Project 2

Amitabh Mishra

10/13/2024

Elijah Reyna and Ryan Wadd

The Experiment

In this experiment, we tested how increasing thread counts in a program influenced the time it took to concurrently compute Collatz Sequence stopping times. We computed the stopping times of values 0-6678923, with thread counts of 1-8. The data was collected from running the code on a computer with 16 Gb RAM, and an Intel Core i7-8750H CPU @ 2.20GHz, which was 6 cores and 12 threads, and an 384KB L1 cache, 1.5MB L2 cache, and 9.0MB L3 cache.

The following is a histogram of the stopping times of the Collatz sequences of values 0-6678923. For each sequence, a stopping time was calculated, and the graph reflects how many of each stopping time was found in the range of N values specified.

Results

The following line chart shows the computation time for each number of threads, for both locking and nonlocking threads.

Based on the data we got, the time taken to compute the Collatz sequences concurrently decreases with additional threads, up to a point. It seems the locks in the locking threads increase the amount of time taken to compute after 4 threads, which makes sense given the locking and unlocking that must occur to ensure data integrity. This locking forces threads to wait, and at a certain point it creates a bottleneck, no longer gaining the benefit of multiple threads. The optimal amount for our program was 4 threads.

The nonlocking threads don’t ensure the integrity of the data, leading to duplicate or missing entries into the histogram. However, since there is no locking done with those threads, the bottleneck that the locking threads experienced is nonexistent, and increasing threads continues to drive down the computation time.

Conclusion

Using threads is a great way to reduce the amount of time a program takes to compute a solution. However, if they access shared data, the race problem needs to be solved. This will increase computation time and will force programmers to find the optimal number of threads to run computations concurrently while ensuring data integrity.