Prime Numbers Concurrency Project

James Hall
C00007006

Concurrent Device Development

Institute of Technology Carlow

Tutor: Dr Joseph Kehoe
Date: 26th February 2021

Table of Contents

Abstract

The purpose of this document is to review the differences between sequential and concurrent code. The code sequential and concurrent code only differ in the use of openMP call "#pragma omp parallel for num_threads(T)" before the loops in the primeInRange() and twinPrimeInRange() functions. With the "for" OpenMP automatically divides the work to be done inside the following for loop between threads.

The sequential code is compared to the concurrent code, and the Absolute and Relative speed ups are calculated from the test results with a number of different core counts. Finally, the scalability of the code is tested by comparing the sequential code with the concurrent code running with the optimal number of cores, using a set of increasing number values.

PseudoCode

```
bool isPrime(int n)

        int N=n;
        // Corner cases
        if (N <= 1)  return false
        if (N <= 3)  return true

        // This is checked so that we can skip
        // middle five numbers in below loop
        if (N%2 == 0 || N%3 == 0) return false

        for (int i=5; i*i<=N; i=i+6){
        if (N%i == 0 || N%(i+2) == 0)
        return false

        return true

void primeInRange(int L, int R, int T)
        int i;
        int count = 0;

        // Traverse each number in the
        // interval with the help of for loop

        Start threading for num_threads(T)
        for (i = L; i <= R; i++)
                if(isPrime(i))
                        //Ensure count is counted correctly with atomic
                        Increment count atomically
                        count++
        printf("Total number of primes: %d \n", count)
```

```
void twinPrimeInRange(int L, int R, int T)
	int i;
	int count = 0;

	// Traverse each number in the
	// interval with the help of for loop
	/Start threading for num_threads(T)
	for (i = L; i <= R; i++) {
		if (isPrime(i) && isPrime(i+2)){
			//Ensure count is counted correctly with atomic.
			Increment count atomically
			count++
		printf("%d %d \n", i, i+2)
	printf("There are %d twin primes between %d and %d. \n", count, L, R)

int main(int argc, char *argv[])
	// 1 to Given value
	int L = 1
	char *passedValue = argv[1]
	char *threadCount = argv[2]
	int R = atoi(passedValue)
	int threads = atoi(threadCount)

	// Function Call
	twinPrimeInRange(L,R,threads)
	primeInRange(L,R,threads)
```

Analysis of Primes

Table 1. Number of Twin Primes in Various Ranges.

Sr.No.	Range 1-x	Ten Power (x)	Number of Twin Primes $_2\pi(x)$
1.	1-10	10^1	2
2.	1-100	10^2	8
3.	1-1,000	10^3	35
4.	1-10,000	10^4	205
5.	1-100,000	10^5	1,224
6.	1-1,000,000	10^6	8,169
7.	1-10,000,000	10^7	58,980
8.	1-100,000,000	10^8	440,312
9.	1-1,000,000,000	10^9	3,424,506
10.	1-10,000,000,000	10^{10}	27,412,679
11.	1-100,000,000,000	10^{11}	224,376,048
12.	1-1,000,000,000,000	10^{12}	1,870,585,220
13.	1-10,000,000,000,000	10^{13}	15,834,664,872
14.	1-100,000,000,000,000	10^{14}	135,780,321,665
15.	1-1,000,000,000,000,000	10^{15}	1,177,209,242,304
16.	1-10,000,000,000,000,000	10^{16}	10,304,195,697,298

For the following runs, 100 million was the number chosen for testing the code.

In the following, run 1 was performed on a different day to runs 2 and 3. The ambient temperature in the room was higher for runs 2 and 3 this is reflected in slightly slower run times.

