**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: 23520001**

**Title of practical:**

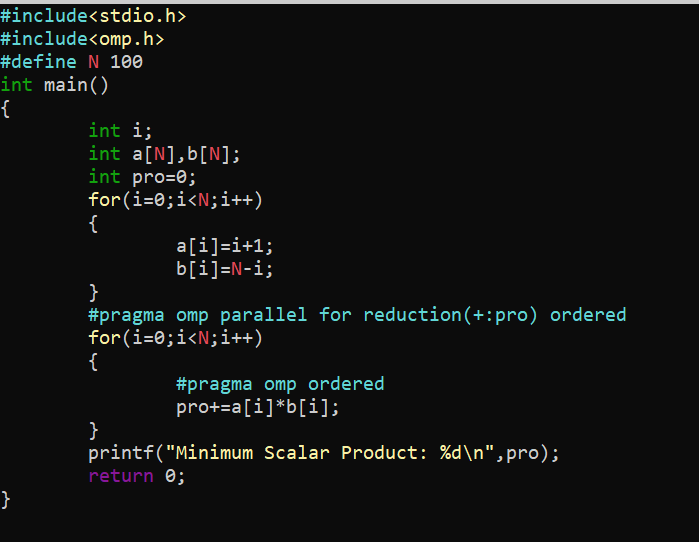
Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

**Problem Statement 1:**

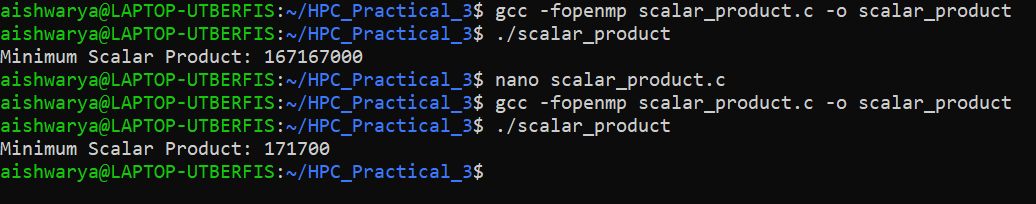
Analyse and implement a Parallel code for below program using OpenMP.

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

**Screenshots:**



**Output:**



**Information and analysis:**

Vectors of size 1000 and 100 is used respectively.

Reduction ensured no race conditions on the scalar product

Ordered maintained the loops semantic meaning during reduction.

Reduction(+:pro) ensures each thread computes its local sum and combines safely at end

