CONCURRENCY PROJECT

Prime numbers and their twins

Abstract

This document analyses a C++ program in sequential, parallel and concurrent execution. The C++ program calculates the number of primes and their twin pairs less than the input number.

Tataru Theodora

C00231174

CONTENTS

TABLE OF FIGURES

INTRODUCTION

This report was produced to observe the performance of a program in sequential, parallel and concurrent execution.

The program outputs the number of primes and the number of twin prime pairs less than the input number.

Graphs, absolute and relative speeds were detailed in this report to serve the analysis of the prime program's execution to determine the most efficient performance the program can achieve, taking advantage of the OpenMP library.

The program was developed in C++ and executed on a computer with 4 CPUs Intel Core i3.

ABOUT OPENMP

The most popular type of programming is sequential, a case in which the executing code runs each line of code in sequential order.

OpenMP is a library dedicated to parallel programming using symmetric multi-processor or shared-memory processors. When a program runs multiple threads in its execution, the threads of that process share the same memory and data (Mattson, n.d.).

The OpenMP library achieves the parallelism of a process exclusively through threads. A thread is the most basic unit of CPU utilization. Threads exist within a process, where if a process is single-threaded, it has one control low, and when a process is multi-threaded contains several flows of control within the same address space (Pandit, 2021).

To execute a program that is multi-threaded in parallel, the number of threads needs to be less or equal to the number of CPU cores on the system. If the number of threads exceeds the number of cores, some threads will execute concurrently. (Mattson, n.d.)

The beauty of using OpenMP is that OpenMP is not an automatic programming mode, offering the programmers complete control over threads and parallelization (Pandit, 2021).

When using OpenMP, the program uses one thread in the sequential section of the code and several threads for the code sections that run in parallel. One thread that runs from the beginning of the execution until the end is called the **main thread**, and the code sections for parallel execution cause additional threads to fork. These threads are called **slaves** (Mattson, n.d.).

Using OpenMP, to mark a section of the code that needs to be parallelized, the **#pragma omp** command is used. When the code's execution reaches this statement, additional slave threads will fork, each thread executing independent that part of the code in parallel or concurrent with the other threads (Mattson, n.d.). **#pragma** is a preprocessing directive.

Each thread has a unique ID that can be obtained with the **omp_get_thread_num()** command. The master thread always has the id 0 (zero).

Before explaining some OpenMP commands and how they behave, some concepts need to be understood, such as:

- All variables and data are shared among threads by default. Extra care needs to be taken if the threads update the data to avoid race conditions.
  - **private(variableOne, variableTwo)**
    When using this command in the same line of **#pragma omp parallel** ensures that each thread has its own copy of the variables described as private
  - **shared(variableThree, variableFour)**
    By default, all variables are shared between threads, but they can also be explicitly declared as shared.
  - **reduction(+:sum)**
    The reduction command ensures that all threads combine their result into a single value. Each thread creates a copy of the reduction variable, and at the end, their values are combined in one variable. The operands available for reduction are: +, -, *, /, &, |, ^, && and ||.
- To avoid race conditions between threads, two mechanisms can be applied:
  - **#prag omp critical**
    The critical command describes a portion of the code that can only be executed by one thread at a time.
  - **#pragma omp atomic**
    An atomic command ensures that a variable is updated in a single, unbreakable step. Only certain operators can be used with the atomic command, such as x++, ++x, x--, --x, -, +, *, /, &, |, << or >>. The atomic command cannot protect any other statements.
- Scheduling is a big part of OpenMP for parallel loops as the amount of work in each iteration is not always the same for each thread;
  - **schedule(dynamic)**
    When the dynamic scheduling is used, the iterations are put into a queue, and as threads become idle, they are forced to take iterations from the queue

- **schedule(static)**
  When static scheduling is used, the iterations are divided evenly between threads

OpenMP is one of the most efficient and low-level way of parallelizing code, as it hides the low-level details allowing the developer to mark the beginning of parallel code with simple commands (Mattson, n.d.):

- **#pragma omp parallel**
  The code following this command inside the curly brackets runs in parallel
- **#pragma omp parallel for**
  This command is written before a **for** loop in the code and divides the iterations of the loop between threads. All variables inside the loop are shared except for the "**for"** loop control variable.
  Before exiting the loop, all threads must wait for all threads to complete the execution.
- **#pragma omp parallel sections** and **#pragma omp section**
  Parallel sections are used to execute independent tasks that are difficult to manipulate with a parallel **for** loop.
- **#pragma omp task** and **#pragma omp single**
  Tasks are flexible, but the order of execution is not guaranteed; therefore, these tasks must be able to run in any order and are used primarily for recursive methods or where the number of iterations is unknown.

