COP4520 HW1
Elijah Smith

Design Summary

This program spawns eight threads to each compute a portion of the total 10^8 primes. Each thread takes the next highest number from a counter that has not been checked for primality and increments that counter safely such that no other thread will check that same number. If that number is prime, that thread will safely increment the total primes found counter by one and the sum of primes counter by the value of the new prime. It will also safely store the new prime at the appropriate position in a list of the 10 largest primes, quickly bumping off the smallest prime on that list if there are now more than 10 stored. Once these operations have been performed, the thread releases all these resources at once for other threads to modify after finishing their own primes and waits to safely access the next highest prime. Once we check 10^8, any threads that access the next highest prime that is larger than 10^8 will shut down. Once all threads have shut down, we can safely output our pre-calculated results.

This approach eliminates synchronization issues between threads by locking the "get the next largest number" operations into a synchronized method and the "save this prime and increment things" operations into another. No two threads can be performing either set of operations at the same time, so only one thread will ever check any number and only one thread at a time can modify the top ten primes list and change result counters. Additionally, in order to guarantee that each thread does an equal share of work, threads that have finished checking any number grab the next highest value to check since as these numbers get close to 10^8 they take longer to evaluate. Since each number takes an increasing amount of time to check, we cannot divide the range of numbers linearly because threads with smaller value ranges would finish faster than threads with higher value ranges and more time-consuming computations. Instead, each thread runs consistently as we go through the range by grabbing the next number available to check and all threads should end at roughly the same time. Finally, to verify the primality of any number a thread currently has, we only need to check that number for divisibility by 2, 3, and any integer $6k \pm 1$ for all integers $k \geq 1$ (proof here: https://en.wikipedia.org/wiki/Primality_test).

The initial implementation for checking primes checked the current number for divisibility by all numbers less than or equal to the square root of the current number. With only one thread, this program took ~91 seconds to execute. When increasing the thread count to eight, the program's execution was reduced to ~14 seconds (a 6.5x speedup). After optimizing the primality check by only checking divisibility by 2, 3, and $6k \pm 1$ as mentioned above, the execution time was reduced to ~12 seconds (for a 1.17x speedup). Finally, by converting the recording method from a sync method to just having a sync block, the runtime was reduced to ~10.5 seconds. Other testing results:

Range	Output
1, 100	1 25 1060 53 59 61 67 71 73 79 83 89 97
1, 10000	4 1229 5736396 9887 9901 9907 9923 9929 9931 9941 9949 9967 9973
1, 1000000	115 78498 37550402023 999863 999883 999907 999917 999931 999953 999959 999961 999979 999983