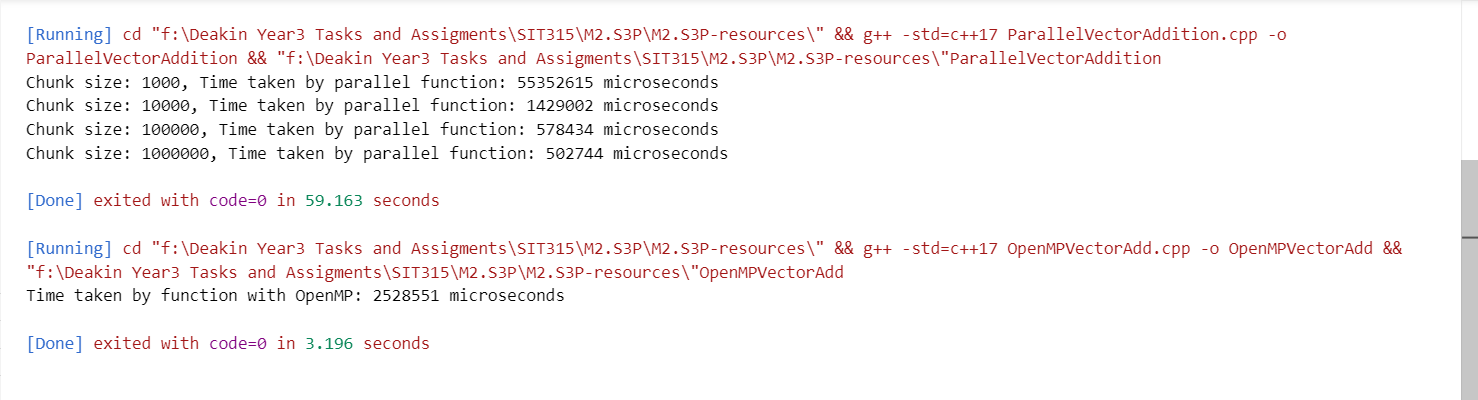
Activity 1

1.2.



In this test, the vector addition implemented using std::thread outperformed the OpenMP implementation when the chunk size was set to a larger value (1,000,000), with the former taking approximately 502,744 microseconds compared to OpenMP's 2,528,511 microseconds. This indicates that while OpenMP is generally easier to use and typically does not require manual tuning, a carefully optimized std::thread implementation can be more efficient in specific scenarios.

Activity 2

1.

Adding the default(none) clause in the OpenMP directive forces explicit declaration of how each variable is shared among threads. To prevent compilation errors, variables like the vector pointers (v1, v2, v3) should be declared as shared, as all threads need to access the same memory locations, while the loop index (i) should be private to ensure each thread operates on different elements. Incorrectly marking vector pointers as private would lead to each thread working on an uninitialized or separate copy of the vector, causing the program to produce incorrect results due to threads not updating the intended shared data.

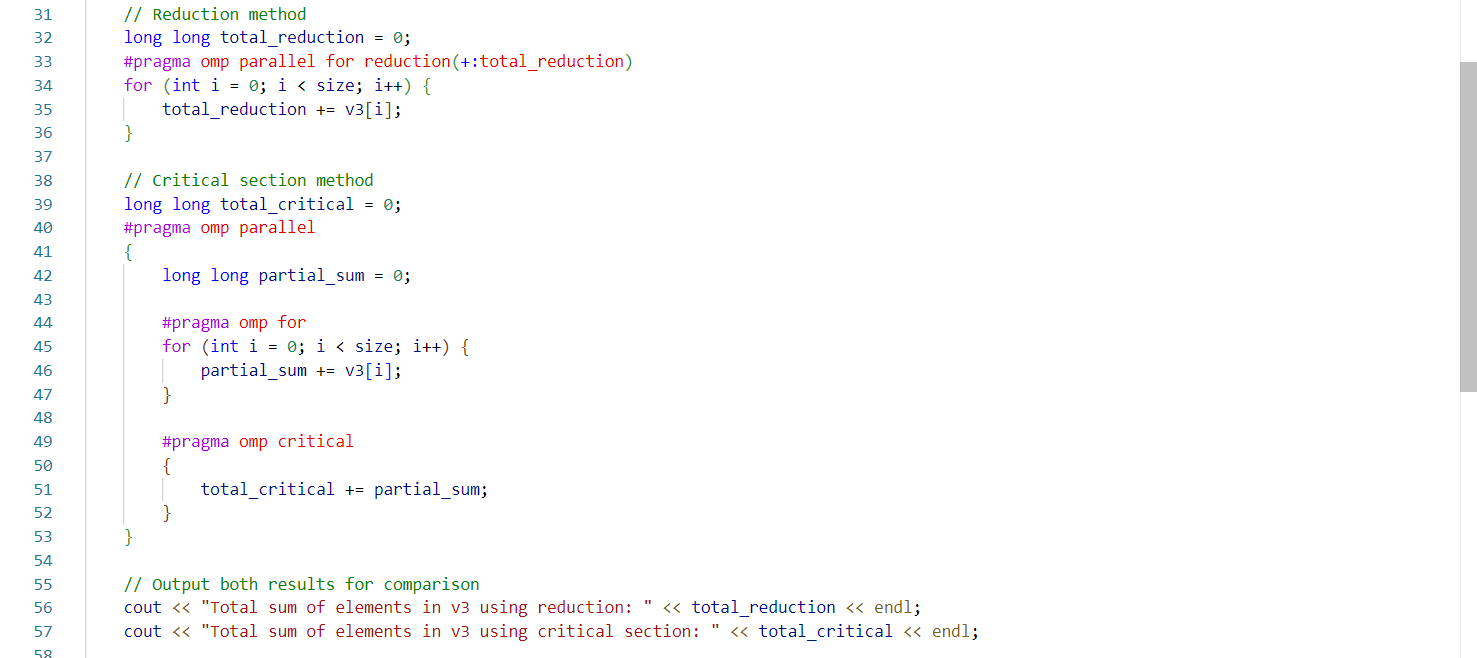
2.

