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

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

**Course:** High Performance Computing Lab

**Practical No. 3**

**Exam Seat No: 21510048**

**Name-Sujan Mujawar**

**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)

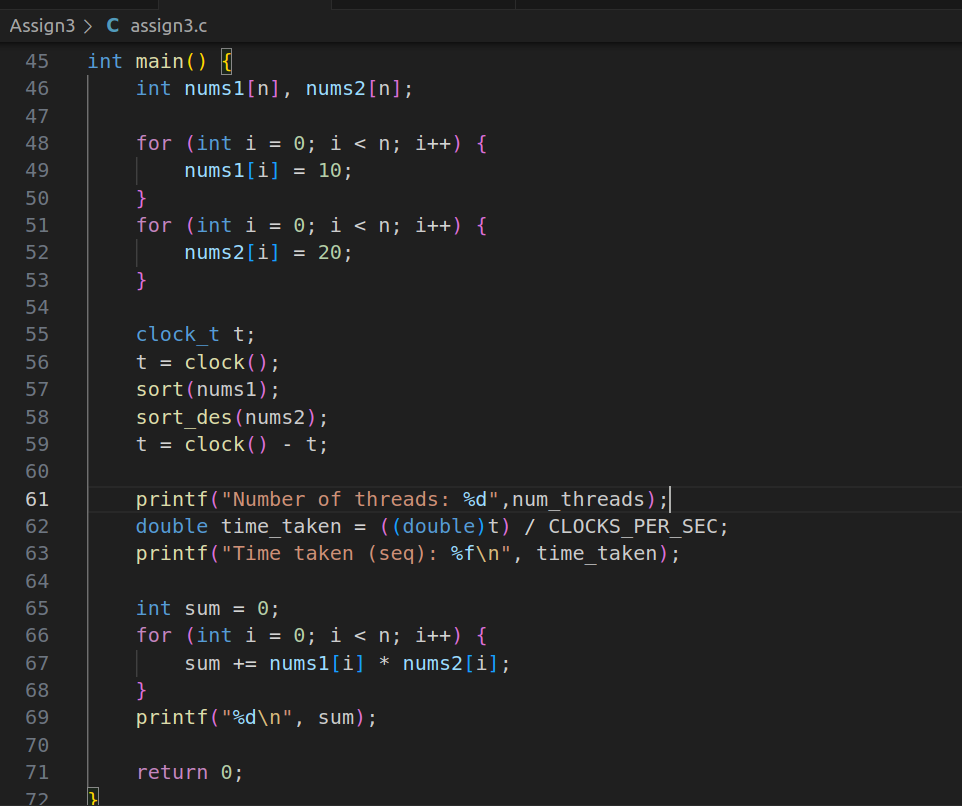
**Screenshots:**

**Code -**

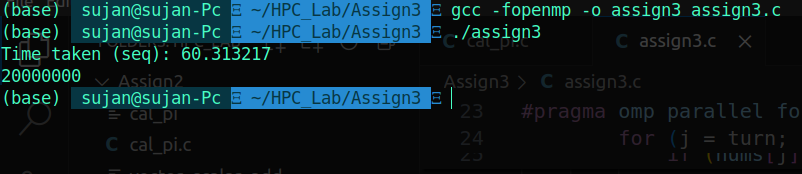
****

****

****

****

**Output:**

****

**Analysis:**

|  |  |
| --- | --- |
| Number of threads | Time required to complete |
| 2 | 14.39 |
| 4 | 15.947185 |

**Information :**

The increase in execution time with more threads indicates that the overheads associated with thread management and the inefficiency of the sorting algorithm are significant factors affecting performance. The odd-even transposition sort algorithm is likely not well-suited for parallel execution, and the overhead of managing more threads is overshadowing the potential benefits of parallelism.

