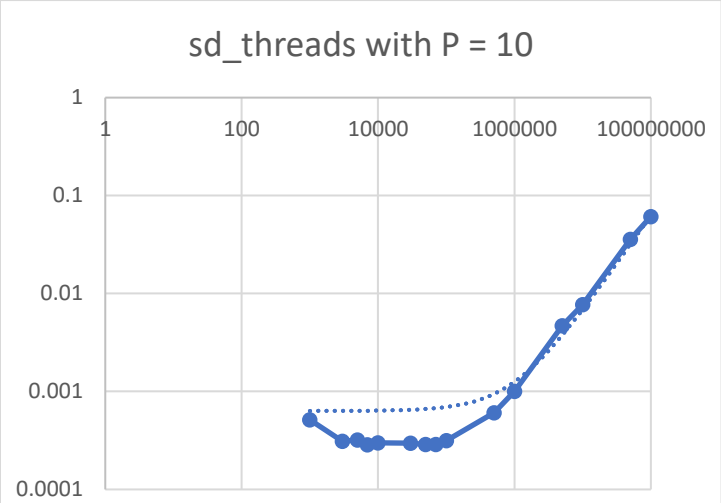
David Vondran

3/21/2022

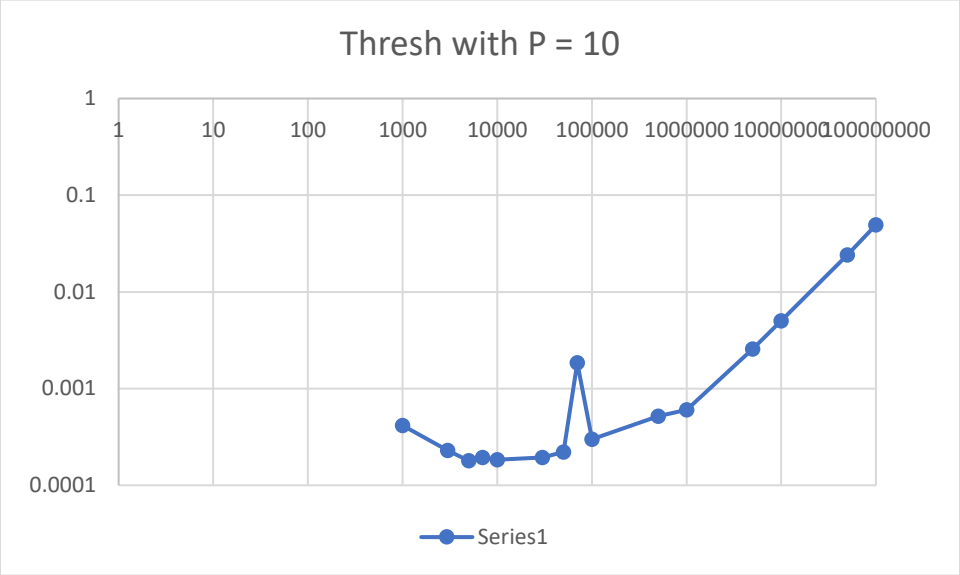
Hw3

For this assignment, I ran into some issues regarding time, with many other assignments and exams occupying the same time space as this project. For part 1 of this assignment, I was able to thread the given code by utilizing a struct to maintain the various relative variables that I need, and broke the loop into sections of size N/P, with different values being able to be made based on the values of N and P. This allows for the concurrent execution of the different parts of the loops, meaning that a significant amount of time is saved. I additionally find the current max and min of each section of the inputs in this function, which helps to save time in that the values are not required to be iterated through a second time for this purpose. Shown below is a graph showing the time it takes to execute the calc_threads method with different values of N, all with the number of processors set to 10. I have scaled the graph along both axis to show the logarithm of the values, as it makes it more readable and easier to understand. As we can see, for around N = 3000 to N = 100000, the time it takes to execute the method remains generally similar. This is because the additional time which is noticed later comes from memory access bottlenecks rather than actual computational bottlenecks. After making use of all of the processor's cache, the program is forced to increasingly rely on RAM, which increasingly has longer access time. This leads to the significant and accelerating growth of the runtime of the method. After running the threaded vs serial versions of the methods, we were able to see that the time generally dropped from around 3.5 to 4 seconds in the serial version to around .6 seconds in the threaded version. This is a very significant decrease in runtime. As the number values of N continue to grow, we would expect that the runtime would continue to grow linearly, as the memory bottleneck is that prevalent issue and likely cannot be more optimized.



For the threshold calculation method, we were able to accomplish similar reduction in overall time when compared to the serial method by performing threading on the loop. This allows for the program to iterate through many sections of the code at the same time instead of having to go through on one thread, saving time. The time was dropped from around d 2.7 to 3 seconds in the serial version to around .5 seconds in the threaded version. With the threshold calculation method, we see a very similar story, in that the runtime remains generally the same from around N = 6000 to N = 100000. This is again because of the effective use of the cache by the program, which

increasingly becomes more congested and strained as the values continue to increase.



Unfortunately due to time constraints, I was not able to effectively complete part 2 of the assignment, which required for me to thread the matrix multiplication calculation. However, I had started to setup the code, which I had included in the submission files. Given that there are 3 sets of loops in this code, it is likely that we can parallelize multiple sections of the code within one another in this section, which would have exponential decreases on the amount of time. For this section, we would expect for the threaded optimized code to do the best when compared to all the other versions of the method, as this is the one that has been reduced most effectively and is being executed concurrently with itself. This would be expected to have the most effect on the time. However, we would expect for this to scale linearly once the cache bottleneck has been reached, as it is still subject to the same memory issues as the other versions of the code.