Project 2: Multi-threaded Collatz Stopping Time Generator

Ethan Tran, Daniel Eckman
COP4634
10/19/2025

Description

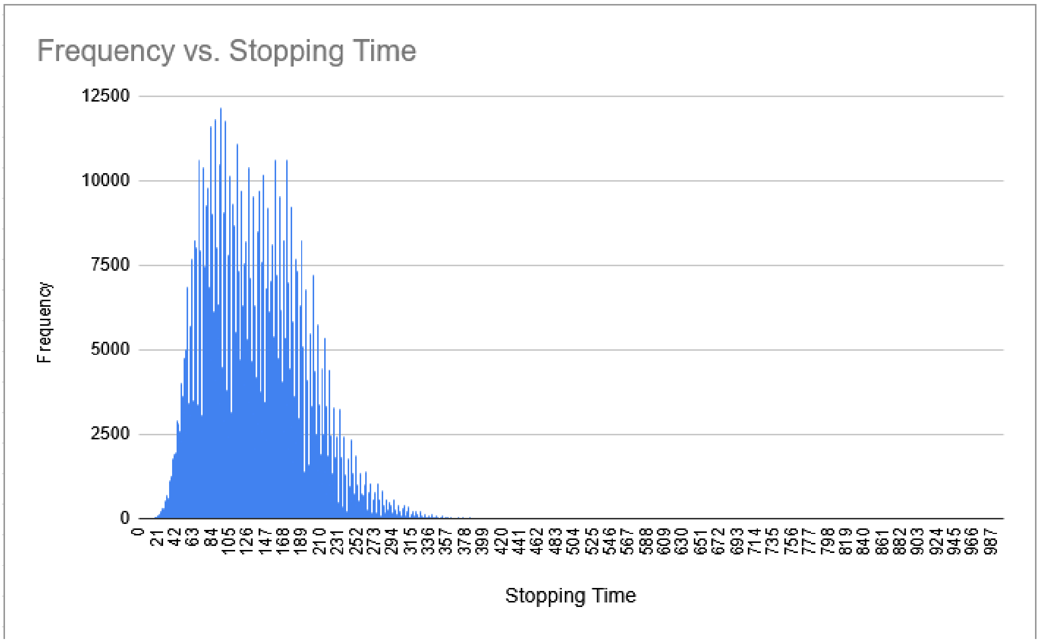
In this experiment, we created a program in C++ that calculates a range of Collatz sequence stopping times for a given input, done optionally on multiple threads. These threads could also optionally be allowed to work without a mutual exclusion lock, such that race conditions would be allowed. The data gathered using the finished program was collected on a computer with an AMD Ryzen 7 7800X3D processor, which has 8 cores and a clock speed of 4.20 GHz, along with 32 GB of installed RAM.

Experiment

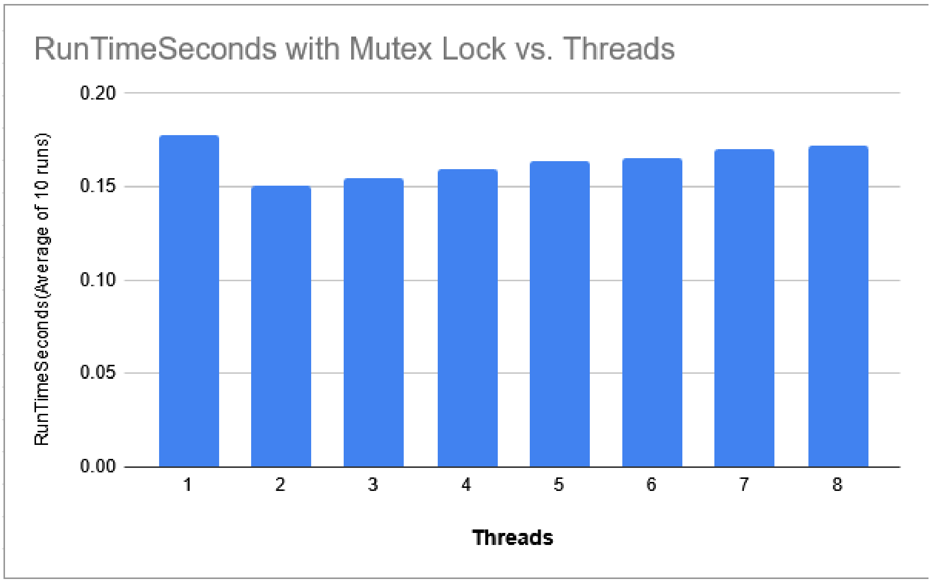
The mt-collatz program was run with the following command once compiled: "**./mt-collatz 1000000 4 -nolock > histogram.csv 2> runtime.csv**" to collect the data on each Collatz sequence's stopping time.

Results

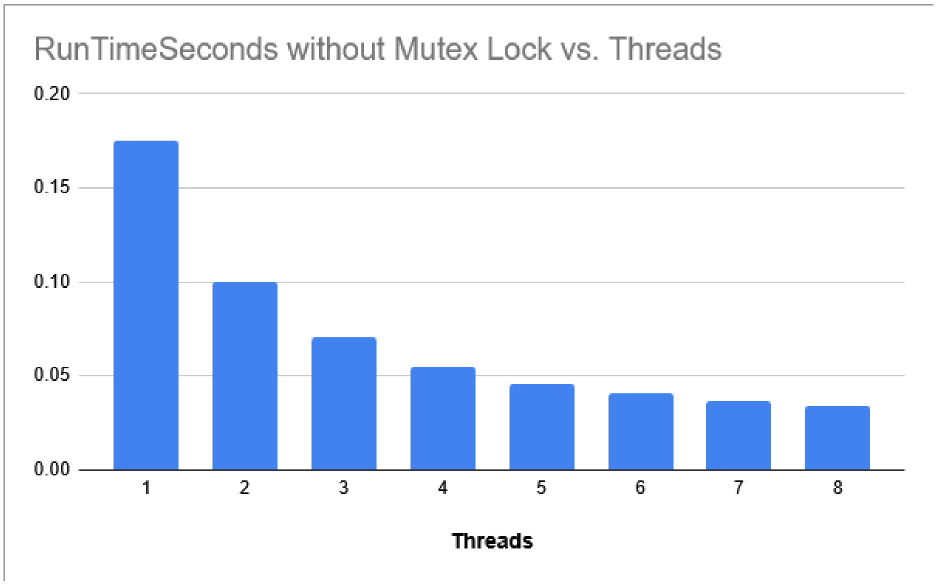
Below is a chart of the accrued data, where x-axis represents the stopping time from 1 to 1,000, and the y-axis represents how many Collatz sequences in the range 1 to 1,000,000 have the stopping time on the x-axis.



These next two charts represent the runtime of the program compared to the amount of threads it was instructed to use, separated by whether or not race conditions were allowed.
Race conditions disallowed:



Race conditions allowed:



Analysis

When using a mutual exclusion lock, the program ran slower as the thread count tended toward the maximum of 8, with the notable exception of thread count 1 to 2, where the runtime went down considerably between them. Without the mutual exclusion lock, the program ran faster as the thread count increased. This implies that using more threads in a safe way (with mutual exclusion locks) is not always the fastest or most efficient way to solve a problem, but not using mutual exclusion locks with multiple threads can lead to marginally inconsistent results, as a tradeoff for speed.

Conclusions

This experiment has demonstrated the tradeoff between performance and thread safety that comes about with multi-threaded programs. As such, using a mutual exclusion lock led to worse performance but higher type safety due to the lock's added overhead, while not using a lock led to higher performance but possible inconsistency in experiment results.