Isa Maguire
Parallel and Distributed Final Report (Primes)
5/9/23

First step to improving the runtime of prime computation was to introduce some basic multithreading. After getting the small primes up to ROOT_MAX, I created a threadpool to compute the rest of the primes. I chose ROOT_MAX for small primes because in the worst case (finding primes up to MAX_VALUE) you would only need to search for multiples up to the square root of that number. We don't need any primes past that for prime computation.

Next step was for me to write a callable that returned the indices of all primes within a certain range. I just used the basic for loop from Primes.primeBlock, with a bitset to represent the indices. Any time I came across a number that wasn't prime, I set the bitset to true. I used MAX_VALUE / 128 for the task size after experimenting with a few different values. After getting the indices of the prime numbers I went through each future, and iterated through every bit to get the prime number associated. This approach got me a runtime of about 2600ms on the hpc cluster.

Then I added a check in the for loop that calculates indices of the primes, to see if the current number in smallPrimes was greater than the square root of the max value in the range. The runtime didn't change at all, if anything it was a tiny bit worse on average.

Here I noticed that as the range of numbers increased, each task was getting longer and longer. This was because as the max value increased, you had to check more and more of smallPrimes. My attempt at offsetting this was creating staggered task sizes to even out the runtimes. I had a max and min task size as well as a function that calculated the next task size (multiply current task size by a multiplier) before passing it to the threadpool. I wasn't able to mathematically optimize this but after playing around with the values I was able to get my runtime down to about 2250. The process I used to find the best value was to change the base task size until I found the number of tasks with the fastest runtime, which was around 250. I then set max task size, min task size, and the multiplier so that the minimum task size would be reached by the last task or a little before. I had to keep tweaking these number to make sure I got the right number of tasks but after a few hours I settled with a max task size of MAX_VALUE / 128 and a min task size of MAX_VALUE / 384, with a multiplier of 0.993.

My next optimization was calculating the prime values in the same thread as the indices and instead of returning a bitset, I returned an array of integers. This slowed me back down to about 2700ms. I figured this was because each thread was now executing more tasks and taking longer. I shortened the task sizes but was only able to speed it up to about 2400ms. After this I went through the bitSet documentation again and found a way to only iterate over the true or false values in the bitset. I used this for finding the primes and got my runtime back to around 2200. I didn't know why it was going so slow here so I changed my task sizes back to uniform to see if the task sizes could be further optimized for this setup. I immediately saw a speedup, and

kept testing to try and find the ideal task size for this setup. After settling on ROOT_MAX * 25, I saw my runtime shoot to 1811ms.

The only thing I really could do after this was parallelize and vectorize prime writing. I created a new runnable task and passed each set of calculated primes into this task, which was then sent to the thread pool. I couldn't do this in the same thread as the prime calculations because I needed to get the index into primes where I should start writing. First I just set the task to write primes without vectors. I didn't see much of a change in runtimes with this so I tried vectorizing the prime writing, which also didn't result in any speedup. I kept trying but no matter how many times I ran it I couldn't replicate the 1811ms from the last improvement, so I got rid of parallel prime writing and vectors. I also learned that the operating system already vectorized the writing of primes automatically, so I decided to stop here for primes since there didn't seem to be anything else to do, plus I was already faster than the past first place.