Report : Star Catalog Multithreading

The goal of this assignment was to improve the efficiency and find the optimal thread count for the best performance of a program that calculates the average angular distance between 30,000 stars in the Tycho Star Catalogue. The program initially ran on a single thread (unthreaded) and ran serially. Hence, it took a significant amount of time to run.

The additional libraries/header files that I included in my implementation were “pthread.h” and “sys/time.h”. I used “pthread.h” to support POSIX threads. POSIX threads, through multithreading and correct implementation, can provide improved performance (in terms of effective resource usage, and CPU usage), and reduce the synchronization overhead, which is done with the help of mutexes (mutual exclusions), condition variables, etc. They help in the coordination of access to shares resources between threads. I also referred to the code sample thread_param.c provided by Professor Bakker in the course repository, to create and join threads. To divide the work among threads, I divided the large array into smaller chunks, resulting in less work for each thread.

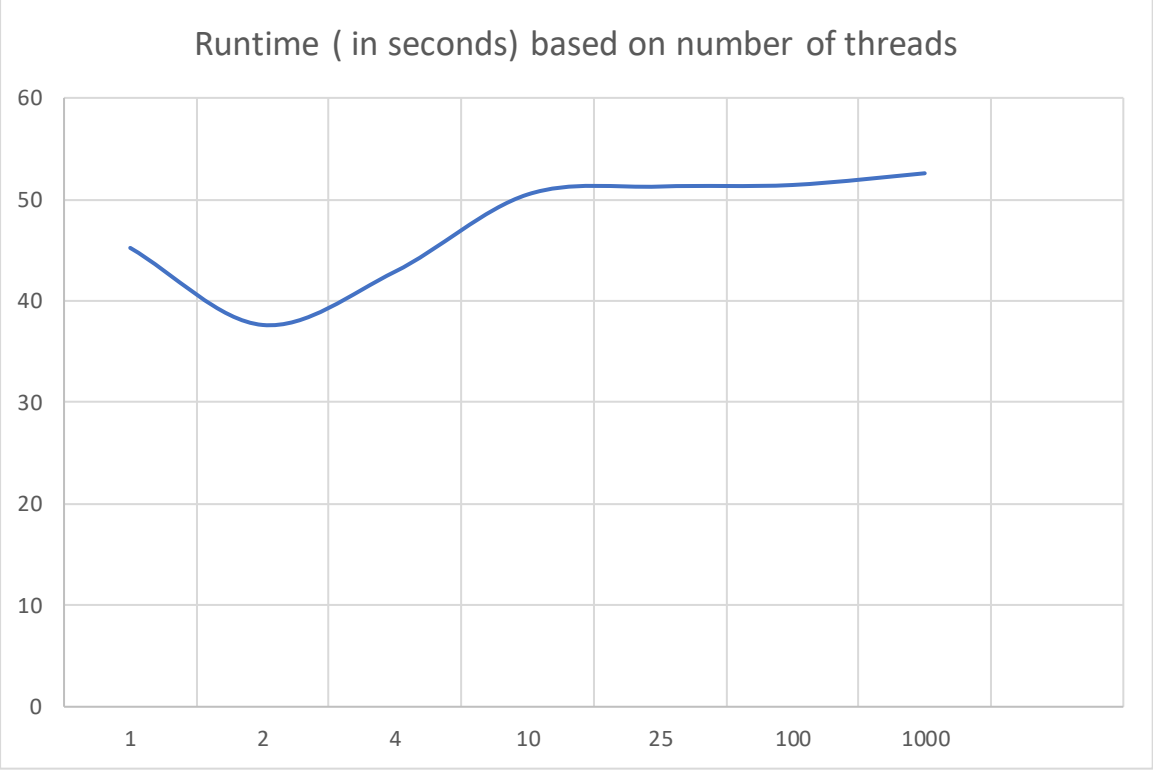
I used “sys/time.h” for the accurate measurement of the runtime of my code. With the help of the function gettimeofday() [with reference to code sample “time.c”, provided by Professor Bakker in the course repository], used before and after the calculation of the average angular distance for each thread, I was able to accurately measure the runtime of the code with varied number of threads. I used “sys/time.h” over “time.h” as the results provided by the former are more precise and are better for optimizing performance in time-critical applications. And additionally, functions like clock(), provide the CPU’s runtime, which could be different when compared to the runtime from the user’s point of view.

My code was run on GitHub Codespaces, a cloud-based development environment that allows developers to create and manage fully featured development environments directly from within GitHub, by creating a cloud-hosted development environment for a repository. So due to shared computing resources used by GitHub Codespaces, the performance on

Codespaces, could contain inconsistencies. This is due to non-deterministic performance on the cloud provider. My results upon execution of the code are as follows:

Number of Threads	Execution Time (in seconds)
1 (unmodified)	45.217611
2	37.622672
4	42.878023
10	50.474338
25	51.261311
100	51.412819
1000	52.569813

The graph containing these values is below:



Conclusion:

Based the results as shown above, the optimal thread count to achieve the highest level of performance from this code is **2**. Apart from the unmodified code, it was observed that the runtime increased as the number of threads being used increased. This could be because: when the workload is distributed among too many threads, each thread will receive a very small amount of work which leads to the overhead associated with starting and stopping threads becoming greater than the actual productive work, ultimately becoming inefficient. It also results in overhead due to the way they compete for limited hardware resources. Therefore, for best results, it is better to have a lower thread count.