Prime Numbers Concurrency Project

Kyle Hennessy - 26th February 2021 - Institute of Technology, Carlow

## Github

[Twin Primes Project](https://github.com/KyleHennessy/TwinPrimesProject)

## Specification

There are an infinite number of prime numbers. Amongst the primes there are what are known as twin primes. A twin prime is a prime number that is either 2 less or 2 more than another prime number—for example, either member of the twin prime pair(41, 43). In other words, a twin prime is a prime that has a prime gap of two. Sometimes the term twin prime is used for a pair of twin primes; an alternative name for this is prime twin or prime pair. It is (currently) unknown whether there are an infinite number of twin primes.

Your task is to write a parallel program that counts the number of primes less than n for any number n and also finds and lists all twin primes less than n. The code should run on linux.

## Concurrent vs Sequential

Sequential processes are generally considered to be a lot slower than concurrent processes. This is because a sequential process must be executed in the exact order as it is written in the code, or in other words, must be executed step by step. This is perfectly acceptable when dealing with small amounts of data and speed is not important, but quickly becomes inefficient when dealing with very large amounts of data or in scenarios where it is critical to perform multiple calculations at once.

Concurrent processes on the other hand eliminates the need for the step-by-step fashion of execution, where calculations can be done in any order without impacting the output of the process.

This is achieved through the use of threads located on the cpu. It allows you to essentially divide the work among each thread which will significantly impact the efficiency of the process. It is important to note however, that a concurrent process is not guaranteed to be faster than a sequential process, as it actually takes a relatively long time to initialize threads and allocate a specific task to that thread, so when dealing with small amounts of data, such as finding every prime number of to 50, it is actually better to do this sequentially. But if the same task was to be done with say 1,000,000 numbers, that is where concurrency will yield much better results in terms of speed up.

## 

## Pseudocode

StartTimer  
#Multithreaded For Loop To Find Prime Numbers

For each number less than max number:  
 If number is prime:

#Critical section, only one thread can do this at a time

Add number to collection of prime numbers

If number + 2 is a prime:

Add number to collection of twin primes

Add number + 2 to collection of twin primes

EndTimer

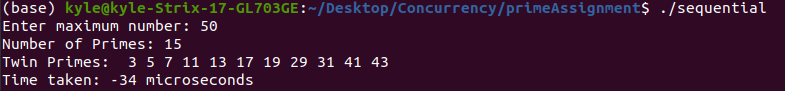
Print size of prime numbers collection

Print contents of twin primes collection

Print time taken

## Speedup

When calculating the speedup of the program, it is important to note that when using a low number of prime numbers, e.g every prime up to 50, there is absolutely 0 benefit to multithreading as it actually takes longer to initialize the threads than running it sequentially as seen in the figure below.



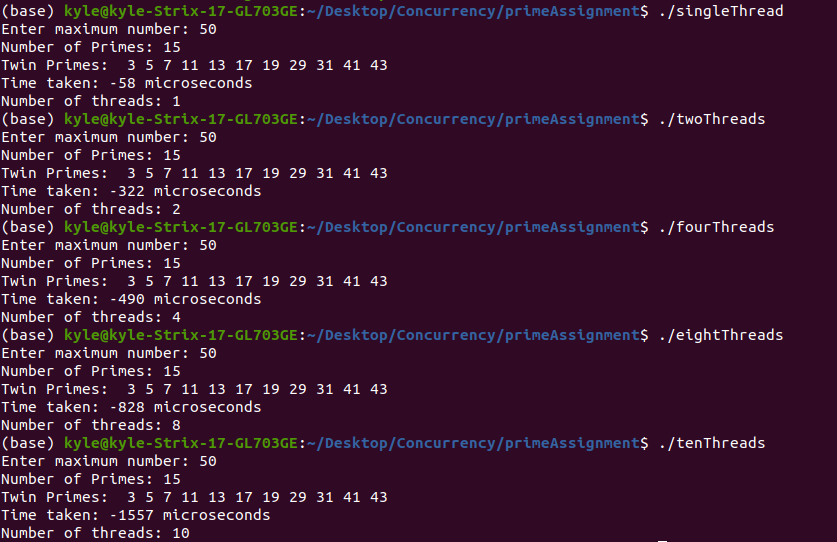
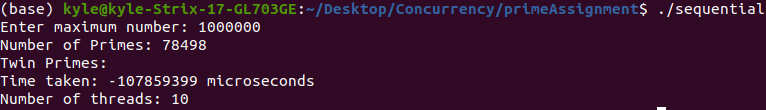
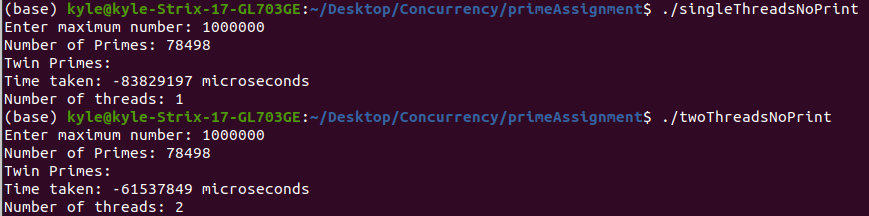


