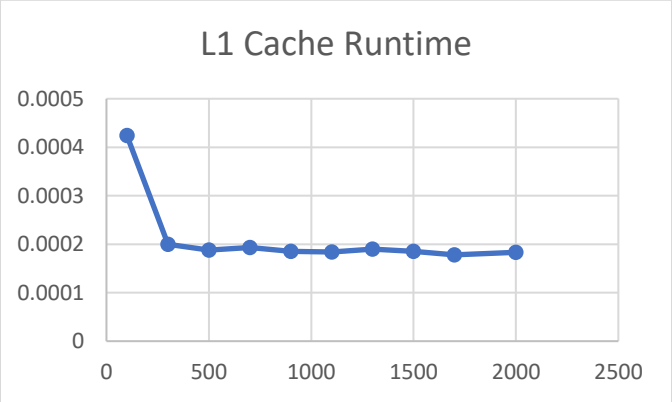
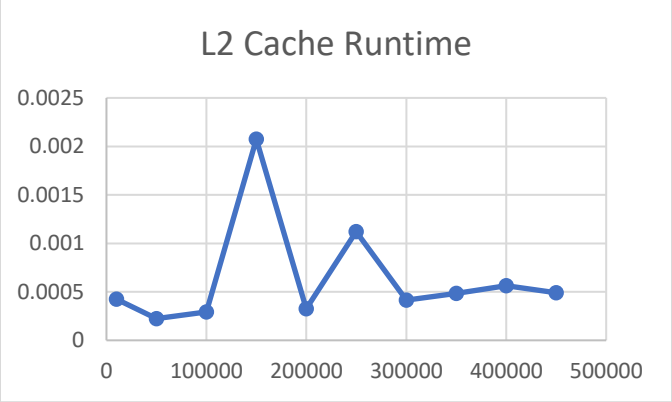
David Vondran

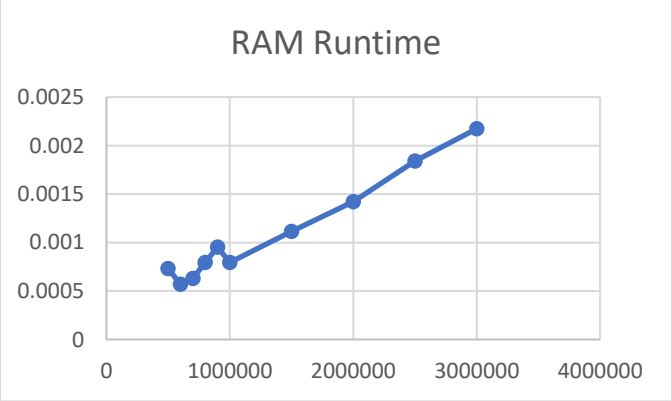
3/21/2022

Hw3

For this assignment, I was able to accomplish the tasks very well, with expected results replicated by the data shown. I ran the implementation of the code on the standard ACI node of the cluster, which contains an Intel Xeon 2.8GHz processor. This processor utilizes L1 cache of 2x16 Kb and L2 cache of 2x2 Mb, meaning L1 cache values will hold 2000 8-bit values and L2 will hold 500000 8-bit values. This is the reason for my testing values. For part 1, working on the std thread part of the assignment, I implemented an object which would contain information regarding start and stop positions, result of the section, and the min and max values of the section. One section included finding the mean and the min and max for each of the allotted iteration values defined in the thread object, 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. For each of these section, the final value was calculated through summation of the individual sections values, of by comparing the values to find the global overall values. 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 3 graphs shown below, we can see the algorithm runtime growth for the standard deviation. We can see that for the level 1 cache values, the runtimes are generally the fastest, hovering around .0002 seconds of execution time. This is because the level 1 cache has the fastest access values, meaning that the memory storage is able to be handled far quicker than it would for the larger values of N. For the level 2 cache things become interesting with the runtime values having spikes and acting very sporadically. This can likely be attributed to the overhead cost interacting strangely with the size of N and returning inconsistent results regarding runtime. However, it is worthwhile to note that the y-axis scale is very small and thus there is not too much inconsistency in the values. The RAM values grow generally in a linear form, as can be expected, because as the problem size increases the execution time will increase along with it, as the final state of memory access has been reached. It is likely that this behavior will continue with further increasing values of N.

