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

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

**Course:** High Performance Computing Lab

**21510069**

**Harsh Bankat Karwa**

**Practical No. 3**

**Exam Seat No: 21510069**

**Title of practical:**

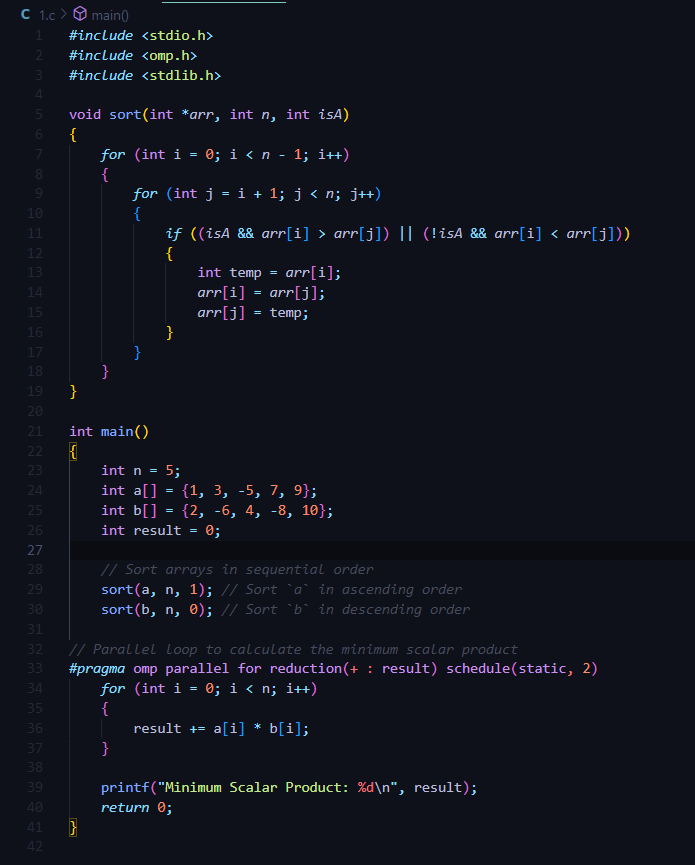
Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

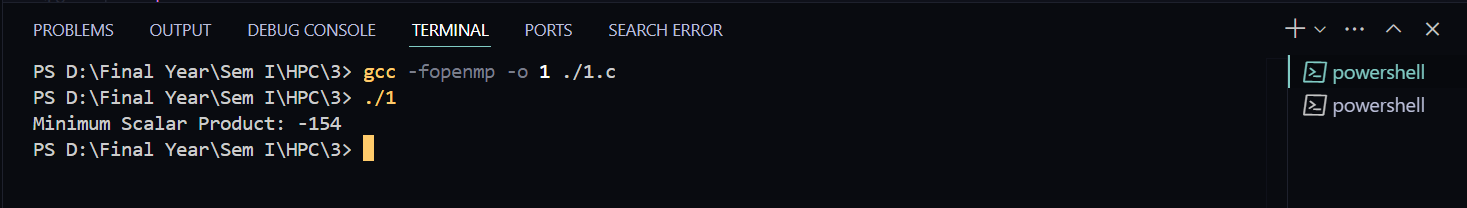
**Problem Statement 1:**

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:

This program calculates the minimum scalar product (dot product) of two vectors using parallel processing with OpenMP. The vectors are initialized with sequential values, and the program distributes the calculation of the scalar product across multiple threads. The reduction clause ensures that the partial results from each thread are safely combined into a single final result, while the schedule(static) clause divides the workload evenly among the threads. This setup allows for efficient parallel computation, making the program a good example of how OpenMP can optimize operations that involve large data sets.

