Collatz Report

COP4634 Team 8

Project 2 - Using pthreads

Benjamin Minor & Carson Wilber

24 October 2018

Introduction

The collatz problem analyzes a specific function of n:

$$a_i = \begin{cases} n & \text{for } i = 0 \\ f(a_{i-1}) & \text{for } i > 0 \end{cases} \quad f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod 2 \\ 3n+1 & \text{if } n \equiv 1 \pmod 2. \end{cases}$$

To solve this problem for some bound N, parallel processing can be used that incrementally calculates the collatz stopping time for each value n = 2, …, N. What follows is an analysis of implementing, testing, and analyzing the results of a parallel processing solution to identify trends and the general effectiveness of using multiple process-level threads.

This project does not take into account memoization or other optimization methods for speeding up stopping time calculations.

Implementation

Collatz is implemented with an iterative loop starting with a = n and calculating the following sequence value appropriately while a > 1. This prevents a stack overflow due to function callback pointers as mentioned under “Challenges” in the original solution.

Testing

Testing consisted of using N = 400,000 and running with threads in increments of 5, starting with 1 then incrementing after T = 5. The timing outputs are taken for each of four runs and then averaged.

Collatz stopping times were separately calculated for N = 4,000,000 to more finely demonstrate the effects of locking versus non-locking runs.

Challenges

One prominent challenge arose during the implementation of mt-collatz.

Collatz was originally implemented as a recursive function, returning the stopping time as a function of itself while a > 1. This resulted in a stack overflow as each thread occupied a large portion of the stack with function callbacks. When the collatz stopping time could be up to 111, for example, even for the value n = 27, the stack is quickly filled and causes a critical failure.

Therefore, the implementation of collatz was changed to an iterative function.

One prominent challenge arose during the testing of mt-collatz.

It must be noted that on the UWF SSH server, users are limited to 32 processes. The user session starts 2 threads; the execution of mt-collatz starts 1; and all process threads are treated as kernel threads, restricting mt-collatz to starting only up to 29 additional threads.

Therefore, testing could not be conducted with T = 30 or more.

Notes

As a result of the challenge listed above, the values provided in Figs. 1 & 3 below for T = 30 threads are approximated using T = 29.

Results

Run Times

N = 400,000 (locking)	Run 1	Run 2	Run 3	Run 4	Avg
Threads: 1	17.9211	17.5084	17.8199	17.7349	17.746075
Threads: 5	18.3969	18.3571	18.921	18.4999	18.543725
Threads: 10	18.7386	18.5951	17.2275	17.7261	18.071825
Threads: 15	14.612	12.4224	14.1084	12.431	13.39345
Threads: 20	18.0402	16.9311	16.1152	18.5009	17.39685
Threads: 25	11.3207	13.1464	14.2034	16.0983	13.6922
Threads: 30	10.4515	11.7327	13.076	11.8434	11.7759

Fig 1. Run times and averages for mt-collatz with locking and N = 400,000.

N = 400,000 (-nolock)	Run 1	Run 2	Run 3	Run 4	Avg
Threads: 1	16.5858	15.9542	16.08	15.0658	15.92145
Threads: 5	16.4541	17.2004	16.4738	17.6069	16.9338
Threads: 10	16.7474	16.4156	16.6903	16.695	16.637075
Threads: 15	17.6068	17.6514	17.3332	17.2363	17.456925
Threads: 20	16.6413	17.0635	16.1757	17.1561	16.75915
Threads: 25	19.5608	16.1112	16.5762	16.7637	17.252975

Threads: 30	16.7873	14.5336	16.9211	14.1769	15.604725

Fig 2. Run times and averages for mt-collatz with -nolock flag and N = 400,000.