Sequential

Unoptimized Prime method, using a bool variable and returning it at the end of the method.

```
99999587 99999589
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    39m7.498s
user    38m58.577s
sys     0m8.781s
```

Absolute time sequential: 39m 7.498s

Optimized Prime method returning if false or true found.

```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    2m50.009s
user    2m47.572s
sys     0m2.426s
```

Absolute time sequential:

Run	Time
1	2m 50.009s
2	2m 54.259s
3	2m 52.519s
Ave.	2m 52.262s = 172.262s

Parallel 1 Core

```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    2m52.667s
user    2m48.162s
sys     0m4.500s
```

Absolute time parallel:

Run	Time
1	2m 52.667s
2	2m 52.420s
3	2m 51.377s
Ave.	2m 52.141s = 172.141s

Parallel 2 cores

```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    1m48.708s
user    2m50.400s
sys     0m2.732s
```

Run	Time
1	1m 48.708s
2	1m 49.470s
3	1m 48.661s
Ave.	1m 48.946s = 108.946s

$$S_n = \frac{T_s}{T_p(n)}.$$

Absolute speedup:
2m 52.262s/ 1m 48.946s
172.262s / 108.946s = ~1.581

Relative speedup:
2m 52.141s/ 1m 48.946s
172.141s / 108.946s = ~1.580

Parallel 3 cores

```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    1m18.668s
user    2m55.907s
sys     0m3.595s
```

Run	Time
1	1m 18.668s
2	1m 20.910s
3	1m 21.920s
Ave.	1m 20.499s = 80.499s

Absolute speedup:
2m 52.262s/ 1m20.499s
172.262s / 80.499s = ~2.140

Relative speedup:
2m 52.141s/ 1m20.499s
172.141s / 80.499s = ~2.138

Parallel 4 cores

```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    1m0.122s
user    2m53.816s
sys     0m4.717s
```

Run	Time
1	1m 0.122s
2	1m 3.419s
3	1m 3.109s
Ave.	1m 2.216s = 62.216s

Absolute speedup:
2m 52.262s/ 1m 2.216s

172.262s / 62.216s = ~2.733

Relative speedup:
2m 52.141s/ 1m 2.216s
172.141s / 62.216s = ~2.767

Parallel 5 cores

```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    0m50.457s
user    3m1.984s
sys     0m4.560s
```

Run	Time
1	50.122s
2	53.187s
3	53.458s
Ave.	52.256s

Absolute speedup:
2m 52.262s/ 52.256s
172.262s / 52.256s = ~3.297

Relative speedup:
2m 52.141s/ 52.256s
172.141s / 52.256s = ~3.294

Parallel 6 cores

```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    0m42.468s
user    3m1.188s
sys     0m4.419s
```

Run	Time
1	42.468s
2	45.532s
3	45.594s
Ave.	44.531s

Absolute speedup:
2m 52.262s/ 44.531s

172.262s / 44.531s = ~3.868

Relative speedup:
2m 52.141s/ 44.531s
172.141s / 44.531s = ~3.866

Parallel 7 cores

```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    0m37.144s
user    3m2.850s
sys     0m4.731s
```

Run	Time
1	37.144s
2	41.518s
3	41.615s
Ave.	40.092s

Absolute speedup:
2m 52.262s/ 40.092s
172.262s / 40.092s = ~4.297

Relative speedup:
2m 52.141s/ 40.092s
172.141s / 40.092s = ~4.294

Parallel 8 cores

```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    0m34.425s
user    3m8.267s
sys     0m5.533s
```

Run	Time
1	34.425s
2	37.939s
3	39.529s
Ave.	37.298s

Absolute speedup:
2m 52.262s/ 37.298s

172.262s / 37.298s = ~4.619

Relative speedup:
2m 52.141s/ 37.298s
172.141s / 37.298s = ~4.615

Concurrent 16 cores

```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    0m29.941s
user    3m17.347s
sys     0m7.286s
```

Run	Time
1	29.941s
2	33.674s
3	33.587s
Ave.	32.304s

Absolute speedup:
2m 52.262s/ 32.304s
172.262s / 32.304s = ~5.333

Relative speedup:
2m 52.141s/ 32.304s
172.141s / 32.304s = ~5.329

After this point the speedup is within margin of error, so there is no further need for Speed up calculations.