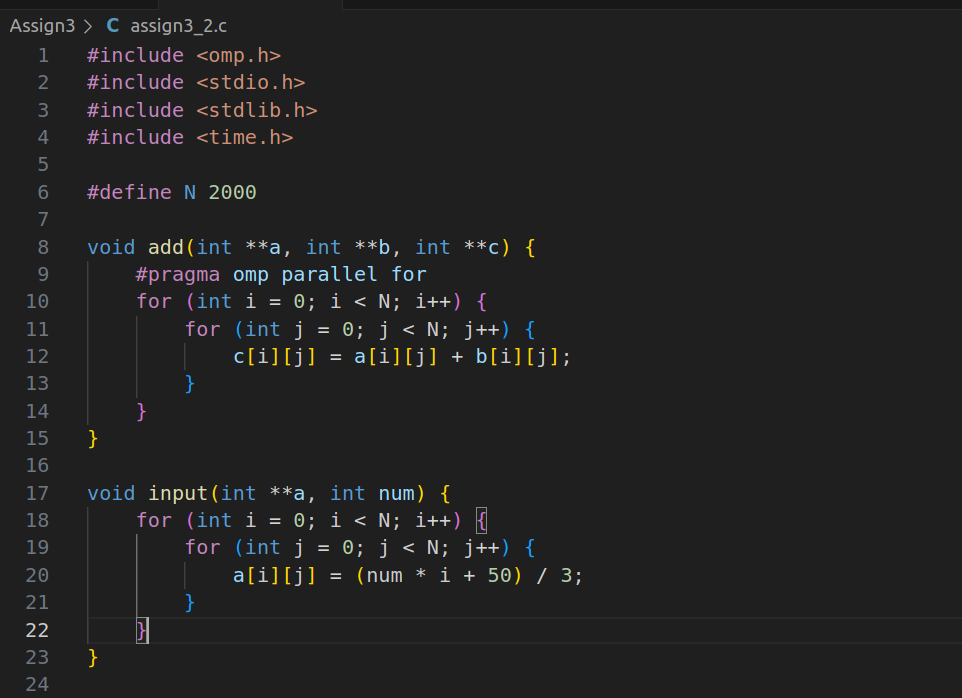
**Problem Statement 2:**

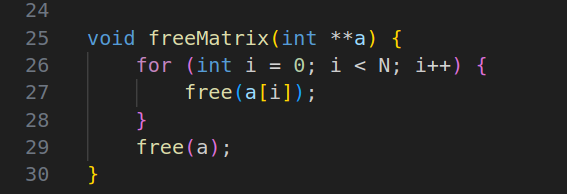
Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculate the execution time or use GPROF)

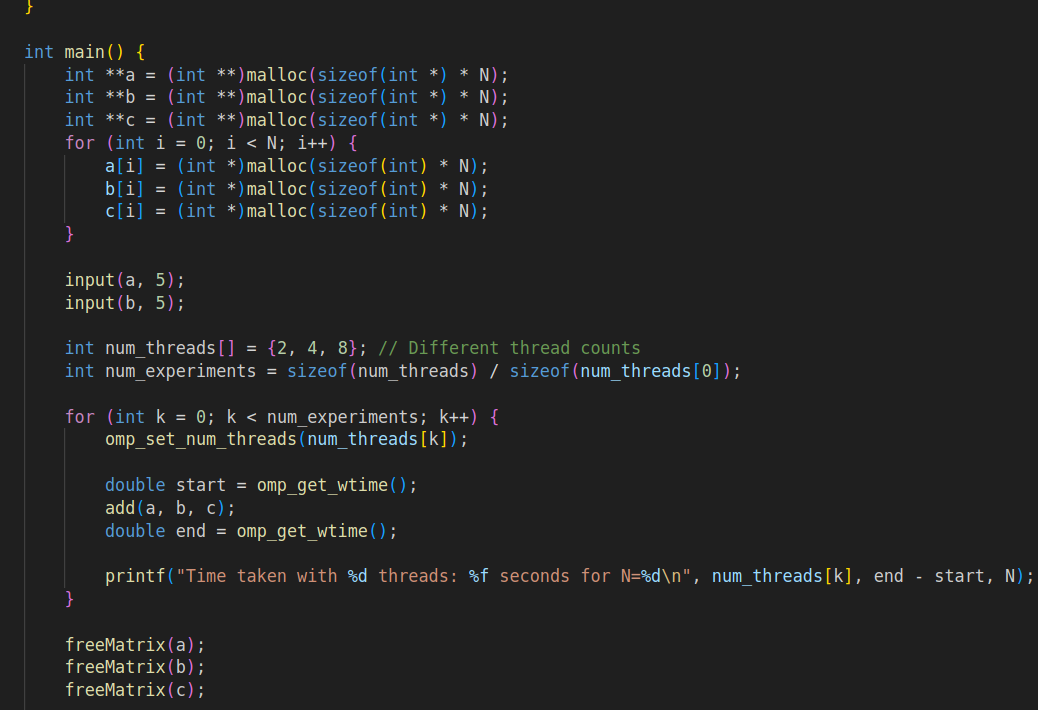
i. For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads.

