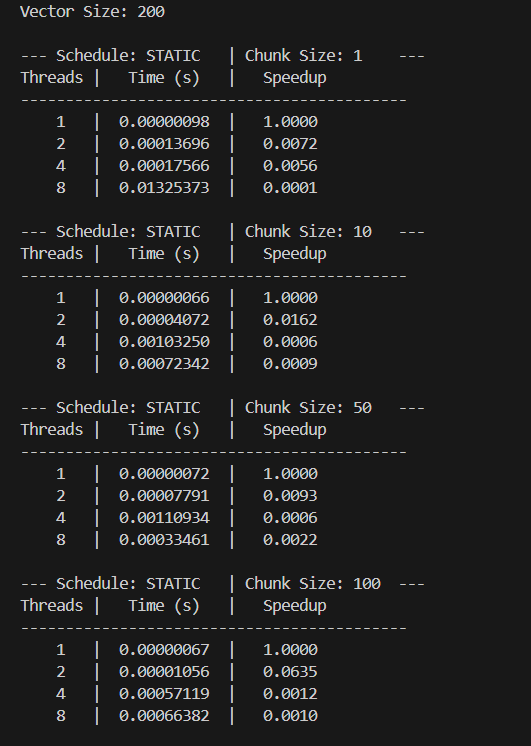
**- This indicates the program is memory-bound, limited by RAM speed, not CPU processing power.**

**- Performance degrades beyond 4 threads because parallel overhead costs more than any gains once the memory bus is saturated.**

**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:**

**Significant Slowdown The primary result was not a speedup but a major slowdown. The parallel code, taking around 200 microseconds, was consistently over 100 times slower than the serial version, which took only ~1.8 microseconds.**

**- Cause: Dominant Parallel Overhead This slowdown is due to parallel overhead. This is the time your system spends creating, managing, and synchronizing the threads. For a tiny job like this (200 additions), this management cost far outweighs the actual computational work.**

**- Scheduling Policies Explained The two strategies tested are different: schedule(static) pre-assigns work in fixed chunks (low overhead), while schedule(dynamic) lets threads pull work as they become free (higher overhead but more flexible).**

**Github Link: https://github.com/Jai-173/HPCL**