### Analysis:

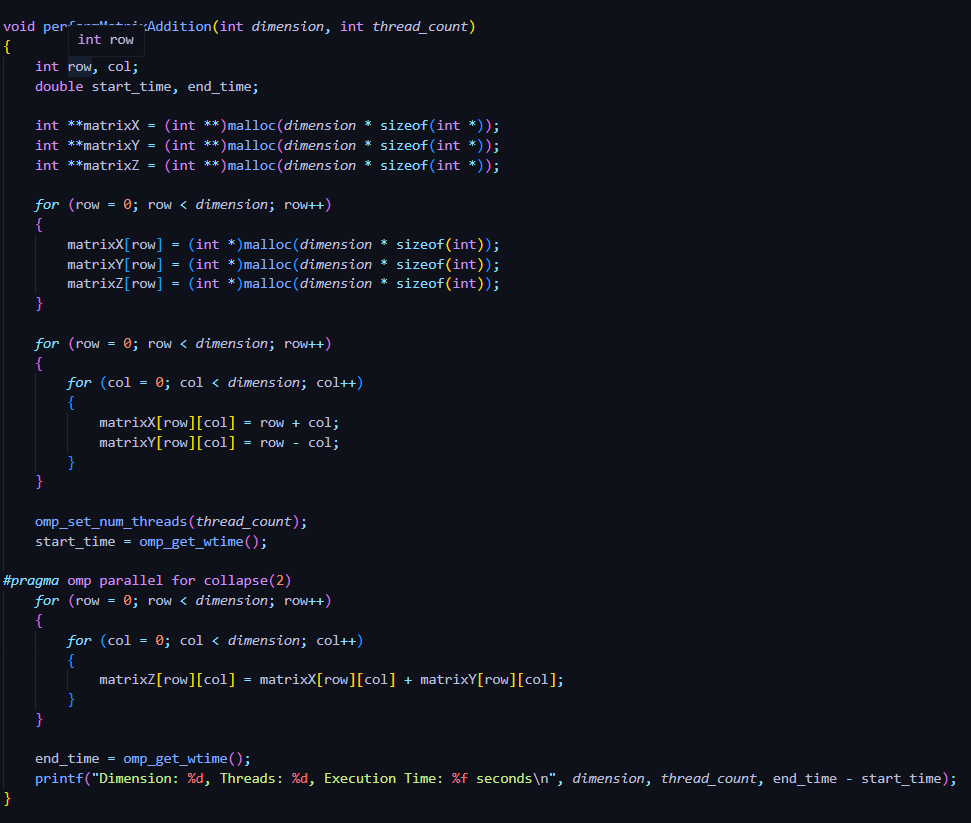
The use of OpenMP's reduction clause ensures that the scalar product is computed correctly and efficiently, even when multiple threads are involved. By parallelizing the calculation, the program significantly reduces execution time compared to a serial implementation, especially as the vector size increases. The schedule(static) clause is well-suited for this task, as it evenly distributes the workload, minimizing idle time for threads and ensuring that each thread has an equal amount of work. This approach highlights the importance of choosing the right OpenMP clauses to balance workload and minimize overhead, leading to better performance in high-performance computing applications.