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

**Screenshots:**

****

****



**Information and analysis:**

|  |  |
| --- | --- |
| N | Time |
| 2000 | 0.008797 |
| 1000 | 0.001675 |
| 500 | 0.006163 |
| 250 | 0.000494 |

|  |  |  |  |
| --- | --- | --- | --- |
| N | thread\_num=2 | thread\_num=4 | thread\_num=8 |
| 250 | 0.000398 | 0.000349 | 0.010385 |
| 500 | 0.000502 | 0.000344 | 0.001230 |
| 1000 | 0.002052 | 0.001141 | 0.002699 |
| 2000 | 0.006030 | 0.006261 | 0.005401 |

**Conclusion:**

**The observed scaling behavior does not entirely match the ideal expectations for parallel performance:**

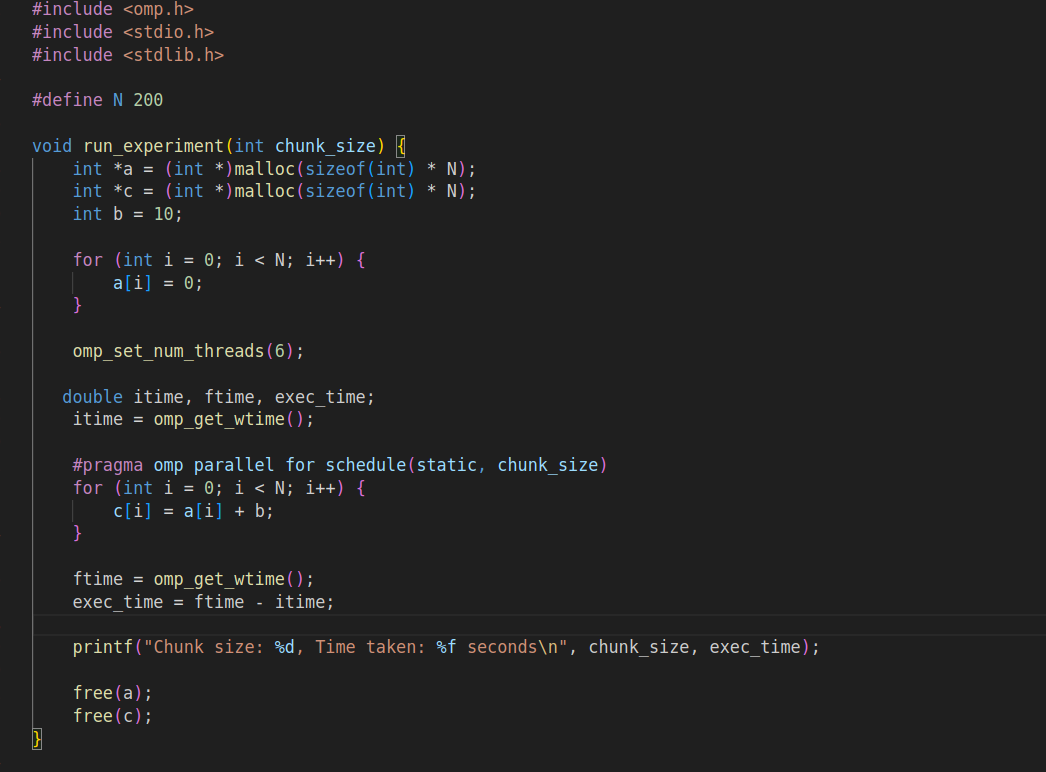
* **For smaller matrices**, the overhead of managing more threads dominates, leading to increased execution times.
* **For larger matrices**, the benefits of more threads may start to outweigh the overhead, but optimal performance is not always achieved.

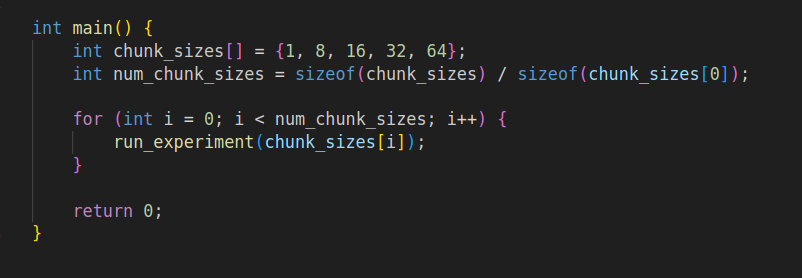
**Problem Statement 3:**

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.

