Multi-Threaded Collatz Stopping Time Generator

Introduction

This project explored the use of threads to solve the collatz stopping time problem for number in a range from 1 to N. N is the desired range of elements in an array that you want. For our case or N range is going to be 100,000. The goal is to compute the stopping times with using different number of threads, compare the difference, and see the impact of use of multiple threads; and use of the mutex locks to avoid race conditions.

Experiment Setup

The program was tested on my home laptop that has an 11th gen Intel® Core ™ i7-11800H with 2.30 Hz. It has 8 cores, 16 threads, and 16 GB of RAM. The program was run with the "-nolock" option to measure the run time. It was run about 5 times each to minimize disruption of other processes. The collatz stopping times were calculated for thread counts of 1, 2, 3, 4, 5, 6, 7, 8.

The expected output of the values of the histogram should be formatted by stopping time or number of element in the array, Frequency.

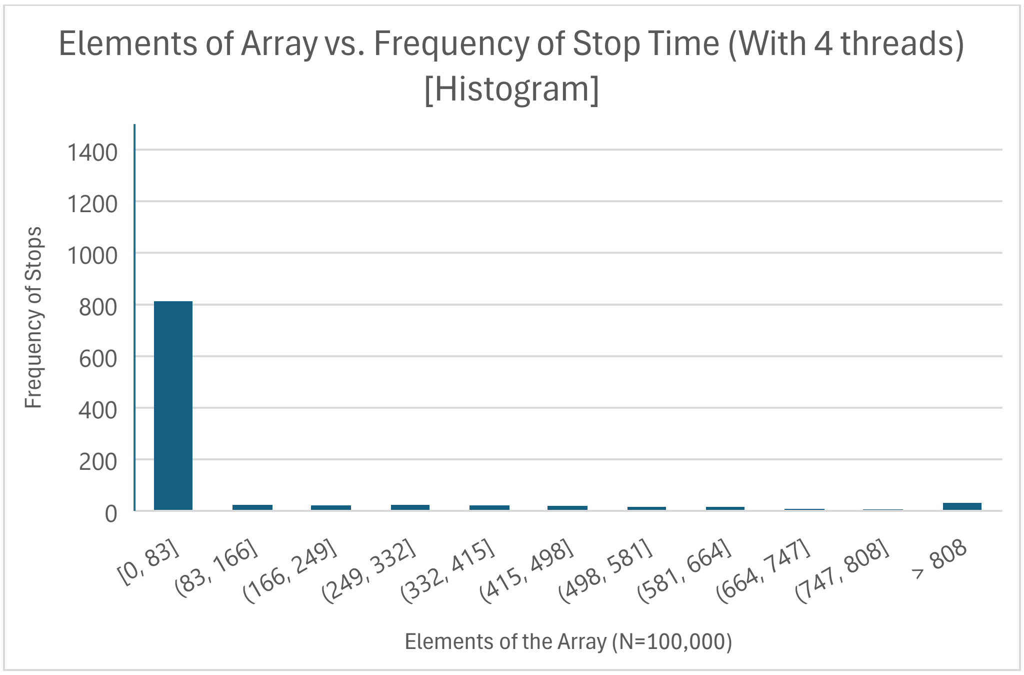

Figure 1.

Figure 1 shows the frequency of stops as the process goes through the elements of array at 4 threads. It shows that earlier into the array is subjected to have more stops then preferred the latter end of the array.

Runtime Results

We measured the runtime for computing the Collatz stopping times with varying thread counts and synchronization options. The following compares the averages of stopping times with different number of thread mutex functionality across 100,000.

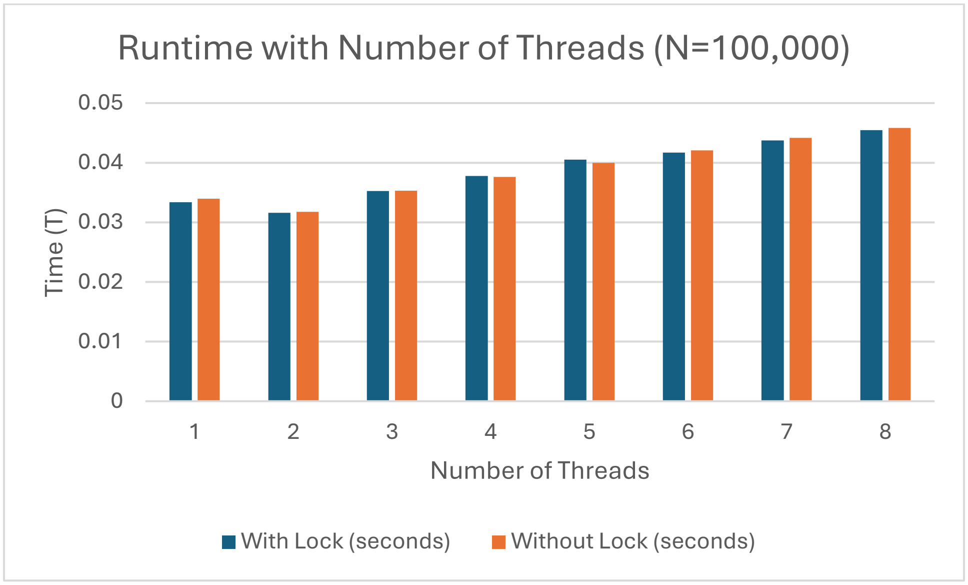

Figure 2.

Figure 2 shows the average of seconds each thread took to go through all the elements with the stopping frequency. It shows that the more threads that took to go through the processes took more time compared to the lower number of threads. It also showed that the race condition we implemented lead to more time but more accurate depictions of the histogram.