Stopping Times

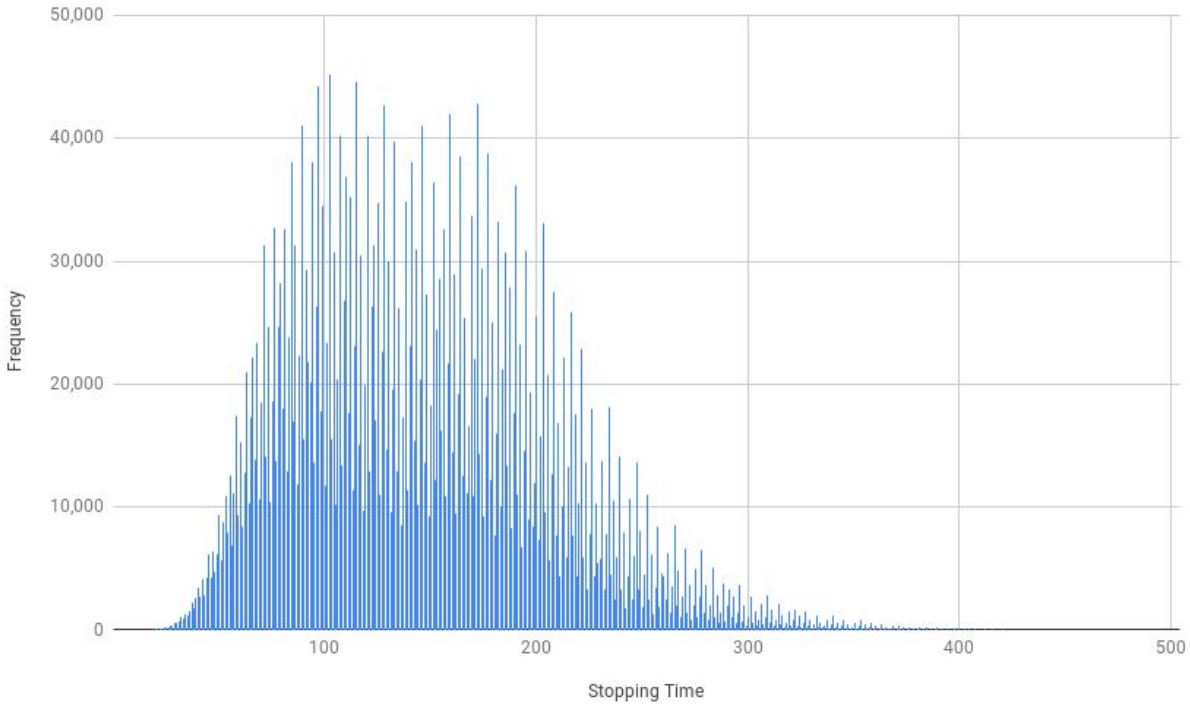

Fig 3. Stopping times calculated during a run with locking. No frequency exceeds 50,000. A regular pattern appears at different scales across the chart. The frequency peaks at 45,202 at Stopping Time = 102. The sum of all frequencies is 3,999,999.

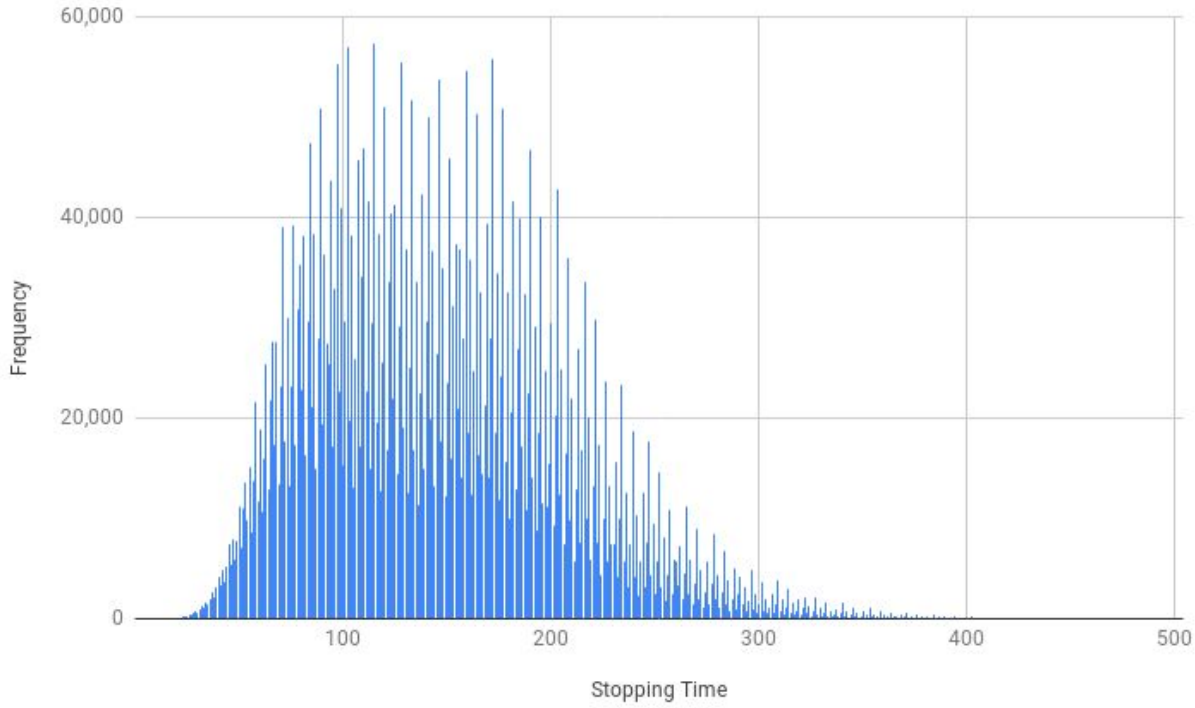

Fig 4. Example of stopping times calculated during a run with -nolock. Multiple frequencies exceed 50,000. Regular patterns from Fig 3 are now irregular, for example, around Stopping Time = 110. The frequency peaks at 57,295 at Stopping Time = 115. The sum of all frequencies is 4,944,287.

Analysis

The sum of the frequencies for either case should be equal to 3,999,999 when N = 4,000,000, as the collatz stopping times are calculated once for all integers n = 2, …, N.

From the sum of frequencies when the -nolock flag is provided, and from the difference between Fig. 3 and Fig. 4, it can be calculated that the non-locking test duplicates collatz calculations for 944,288 values in n = 2, …, N. Some values may have been calculated up to 8 times, as each thread has the opportunity to calculate each value of n once, and there are 8 threads.

Conclusion

Without a mutex lock preventing duplicate calculations of collatz, the multiple threads perform a large number of unnecessary calculations and incorrectly update the histogram data. This results in a more homogenous set of results, but an overall slowdown for a higher number of threads when the -nolock flag is set as more time is needed to complete the extra calculations.

With threading enabled, the results vary more, but the more efficient combinations of parameters are significantly faster, on the order of 3-6 seconds.

Looking more specifically when locking is enabled, it appears that the most efficient values of T are 15, 25, and 30 for the given values tested with times in the 11.8 to 13.7 range. With a higher number of threads, there may be an eventual trend upward in time as the context switching and mutex locking needed for safe parallel processing consume too much time for significant returns.