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

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

**Course:** High Performance Computing Lab

**Practical No. 3**

**Exam Seat No:**

**Title of practical:**

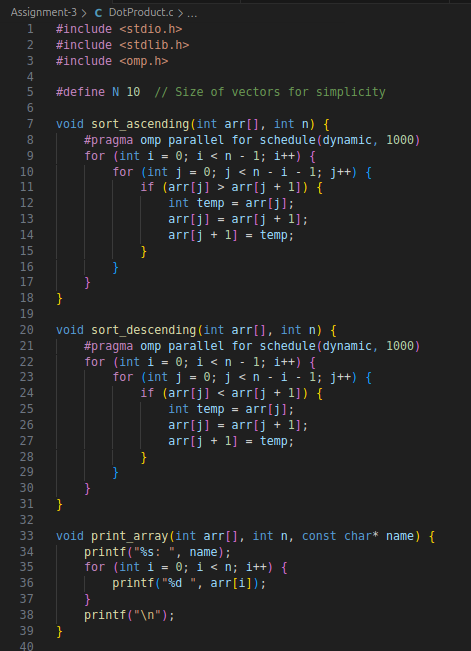
Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

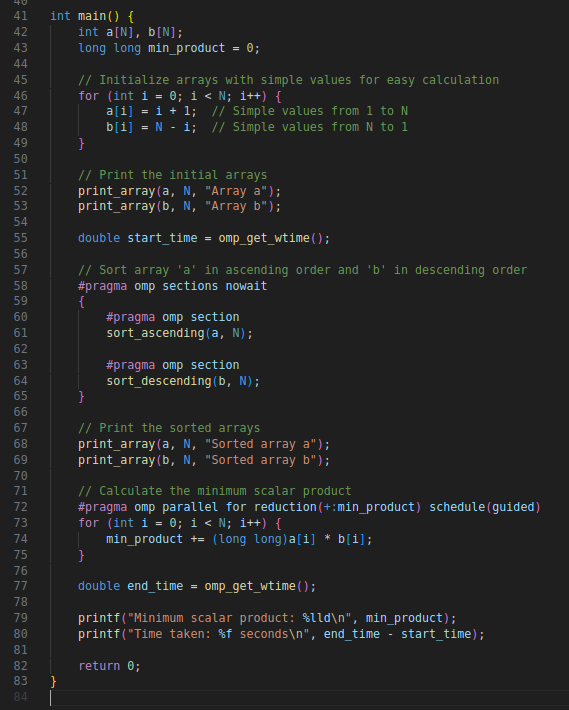
**Problem Statement 1:**

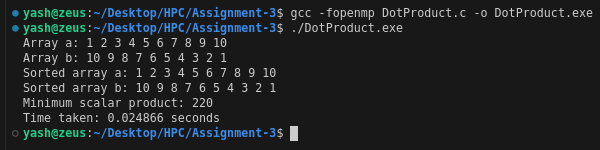
Analyse and implement a Parallel code for below program using OpenMP.

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

**Screenshots:**

****

****

****

**Information and analysis:**

**parallel for**: Distributes loop iterations among threads.****

* schedule(dynamic, 1000): Balances workload dynamically with chunk size.
* sections **and** section: Executes different code sections in parallel.
* reduction: Manages accumulation of results safely.
* schedule(guided): Dynamically assigns iterations with decreasing chunk size.
* ordered: Ensures code within the ordered block runs in the order of loop iterations.

****Parallelizing Sorting**:** Sorting operations are divided among multiple threads, reducing the time spent sorting.

* ****Concurrent Execution**:** Sorting of multiple arrays is done simultaneously, which cuts down the total execution time.
* ****Parallel Scalar Product Computation**:** The scalar product calculation is also parallelized, speeding up the computation.

