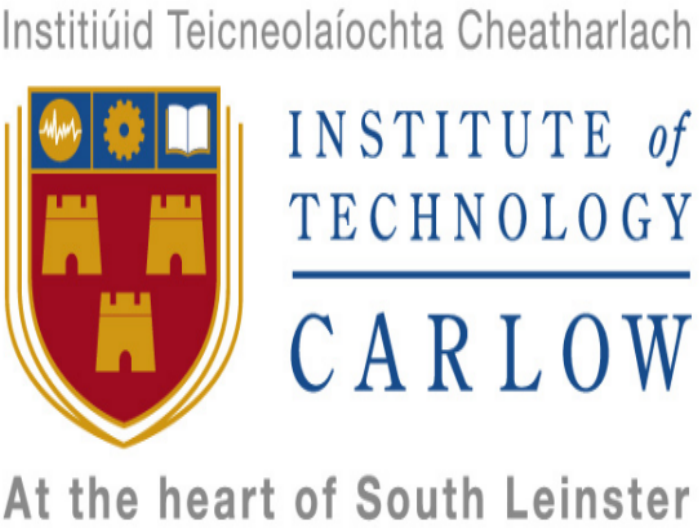
Concurrency Project



Student Name: Molin Sun

Student Number: C00266170

Date of submission: 1st September

Introduction

This report is mainly to measure the performance of the program by analysing the program that runs in sequential, concurrent and parallel. In this document, I will use both absolute speedup and relative speedup to measure the efficiency of the program.

The program is to calculate the number of primes less than n (n could be any number) and list all twin primes less than n. In this project, we will use openMP to parallelize the program.

Research about OpenMP

OpenMP [1] is an application program interface (API) which provides a portable and extensible programming model for developers of shared memory-based parallel programs. Its API supports C/C++ and Fortran on a variety of architectures.

OpenMP [1] can be used to explicitly guide multithreaded shared memory parallelism in C/C++ programs. It does not interfere with the original serial code because OpenMP instructions are generated in pragmas interpreted by the compiler.

There is an example that can help us to understand OpenMP quickly and easily.

The code below shows a Hello World example which is sequential.

```
#include <stdio.h>
int main() {
    printf( "Hello, World!\n" );
    return 0;
}
```

So how do we use OpenMP to make it parallel? We just need add “#pragma omp parallel private(thread_id)” before the task that needs to be executed in parallel.

```
#include <stdio.h>
#include <omp.h>
int main() {
    int thread_id;
    #pragma omp parallel private(thread_id)
    {
        thread_id = omp_get_thread_num();
        printf( "Hello, World from thread %d!\n", thread_id );
    }
    return 0;
}
```

After having a basic understanding of OpenMP, let’s learn how to use it to make our program parallel.

First point, if you want to use OpenMp, you need to include the OpenMP header.

```
#include <omp.h>
```

Second point, each thread has its own unique id and we use **omp_get_thread_num()** to get the id of the thread.

#pragma omp parallel {...} to parallelize the task we need to do. The task is written in {}. And if the task is looping statements, we need to use **#pragma omp parallel for {...}.**

```
#pragma omp parallel
{
  task;
}
```

```
#pragma omp parallel for // parallel for loop
for( ... ) {
  task;
}
```

#parallel NUM_threads () to limit the number of parallel threads. **#parallel NUM_threads (3)** means that limit the number of parallel threads to 3.

#pragma omp atomic is an atomic command that ensures variables are updated in a single and unbreakable step.

Real/user/sys time

Adding the time command to program execution can roughly calculate the time spent during program execution. There are usually three values: real time, user time and sys time.

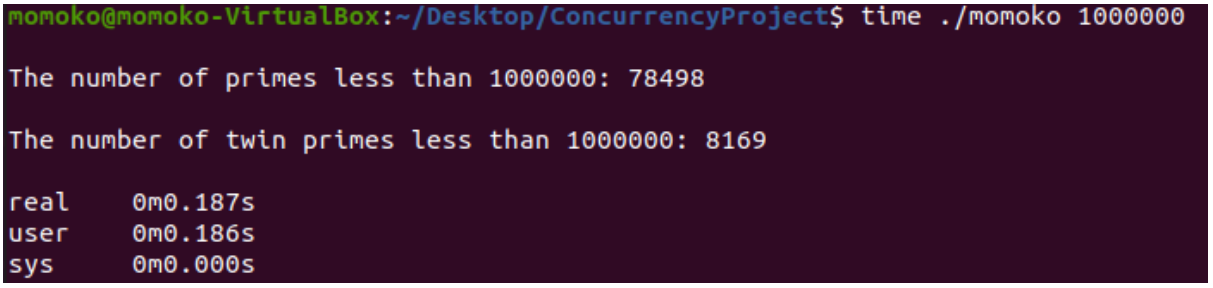


Figure 1 “the result of time command in linux”

real time: is the time experienced by the user. In other words, the amount of time the user waits for the program to execute.
user time: All the time spent in user space during the execution of a program, i.e. the CPU time spent in user mode of the program.
sys time: The amount of time spent in the kernel space during program execution, which is the CPU time spent by the program in kernel calls.

In this program, we use real time to measure the performance of the program.

