**Class:** FY M.Tech (Computer Science and Engineering)

**Year:** 2024-25 **Semester:** 2

**Course:** High Performance Computing Lab

**Practical No. 3**

**PRN:245519003\_DeepaHanamapure**

**Title of practical:**

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

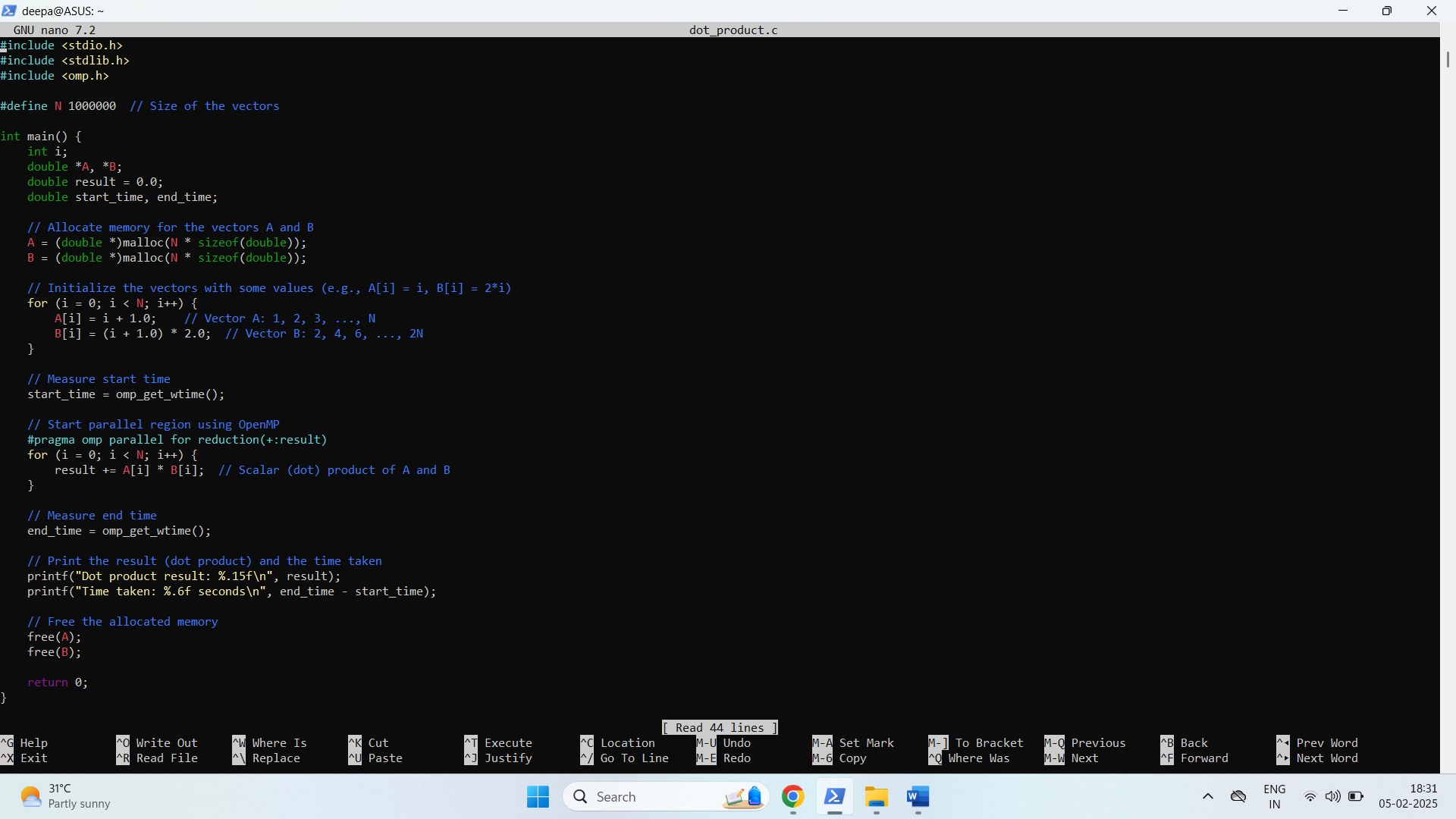
**Problem Statement 1:**

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

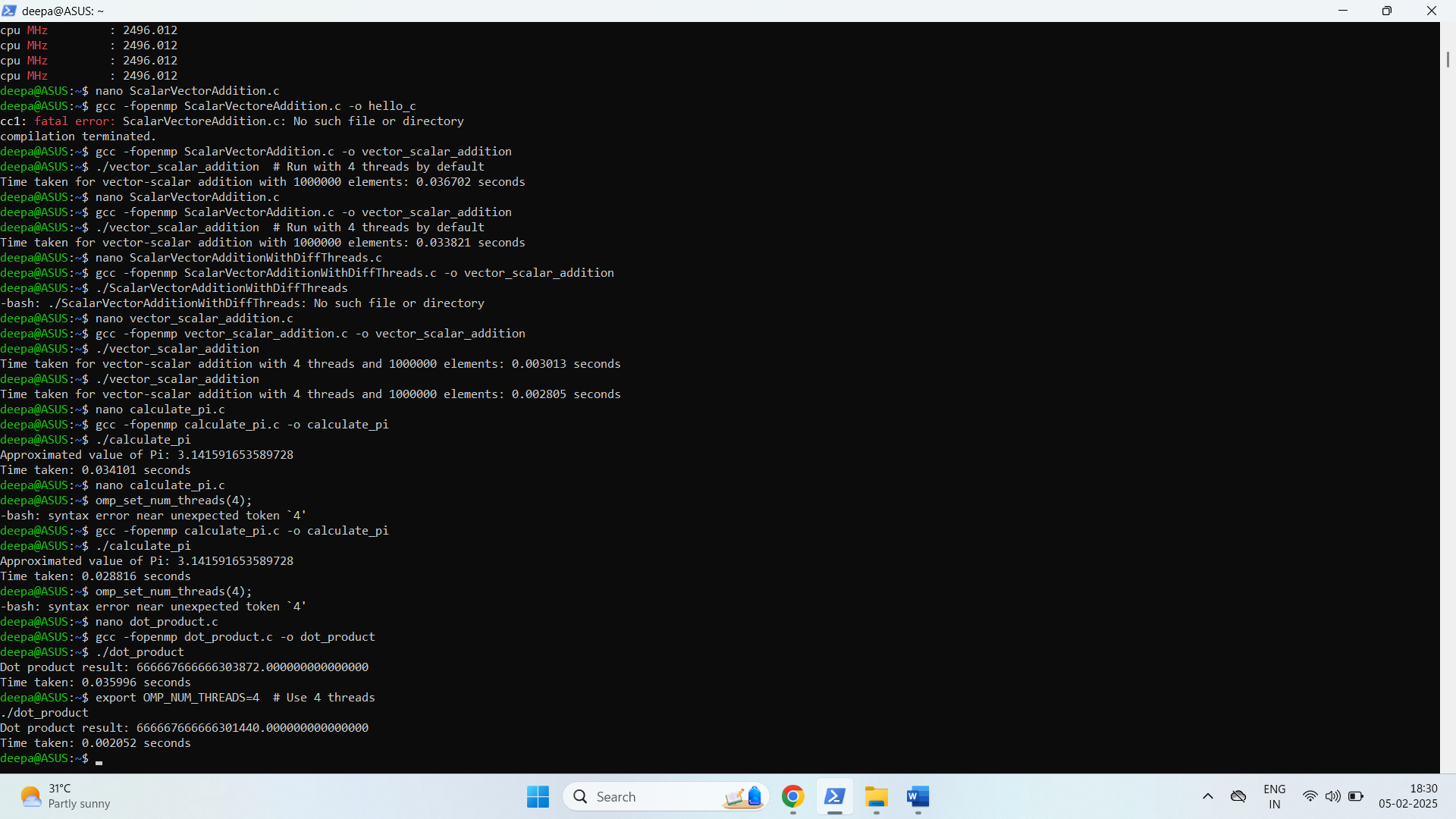
// C Program to find the minimum scalar product of two vectors (dot product)

**Screenshots:**

**Code:**



**Output:**



**Information and analysis:**

* Vector Initialization:
  + We initialize vector A as the sequence [1, 2, 3, ..., N] and vector B as [2, 4, 6, ..., 2N] for simplicity.
* Parallelization:
  + We parallelize the loop that computes the scalar product using the OpenMP directive #pragma omp parallel for.
  + The reduction(+:result) clause ensures that the results from each thread are safely summed up into the result variable.
* Performance Measurement:
  + We measure the execution time using omp\_get\_wtime() to see how long the parallel computation takes.
* Memory Allocation:
  + We dynamically allocate memory for vectors A and B based on the size N.

**Performance Analysis:** Execution Time: As the number of threads increases, the execution time decreases (up to the number of available CPU cores).

Scalability: The performance improves as the number of threads increases for larger N (i.e., the program scales well with the number of threads for large 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.

