Program 1 Report

Will Townsend

October 21, 2022

Overview

Our objective for this program was to implement the Monte Carlo algorithm to estimate π using multithreading. I was successful in implementing multithreading to the algorithm using Python's threading library, and it did not take too long to accomplish. The results show a speed up of the program and an increase in efficiency of the algorithm.

Speedups and Efficiencies

Since the algorithm deals with random data all the calculated speedups and efficiencies were calculated by the average seconds of 10 different runs of each number of tosses and threads. This means the output of these will be different each time but I will calculate the speedup and efficiencies using the first average serial time calculated by the program along with each respective average parallel time calculated. If we look at the 3 tables, the data is skewed due to some limitations of my personal device and other factors. However, I played around with the program (increasing problem sizes, iterations for the average times, and number of threads) and came to the conclusion that the given problem with my implementation is weakly scalable: efficiency stays fixed as problem size increases along with number of threads.

10,000 tosses Avg Serial Runtime (sec): 0.06543

Threads	2	4	8
Avg Parallel Runtime (sec)	0.01965	0.02880	0.02696
Speedup	3.32977	2.27187	2.42692
Efficiency	1.66488	0.56797	0.30336

1,000,000 tosses Avg Serial Runtime (sec): 4.19469

Threads	2	4	8
Avg Parallel Runtime (sec)	2.04672	1.69721	1.31609
Speedup	2.04946	2.47152	3.18724
Efficiency	1.02473	0.61788	0.39840

10,000,000 tosses Avg Serial Runtime (sec): 42.73507

Threads	2	4	8
Avg Parallel Runtime (sec)	22.50747	22.08526	20.83265
Speedup	1.89871	1.93500	2.05135
Efficiency	0.94935	0.48375	0.25642