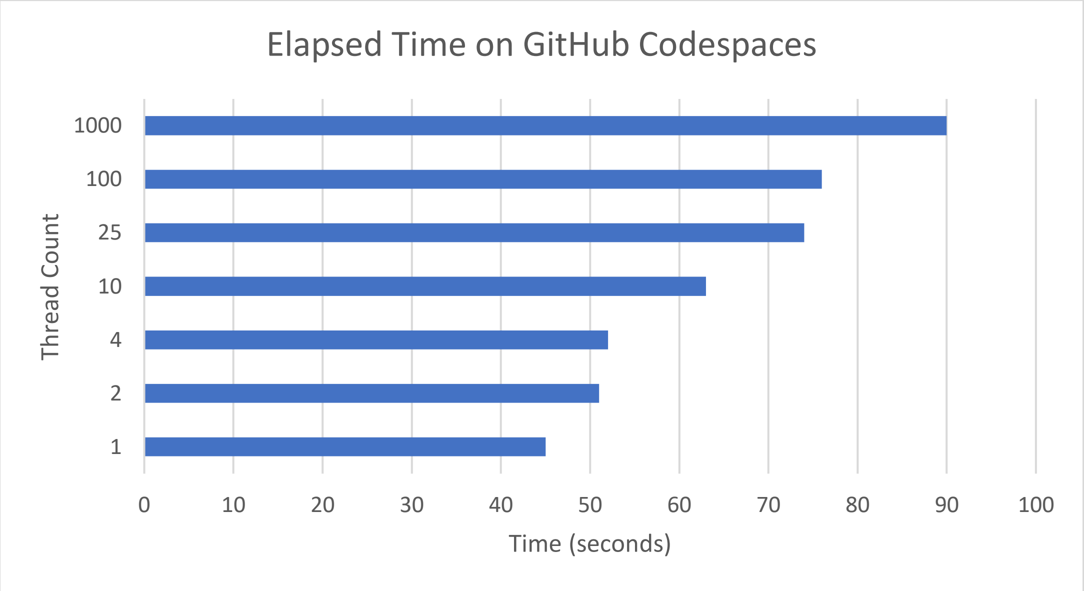
Star Catalog Multithreading Report

In this assignment, I was tasked with writing a multithreaded application in C to determine the optimal thread count for best performance when calculating the average angular distance between 30,000 stars in the Tycho Star Catalogue. The unthreaded application when ran serially takes a significant amount of time to run. To improve this time and execute the program faster, I implemented support for POSIX threads using the pthread.h library. This library allows the program to control multiple different flows of work that overlap in time. A mutex was implemented to protect resources, ensure consistent data, and perform valid calculations when using multiple threads. The data was recorded after running the program multiple times both locally and on GitHub Codespaces using 1, 2, 4, 25, 100, and 1000 threads.

While running the application I recorded the elapsed time for each thread count to determine the optimal number of threads. I implemented timing using the sys/time.h library. I used the gettimeofday() function before thread creation and again after joining the threads, and then subtracted the start time from the end time to get my total time. I chose this timing method over others because I wanted the elapsed time that the user experiences rather than the CPU time.

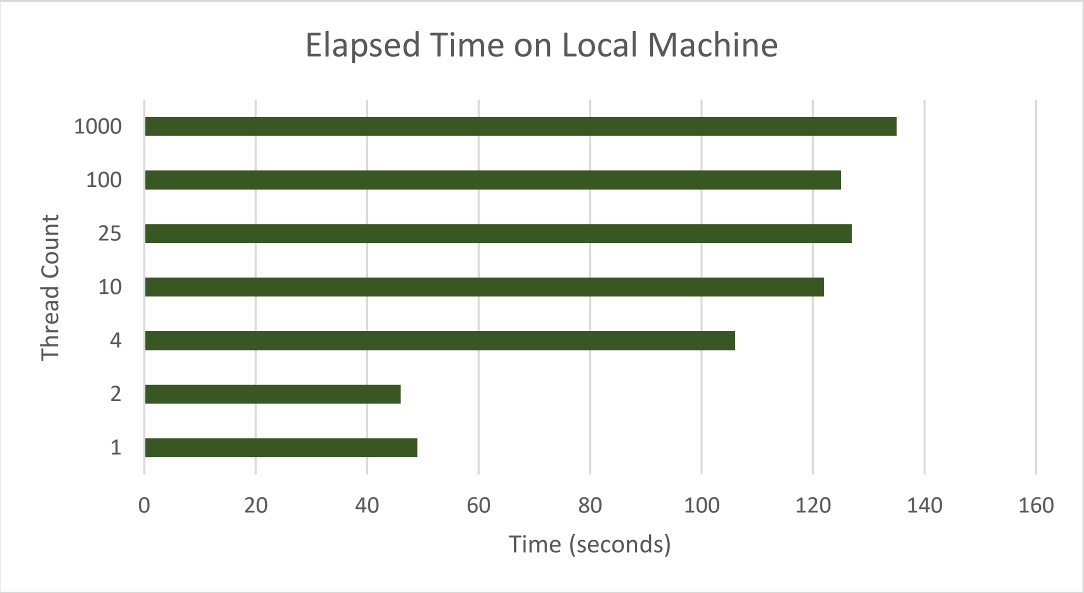
GitHub Codespaces Results



Thread Count	Elapsed Time (seconds)
1	45
2	51
4	52
10	63
25	74
100	76
1000	90

The program first ran on Codespaces, a cloud-based development environment that uses a container to provide you with common languages, tools, and utilities for development. A slight decrease in elapsed time was expected when using 2-4 threads, and then an increase once the thread count exceeded 4. However, despite acquiring correct calculations and no deadlocks or race conditions, the time increased regardless of the number of threads used. This is due to non-deterministic performance on the cloud provider. The shared computing resources don’t provide consistent performance.

Local Machine Results



Thread Count	Elapsed Time (seconds)
1	49
2	46
4	106
10	122
25	127
100	125
1000	135

After obtaining undesirable results due to the nature of Codespaces. The program was run on my local machine, an M1 Pro MacBook. This time, since not sharing computing resources, the results were closer to the hypothesis expected, that is more than one thread would decrease execution time, regardless of how little. There was a consistent 3-5 second decrease when running in parallel with two threads compared to when ran serially. After increasing the thread count past two the time increased exponentially.

Conclusion

After reviewing the data from runs on both platforms, to achieve the top performance the optimal thread count is two threads and should be ran on a local machine rather than a virtualized machine. I suspect four threads might also result in better performance if executed on a different machine, despite the results documented here. There are several reasons why performance can decrease when implementing a large quantity of threads. When a fixed quantity of work is divided among too many threads, each thread receives so little work that the overhead associated with initiating and stopping threads overwhelms the productive work. It also results in overhead due to the way they compete for limited hardware resources. Finally, if there is a mutex it can be incredibly CPU intensive for multiple threads to lock and unlock it. Therefore, for the best results keep the thread count low and on a local machine.