Figure 1 - Sequential is best for smaller computations

However if a high number of primes was being calculated, e.g every prime number up to 1,000,000, that is where there will be major improvements in the time to compute when using multithreading vs. sequentially, as seen in the figure below.





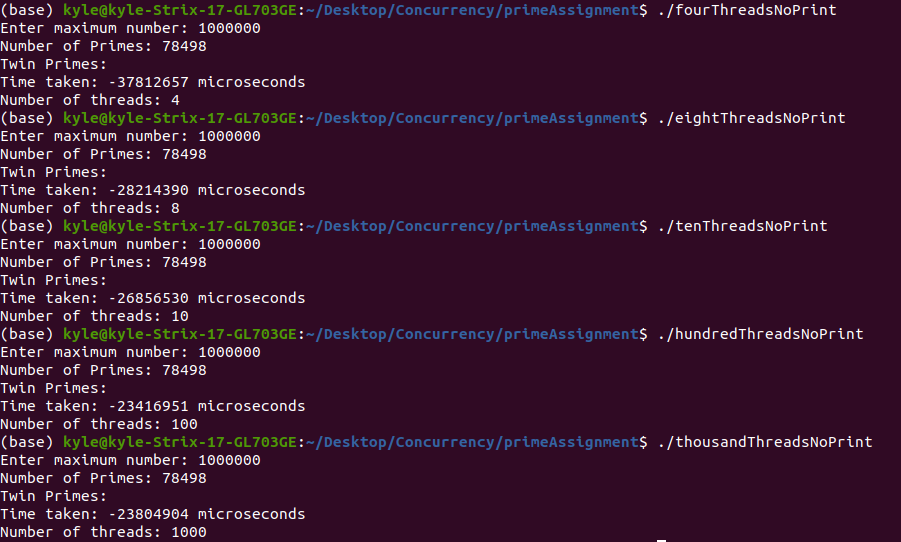


Figure 2 - Multithreading much faster for large computations

|  |  |
| --- | --- |
| Number of Threads | Time to Compute |
| Sequential | 107.86 seconds |
| 1 | 83.83 seconds |
| 2 | 61.54 seconds |
| 4 | 37.81 seconds |
| 8 | 28.21 seconds |
| 10 | 26.86 seconds |
| 100 | 23.47 seconds |
| 1000 | 23.80 seconds |

This will be the time that will be used to calculate the speed up of the program. The formula for calculating the speedup is as follows:

Absolute speed up = sequentialSpeed / thread1 speed

Relative speed up = thread1 speed / thread(n) speed

Absolute speedup = 107.86/83.83 = 1.2867

Relative Speedup for each thread:

|  |  |
| --- | --- |
| Number of threads | Result (83.83/thread(n) speed) |
| 2 | 1.3622 |
| 4 | 2.2171 |
| 8 | 2.9716 |
| 10 | 3.1210 |
| 100 | 3.5718 |
| 1000 | 3.5223 |

## Scalability

Using the relative speed up, it is possible to graph the scalability of the software. In other words, at what point does adding more resources actually increase the time it takes to compute? As seen in the figure below, when there are 1000 threads, it actually has worse performance than with 100 threads. 100 threads is where the maximum performance can be achieved by the cpu, and since it uses processing power to create and initialize threads, it ends up just increasing the overall time to compute.

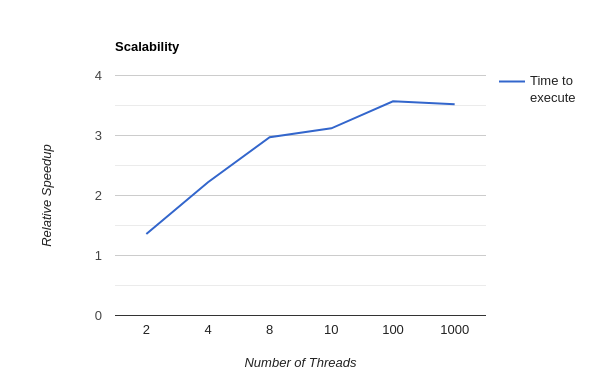


Figure 3 - Scalability of the “twin primes” program.