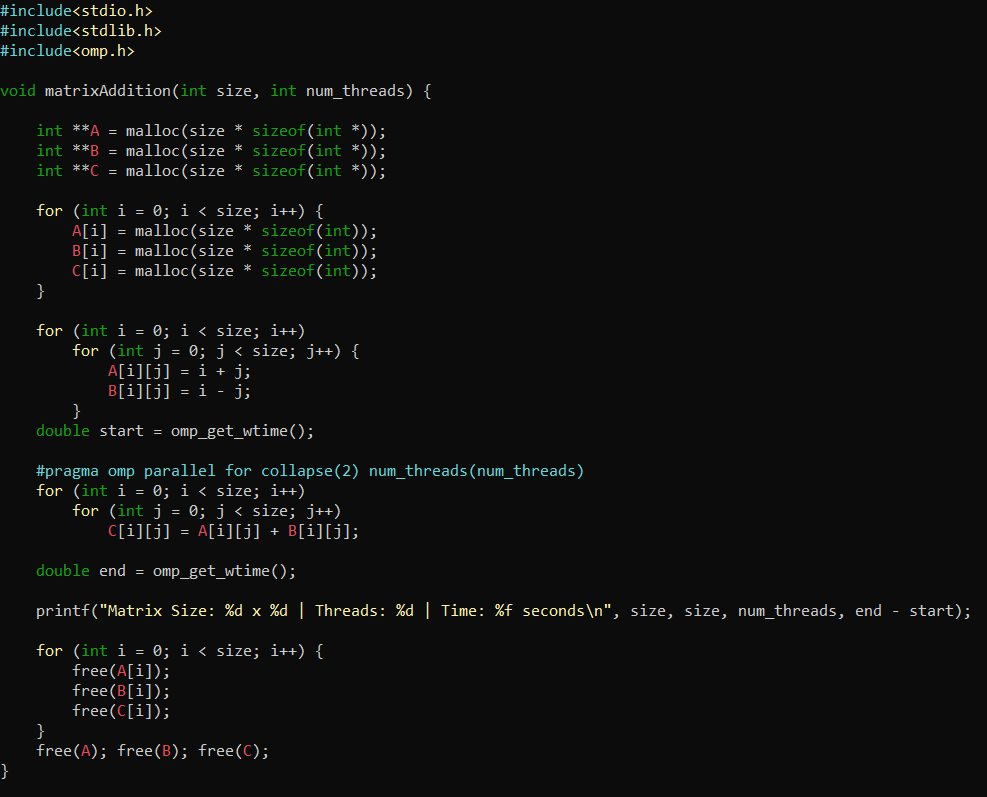
**Problem Statement 2:**

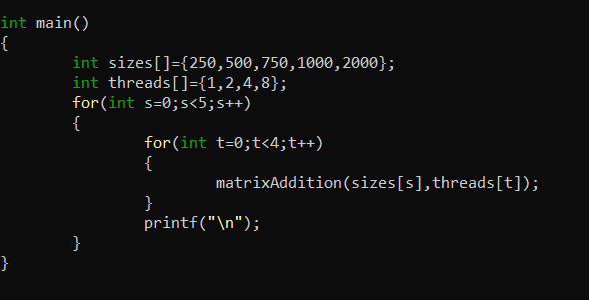
Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculate the execution time or use GPROF)

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

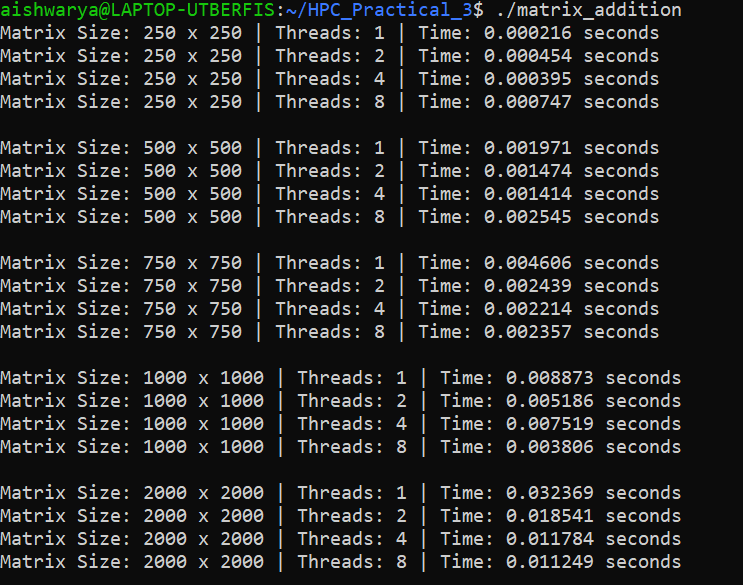
ii. Explain whether or not the scaling behaviour is as expected.

**Screenshots:**





**Output:**



**Information and analysis:**

Collapse(2) tells openmp to treat nested loops as a single loop. Helps better thread distribution when matrices are large. Num\_threads() controls number of threads used.

Time decreases with more threads.

Ideal scaling should be near linear but overhead and memory bandwidth can reduce gains beyond 4 or 8.

