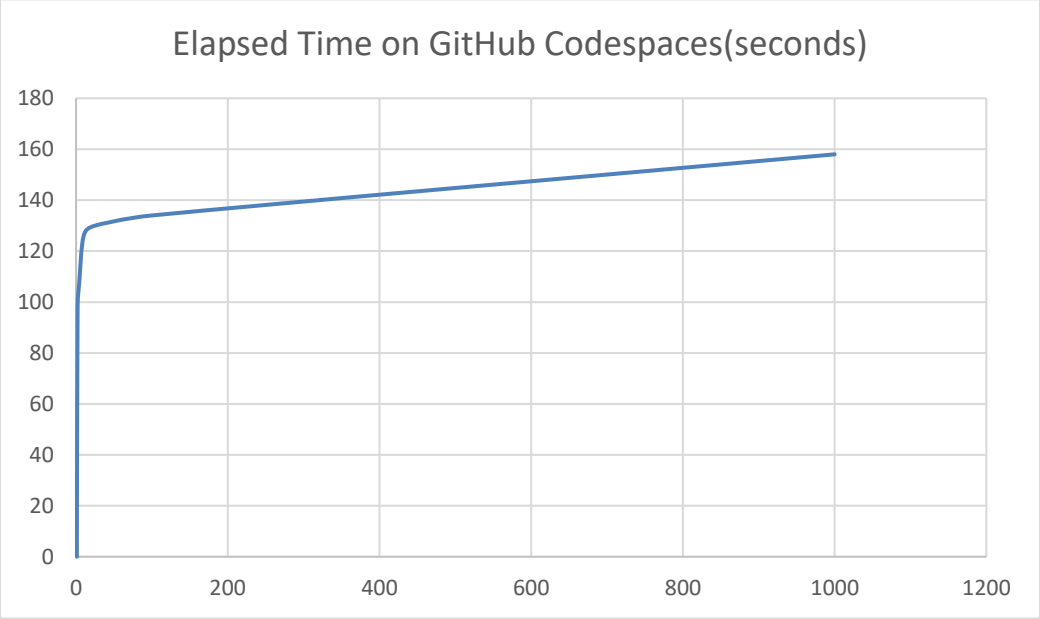
Inshaad Merchant

Star Catalog Multithreading Report

I had to create a multithreaded C program for this project in order to figure out the ideal thread count for computing the average angular distance between 30,000 stars in the Tycho Star Catalogue. When executed serially, the unthreaded application consumes a large amount of processing time. I used the pthread.h package to build support for POSIX threads to decrease this time and speed up the program's execution. This library enables the software to manage several, overlapping work flows of various types. While employing several threads, a mutex was designed to safeguard resources, guarantee consistent data, and carry out accurate calculations. 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.

I kept track of the amount of time each thread consumed while the program was running to calculate the ideal number of threads. I utilized the time.h header file and clock() function to implement timing. Before creating the threads and again after merging them, I used the clock() method. I then subtracted the start clock from the end clock and then divided by CLOCKS_PER_SEC to get my total time. Because I wanted the user-experienced time to pass rather than the CPU time, I picked this timing approach over others.

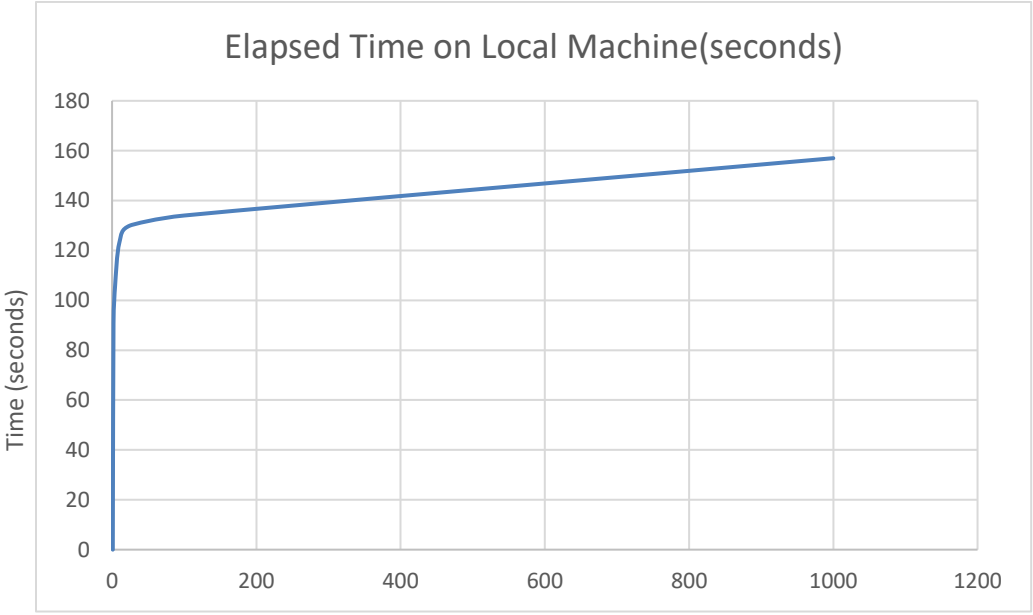
GitHub Codespaces Results



Thread Count	Elapsed Time (seconds)
1	0
2	98
4	107
10	126
25	130
100	134
1000	158

The software was initially run on Codespaces, a cloud-based development environment that employs a container to provide you access to popular programming tools, languages, and utilities. While employing 2-4 threads, an increase in elapsed time was anticipated, and after the thread count approached 4, a minor increase from thread count 4 to 100 but after 100, there was a steep increase seen in the elapsed time after I implemented it with thread count 1000. Yet, regardless of the number of threads utilized, the duration rose even though the computations were accurate and there were no deadlocks or race situations. This is a result of the cloud service provider's non-deterministic performance. The performance of the pooled computer resources varies.

Local Machine Results



Thread Count	Elapsed Time (seconds)
1	0
2	91
4	105
10	123
25	130
100	134
1000	157

After seeing the results on Codespaces. I tested the software on my local machine on VS Code on my Windows laptop with an i7 processor. This time, since computing resources weren't shared, the outcomes were more in line with the predicted conclusion, which is that having more than one thread will increase execution time, no matter how slightly or rapidly. When two threads were operating simultaneously as opposed to when they were serially, there was a constant increase of 5-15 seconds. The time rose exponentially when the number of threads was raised to two but at the end when I found out the time with thread count 1000, it was almost the same as that tested on Codespaces.

Conclusion

After examining the data from runs on both platforms, it has been determined that all threads are ideal number to use except 1 and either local system, or a virtualized one, could be used to run the program since both produced approximately same results. Performance may suffer when several threads are implemented for a variety of reasons. Each thread receives so little work when a fixed amount of labor is split across too many threads that the overhead of starting and terminating threads outweighs the useful work. Due to the way they vie for scarce hardware resources, it also causes overhead. Last but not least, locking and unlocking a mutex when used by several threads may be extremely CPU-intensive. Thus, keep the thread count low and use a local computer for the best performance.