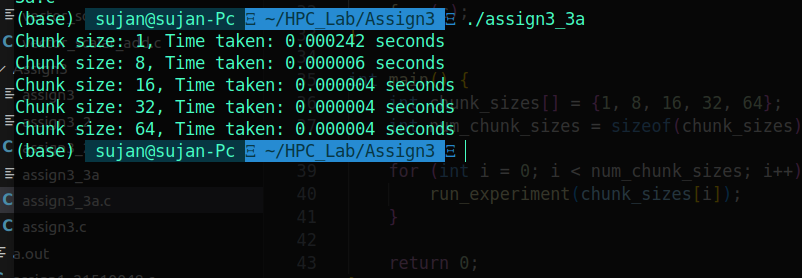
**Screenshots:**

**For Static Scedule**

****

****

**Information and analysis:**

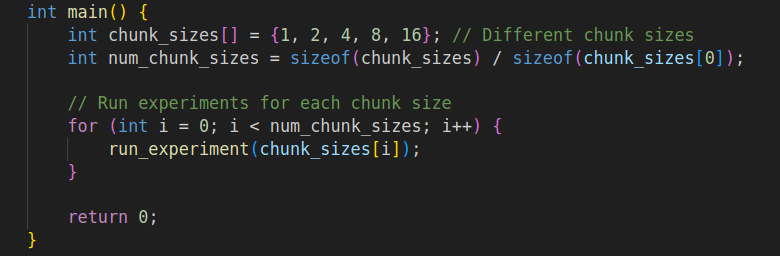
****

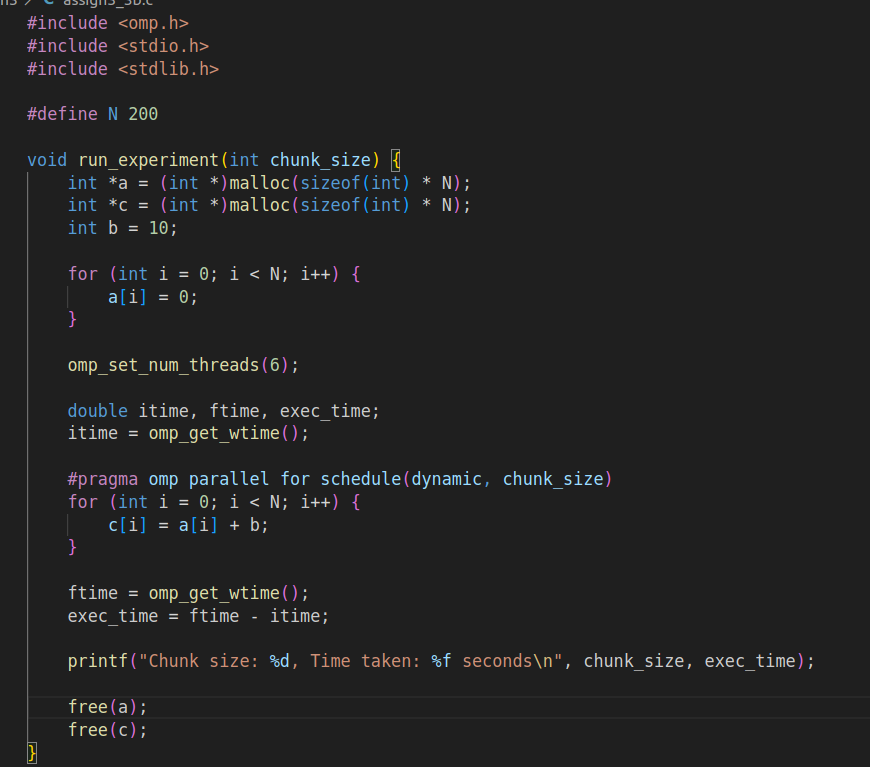
**Analysis:**

**Performance Improvement with Larger Chunk Sizes:**

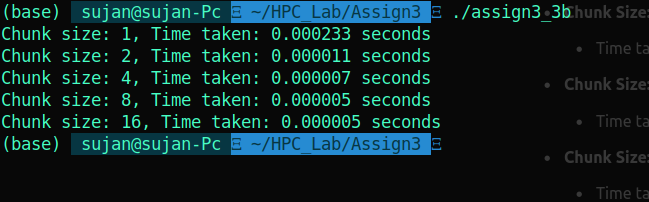
* **Chunk size of 1**: This has the highest execution time. A chunk size of 1 means that each thread processes one element at a time, which can lead to higher overhead due to frequent thread synchronization and less efficient load distribution.
* **Chunk sizes of 8, 16, 32, and 64**: These have significantly lower execution times. The performance improvement with larger chunk sizes due to less overhead from thread management and more efficient data access patterns.
* **Diminishing Returns**:
* For chunk sizes 16, 32, and 64, the execution times are essentially the same (0.000004 seconds). This indicates that after a certain point, increasing the chunk size further does not yield additional performance benefits.

**For dynamic Schedule:**

**ScreenShots:**

****

**Output:**

****

**Analysis:**

**Performance Improvement with Larger Chunk Sizes:**

* **Chunk size of 1**: This has the highest execution time, indicating that managing very small chunks incurs high overhead.
* **Chunk sizes of 2, 4, 8, and 16**: The execution time decreases as the chunk size increases. Larger chunk sizes reduce the frequency of scheduling overhead, as threads handle larger blocks of work before requesting more.
* **Diminishing Returns**:
* For chunk sizes of 8 and 16, the execution times are the same (0.000005 seconds). This suggests that, beyond a certain point, increasing the chunk size further does not yield additional performance improvements.

**Demostration of nowait**



**Output:**



**Github Link:**