Jacob Huckelberry

Section B1

10 December 2021

Dr. Matthews

Unoptimized	t = 1	t = 2	t = 4	t = 8
Run 1	9.509	5.088	2.769	2.076
Run 2	9.499	5.079	2.766	2.085
Run 3	9.509	5.077	2.766	2.089
Average	9.506	5.081	2.767	2.083

Optimized (-O3)	t = 1	t = 2	t = 4	t = 8
Run 1	3.759	2.088	1.231	0.998
Run 2	3.755	2.089	1.223	0.995
Run 3	3.763	2.092	1.225	0.996
Average	3.759	2.090	1.226	0.996

Speedup	t = 2	t = 4	t = 8
Unoptimized	1.871	3.435	4.56
Optimized	1.799	3.066	3.774

General Strategy:

When implementing my parallel solution, I went through several designs. My original design that ran in about seven seconds included parallel memory allocation and two parallel strtoking functions. This implementation had several issues and a lot of moving parts, so I needed to try something new. After coming across the example in section 14.6 of the book, I knew that design was the most effective way to solve this problem. I studied how the example used strtok in parallel to split up the string and implemented a very similar design to split mine up. The most challenging part of this implementation was figuring out where to place the start and end for each thread. Solving this issue took upwards of six hours. I solved this issue through trial and error with print statements and running my program on the test.txt file. After getting the correct output, I had an epiphany on how I could make my solution much quicker. I was originally placing each thread's token in a local array and looping through it with a mutex to add them to the hash table. I then realized that I could make a local hash table and then add the elements of the hash table to the global hash table using the same process. This would take the total number of loops done from 67 million down to about 800. This took my runtime with eight threads from 6 seconds to 2.08 seconds.

Benchmarking Results:

My benchmarking results showed continuous speed up on both the O3 optimized and unoptimized versions. I would venture to say that this is probably about as fast as this program can possibly run. The two things that took the most time in the serial version were strtoking the file and adding the states to the hashtable. I don't see how my parallel strtok solution could get much faster, and the hashtable function is from a library that runs at constant time. I believe I provided a solution to everything that would have caused the program to significantly slow down.

Impact and Personal Reflection:

From this project, I learned that parallel computing is essential when tracking network probes. I would imagine that there are records much larger than 67 million lines of network probes and to run those records serially would be incredibly time consuming. I would not say that I observed any limitations presented by pthreads when implementing this program. You cannot write to the same space using pthreads, but I would imagine this would be the case for any other threading library for any other language. With that being said, other higher-level languages likely deal with this issue under the hood so the programmer doesn't have to deal with it. I struggled with literally everything when doing this project; it was easily the most challenging and stressful assignment I have ever had. I have never spent over a week and twenty plus hours trying to figure out how to solve a problem. However, I did learn a lot through this struggle and am now incredibly confident in my ability to use C. The most valuable things I learned through all this struggle were the finer details of C. I learned how to use structs, as they came in very handy when implementing my parallel solution. I gained a far deeper understanding for how to use pthreads through trial and error. Overall the struggle was worth it because I gained a deep understanding for how to use C and seeing the program finally work was incredibly satisfying.