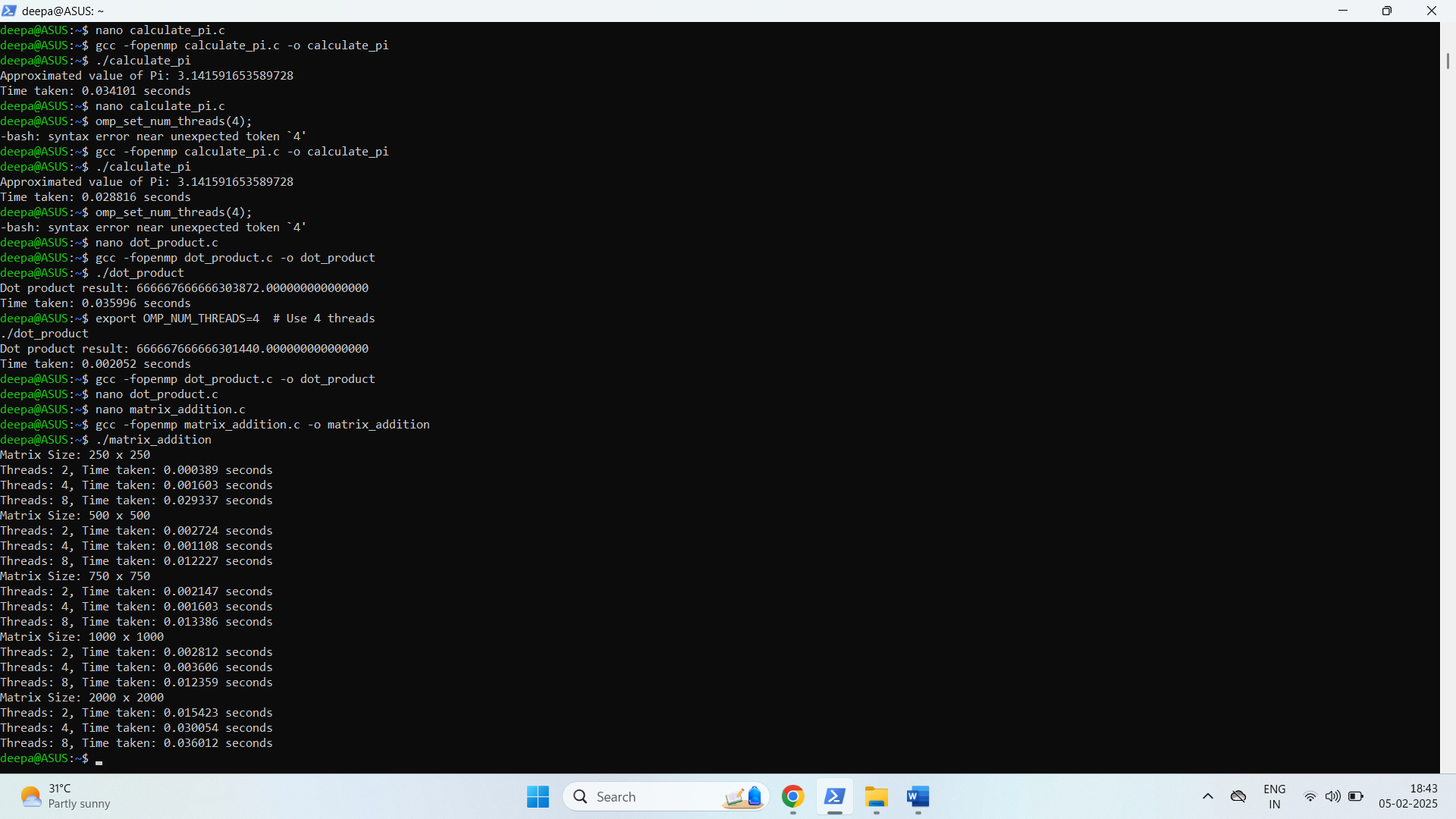
ii. Explain whether or not the scaling behaviour is as expected.

**Screenshots:**

**Input:**



**Output:**



**Information and analysis:**

1. Matrix Size: We define different matrix sizes (250x250, 500x500, 750x750, 1000x1000, and 2000x2000) to evaluate the performance.

2.Matrix Initialization: Matrices A and B are initialized with random values between 0 and 99 for simplicity.

3. OpenMP Parallelization:

* The matrix addition is parallelized using #pragma omp parallel for.
* This parallel directive allows each element addition (C[i][j] = A[i][j] + B[i][j]) to be done by different threads.

