Bong Lee
CS 218
Asst12 WriteUp
****

Sequential Timed Executions

Timed Test #1

real 0m9.599s

user 0m9.574s

sys 0m0.016s

Timed Test #2

real 0m9.895s

user 0m9.893s

sys 0m0.000s

Timed Test #3

real 3m39.105s

user 3m39.039s

sys 0m0.016s

Timed Test #3

real 3m45.171s

user 3m45.016s

sys 0m0.072s

****

Parallel Timed Executions

Timed Test #1

real 0m3.311s

user 0m9.919s

sys 0m0.000s

Bong Lee
CS 218
Asst12 WriteUp

Timed Test #2

real 0m3.352s

user 0m10.035s

sys 0m0.004s

Timed Test #3

real 1m17.643s

user 3m52.508s

sys 0m0.148s

Timed Test #3

real 1m18.549s

user 3m55.437s

sys 0m0.064s

Bong Lee
CS 218
Asst12 WriteUp

With Lock

Start Prime Count

Prime Count: 2422003

Completed.

****

[1mPrimes Program[0m

Start Prime Count

Prime Count: 25764657

Completed.

No Lock

Start Prime Count

Prime Count: 2471317

Completed.

****

[1mPrimes Program[0m

Start Prime Count

Prime Count: 26225361

Completed.

Bong Lee
CS 218
Asst12 WriteUp

Sequential vs Parallel

The difficulty of checking for prime numbers increases significantly as the size of the number increases. A single core checking the prime numbers will have to finish checking the current number before moving onto the next number. This process is very inefficient and slow. Having three cores significantly increases performance. Each core is able to check a number which means that three numbers can be check simultaneously. Theoretically that should increase the performance by three times.

Yes Lock vs No Lock

Having three cores work together presents a new kind of challenge. When the cores must change a variable, a race condition might occur when multiple cores try to change a variable at the same time. Without a proper way to lock the variable the resulting data can be lost or extra results can be gained. In this case extra primes were counted when the lock was not present.