Concurrent 32 cores

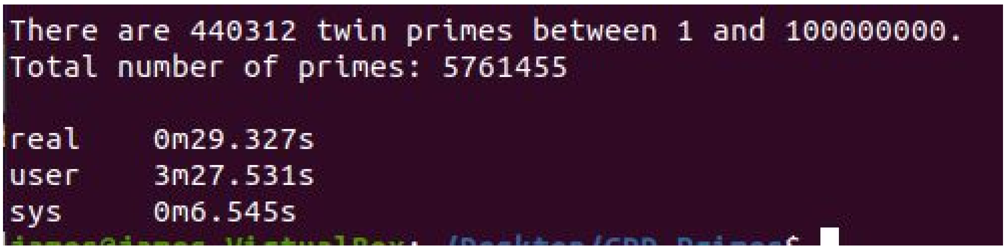
```
There are 440312 twin primes between 1 and 100000000.
Total number of primes: 5761455

real    0m28.528s
user    3m17.051s
sys     0m7.001s
```

Run	Time
1	28.528s
2	32.308s

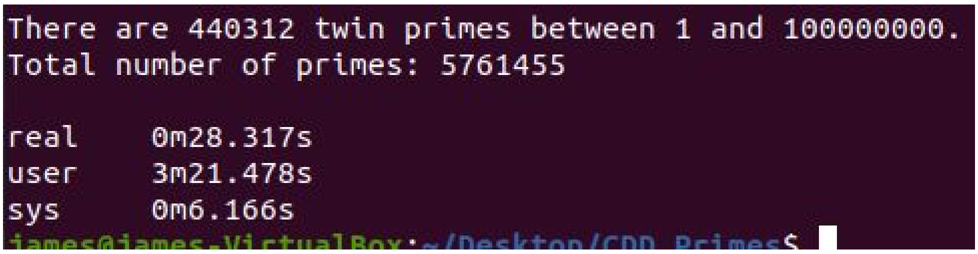
3	<u>32.713s</u>
Ave.	31.183s

Concurrent 64 cores



Run	Time
1	29.327s
2	31.841s
3	<u>32.330s</u>
Ave.	31.166s

Concurrent 128 cores

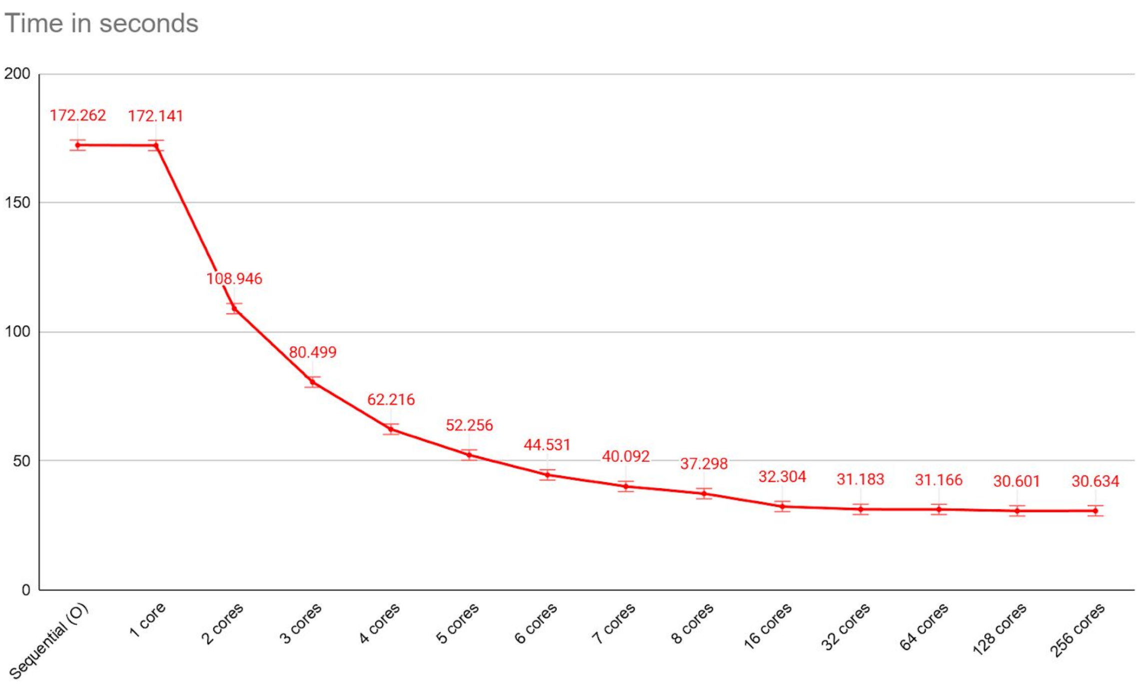


Run	Time
1	28.317s
2	31.941s
3	<u>31.546s</u>
Ave.	30.601s

Concurrent 256 cores

Run	Time

1	28.257s
2	32.078s
3	31.568s
Ave.	30.634s

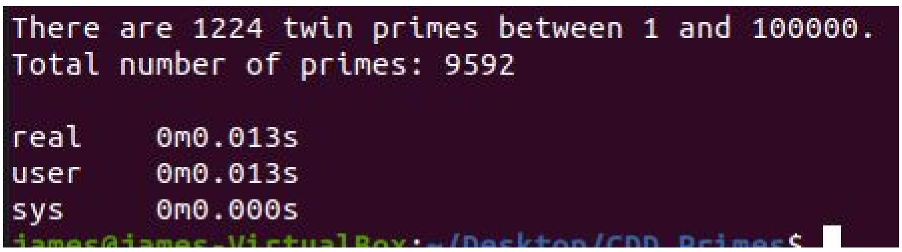


Scalability

For these tests, a number was chosen and run with the sequential code, and the best result per core concurrent code. This number was then doubled until a sufficient measure of scalability could be established.

Sequential

Number to test = 100000



Number to test = 200000

```
There are 2160 twin primes between 1 and 200000.
Total number of primes: 17984

real    0m0.032s
user    0m0.031s
sys     0m0.000s
james@james-VirtualBox:~/Desktop/CDD Primes$
```

Number to test = 400000

```
There are 3804 twin primes between 1 and 400000.
Total number of primes: 33860

real    0m0.078s
user    0m0.078s
sys     0m0.000s
```

Number to test = 800000

```
There are 6766 twin primes between 1 and 800000.
Total number of primes: 63951

real    0m0.197s
user    0m0.184s
sys     0m0.012s