There are many other features that OpenMP can provide to developers, but for this assessment purpose, the ones explained above are the main important ones.

OPENMP CONSTRUCTS COST

Using OpenMP comes with its costs; as an example, the following costs were calculated on an old Intel 3.0 GHz machine and detailed by *"Multi-core programming: increasing performance through software threading" –Akhter & Roberts, Intel Press, 2006.*

Cost of OpenMP constructs

Construct	microseconds
parallel	1.5
barrier	1.0
schedule (static)	1.0
schedule (guided)	6.0
schedule (dynamic)	50
ordered	0.5
single	1.0
reduction	2.5
atomic	0.5
critical	0.5
lock/unlock	0.5

Figure 1 "OpenMP Construct Costs"
Source: (Pandit, 2021)

PERFORMANCE

To measure the performance of the program executing the **time ./executable** command is used. The command will output at the very end of the execution an output similar to:

Parallel execution for N = 1000000: 189186515 milliseconds
real 3m9.192s
user 6m2.502s
sys 0m0.626s

Where (Ferreira, 2020):

- **Real** – is wall clock time, from the time the execution starts until the end. As if it was measured with a stopwatch
- **User** – is the amount of time that the CPU spends in user mode within the process. Other processes time is not included here
- **Sys** – is the amount of time the CPU spends in the Kernel within the process, the amount of time the CPU spends in system calls within the Kernel

The rule of thumb, to interpret the time outputs, says that if (Levon, 2012):

- **Real < User** – the process is CPU bound, and takes advantage of the parallel execution on multiple cores or CPUs
- **Real ~= User** – the process is CPU bound, and takes no advantage of the parallel execution
- **Real > User** – the process is I/O bound, and the execution on multiple cores has no advantage

This report focuses on the **real** time output to measure the performance of the program in sequential, parallel and concurrent execution.

ABSOLUTE AND RELATIVE SPEED

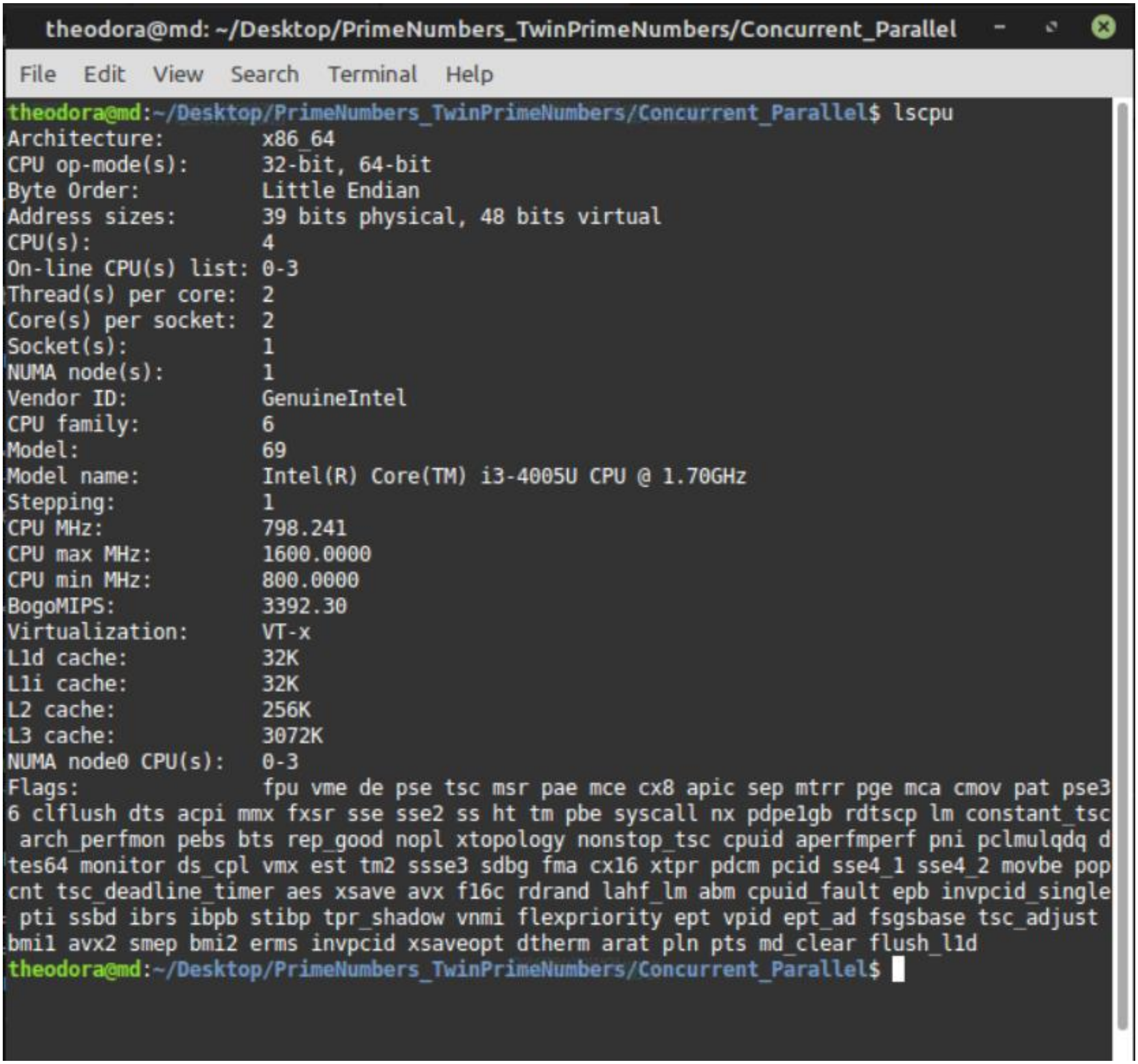
The absolute speed is calculated with the following formula:

