Grant Kinsley (gkinsley)
Ethan Duong (duonget)

231P: HW 3

A description of the system where the programs were executed: Processor, RAM, and Operative System

For a program to be executed, the operating system must choose a processor to execute a process of that program. Once a processor is chosen, it will retrieve that program's instructions from the RAM and start executing those instructions.

A comparative table showing, on the search up to 10^7, the execution times of the single-thread version, and the multi-thread with 1, 2, 4, 6, and 8 threads

Searching for primes up to 10^7

Exec type	Trial 1: Exec time (s)	Trial 2: Exec time (s)
Sequential	29.49283270	11.09210960
Multi-thread (1 thread)	29.13394360	12.17451280
Multi-thread (2 thread)	21.49251200	7.37372870
Multi-thread (4 thread)	13.96216980	4.62890790
Multi-thread (6 thread)	13.64732150	3.98147260
Multi-thread (8 thread)	10.94066280	3.88520120

An analysis of the performance of the programs with emphasis on correctness and speedup

We can see from the two trials that there is a speedup on execution time as we increase the number of theads used in the multi-threaded execution types. We see the biggest speed up between 1 and 2 threads, and between 2 and 4 threads used. The speed up from 4 to 6 and then 6 to 8 threads is not as significant, but there is still an analytic difference.

We made sure that results and output from all executions resulted in correct prime numbers found:

```
~/parallel-hw/hw3_prime_numbers$ diff <(sort m_002_10000000.log) <(sort s_001_10000000.log)
~/parallel-hw/hw3_prime_numbers$ diff <(sort m_004_10000000.log) <(sort s_001_10000000.log)
~/parallel-hw/hw3_prime_numbers$ diff <(sort m_006_10000000.log) <(sort s_001_10000000.log)
~/parallel-hw/hw3_prime_numbers$ diff <(sort m_008_10000000.log) <(sort s_001_10000000.log)
~/parallel-hw/hw3_prime_numbers$ diff <(sort m_001_10000000.log) <(sort s_001_10000000.log)
~/parallel-hw/hw3_prime_numbers$
```

An analysis of the sequential version versus the multi-thread version with only one thread. Is there a difference? How do you explain the results? Add plots and figures as necessary to explain your results

A multithreaded version with only one thread will essentially execute the same as the sequential version. This is because a sequential process can be thought of as a process with just one thread. We can see this reflected in the results as the execution time is nearly identical between the two trials.

In the second trial, the multithreaded version with one thread even takes a little more time to execute. This can be explained by the context switches necessary to assign the singular thread to execute for each number, when the sequential verison just keeps executing.

What happens if Simultaneous Multi-Threading (or “Hyper-Threading”) is enabled or disabled

If SMT is enabled, then each core appears as 2 cores to the operating system. This would increase the speedup of the execution time of all the multithreaded versions of execution, since threads would have “2” cores that they can access to execute instructions. The executions with at least two threads would benefit from hyperthreading the most. The execution with just one thread would still perform similarly to the sequential execution.

We might see small speedups in the sequential version, since there are two cores available to execute the process. But the total work required for the process would still not be split up and done in parallel, even with hyperthreading.