james@james-VirtualBox:~/Desktop/CDD Primes$
```

Number to test = 1600000

```
There are 12260 twin primes between 1 and 1600000.
Total number of primes: 121127

real    0m0.509s
user    0m0.438s
sys     0m0.072s
```

Number to test = 3200000

```
There are 22137 twin primes between 1 and 3200000.
Total number of primes: 230209

real    0m1.350s
user    0m1.162s
sys     0m0.189s
```

Number to test = 6400000

```
There are 40092 twin primes between 1 and 6400000.
Total number of primes: 438410

real	0m3.731s
user	0m3.331s
sys	0m0.259s
james@james-VirtualBox:~/Desktop/CDD Primes$
```

Number to test = 12800000

```
There are 72946 twin primes between 1 and 12800000.
Total number of primes: 837099

real	0m9.402s
user	0m8.848s
sys	0m0.554s
james@james-VirtualBox:~/Desktop/CDD Primes$
```

16 cores

Number to test = 100000

```
There are 1224 twin primes between 1 and 100000.
Total number of primes: 9592

real	0m0.013s
user	0m0.008s
sys	0m0.025s
```

Number to test = 200000

```
There are 2160 twin primes between 1 and 200000.
Total number of primes: 17984

real	0m0.024s
user	0m0.043s
sys	0m0.027s
```

Number to test = 400000

```
There are 3804 twin primes between 1 and 400000.
Total number of primes: 33860

real	0m0.044s
user	0m0.104s
sys	0m0.041s
```

Number to test = 800000

```
There are 6766 twin primes between 1 and 800000.
Total number of primes: 63951

real    0m0.089s
user    0m0.226s
sys     0m0.092s
```

