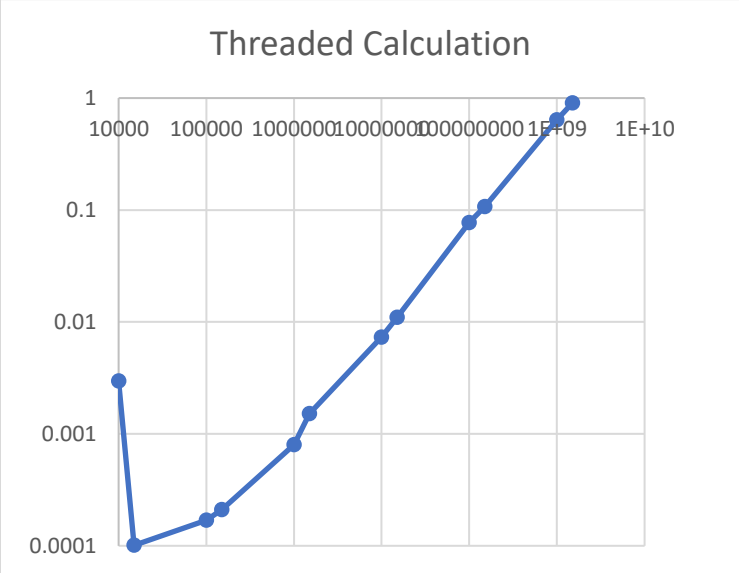
David Vondran

3/31/2022

Hw4

For this assignment, I was able to accomplish the tasks very well, especially considering that the task given was very similar in nature to that of homework 3, but with significantly simpler implementation of the code. Utilizing pragma OMP, I was able to introduce threading in a significantly simpler manner than the multi-step process that was required for std threads to be implemented. For part 1, working on the std thread part of the assignment, I implemented 3 total parallel sections. One included finding the mean and the min and max, the second was utilized to find the std calculation, and the final was used in the threshold method to find the value c for building the threshold array. I could have found a way to implement threading on the final loop in this section, however I ran into difficulty on this part. For all of the parallel regions I implemented, I used P threads so that this number of threads could be changed depending on the desires of the user without having to change any hardcoded options. For the std method, I was able to get my threaded solution down to around .6 seconds, while the serial solution remained around 2.7-3 seconds. This is a significant decrease in the amount of time, and leads to significant savings over larger iterations. Given the graph below, we are able to see the compute time given exponentially increasing values of N. We can see that it follows an exponential pattern given that both of the axes are logarithmically scaled. For the threshold method, I was able to get the execution time down to around .9 seconds, while the serial version of this method took around 1.7 seconds. This is a significant decrease in execution time as well, cutting time almost in half. Unlike the previous homework, there is not much visible difference in compute time between the different levels of cache. This is likely because of the specific values chosen to iterate through, and the scale of the graph making it difficult to represent effectively.



As for part 2, we can see a similar story unfolding to part 1, except without the logarithmically scaled y axis. This is because the values of the y axis are within close enough proximity to be all shown effectively in a linear fashion. For this part of the assignment, I included one parallel region encompassing the entire code chunk, and a parallel for within the section encompassing the most outer for loop. I attempted to include further parallelization on the inner loops, however this was not allowed by the OMP package. However, even with only the outmost loop parallelized, I was able to get the standard run with N=2000 down from 18 seconds on the serial run to around 4 seconds on the parallel run. This was accomplished with P=10 processors. Below is shown a graph showing the run time of this algorithm with increasing values of N and the execution time. It grows exponentially as well with no

distinguishable cutoffs between the cache values.