Multithreading Programming Assignment

Arvin Saju

1001705173

CSE 3320: Operating Systems

Professor Bakker

Introduction/Overview:

The goal of this programming assignment was to practice the application of using multithreading to speed up a program that was provided to us while still having the same results from the program. The program to be multithreaded was written to calculate the average angular distance between different stars based on a data file that was being analyzed that contained a trimmed list of 30,000 stars to keep runtimes more practical compared to if the entire database was used. Due to the number of calculations that have to completed, this code takes a rather long time to run when serially. To resolve this problem, the goal of this assignment was to alter the program, so it was capable of having a user entered number of threads to run different calculations simultaneously allowing the program to ideally finish execution significantly faster

Methodology:

In order to make this possible, first some of the different libraries that had to be included in addition to the originally included libraries were the time and pthread libraries. The time library was included so that I could have a way of determining the runtime of the program. The timing method used was to create a two variables that could hold the start and end clocks of the programs. The start clock of the program was stored slightly before the individual threads were created in the program, while the end clock of the program was stored shortly before the end of the main function. To calculate the elapsed runtime, the two were subtracted and then divided by the number of clocks per second constant from the time library. The reason this method was used is because it is quite accurate while also being simple to implement. As for the pthread library, it was included to allow the multithreading of the program to actually be done, as multithreading required functions like pthread_create, pthread_join, as well as the mutex controls that are all defined in the pthread library.

Data:

Threads	Time	Minimum Distance	Average Distance	Maximum Distance
1	47.27326	0.000225	31.904232	179.56972
2	45.87103	0.000225	31.075228	179.56792
4	46.21073	0.000225	37.315709	179.56792
10	46.51395	0.000225	27.787416	179.56972
25	46.13802	0.000225	29.204036	179.56972
100	46.59526	0.000225	28.756486	179.56972
1000	47.12385	0.000225	32.020915	179.56972

Figure 1. Multithreaded Star Catalog Calculations and Runtime

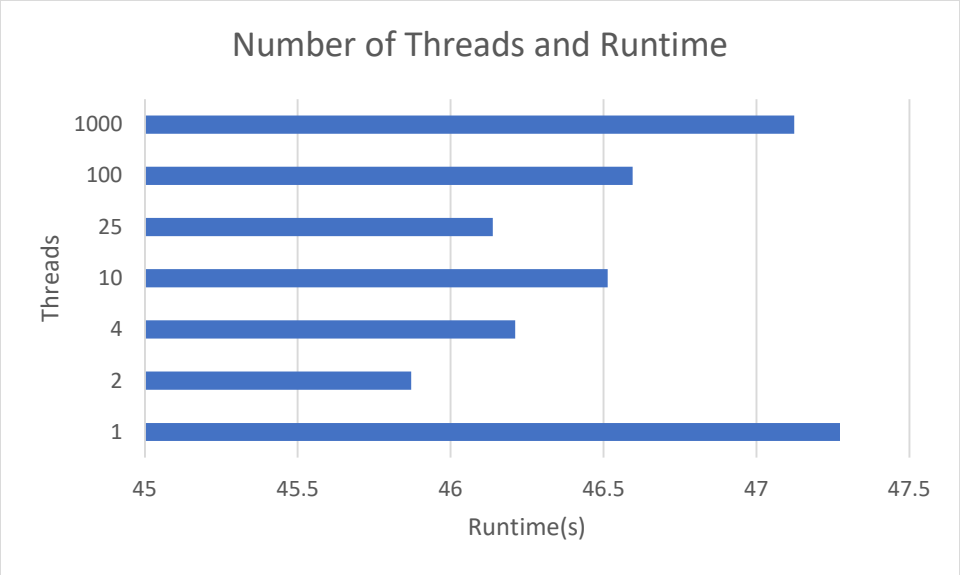

Figure 2. Graph of Time Comparison for Different Number of Threads

Discussion of Results:

The data table and graph provided above show the data that was obtained from this program. From this data it can be noted that while the range of differences in runtime was smaller than expected, the number of threads being run does seem to affect how long the program takes to run. Ideally, on a local system, as you increase the number of threads, the time should continually decrease up to a certain number of threads based on the system the program is being run on. However due to the codespace being a remote system that is limited to 2 cores, the best performance seems to be at 2 threads. As the number of threads increase beyond 2, the time once again begins to increase although it still remains faster than a single threaded program up to a 1000 threads. This is due to the fact that codespaces is remote and has to split resources between all the different organizations and users that need to run programs, so the results are not what you would obtain on a local system where times would decrease as threads increase. Another thing that can be noted from the data is that while the minimum and maximum values remain the same across all number of threads as expected, the average value changes based on the number of threads. Ideally the average value should be the same regardless of threads, however it seems that the calculation of averages is slightly off in some places, however this issue was unable to be resolved due to the sheer size of the dataset that needs to be computed and limiting factors of time especially since every change to the code that needs to be compiled can take almost an entire minute to run. However, in spite of this, it can be seen that multithreading is a valuable approach to speed up data analysis and other functions especially when dealing with large sets of data.

Conclusion:

To summarize, multithreading can help process tasks that can often be time consuming in a more efficient manner. The optimal number of threads to run depends on the system itself and the number of cores it has. In most systems, although there are exceptions, the fastest result should be obtained when there are the same number of threads as there are cores in the system. This will vary from system to system, and in the case of Codespaces, the number of cores available was two cores and the fastest runtime in calculating the average angular distance of the stars was indeed when there were two threads. While having more than one thread will usually be faster as it can run functions simultaneously, if there are too many threads the performance will also suffer as shown from the data gathered from the testing of this program.