Code

```
/*! \fn int isPrime(int number)
    \param number  the number which is checked whether it is a prime
    \brief returns whether a number is prime
*/
bool isPrime(int number)
    if (number <= 1) return false;
    if (number <= 3) return true;

    if(number%2 == 0 || number%3 == 0) return false;

    for(int i = 5; i*i <=number; i=i+6)

      if(number%i == 0 || number%(i+2) == 0) return false;

    return true;
```

This method is to determine whether the number is a prime. If the number is a prime, return true, otherwise return false.

```
/*! \fn int numberPrimes(int first, int last)
    \param first The first number to check
    \param last The last number to check
    \brief returns the number of primes less than n
*/
void numberPrimes(int first, int last)

    int i;
    int count = 0;
    //#pragma omp parallel for num_threads(2)
    for(i = first; i <= last; i++)

      if(isPrime(i))

        #pragma omp atomic
        count++;

    printf("\nThe number of primes less than %d: %d\n", last, count);
```

This method is to calculate the number of primes less than n.

The system

```
momoko@momoko-VirtualBox:~/Desktop$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          4
On-line CPU(s) list:             0-3
Thread(s) per core:              1
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           158
Model name:                      Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
Stepping:                        9
CPU MHz:                         2496.002
BogoMIPS:                        4992.00
Hypervisor vendor:               KVM
Virtualization type:             full
L1d cache:                       128 KiB
L1i cache:                       128 KiB
L2 cache:                        1 MiB
L3 cache:                        24 MiB
NUMA node0 CPU(s):               0-3
Vulnerability Itlb multihit:     KVM: Mitigation: VMX unsupported
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT Host state u
                                 nknown
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user
                                  pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, STIBP disab
                                 led, RSB filling
Vulnerability Srbds:             Unknown: Dependent on hypervisor status
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtr
                                 r pge mca cmov pat pse36 clflush mmx fxsr sse s
                                 se2 ht syscall nx rdtscp lm constant_tsc rep_go
```

Figure 2 “The CPU information about the system”

Sequential

When we run the program in sequential, we need to comment the #gragma sentence.

In order to get the accurate result, in each case we run 3 times and get the average real time.

Sequential: n = 10,000

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000

The number of primes less than 10000: 1229

The number of twin primes less than 10000: 205

real    0m0.003s
user    0m0.003s
sys     0m0.000s
```

Figure 3

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000

The number of primes less than 10000: 1229

The number of twin primes less than 10000: 205

real    0m0.003s
user    0m0.003s
sys     0m0.000s
```

Figure 4

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000

The number of primes less than 10000: 1229

The number of twin primes less than 10000: 205

real    0m0.003s
user    0m0.000s
sys     0m0.003s
```

Figure 5

Average(real time):0.005s

Sequential: n = 100,000

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000

The number of primes less than 100000: 9592

The number of twin primes less than 100000: 1224

real    0m0.013s
user    0m0.013s
sys     0m0.000s
```

Figure 6

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000

The number of primes less than 100000: 9592

The number of twin primes less than 100000: 1224

real    0m0.012s
user    0m0.012s
sys     0m0.000s
```

Figure 7

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000

The number of primes less than 100000: 9592

The number of twin primes less than 100000: 1224

real    0m0.011s
user    0m0.007s
sys     0m0.004s
```

Figure 8

Average(real time):0.012s

Sequential: n = 1,000,000

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 1000000

The number of primes less than 1000000: 78498

The number of twin primes less than 1000000: 8169

real    0m0.186s
user    0m0.182s
sys     0m0.004s
```

Figure 9

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 1000000

The number of primes less than 1000000: 78498

The number of twin primes less than 1000000: 8169

real    0m0.187s
user    0m0.183s
sys     0m0.000s
```

Figure 10

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 1000000

The number of primes less than 1000000: 78498

The number of twin primes less than 1000000: 8169

real    0m0.186s
user    0m0.184s
sys     0m0.000s
```

Figure 11

Average(real time):0.186s

Sequential: n = 10,000,000

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m4.168s
user    0m4.166s
sys     0m0.000s
```

Figure 12

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m4.287s
user    0m4.276s
sys     0m0.008s
```

Figure 13

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m4.199s
user    0m4.180s
sys     0m0.008s
```

Figure 14

Average(real time):4.218s

Sequential: n = 100,000,000

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    1m47.797s
user    1m47.648s
sys     0m0.068s
```

Figure 15

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    1m48.105s
user    1m48.056s
sys     0m0.012s
```