$$speed = \frac{sequential\ time}{concurrent/parallel\ \ time\ (multiple\ cores)}$$

The relative speed is calculated with the following formula:

$$speed = \frac{concurrent\ speed\ (1\ thread)}{concurrent/parallel\ \ time\ (multiple\ cores)}$$

ABOUT THE SYSTEM



```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers/Concurrent_Parallel
File  Edit  View  Search  Terminal  Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers/Concurrent_Parallel$ 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:  2
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               69
Model name:          Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz
Stepping:            1
CPU MHz:             798.241
CPU max MHz:         1600.0000
CPU min MHz:         800.0000
BogoMIPS:            3392.30
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            3072K
NUMA node0 CPU(s):   0-3
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse3
6 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc
 arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq d
tes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe pop
cnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm cpuid_fault epb invpcid_single
 pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust
bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm arat pln pts md_clear flush_l1d
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers/Concurrent_Parallel$
```

Figure 2 "System configuration command line"

Figure 3 "CPUs"

As seen in **Figure 2** and **Figure 3,** the system on which the program executes has four cores, and an Intel i3 CPU.

THE CODE

Check if a number is prime

```
bool isPrime(int n)
    int number = n;
    int half = number/2;
    if(number <= 1) return false;
    if(number <= 3) return true;
    if(number%3 == 0) return false;
    if(number%2 == 0) return false;
    if(number%5 == 0) return false;
    for(int i = 5; i<=half; i++)
        if(number%i==0)
            return false;
    return true;
```

This method returns true if a number is prime and false if the number in question is not prime. This method is not going to be parallelized, as it contains a break statement. In a sequential code, the break statement decreases execution time; however, using OpenMP to parallelize the execution, the break statement becomes an obstacle.

Several other implementations could have been used to void the break statement problem, such as a while loop or a flag to terminate the execution of the loop, but testing the execution time with the alternatives, the program's output was in some cases erroneous, without finding the cause. Therefore, this method is the only method from the program that is not parallelized.

```
/*! \fn void numbersPrimeLessThen(int N)
    \param N The number that represents the upper limit of checking prime numbers
    \brief Method that returns all the prime numbers less than N
*/
void numbersPrimeLessThen(int N)
    int count = 0;
    pragma omp parallel for num_threads(4) shared(count)
    for(int number = 2; number <= N; number++)
        print Thread ::numbersPrimeLessThen: omp_get_thread_num() ;
        if(isPrime(number))
            #pragma omp atomic
            count++;
    printf Number of primes less than N, count
```

