Nolan Anderson

CPE212

Langford

19 April 2020

Project 06 Answers

**a.) Data:**

**a.)** I compared the data by making two graphs. Time is on the “x” axis and array size is on the “y” axis. When looking at the sequential vs parallel time differences, the graphs show that as the array size increases, the parallel implementation increases in it’s time differences. This is most likely because the sequential implementation takes the same route of solving every time, while the parallel will sometimes change up its pattern. It is more random, and therefore the time differences are more random. I split this into two groups, one where the data was limited to 50,000 array size so that you could better see that the parallel implementation is more random than the sequential implementation.

**b.) Explain why certain thresholds failed to execute after a certain point.**

Some thresholds fail to execute because they cannot handle the size of the array that needs to be sorted. If you give a computer a small threshold to work with, then it can only handle so much. If you send too much data for the CPU threads to compute, it will simply give up. Computers can only handle how much they were designed to handle. A small ARM system is only going to compute so much. If you give more wattage, cores, and threads, then the computer will be able to handle those larger arrays. Continually, the RAM of the computer is also limited, and if you overload it with operations then it similarly will fail. Also, many computers are designed with a fail-safe execution. For example, if a CPU is not applied with thermal paste, then it will thermal throttle and shut itself down to prevent further damage to itself.

**c.) explain why there are speedups at higher thresholds**

Given a higher threshold, the computer can allow more of the CPU to be used to execute a task. The higher threshold opens more threads on the CPU to allow for more executions to happen at the same time.

**d.) Explain why smaller thresholds are so slow.**

Smaller thresholds are very slow because it is limiting the whole potential of the CPU. You are only allowing for a certain amount of executions to happen at the same time, and since the execution relies on threads, then it will naturally run slower if there are not as many threads.

**e.) This process is running on a 4 thread ARM system. What type of performance gains would you expect with this running on a modern intel i7? What about a server similar to Blackhawk running on a 40+ Core Xeon?**

ARM systems are usually very limited on CPU cores, RAM speed and amount, and threading. What we are doing is a multithreaded process, as in, we are allowing the CPU to make multiple executions at once. If your CPU has only 4 threads, then it can only do so many processes at a time. The current 10th gen i7 processors, such as the 10750H are running with 6 cores and 12 threads. This allows for significantly more processes to happen at the same time. Further, looking at the Xeon processors (assume 80 threads due to manufactures usually doubling the core count for threads) we can see that a much larger amount of processes can occur at the same time. The more threads, the more processes that can happen at the same time.

**f.) What are some good things that may be done to improve performance further?**

The first thing you can do to improve performance is to find the highest performing algorithm. This will allow for the data to be sorted much quicker and puts less stress on the threads of the CPU. You could also give some of the processing load to the GPU instead, putting less stress on the CPU as well. Lastly, you could always optimize your code to use up memory. If it’s just sitting there, you might as well use it.