Figure 16

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    1m51.930s
user    1m51.890s
sys     0m0.004s
```

Figure 17

Average(real time): 1m49.277s =109.277s

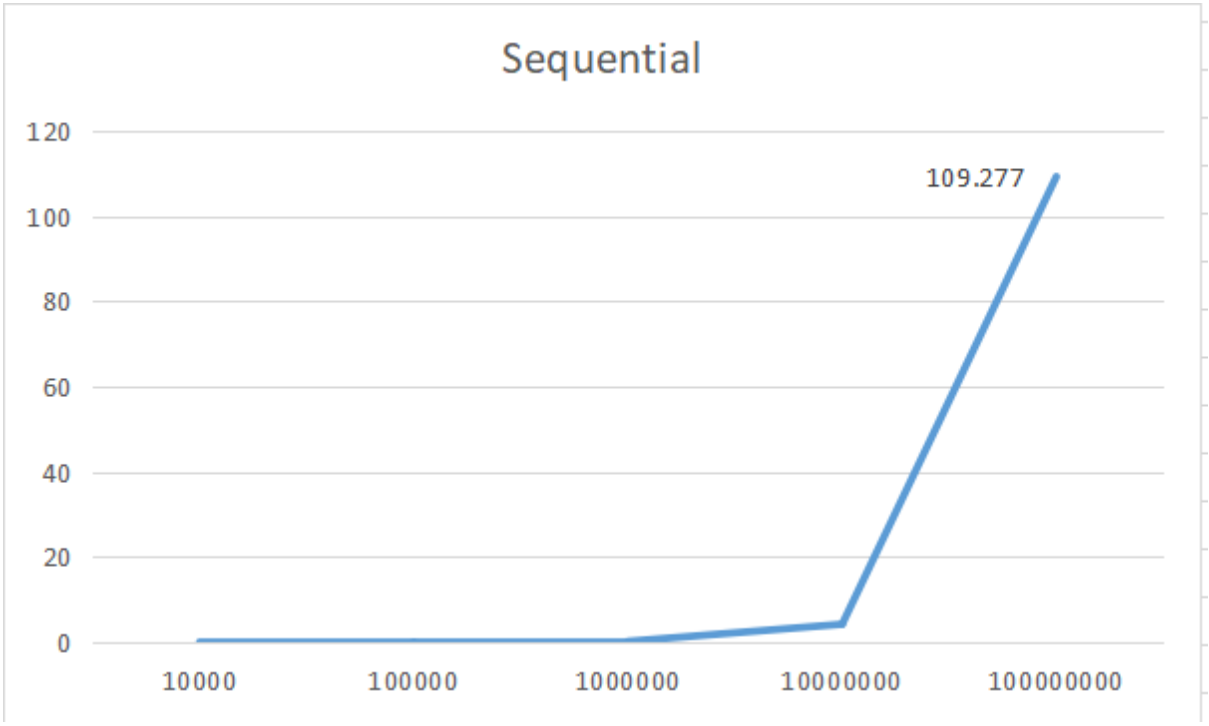


Figure 18

According to the figure above, when the calculative scale is very great, the time spent on calculation increases rapidly.

Concurrency & Parallel (n=10,000,000)

In this part, we run the program in parallel. Because my computer just has 4 CPUs. When thread is 1 or more than 4, the program actually runs concurrently.

Concurrency: n = 10,000,000 and thread = 1

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m4.140s
user    0m4.127s
sys     0m0.004s
```

Figure 19

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m4.299s
user    0m4.257s
sys     0m0.024s
```

Figure 20

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m4.091s
user    0m4.071s
sys     0m0.008s
```

Figure 21

Average(real time): 4.177s

Parallel: n = 10,000,000 and thread = 2

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m2.465s
user    0m4.020s
sys     0m0.000s
```

Figure 22

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m2.483s
user    0m4.024s
sys     0m0.008s
```

Figure 23

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m2.511s
user    0m4.038s
sys     0m0.020s
```

Figure 24

Average(real time): 2.486s

Parallel: n = 10,000,000 and thread = 3

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.796s
user    0m4.153s
sys     0m0.012s
```

Figure 25

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.749s
user    0m4.044s
sys     0m0.016s
```

Figure 26

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.767s
user    0m4.084s
sys     0m0.028s
```

Figure 27

Average(real time): 1.771s

Parallel: n = 10,000,000 and thread = 4

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.423s
user    0m4.297s
sys     0m0.033s
```

Figure 28

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.406s
user    0m4.173s
sys     0m0.017s
```

Figure 29

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.367s
user    0m4.155s
sys     0m0.032s
```

Figure 30

Average(real time): 1.399s

Concurrency: n = 10,000,000 and thread = 8

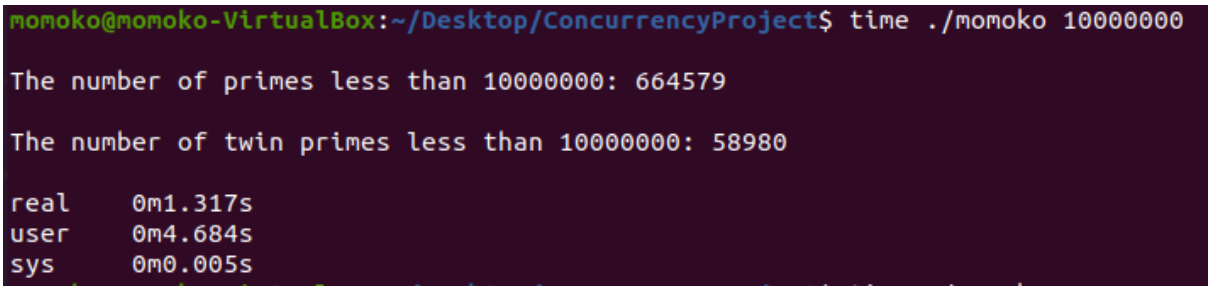


Figure 31

Figure 32

Figure 33

Average(real time): 1.287s

Concurrency: n = 10,000,000 and thread = 16

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.076s
user    0m4.062s
sys     0m0.016s
```

Figure 34

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.130s
user    0m4.248s
sys     0m0.021s
```

Figure 35

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.184s
user    0m4.397s
sys     0m0.052s
```

Figure 36

Average(real time): 1.130s

Concurrency: n = 10,000,000 and thread = 32

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.059s
user    0m3.995s
sys     0m0.024s
```

Figure 37

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.038s
user    0m3.992s
sys     0m0.012s
```

