Jason Gallagher

Steven Kravitsky

**PThread Trapezoidal Integration**

This assignment was parallelizing trapezoidal integration using the PThread library. PThreads are used to parallelize chunks of the intergration using a user specified number of threads. It also allowed us to use the already made computation function in parallel, since each thread was just doing the same computation but with a smaller chunk of the graph. The code was fairly straightforward to parallelize, with the serial code only needing to be slightly changed with global variables and PThread initiation.

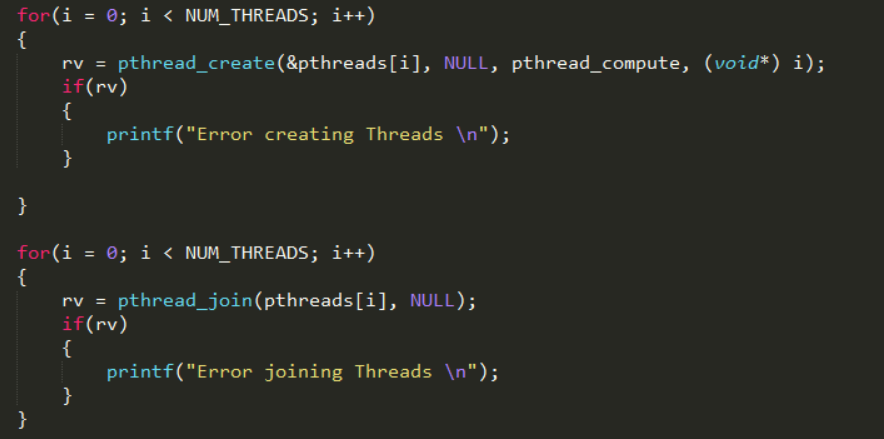


Figure : The first PThreads are initiated to run the pthread\_compute function then joined as they are finished

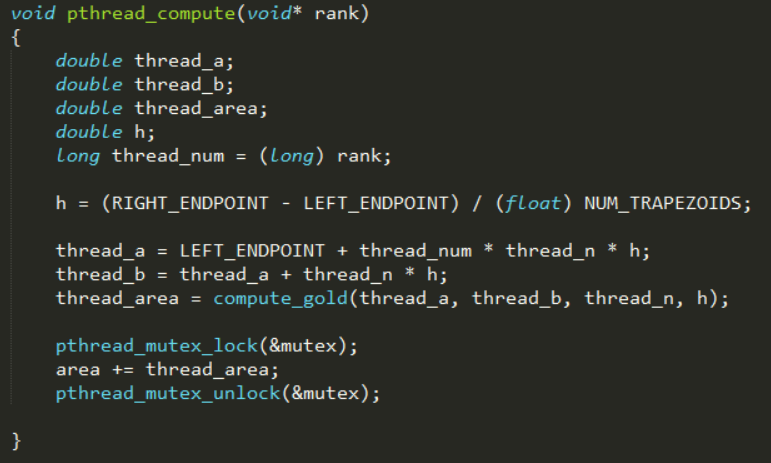


Figure : The pthread\_compute function using global variables to chunk the data and run the compute function

In Figure 1 it can be seen that the threads are being created and they are running the pthread\_compute function. Once the threads are finished they are being joined to remove the threads, since the addition is being done in the function. In Figure 2 it can be seen that the threads are each using global variables to figure out which chunk of the data. “RIGHT\_ENDPOINT”, “LEFT\_ENDPOINT”, “NUM\_TRAPEZOIDS”, “thread\_n”, and “area” are all global variables that were initialized at the start. The threads are figuring out which chunk of the data they are computing then running the compute\_gold function. A mutex is used to make sure that the sum is not being calculated incorrectly because of threads using a wrong area value.

The speedup seen in Figure 3 shows about the results that we were expecting to see when this test was parallelized. As the number of cores used went up the time needed to complete the parallel part of the script went down. At 2 cores the speed went speedup was 1.9x, at 4 cores the speed up was 3.8x, at 8 cores the speedup was 6.9x, and at 16 cores the speedup was 12x. The speedups can be seen in figure 4. This is about what we were expecting when making this code parallel. It is extremely suited to being parallelized since each thread is basically working with completely different data. The only time that there is an issue is when everything is being summed up, but a single mutex was able to fix that with minimal time losses.

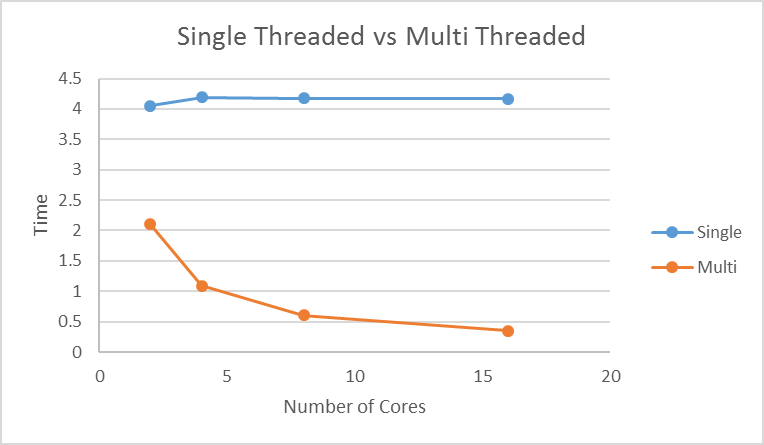


Figure 3: Single vs Multithreaded

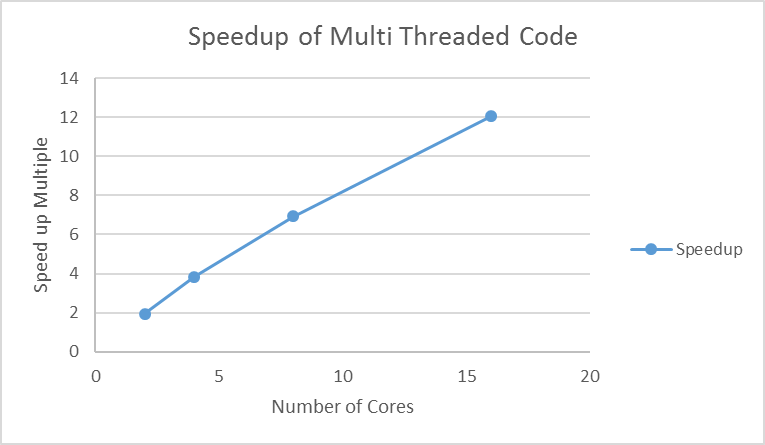


Figure 4: Speed-up