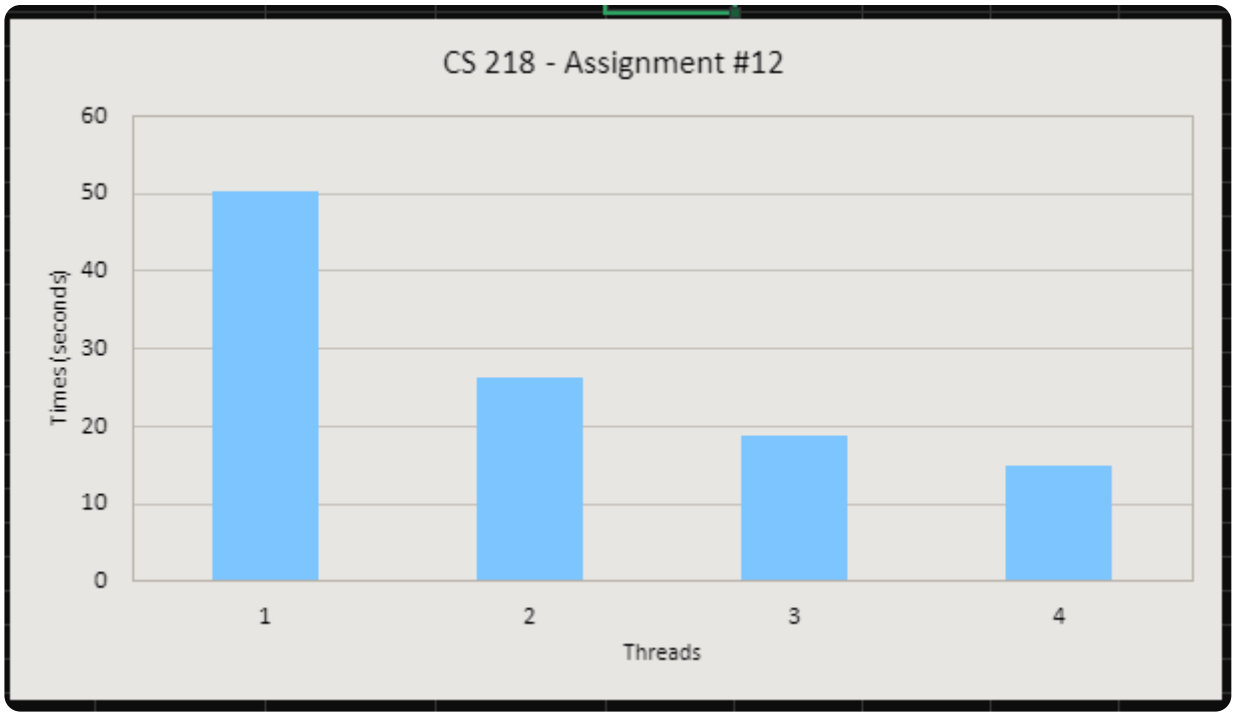
1. Resulst Copy

Timing	1 Thread	2 Threads	3 Threads	4 Threads
real	0m50.072s	0m26.095s	0m18.773s	0m14.606s
user	0m50.061s	0m52.166s	0m56.290s	0m58.377s
sys	0m0.000s	0m0.000s	0m0.000s	0m0.000s
Speed Up	1	1.92	2.67	3.43

2. Plotting Graph



3. The difference with and without the locking calls for the parallel execution

The outcome between a lock and no lock execution is widely different not just in time complexity but also in the accuracy of the desired results.

Our desired situation in a parallel execution is the threads have implemented a locking mechanism. It will verify that no other thread is currently holding the lock.

If it is locked, it waits until it is unlocked, otherwise, it continues executing the code.

This mechanism will enforce limits on access to a resource when there are many threads of execution. At the same time, reach the results faster and ensure data consistency. This is the key feature for parallel execution to work since we don't want two threads to execute at the same time, the same object, which results to often the program will skip over the next resource due to the previous resource being stuck with being executed twice (++2).

The results are different as a consequence of users' design with a lock and no lock. While counting the number for assignment 12, we get overcounting/ undercounting the perfect/ abundant/ deficient numbers due to the exact reason stated above (multiple threads reading the same data at the same time). Hence, losing accuracy and costing time to the program.