Figure 38

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.056s
user    0m3.995s
sys     0m0.036s
```

Figure 39

Average(real time): 1.051s

Concurrency: n = 10,000,000 and thread = 64

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.062s
user    0m4.059s
sys     0m0.025s
```

Figure 40

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.047s
user    0m4.017s
sys     0m0.021s
```

Figure 41

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 10000000

The number of primes less than 10000000: 664579

The number of twin primes less than 10000000: 58980

real    0m1.052s
user    0m4.034s
sys     0m0.028s
```

Figure 42

Average(real time): 1.054s

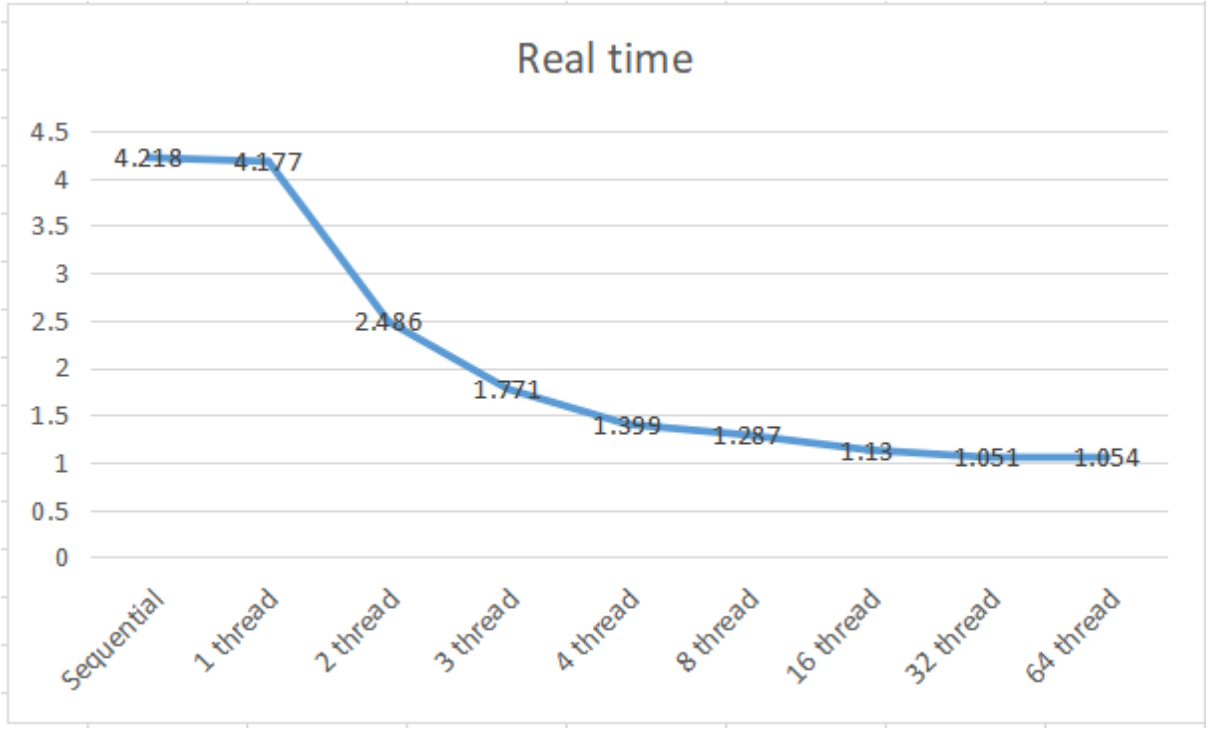


Figure 42

According to Figure 42, as the number of threads increases, the execution time decreases. So concurrency and parallel can improve the running speed of the program.

Absolute speedup and relative speedup（n = 10,000,000）

Formula:

- Absolute speedup = Sequential Time/Parallel Time
- Relative speedup = Single Thread/Parallel Time

Sequential time: 4.218s
Concurrent time (1 Thread): 4.177
Absolute speedup = 4.218/4.177 = 1.010
Parallel time (2 thread): 2.486s
Absolute speedup = 4.218/2.486 = 1.697
Relative speedup = 4.177/2.486 = 1.680
Parallel time (3 thread): 1.771s
Absolute speedup = 4.218/1.771 = 2.382
Relative speedup = 4.177/1.771 = 2.359
Parallel time (4 thread): 1.399s
Absolute speedup = 4.218/1.399 = 3.015
Relative speedup = 4.177/1.399 = 2.986
Concurrent time (8 thread): 1.287s
Absolute speedup = 4.218/1.287 = 3.277
Relative speedup = 4.177/1.287 = 3.246
Concurrent time (16 thread): 1.13s
Absolute speedup = 4.218/1.13 = 3.733
Relative speedup = 4.177/1.13 = 3.696
Concurrent time (32 thread): 1.051s
Absolute speedup = 4.218/1.051 = 4.013
Relative speedup = 4.177/1.051 = 3.974
Concurrent time (64 thread): 1.054
Absolute speedup = 4.218/1.054 = 4.002
Relative speedup = 4.177/1.054 = 3.963

