Ahmed Malabi

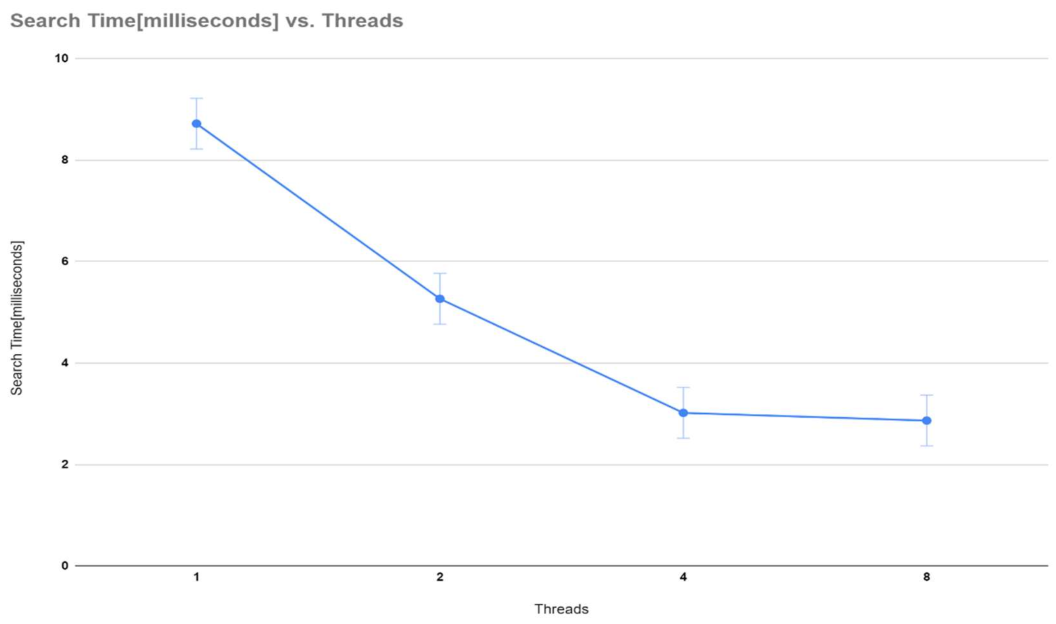
Threading HW Report

Part 1:

1. The problem we are trying to solve is taking the substring.c code provided to us and somehow make it faster in parallel using threading. We tested this by timing the function num_substring() with 1, 2, 4 and 8 threads.
2. The threading library I chose was pthreads. I chose this library as I was already familiar with it from CSE 1325 and this class.
3. The experiment designed for this test was to take one massive string via hamlet.txt and Shakespeare.txt and divvy up sections of the string, then take a smaller string and count how many times it appears in the text. We were making this version of substring.c to test if/how threads can speed up processes.
4.

test #	1 thread	2 threads	4 threads	8 threads
1	8.691	5.285	2.975	2.849
2	8.453	5.723	2.698	2.961
3	8.701	6.25	3.147	2.991
4	9.325	5.141	2.788	2.771
5	8.554	5.103	2.847	2.994
6	8.774	5.034	2.707	2.824
7	8.69	5.662	2.899	2.868
8	8.613	5.137	2.773	2.856
9	8.81	5.03	2.854	3.034
10	8.598	5.231	3.431	2.86
11	8.659	5.045	2.767	1.903
12	9.091	5.045	2.87	2.898
13	8.474	5.008	2.927	2.905
14	9.643	5.019	4.913	2.947
15	8.543	5.158	3.191	3.021
16	8.842	5.095	2.943	3.006
17	8.688	5.37	2.998	1.897
18	8.663	5.241	2.848	2.745
19	8.723	5.303	2.76	3.105
20	8.603	5.756	3.024	2.774
21	8.585	5.201	3.392	3.015
22	7.798	5.02	3.099	3.058
23	8.684	5.29	2.895	3.23
24	8.684	5.033	2.92	3.172
25	9.173	5.526	2.792	2.98
avg	8.72248	5.26824	3.01832	2.86656

Ahmed Malabi



5. After running the program with Shakespeare.txt searching for Romeo using 1, 2, 4 and 8 threads my data shows that doubling the threads leads to diminishing returns. For 1 -> 2 and 2 -> 4 the average time went down by about 40% but, from 4 -> 8 the average time only went down by about 10%. This shows that while adding more threads can speed up time (some runs the times overlapped) after a certain number of threads there is little to no gain.

Part 2:

1. The problem we were solving in part 2 was learning about conditional variables and running two separate functions at the same time. We experimented with this by creating a program that reads characters from a file into a buffer in one function and then sequentially prints them out from the buffer in another.
2. The threading library I chose was pthreads. I chose this library as I was already familiar with it from CSE 1325 and this class.
3. Using a circular queue, I decided the best solution was to allow the producer to fill up to 5 spots at a time keeping track of a tail pointer. I then had my consumer print out up to 5 characters at a time keeping track of a head pointer. These 2 pointers were separate allowing sequential reads and writes to happen without overlap.
4. N/A
5. In conclusion the producer was able to load the buffer without overwriting what the consumer had not printed out yet. While making this program I ran in to a few problems that took a little thinking to solve. One of these problems being to break out of the infinite loop at the right time. For the producer that was easy enough just when reach the end of the file, but for the producer I was running into a problem where it would not finish printing before it exited. To solve this, I had to make sure the consumer did not break until the head pointer was pointing to the same spot as the tail pointer. All in all, I felt this assignment was a good entry into multi-threaded programming and conditional variables.