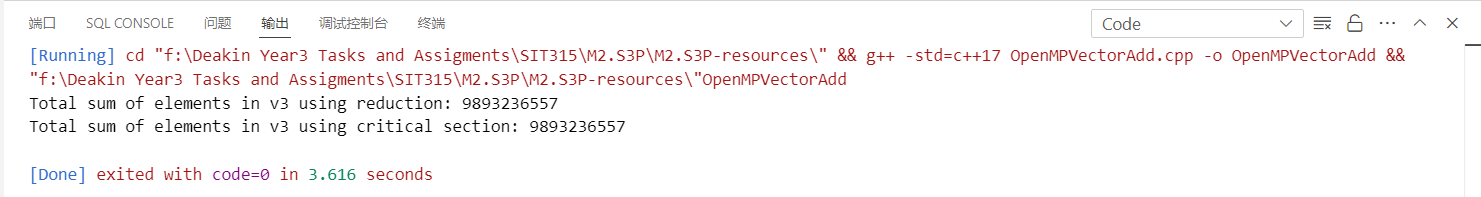


3.

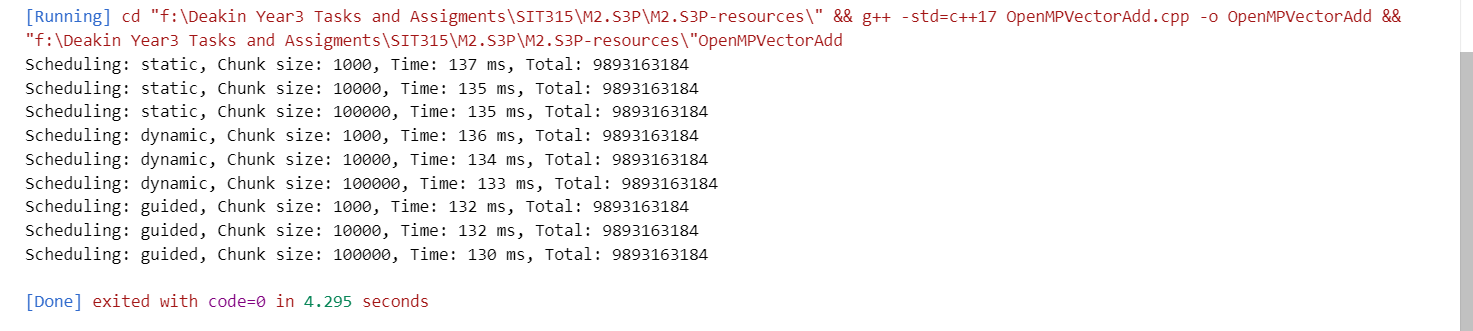


4.





5.



Static scheduling produced nearly identical execution times across all chunk sizes, indicating that the workload was evenly distributed. Dynamic scheduling performed similarly, with only slight variations, suggesting minimal overhead from work reassignment. Guided scheduling showed a slight performance improvement, particularly with larger chunk sizes, due to its balanced approach of starting with larger chunks and gradually decreasing the size, thereby optimizing both load balancing and overhead.