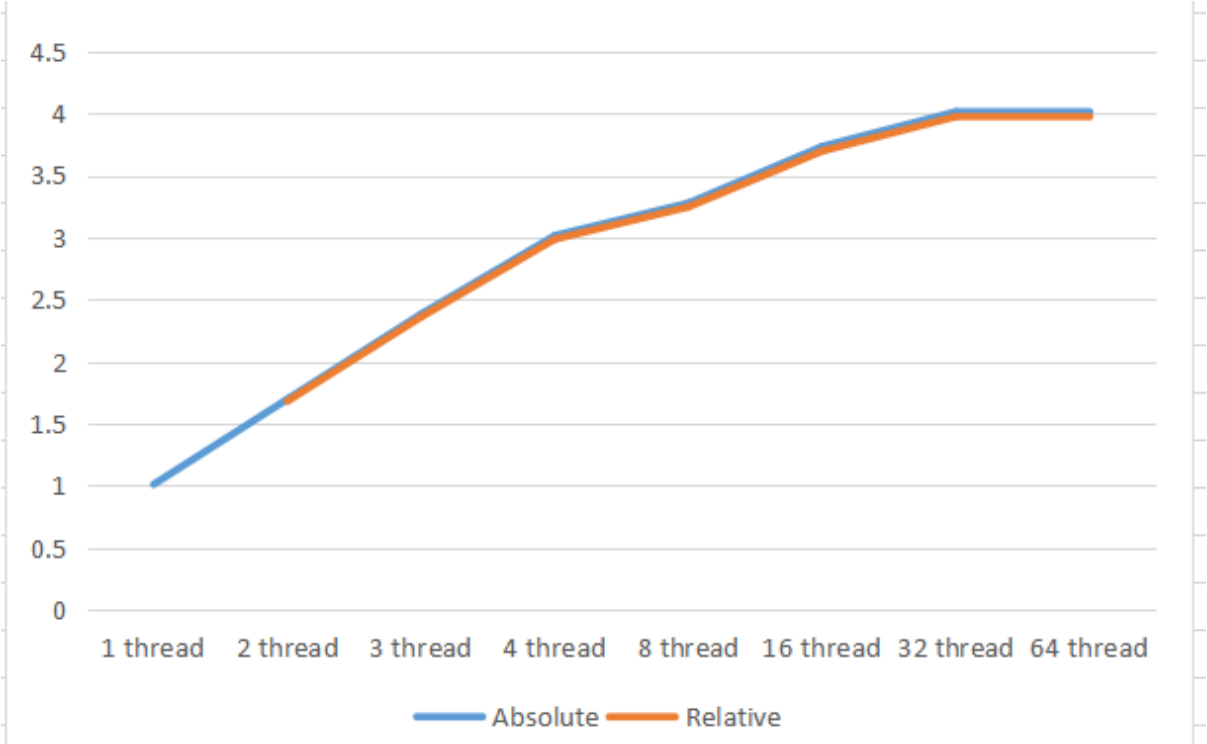


Figure 43 Absolute speed and relative speed with different thread when n = (10,000,000)

Concurrency & Parallel (n=100,000,000)

Concurrency: n = 100,000,000 and thread = 1

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    1m51.747s
user    1m51.693s
sys     0m0.012s
```

Figure 44

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    1m47.568s
user    1m47.476s
sys     0m0.016s
```

Figure 45

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    1m47.209s
user    1m47.173s
sys     0m0.000s
```

Figure 46

Average(real time): 1m48.841s = 108.841s

Parallel: n = 100,000,000 and thread = 2

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    1m4.874s
user    1m42.983s
sys     0m0.012s
```

Figure 47

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    1m5.535s
user    1m44.031s
sys     0m0.004s
```

Figure 48

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    1m5.284s
user    1m43.700s
sys     0m0.016s
```

Figure 49

Average(real time): 1m5.231s = 65.231s

Parallel: n = 10,000,000 and thread = 3

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m45.182s
user    1m41.869s
sys     0m0.108s
```

Figure 50

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m44.669s
user    1m41.167s
sys     0m0.084s
```

Figure 51

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m44.922s
user    1m41.637s
sys     0m0.112s
```

Figure 52

Average(real time): 44.924s

Parallel: n = 10,000,000 and thread = 4

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m34.202s
user    1m42.018s
sys     0m0.133s
```

Figure 53

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m35.093s
user    1m44.631s
sys     0m0.323s
```

Figure 54

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m34.598s
user    1m43.561s
sys     0m0.162s
```

Figure 55

Average(real time): 34.631s

Concurrency: n = 10,000,000 and thread = 8

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m27.890s
user    1m44.063s
sys     0m0.159s
```

Figure 56

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m29.067s
user    1m48.741s
sys     0m0.244s
```

Figure 57

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m28.267s
user    1m44.050s
sys     0m0.187s
```

Figure 58

Average(real time): 28.408s

Concurrency: n = 100,000,000 and thread = 16

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m26.711s
user    1m44.525s
sys     0m0.143s
```

Figure 59

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m27.203s
user    1m46.625s
sys     0m0.326s
```

Figure 60

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m27.163s
user    1m47.100s
sys     0m0.171s
```

Figure 61

Average(real time): 27.025s

Concurrency: n = 100,000,000 and thread = 32

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m26.574s
user    1m44.974s
sys     0m0.326s
```

Figure 62

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m27.182s
user    1m47.598s
sys     0m0.171s
```

Figure 63

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m27.121s
user    1m47.283s
sys     0m0.154s
```

Figure 64

Average(real time): 26.959s

Concurrency: n = 100,000,000 and thread = 64

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m27.525s
user    1m49.052s
sys     0m0.252s
```