This method returns the number of primes less than N.

```
/*! \fn void Primes::twins(int N)
    \param N The number that represents the upper limit of checking for prime twins
    \brief Method checks if 2 prime numbers are twins from the vector of primes
*/
void twins(int N)
    std::cout << "Twins: " << std::endl;
    int sum = 0;
    #pragma omp parallel for num_threads(4)
    for(int i = 2; i <= N; i++)
        if(isPrime(i) && isPrime(i+2))
            printf("\n(%d,%d)",i,i+2);
            #pragma omp atomic
            sum++;
    print Numbers of prime twins: sum);
```

This method returns the number of prime twin pairs less than N.

DEMONSTRATION

According to (PrimePages, 2020) and (PANDE, 2016) there are 78,498 primes and 8,169 prime twins less than 1,000,000, 168 primes and 35 twins less than 1,000 and 25 primes and 8 twins less than 100.

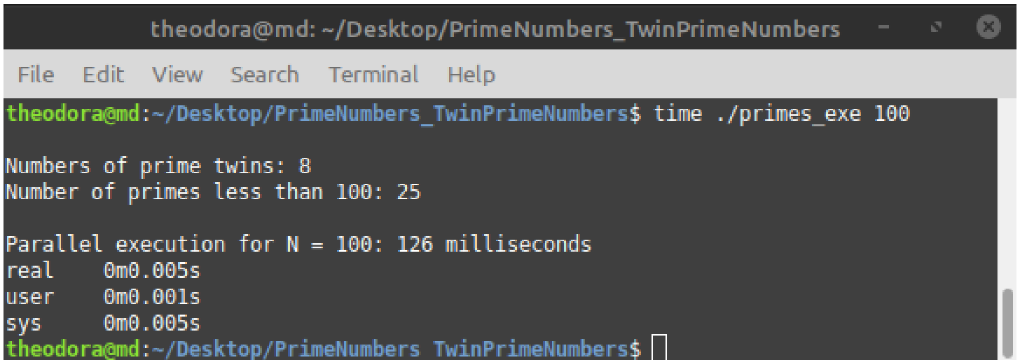


Figure 4 "Number or primes and twins less than 100"

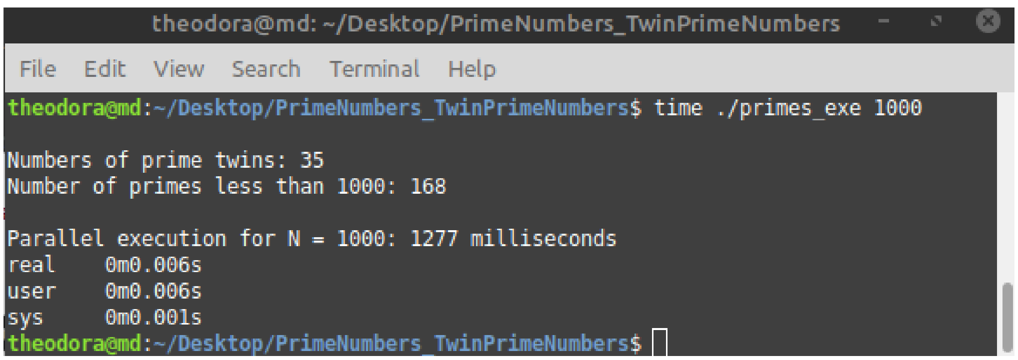


Figure 5 "Number or primes and twins less than 1,000"

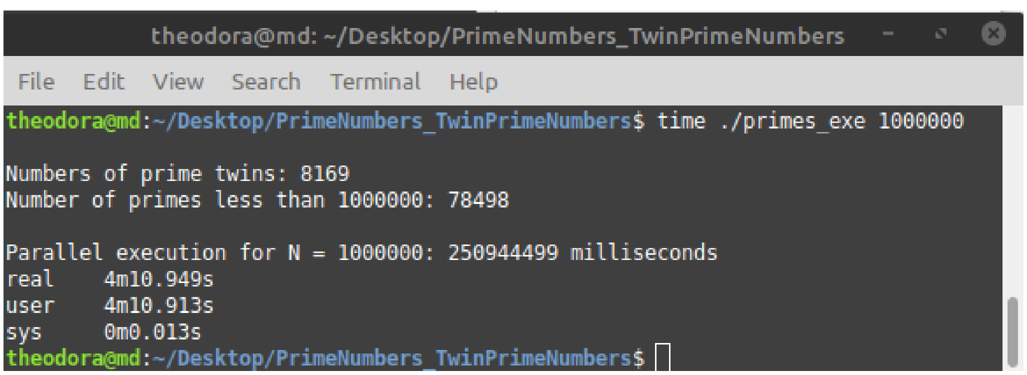


Figure 6 "Number or primes and twins less than 1,000,000"