4. Number of Threads: We vary the number of threads (2, 4, 8) for each matrix size.

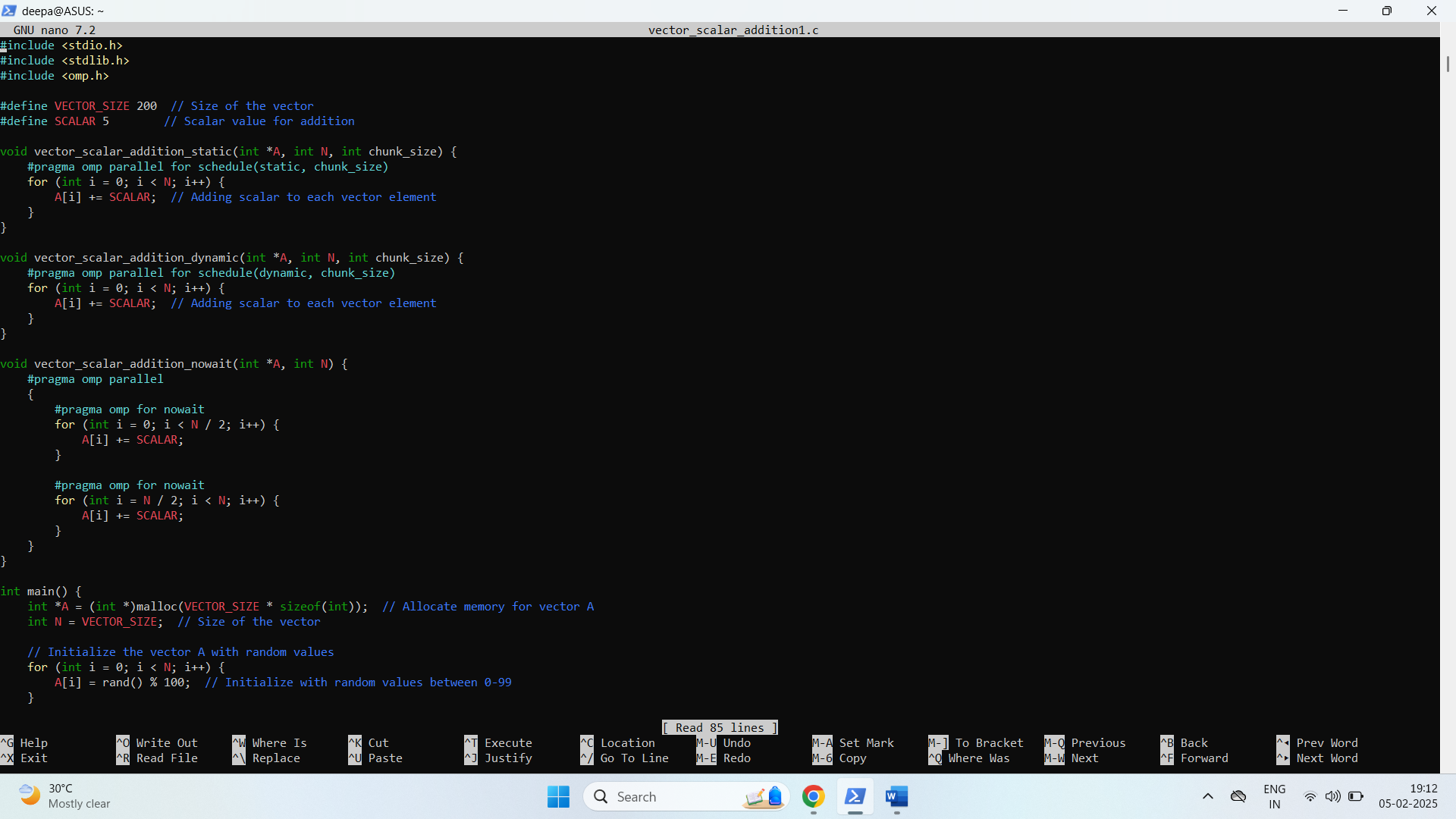
5. Execution Time Measurement: We use omp\_get\_wtime() to measure the time taken for matrix addition in seconds for each matrix size and each number of threads.