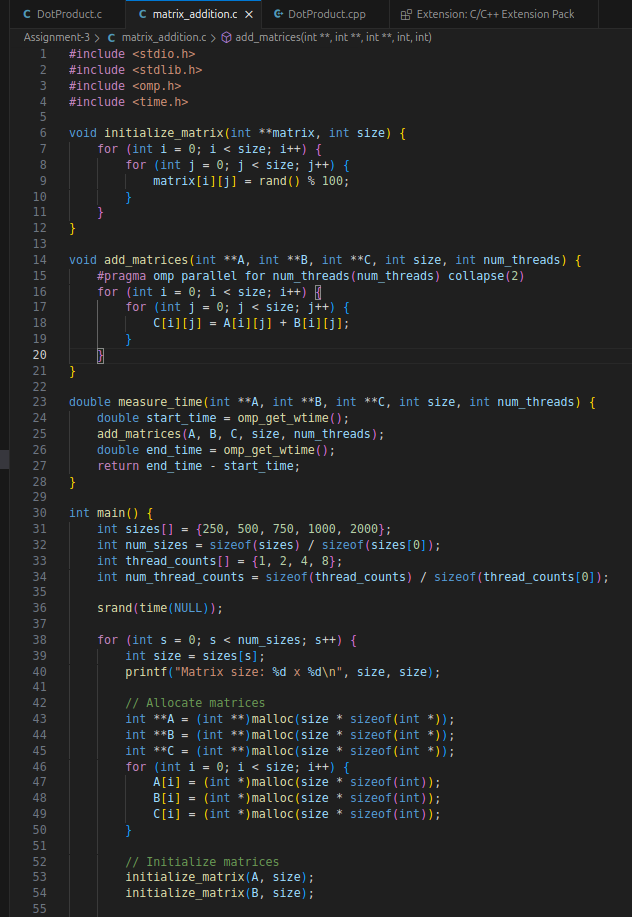
**Problem Statement 2:**

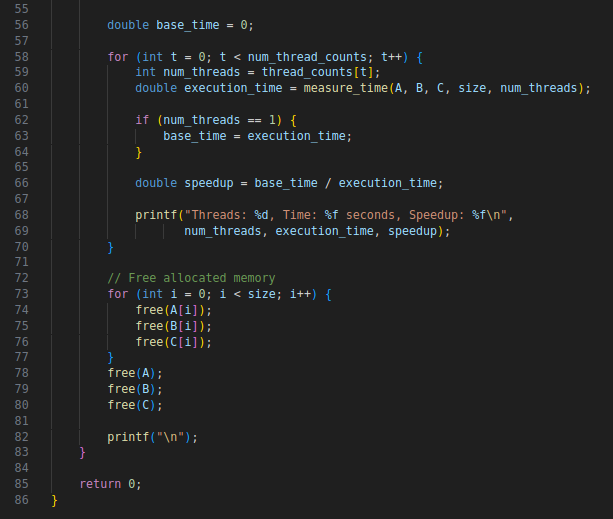
Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculate the execution time or use GPROF)

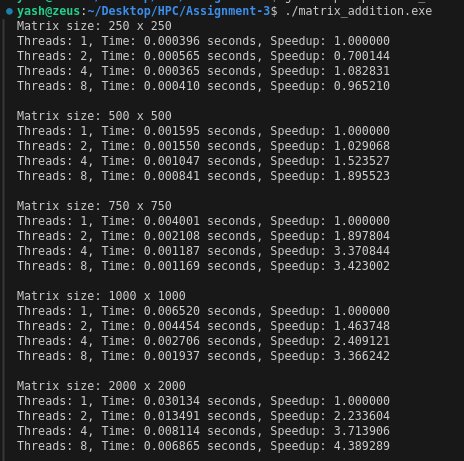
i. For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads.

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

**Screenshots:**

****

****

****

**Information and analysis:**

**Matrix Initialization:**

The code initializes two matrices with random values.

**Matrix Addition:** The matrix\_addition function performs the matrix addition in parallel using OpenMP.

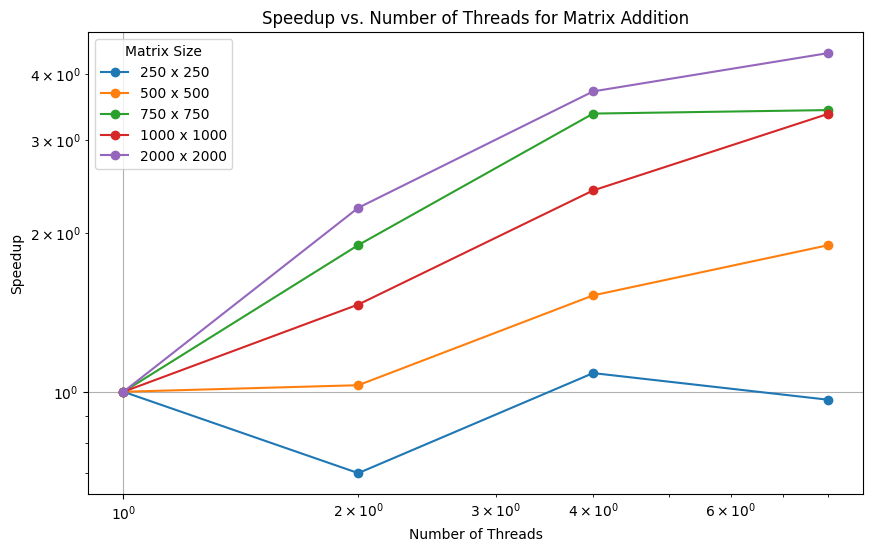
* The #pragma omp parallel for num\_threads(num\_threads) directive is used to parallelize the loop that adds the matrices.