As seen in **Figure 4, Figure 5** and **Figure 6** the "Primes" program outputs are correct answers.

SEQUENTIAL

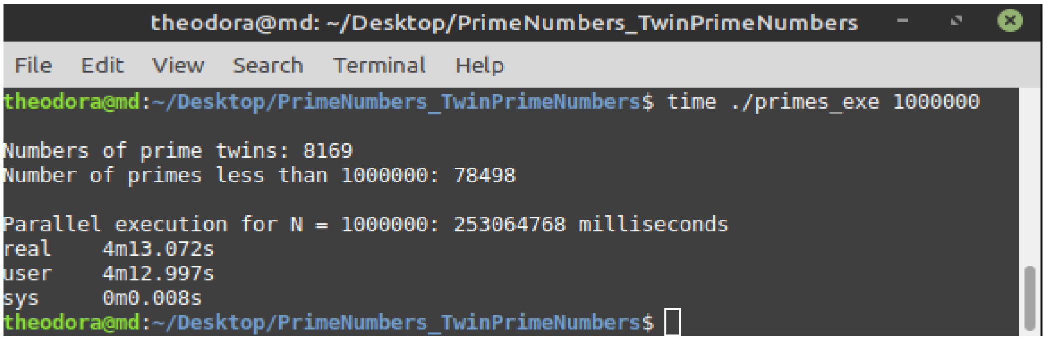


Figure 7

Multiple Runs (real time):

- 4m13.072s
- 4m10.353s
- 4m7.642s

Average: 4m10.355s

SEQUENTIAL VS PARALLEL VS CONCURRENT

NUMBERSPRIMELESSTHEN(1000000)

For this experiment, the method numbersPrimesLessThen() executes in parallel and concurrently in some cases, while twins() executes sequentially.

CONCURRENT – 1 THREAD

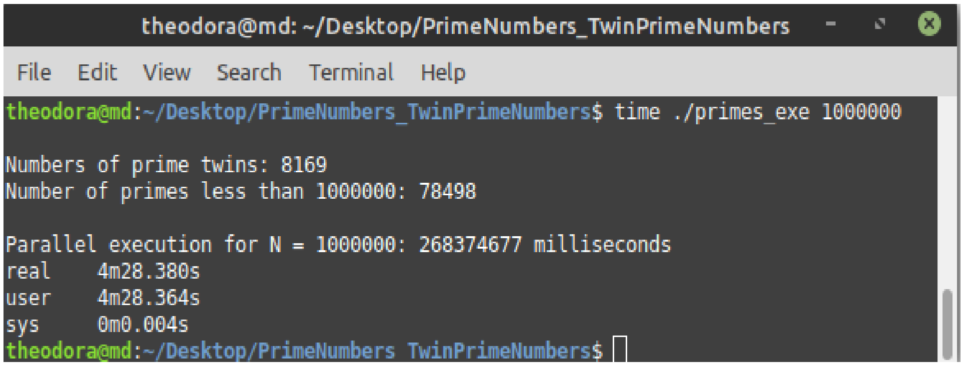


Figure 8

Multiple Runs (real time):