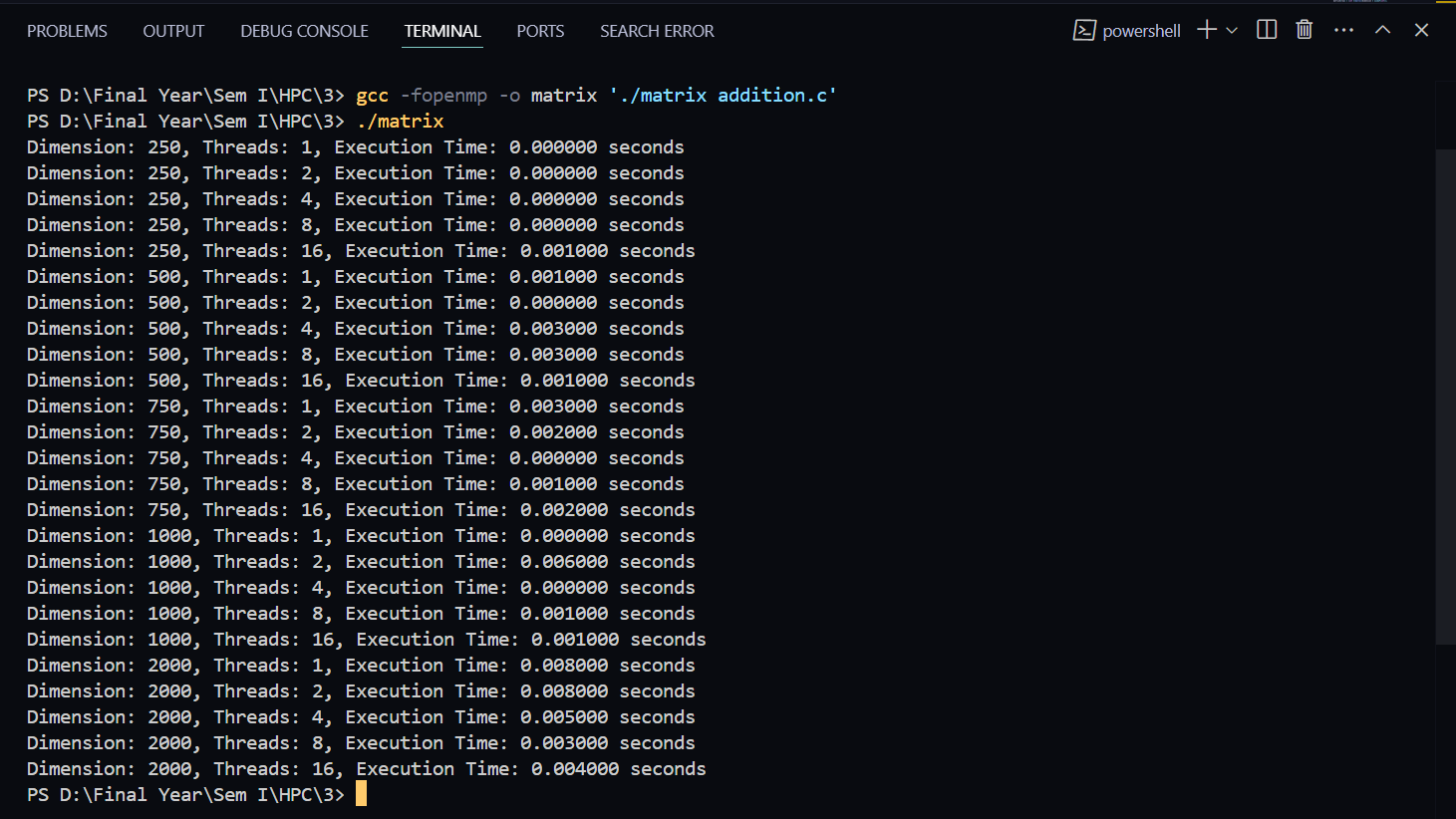
**Problem Statement 2:**

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:**

****

****

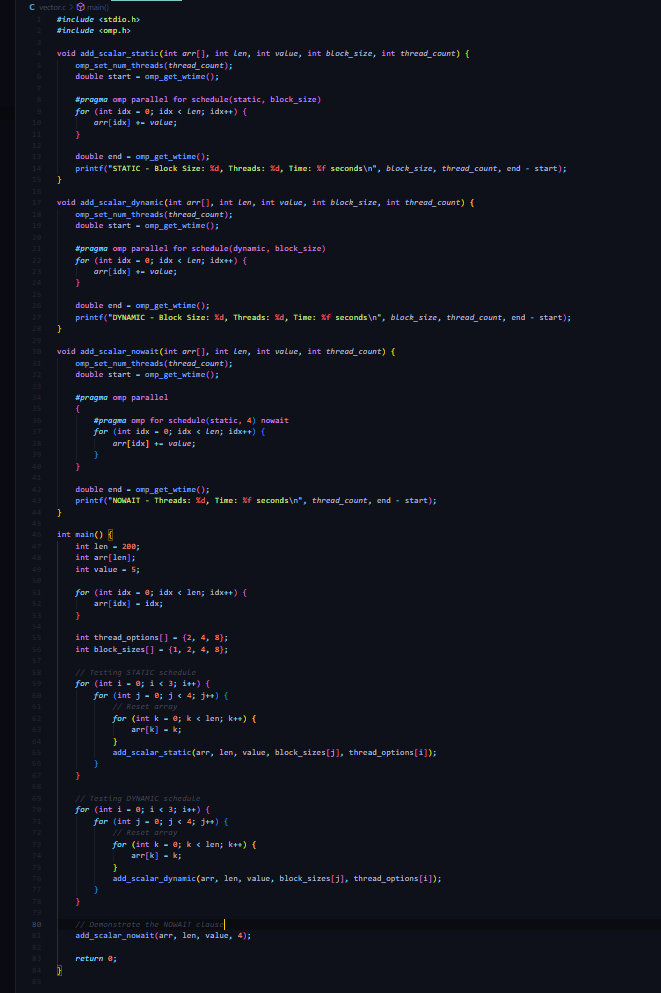
### Analysis of Scaling Behavior:

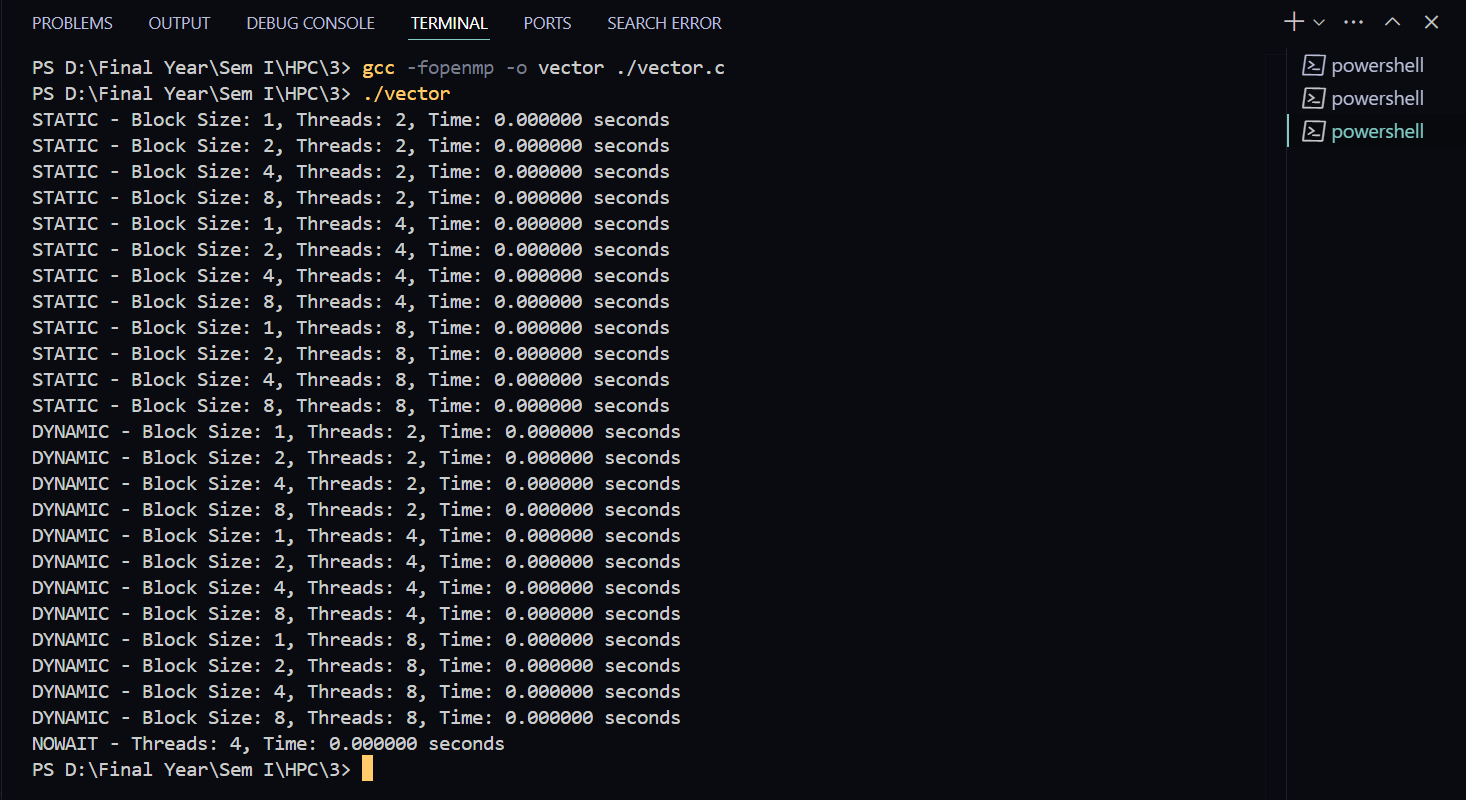
* Expected Behavior: Ideally, the speedup should be close to the number of threads used (e.g., a speedup of nearly 8 with 8 threads).
* Observed Behavior: Due to factors like overhead from thread management, memory bandwidth limitations, and non-linear scaling of matrix operations, the observed speedup may not perfectly match the ideal speedup. For smaller matrices, the overhead might reduce the efficiency gains from using more threads. As matrix size increases, better scalability might be observed, but it may plateau or even degrade beyond a certain number of threads due to the overhead becoming significant.