Figure 65

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m27.123s
user    1m47.721s
sys     0m0.180s
```

Figure 66

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100000000

The number of primes less than 100000000: 5761455

The number of twin primes less than 100000000: 440312

real    0m26.837s
user    1m46.257s
sys     0m0.335s
```

Figure 67

Average(real time): 26.162s

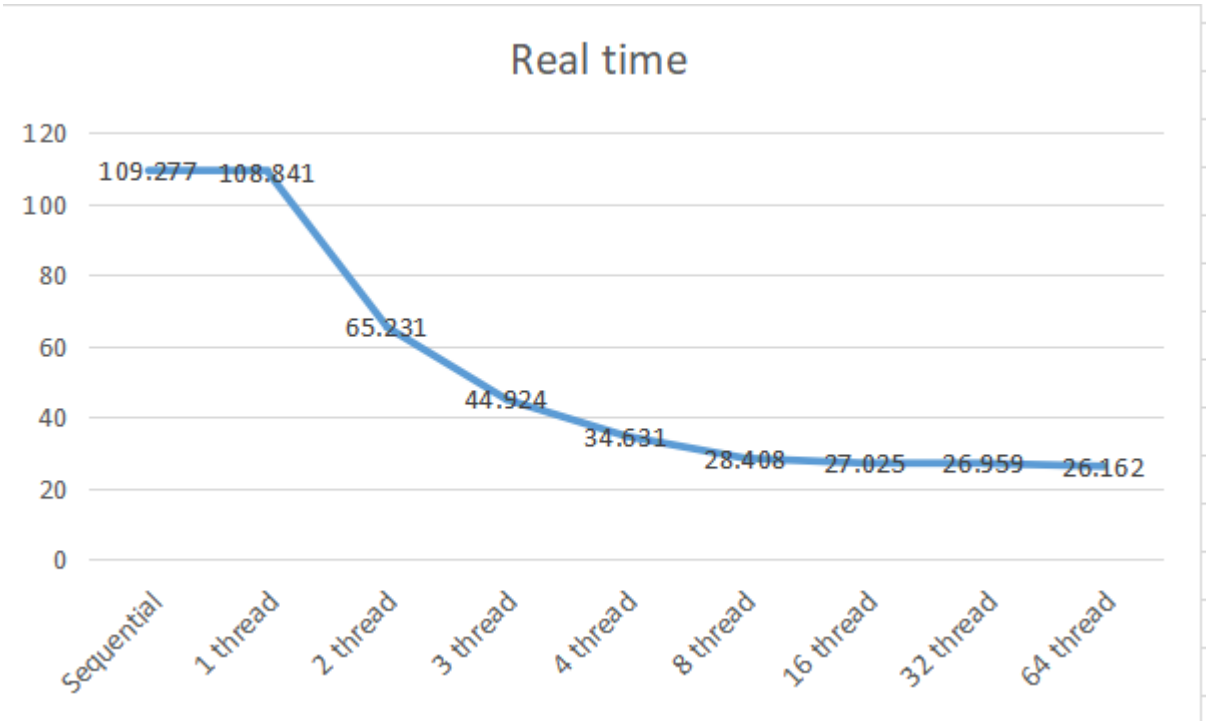


Figure 69

According to Figure 46, as the number of threads increases, the execution time decreases. So concurrency and parallel can improve the running speed of the program. But when the number is greater than 8, the number of threads seems to have no effect on execution time.

Absolute speedup and relative speedup（n = 100,000,000）

Formula:

- Absolute speedup = Sequential Time/Parallel Time
- Relative speedup = Single Thread/Parallel Time

Sequential time: 109.277s
Concurrent time (1 Thread): 108.841s
Absolute speedup = 109.277/108.841 = 1.004
Parallel time (2 thread): 65.231s
Absolute speedup = 109.277/65.231 = 1.675
Relative speedup = 108.841/65.231 = 1.669
Parallel time (3 thread): 44.924s
Absolute speedup = 109.277/44.924 = 2.433
Relative speedup = 108.841/44.924 = 2.423
Parallel time (4 thread): 34.631s
Absolute speedup = 109.277/34.631 = 3.155
Relative speedup = 108.841/34.631 = 3.143
Concurrent time (8 thread): 28.408s
Absolute speedup = 109.277/28.408 = 3.847
Relative speedup = 108.841/28.408 = 3.831
Concurrent time (16 thread): 27.025s
Absolute speedup = 109.277/27.025 = 4.044
Relative speedup = 108.841/27.025 = 4.027
Concurrent time (32 thread): 26.959s
Absolute speedup = 109.277/26.959 = 4.053
Relative speedup = 108.841/26.959 = 4.037
Concurrent time (64 thread): 26.162s
Absolute speedup = 109.277/26.162 = 4.177
Relative speedup = 108.841/26.162 = 4.160

