HW #3 Report

John Yearsley (id# 1440547)

1.

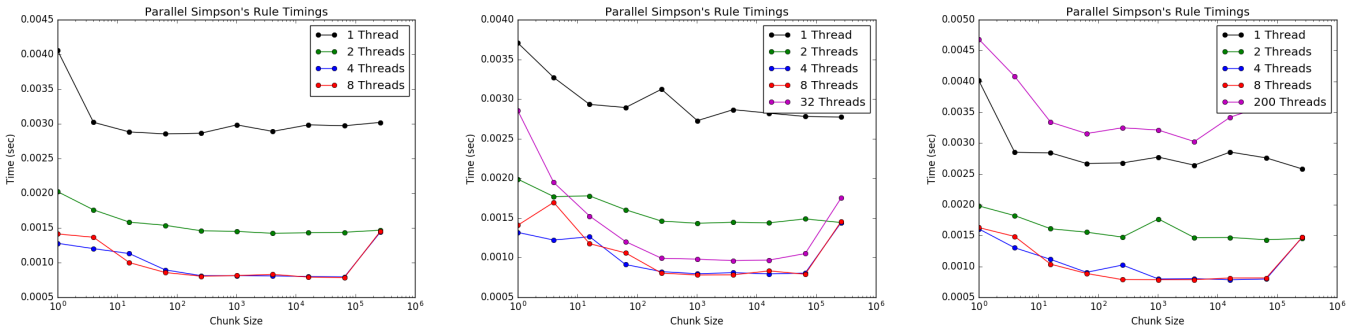

Figure 1: Plots for various tests including the addition of 32 threads as well as 200 threads.

For the analysis of my plot, it should be noted that I used my own 4 core machine.

- It was interesting to see that there was not much of a dip in performance when using 8 threads. The same held true for 16 threads and 32 threads. In fact, it took up to about 200 threads to start to notice an obvious decrease in performance. However it can be seen in the middle plot shown (the one including 32 threads), that 8 threads perform better than 32 threads. This suggests that the system is doing work behind the scenes and is hiding some of the latency via the CPU. In essence the scheduler is faking some of the parallelization, but nonetheless doing a good job!

- As can be seen in each plot, when the chunk size is equal to the problem size (number of points in the sample) 2,4,8 threads all have the same timing! This is not surprising, since if the chunk size is the problem size there is really no difference between 2,4, 8 threads (at least for my machine). The point of parallelization is to split the work between different threads. If the chunk size is the problem size there isn't much "splitting" of the work. However, we can see that the problem is still solved faster than in the case of 1 thread. This is likely due to the system's ability to schedule, i.e. there is less waiting time while memory is being retrieved.

- The chunk size in this problem is relevant since we are given contiguous data and pulling that data from memory to calculate the desired results. Because of this it is in our best interest for memory management to pull an optimal amount of chunks such that when one element is requested several nearest neighbors are sent to the cache. This is different than the in class example where we generated the relevant data within our code, in which case we are not pulling data from memory.

- A first simple guess determined from looking at my plots suggests the second to last chunk size, $N = 2^{18}$ (that's big!), is at least close to an optimal value. This guess stems from the fact that in each plot for each number of threads (excluding 200 threads), the timings are minimal. This is not an exact analysis, and without more experimentation in chunk size it is unclear if there is not a better value. It is important to bear in mind, however, that this optimal chunk size will be highly correlated to the capabilities of my own machine. As far as an optimal chunk size for N in general, the analysis will

be machine dependent. One cannot specify an exact optimal chunk size without knowing cache size, processor speed, etc.