**Timing:** omp\_get\_wtime() is used to measure the execution time of the matrix addition.

**Memory Management:** Allocates and frees memory for matrices.

**Testing Different Sizes and Threads:** The main function tests different matrix sizes and numbers of threads, printing the execution time for each configuration.

**Speedup = Time taken with 1 thread / Time taken with n threads**

****

**Scaling Behavior:**As the number of threads increases, the execution time should decrease, leading to higher speedup, especially for larger matrix sizes. This is due to better utilization of multiple cores.

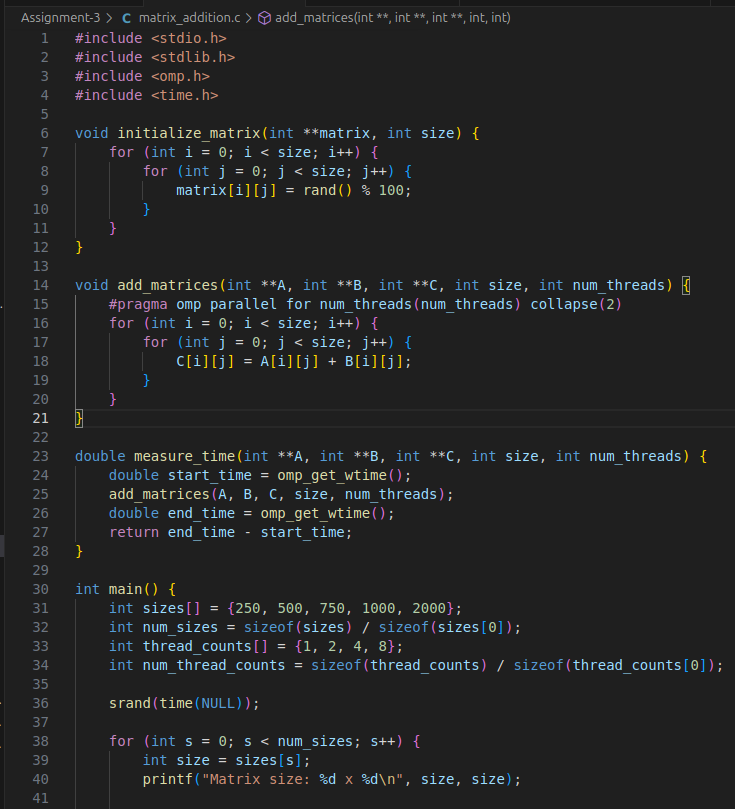
**Problem Statement 3:**

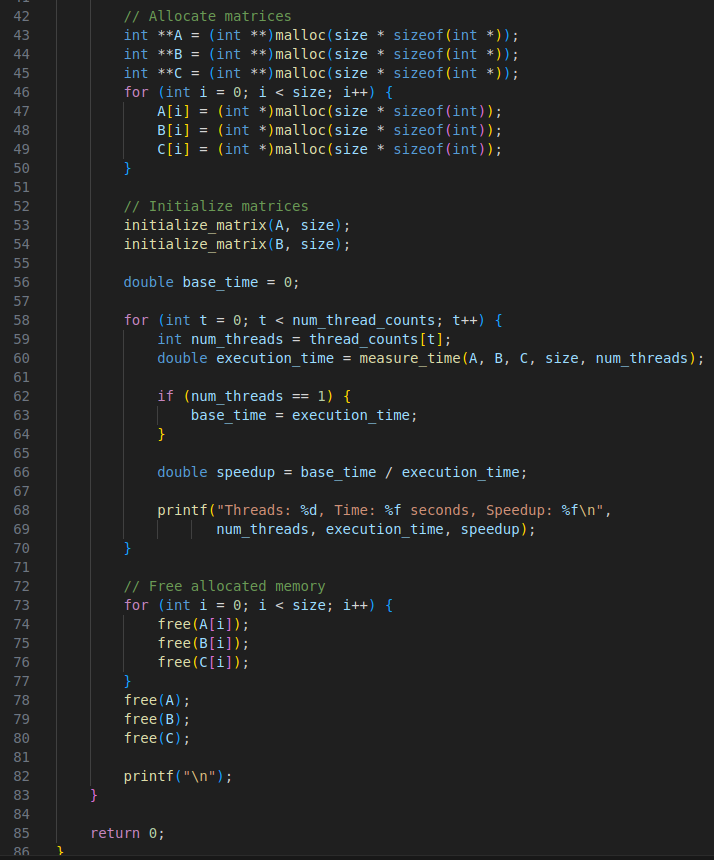
For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following:

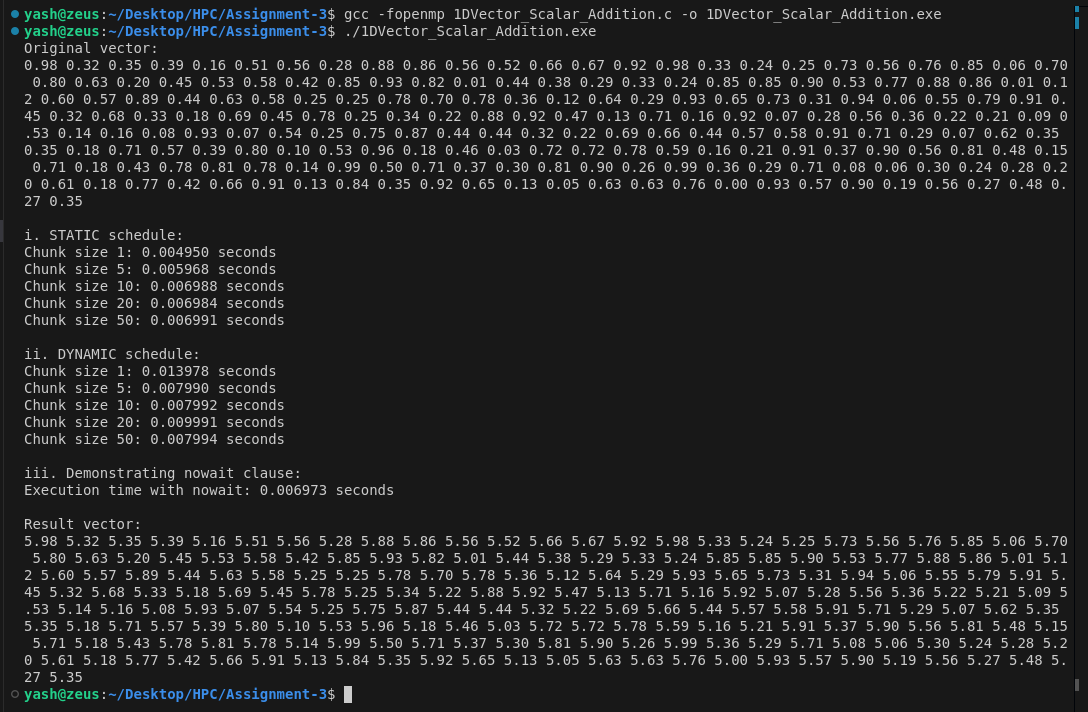
i. Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.

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:**

**Matrix Initialization and Addition:**

* initialize\_matrix: Initializes matrices with random values using parallelized nested loops.
* add\_matrices: Adds two matrices element-wise using parallelized nested loops.
* **Timing and Speedup Measurement:**
* measure\_time: Measures execution time for matrix addition with a specified number of threads.
* **Speedup**: Calculated as the ratio of the time with one thread to the time with multiple threads.
* **Memory Management:**
* allocate\_matrices and free\_matrices: Manage dynamic allocation and deallocation of matrix memory.

**Parallelization:**

* #pragma omp parallel for collapse(2) is used to parallelize the nested loops for matrix operations, improving performance by utilizing multiple threads.
* **Performance Measurement:**
* Timing results show execution time and speedup for different matrix sizes and thread counts.
* **Expected Results:**
* **Smaller Matrices:** May show minimal speedup due to parallelization overhead.
* **Larger Matrices:** Generally exhibit more significant speedup as parallel execution becomes more beneficial.

**Github Link:** <https://github.com/YashNawale26/High-Performance-Computing.git>