**Problem Statement 3:**

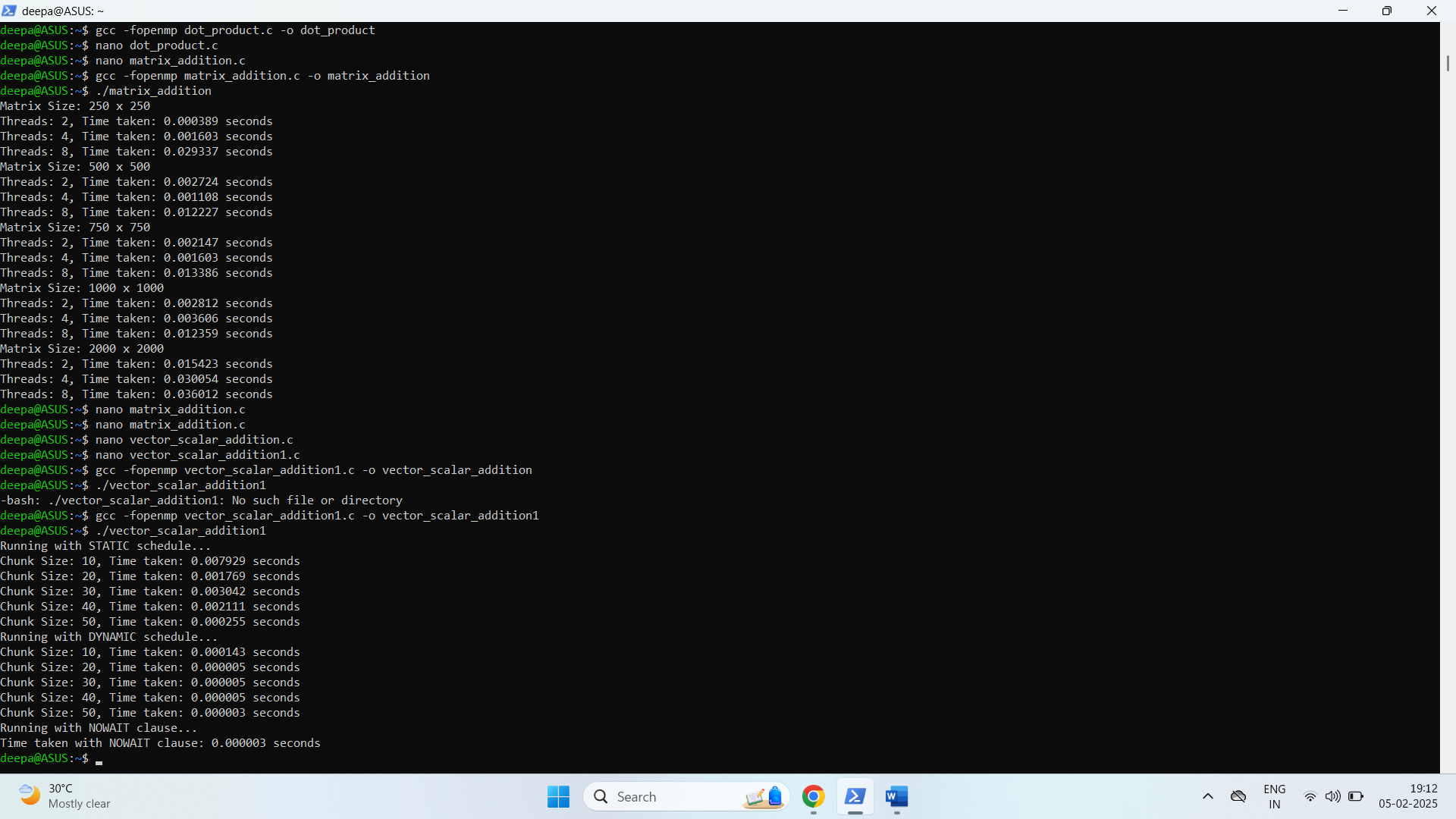
For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following: i. Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup. ii. Use DYNAMIC schedule and set the loop iteration chunk size to various sizes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

**Screenshots:**

**Input:**



**Output:**



**Information and analysis:**

1 Static Scheduling (static):

* The loop is divided into chunks of size chunk\_size and assigned to different threads.
* Each thread processes a contiguous portion of the loop without synchronization.
* The chunk size can be varied to analyze the performance impact.

2 Dynamic Scheduling (dynamic):

* In dynamic scheduling, chunks are dynamically assigned to threads as they finish their work. This allows for better load balancing, especially when the iterations are unevenly distributed.
* The chunk size is again configurable.

3. Nowait Clause:

* The nowait clause allows the threads to skip the implicit barrier at the end of a loop, allowing them to proceed to the next part of the code without waiting for the others to finish.
* This can improve performance if there is no dependency between the iterations.

**Github Link:**