Number to test = 1600000

```
There are 12260 twin primes between 1 and 1600000.
Total number of primes: 121127

real    0m0.204s
user    0m0.682s
sys     0m0.115s
```

Number to test = 3200000

```
There are 22137 twin primes between 1 and 3200000.
Total number of primes: 230209

real    0m0.404s
user    0m1.574s
sys     0m0.272s
```

Number to test = 6400000

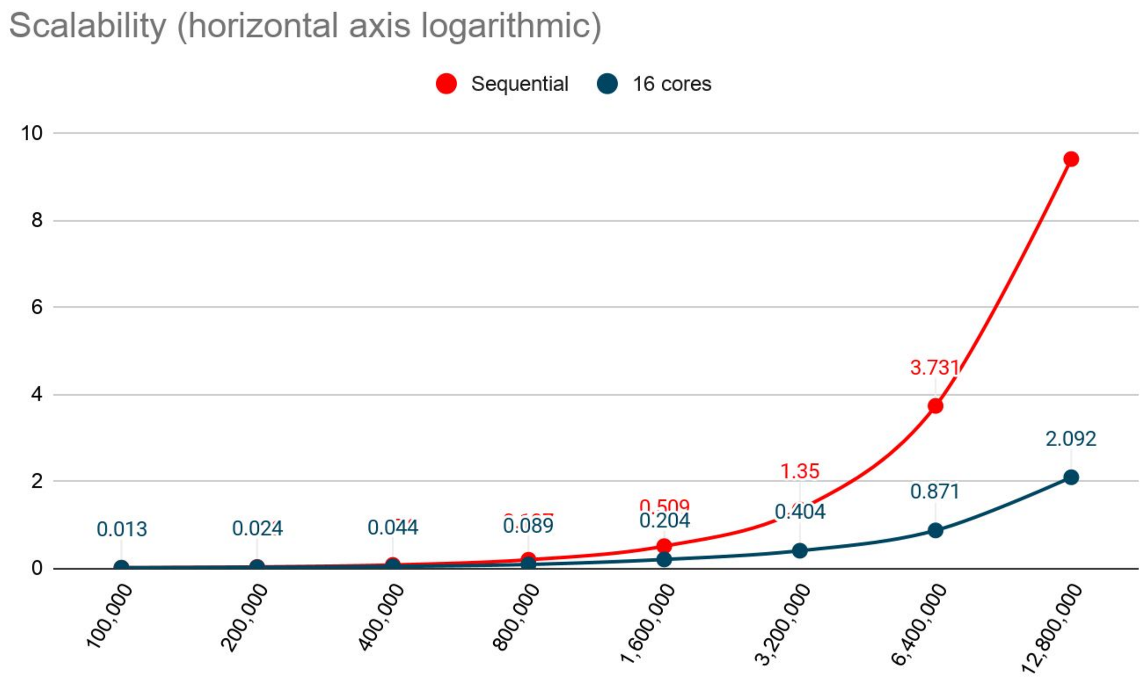
```
There are 40092 twin primes between 1 and 6400000.
Total number of primes: 438410

real    0m0.871s
user    0m4.350s
sys     0m0.308s
```

Number to test = 12800000

```
There are 72946 twin primes between 1 and 12800000.
Total number of primes: 837099

real    0m2.092s
user    0m11.903s
sys     0m0.687s
```



Conclusion

As using OpenMP takes a lot of the head work out of concurrent development, the concurrent and sequential code used did not differ greatly and this led to run times using both with "one" core to be within margin of error from run to run. Running the concurrent code with 16 "cores" lead to the optimal speed up with Absolute and Relative values of ~5.333 and ~5.329 respectively.

With regards to scalability, the difference in speed can be seen to diverge as the number being checked was doubled each time, with the sequential code more than doubling its run time as the number doubled. The concurrent code kept this increase closer to a two times increase and so agap began to develop.

Given more time, it would be interesting to see what other improvements could be made to both the sequential and concurrent code to improve run time for both, including further improvements to the isPrime() method. Problems were encountered with attempts to create threading for the loop in this method and a threaded implementation was dropped and the primeInRange() and twinPrimeInRange() methods were concentrated on instead.