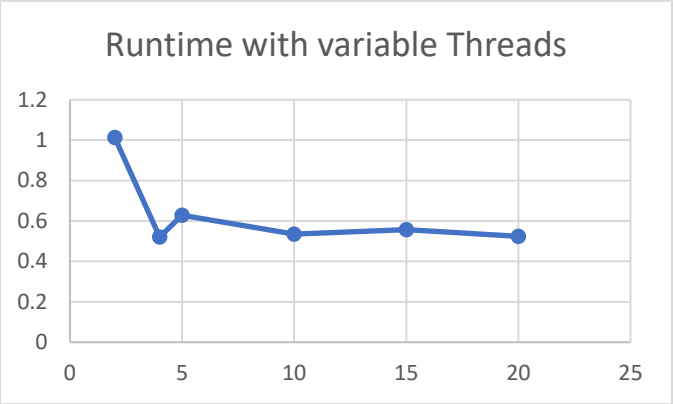




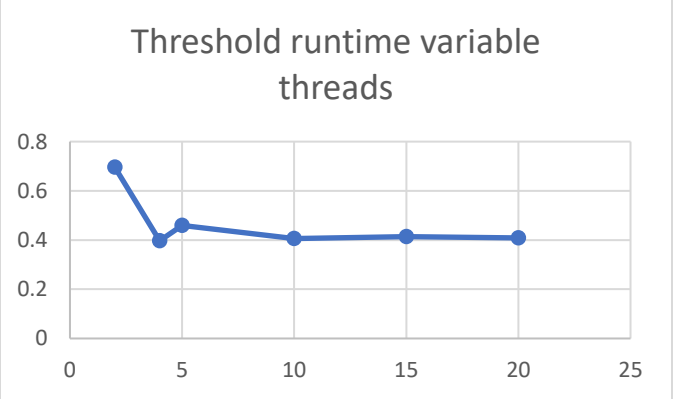


Beyond this, I was able to test the STD method with a variable amount of threads ranging from 2-20. These were chosen because number of cores (20) in the given processor. This was executed on a set N value of 1000000000. We are able to see that generally speaking, as thread count increased, runtime decreased. However, between 10 and 20 the benefits of more threads begin to rapidly fall off and the runtime

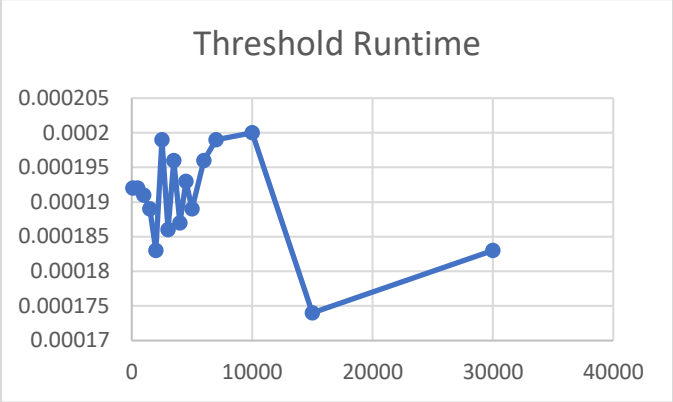
levels off. It is likely that with further increases in number of threads, this value will remain level or even decrease as the combination of threads will have increasing effect on the overall runtime.



As for the threshold method, we were able to see a generally similar runtime chart with the best values coming from more threads, with a larger jump at the beginning. It is likely that with further increases in number of threads, this behavior will continue to be shown, and the values will remain level as the overhead of combining the threads counteracts any computational benefit gained. This was executed on a set N value of 1000000000.



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. For the threshold method, we are able to see that the values of the runtime act very sporadically at the lower values of N. This is likely because of the similar issues observed with the std level 2 cache section, which shows that the relation between time saves by threading and lost to thread combining is very unique to the given size of the problem. However, it is shown that for significantly larger values of N, savings are made as the runtime drops off substantially.



I unfortunately was not able to get part 2 of this assignment working correctly, and thus will not submit anything for this section of the assignment.