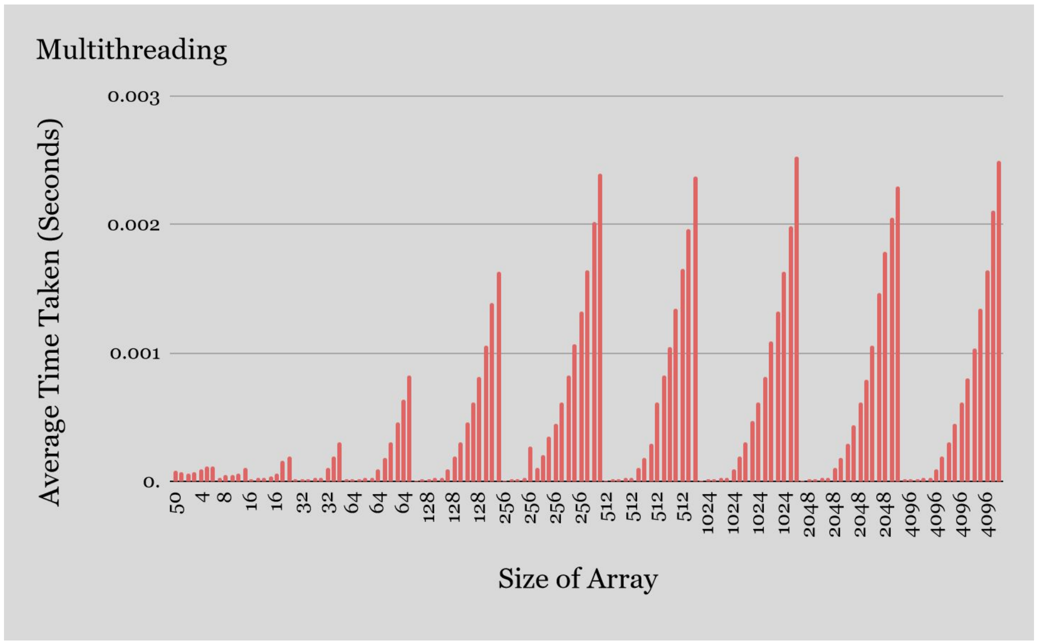
CS 214 Systems Programming
Assignment 2: Spooky Searching
Rithvik Aleshetty: rra76
Steven Nguyen: shn27

Abstract

Project done to test an iterative search of an array of integers with various lengths up to 4096. The test plan is detailed in the testplan.txt file included with this submission. Findings are detailed in this report as well as answers to the questions asked.

Data & Findings

Figure 1: Multithreading Average Time vs Data Size



CS 214 Systems Programming
Assignment 2: Spooky Searching
Rithvik Aleshetty: rra76
Steven Nguyen: shn27

Figure 2: Multiprocessing Average Time vs Data Size

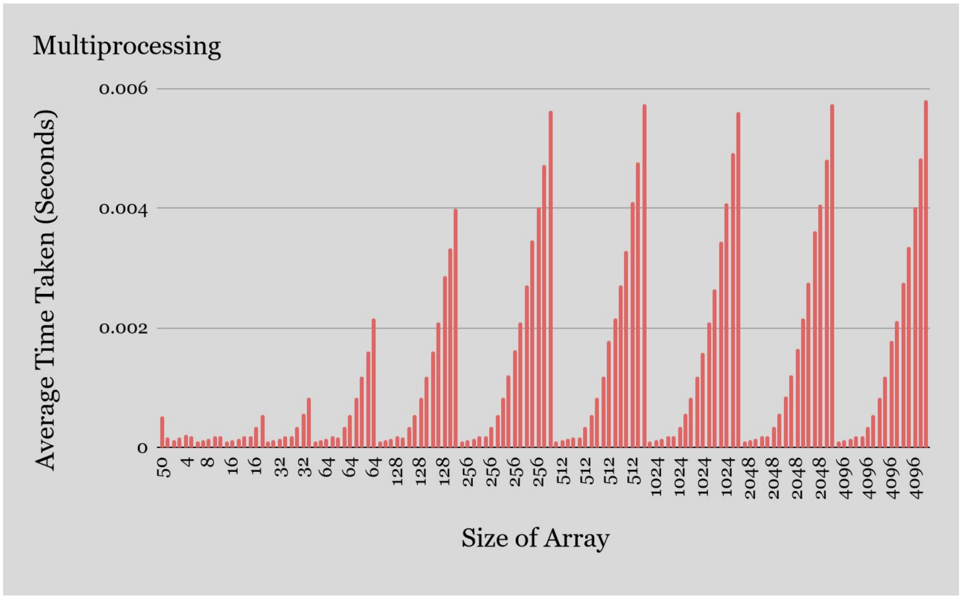


Figure 3: Table of Raw Data

Conclusions/Responses

The test data verifies that threads searching is faster than searching with processes. This can be seen by comparing figure 1 and figure 2. The differences are especially seen in array sizes of greater than 64. The range of the y-axis is also higher in figure 2 in order to account for the increased timings required of multiprocessing.

There is a trend of increasing times with increasing data size although this increase seems to plateau after 256 elements. After array size is increased above 256 elements, there was not any significant difference in the time. This situation was consistent in both threads and processes.

Figure 1 and Figure 2 also show the timings when the number of processes or threads are increased. This is seen in the multiple red bars representing the increased process or thread count. The charts indicate a positive correlation between the number of processes/threads and the time of completion. The more processes or threads there are, the larger the times are.

CS 214 Systems Programming
Assignment 2: Spooky Searching
Rithvik Aleshetty: rra76
Steven Nguyen: shn27

The tradeoff point found for processes vs threads was 9 threads ≈ 1 process. For all of the array sizes above 16, the tradeoff was: the time taken for 9 threads were approximately equal to the time taken for 1 process.

The tradeoff point for parallelism is that splitting the work over more processes and threads is always slower than having fewer threads/processes and this is consistent with Figure 1 and 2.