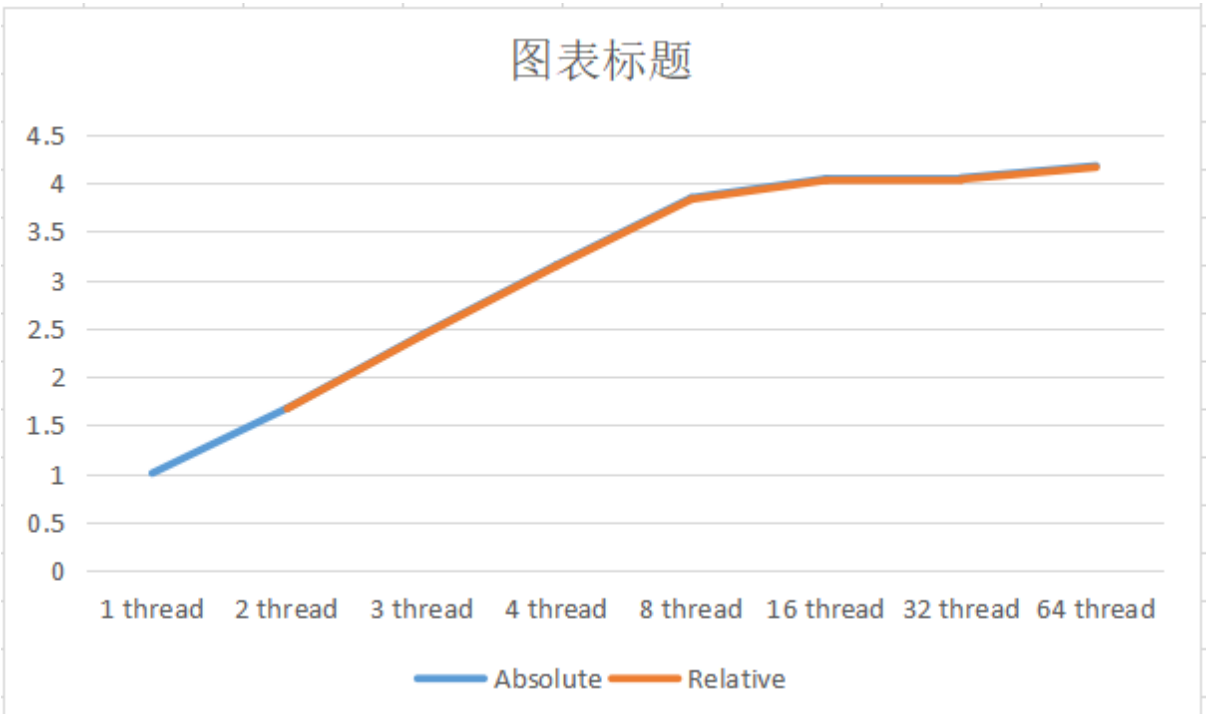


Figure 69 Absolute speed and relative speed with different thread when n = (100,000,000)

Context Switch

According to the result of the calculation above, we know that multithreading can decrease the execution time of the program. But when multithreading is running, we need to switch threads. The technical term of this situation is context switch.

As the figure shown below, we know that context switching takes time. Because when the kernel switches threads, it must saving every settings before running another thread.

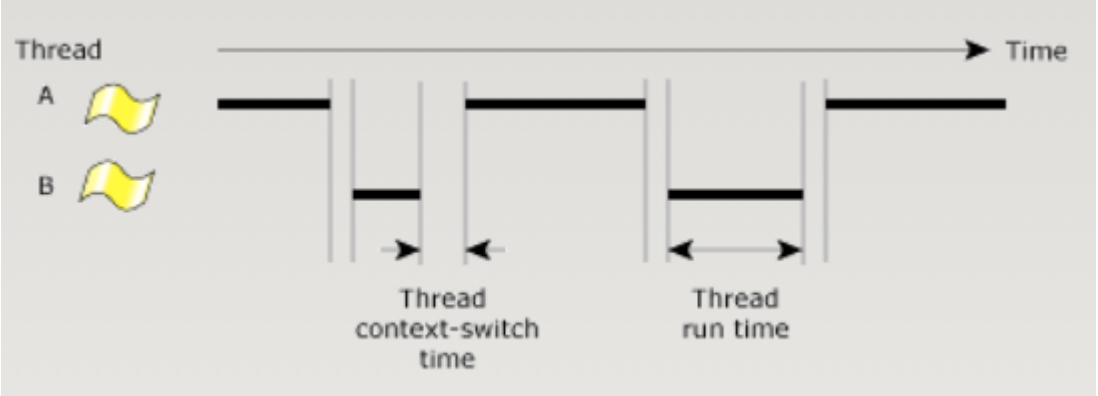


Figure 70 Context Switch

So when there is not a lot of computation, we run the program sequentially and concurrent with 64 threads. And the result is shown below. According to these two pictures, we found that it takes less time when the program runs sequentially. That's because time used to context switching is greater than calculation when the computation is small.

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100

The number of primes less than 100: 25

The number of twin primes less than 100: 8

real    0m0.002s
user    0m0.002s
sys     0m0.000s
```

Figure 71 Sequential when n=100