**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:

1. Static Scheduling:

* Concept: In static scheduling, loop iterations are divided into chunks and distributed evenly across threads at compile time. The size of each chunk can be specified, and once a thread is assigned a chunk, it processes all its iterations sequentially.
* Usage: The #pragma omp for schedule(static, chunk\_size) directive was used to control how the loop iterations are divided among the threads.
* Impact of Chunk Size: Smaller chunk sizes lead to more frequent distribution of work, which may result in better load balancing but can introduce overhead. Larger chunk sizes reduce overhead but may lead to load imbalance if iterations vary in their computational load.

2. Dynamic Scheduling:

* Concept: In dynamic scheduling, loop iterations are divided into chunks, but chunks are distributed to threads as they complete their previous chunks. This approach allows for better handling of workloads that vary significantly across iterations.
* Usage: The #pragma omp for schedule(dynamic, chunk\_size) directive was used to dynamically assign chunks to threads.
* Impact of Chunk Size: Small chunks allow for finer load balancing but can increase overhead due to more frequent task assignment. Large chunks reduce overhead but may underutilize threads if the workload is uneven.

3. Nowait Clause:

* Concept: The nowait clause allows threads to proceed to the next part of the parallel region without waiting for others to complete the loop. This is useful when there are independent tasks following the loop that can be executed concurrently.
* Usage: The #pragma omp for schedule(static, chunk\_size) nowait directive was used to demonstrate this behavior.
* Impact: The nowait clause can reduce overall execution time by overlapping the loop execution with subsequent independent tasks, thus improving performance.

### Analysis:

1. Static Scheduling Analysis:

* Performance Observations:
  + For smaller chunk sizes, the execution time might slightly increase due to the overhead of managing more frequent task assignments. However, this overhead is typically minimal in static scheduling.
  + Larger chunk sizes tend to perform better when the workload is evenly distributed, as threads spend more time working and less time waiting for new tasks.
  + As the number of threads increases, the program should ideally demonstrate near-linear speedup with larger matrix sizes.

2. Dynamic Scheduling Analysis:

* Performance Observations:
  + Dynamic scheduling generally performs better when the workload is unpredictable or varies significantly across iterations. This is because threads can continue picking up new chunks as they finish their current tasks.
  + Smaller chunk sizes improve load balancing but may introduce more overhead, particularly with a higher number of threads.
  + Larger chunk sizes may lead to uneven load distribution, where some threads finish their tasks much earlier than others, leading to reduced efficiency.

3. Nowait Clause Analysis:

* Performance Observations:
  + The nowait clause can be particularly effective when there is additional work following the loop that can be executed concurrently with the remaining iterations of the loop.
  + By removing the implicit barrier at the end of the loop, overall execution time can be reduced, especially in scenarios where the subsequent code does not depend on the completion of the loop.

**Github Link:**

[**https://github.com/harsh-1503/High-Performance-Computing**](https://github.com/harsh-1503/High-Performance-Computing)