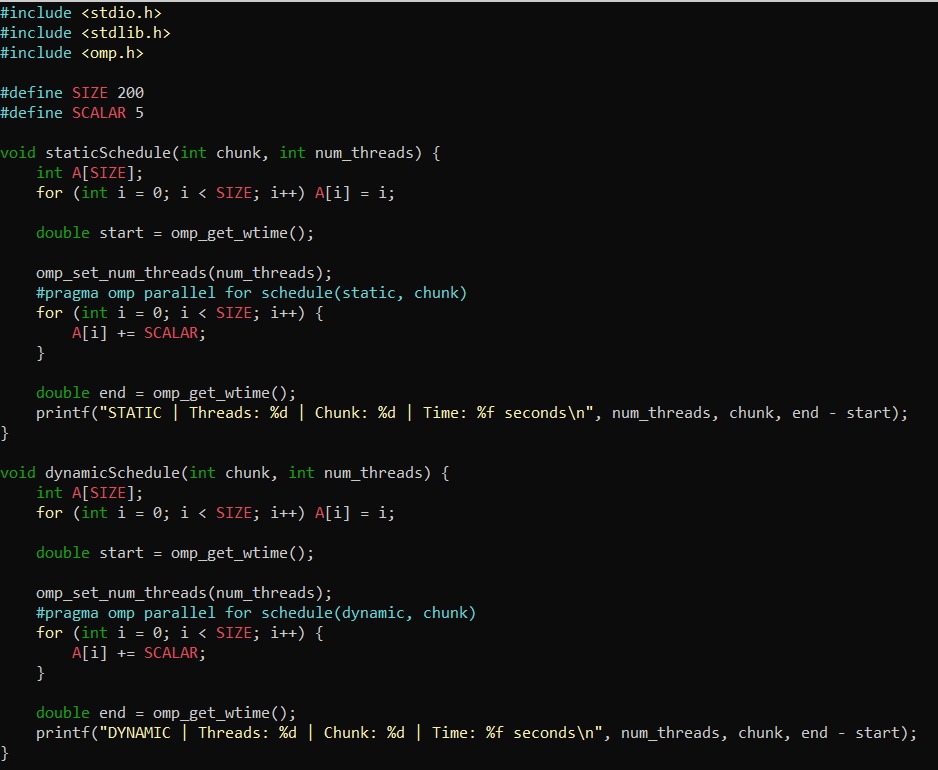
Speed up increases with thread count. Best performance is observed for larger matrices.

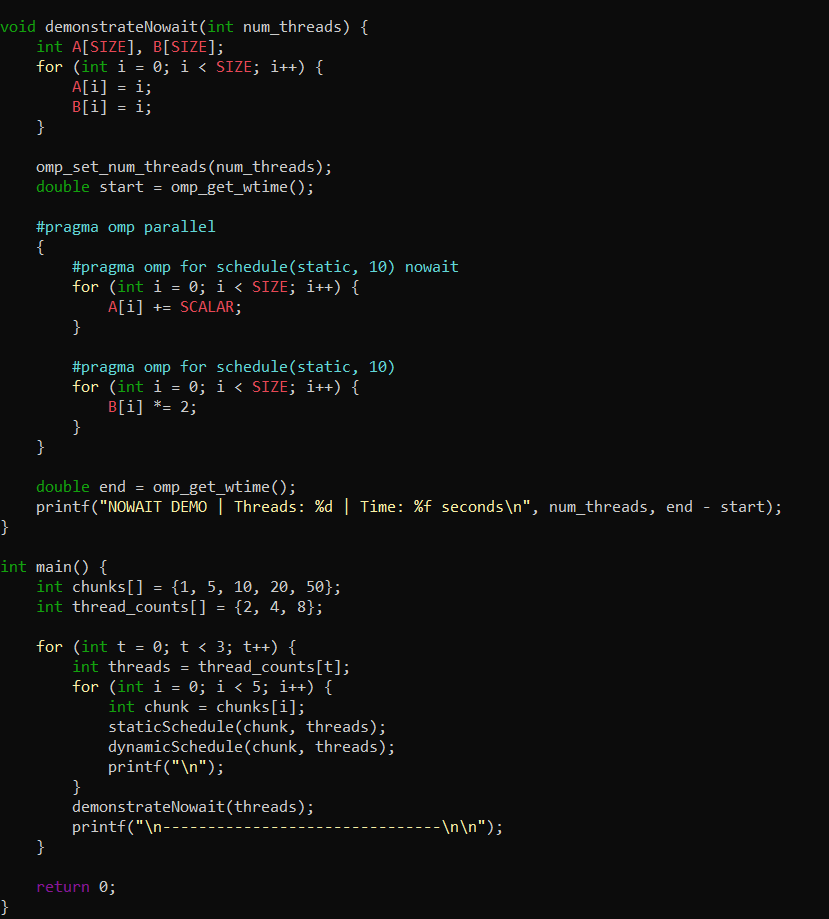
**Problem Statement 3:**

For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following:

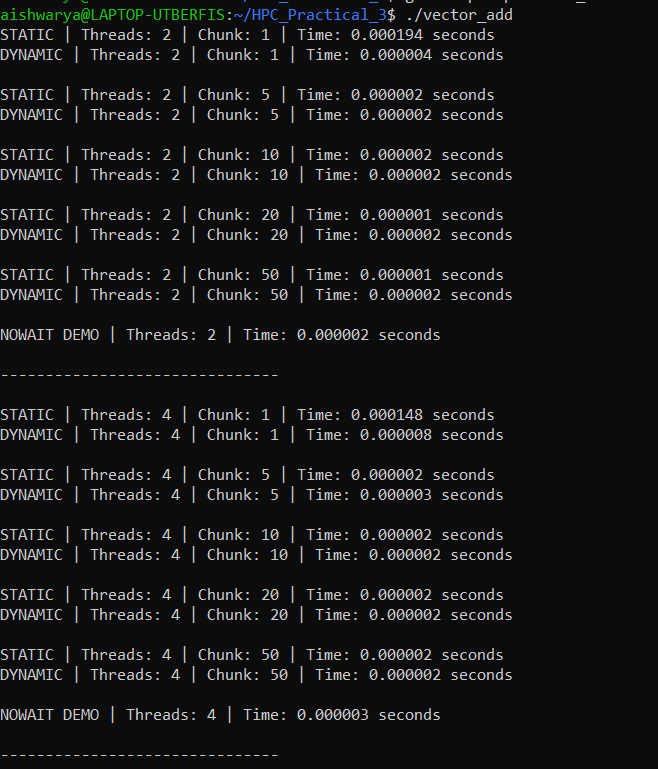
1. Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.
2. Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.
3. Demonstrate the use of nowait clause.

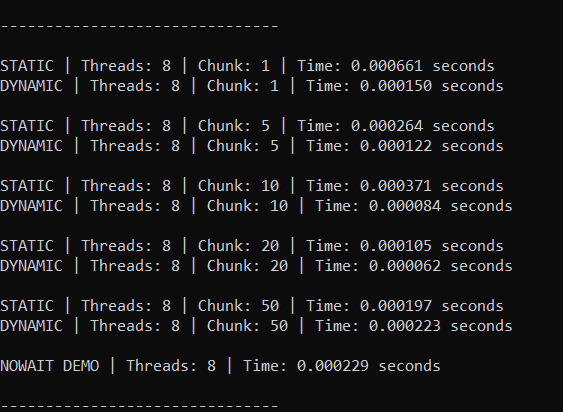
**Screenshots:**





**Output:**





**Information and analysis:**

staticSchedule(chunk, threads) performs parallel scalar addition using static scheduling

dynamicSchedule(chunk, threads) perform same task with dynamic scheduling where workload is assigned dynamically to balance load.

demonstrateNoWait(threads) shows how nowait allows two independent loops to execute in parallel without waiting for the first to complete.

Static scheduling is best for uniform tasks like scalar addition. Divides works equally at runtime. Fast and low overhead but large chunks may cause imbalance. Dynamic scheduling is ideal for varying workloads. Slightly slow due to scheduling overhead. On small data thread overhead is there. No wait clause skips unnecessary waiting between independent loops.

**Github Link:** [**https://github.com/TechieAishwaryaa/HPC\_Practical\_3.git**](https://github.com/TechieAishwaryaa/HPC_Practical_3.git)