```
momoko@momoko-VirtualBox:~/Desktop/ConcurrencyProject$ time ./momoko 100

The number of primes less than 100: 25

The number of twin primes less than 100: 8

real    0m0.006s
user    0m0.000s
sys     0m0.010s
```

Figure 72 Concurrency with 64 thread when n=100

Race Condition

There is a common problem when we write the multithreaded program called race conditions which occurs when two or more threads access and manipulate the same data at the same time and the result of manipulation depends on the order of access. But we cannot control the order of access, because it was controlled by the operating system. Therefore, race conditions will result in a different result each time we run the application.

#pragma omp atomic is to ensure that only one thread can manipulate data at a time, which can ensure the result we calculate is correct. In other words, it ensures that no other thread is allowed to operate until one thread completes.

The following figures shows the result of calculation when comment #pragma omp atomic. We found that the number of twin primes is different every time.

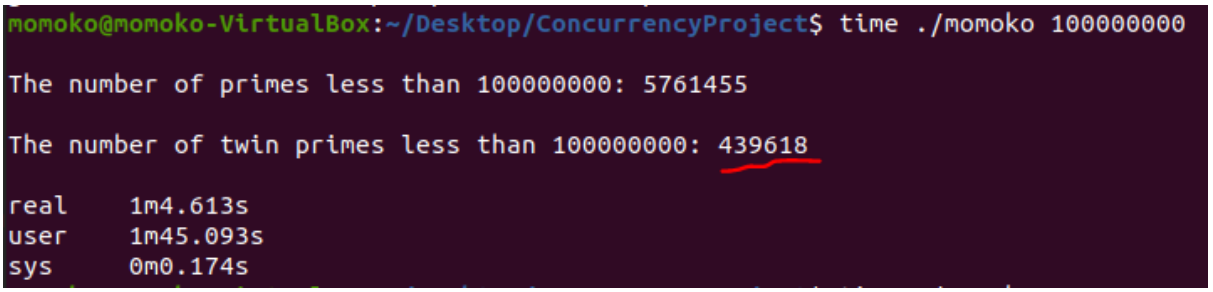


Figure 72 n = 100,000,000 and threads = 8

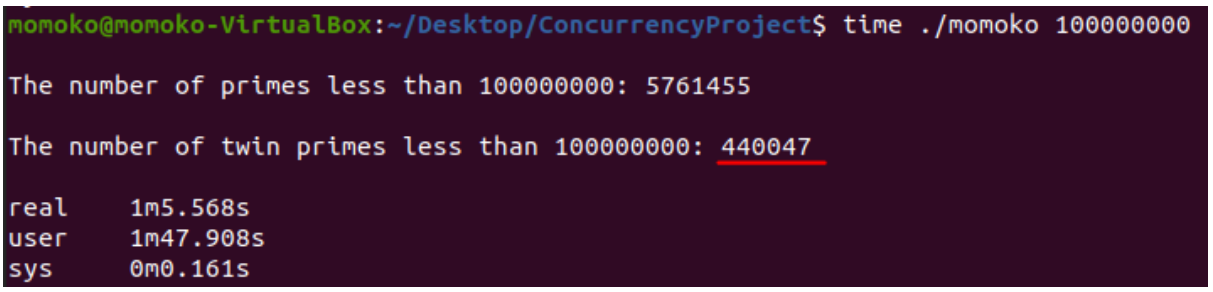


Figure 73 n = 100,000,000 and threads = 8

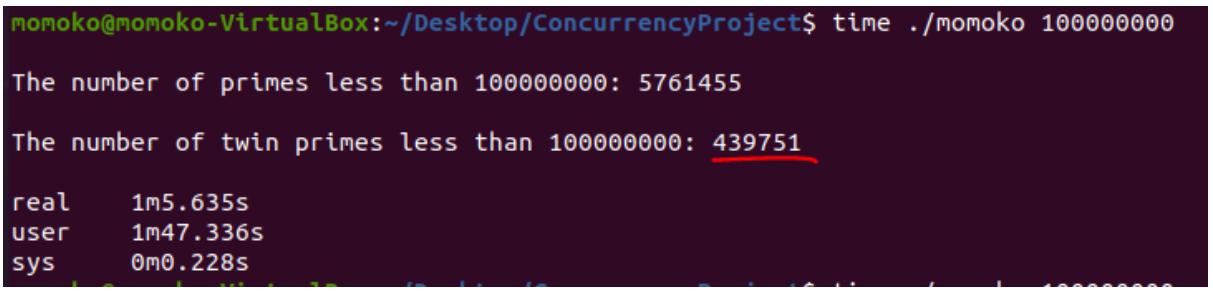


Figure 74 n = 100,000,000 and threads = 8

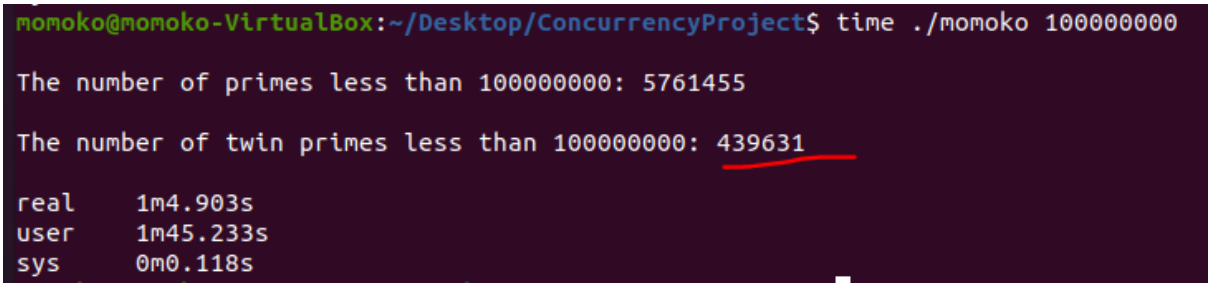


Figure 75 n = 100,000,000 and threads = 8

Conclusion

Multithreading can improve the efficiency when there is a large amount of computations. But we need to avoid race conditions when we develop multithreaded application.