Deep Shinglot CSE-3320-003

Star Catalog Multithreading Report

Overview:

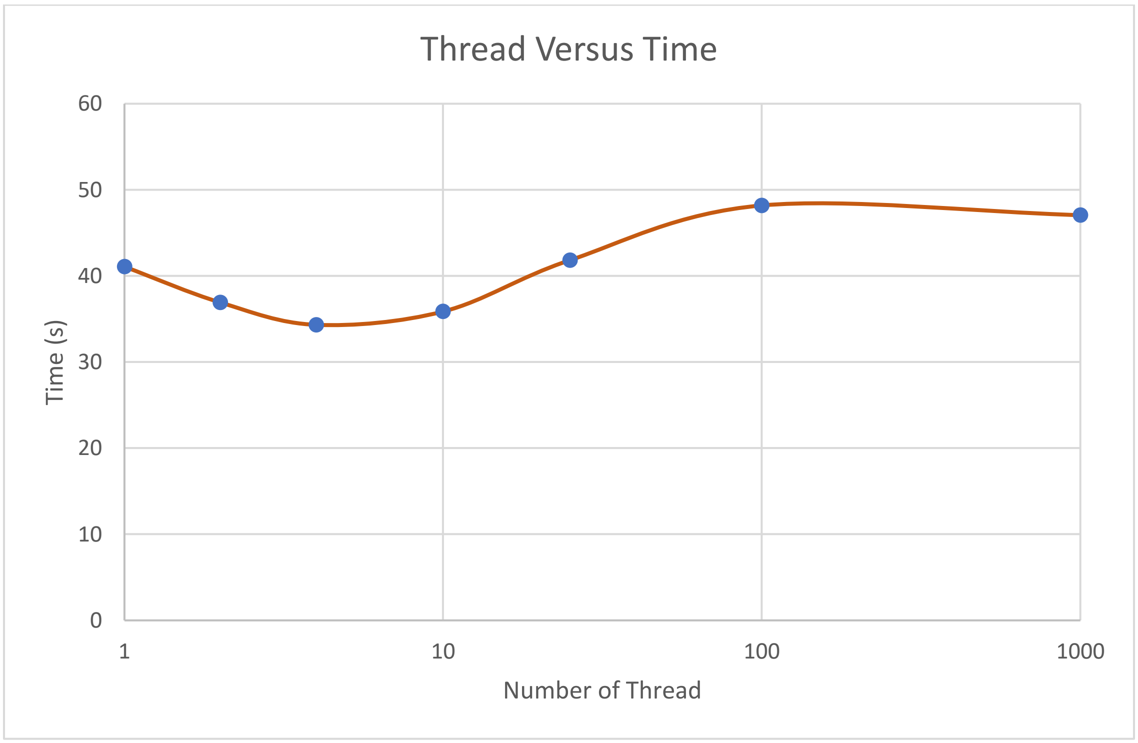
The assignment sets a challenge of calculating the average angular distance between 30,000 stars in the Tycho Star Catalogue with efficient time. With the provided code a significant amount of time is taken, as it is single threaded. For improving time of code execution, multithreading is needed to be implemented for time improvement. Multithreading is multitasking feature which allows the system to execute multiple programs concurrently. One of the multithreading is thread, each thread is a signal sequential flow of control within a program running at same time. This execution of continuous processes conflicts with shared data as threads race to get the priority. The scenario is called race condition. To counter race condition any shared data should be executing threads atomically with consideration of deadlock which is threads approaching lock simultaneously. For proving data consistency, data needs to be recorded using multiple number of threads which are 1, 2, 4, 25, 100, and 1000.

Description:

In the assignment, threads are implemented using POSIX Threads. Leading to adding "pthread" header file, as it holds thread functions. Moreover, the library also provided mutex functions which are necessary for the code for a race condition. Mutex locks and unlock protect data by allowing one thread at a time for accessing shared memory. For measuring improvement of time, preferred to use Input/Output time which gives the elapsed time of a number of threads created. This includes implementing "sys/time.h" to be able to use "gettimeofday" function. The function is implemented before creating multiple threads and after joining which measures the time took by all threads throughout. One of the reasons for picking I/O time would be user experience of latency. All the implementation resulted in giving consistent average, minimum, and maximum distance meaning processed correctly while multithreading.

Results:

Thread Count	Time (s)
1	41.074387
2	36.916604
4	34.322283
10	35.865186
25	41.825447
100	48.184082
1000	47.055113



Discussion:

The increment of time after four thread implementation refers to the Codespace limitation. Since Codespace only provides two cores to process, it comes really inefficient to run more than 4 thread optimized considering a thread per core. Moreover, more than four threads involve switching which consumes more time. Respect to the data, everything comes out to be consistent for each run with any threads. Concluding no deadlock or run condition occurrence ever.

Conclusion:

In conclusion, a way to get most optimized time on Codespace would be by assigning four threads to the execution. By facts and data that is because Codespace only holds 2 cores so to avoid too much switching threads and speed up less or equal to that thread number is best. Otherwise, each thread would get significantly less work assigned that kill productivity. The division of data for four threads seem to work preferably with Codespace. However, on local it seems to be 1000 threads for most optimized time. This is because some more access and hardware resources available to locals.