- 4m28.380s
- 4m27.416s
- 4m27.570s

Average: 4m27.788s

PARALLEL – 2 THREADS

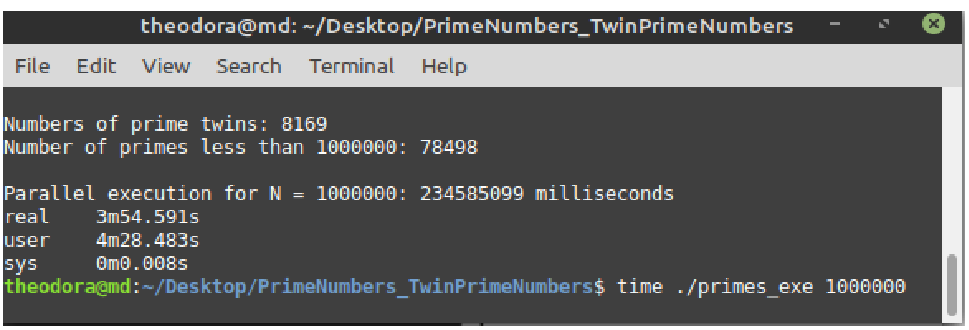


Figure 9

Multiple Runs (real time):

- 3m54.591s
- 3m54.084s
- 3m54.073s

Average: 3m54.249s

PARALLEL – 3 THREADS

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File  Edit  View  Search  Terminal  Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 1000000

Numbers of prime twins: 8169
Number of primes less than 1000000: 78498

Parallel execution for N = 1000000: 216444312 milliseconds
real	3m36.450s
user	4m42.817s
sys	0m0.016s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 10

Multiple Runs (real time):

- 3m36.450s
- 3m36.124s
- 3m36.476s

Average: 3m36.350s

PARALLEL – 4 THREADS

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File  Edit  View  Search  Terminal  Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 1000000

Numbers of prime twins: 8169
Number of primes less than 1000000: 78498

Parallel execution for N = 1000000: 201319363 milliseconds
real	3m21.325s
user	5m19.823s
sys	0m0.072s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 11

Multiple Runs (real time):

- 3m21.325s
- 3m21.426s
- 3m22.247s

Average: 3m21.666s

CONCURRENT – 8 THREADS

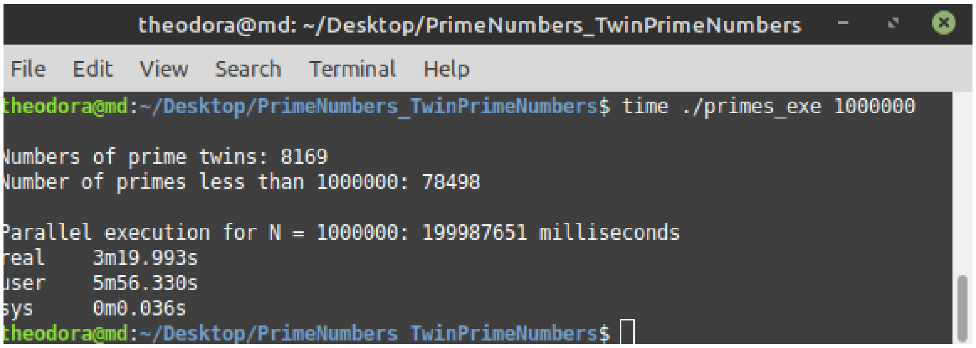


Figure 12

Multiple Runs (real time):

- 3m19.993s
- 3m17.918s
- 3m19.442s

Average: 3m19.117s

CONCURRENT – 16 THREADS

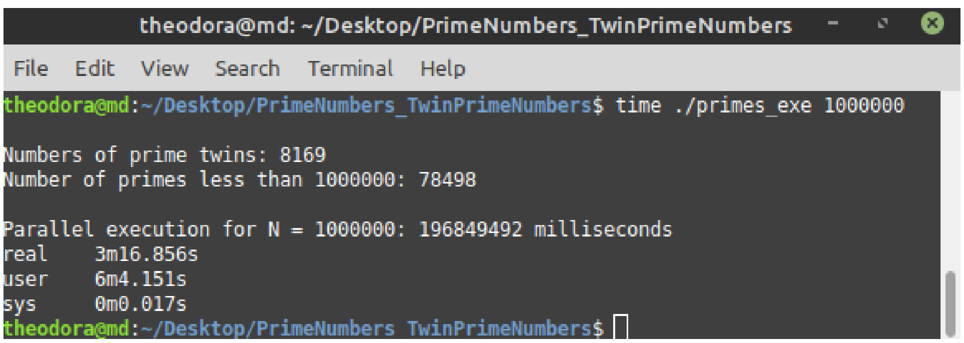


Figure 13

Multiple Runs (real time):

- 3m16.856s
- 3m18.072s
- 3m17.844s

Average: 3m17.590s

CONCURRENT – 32 THREADS

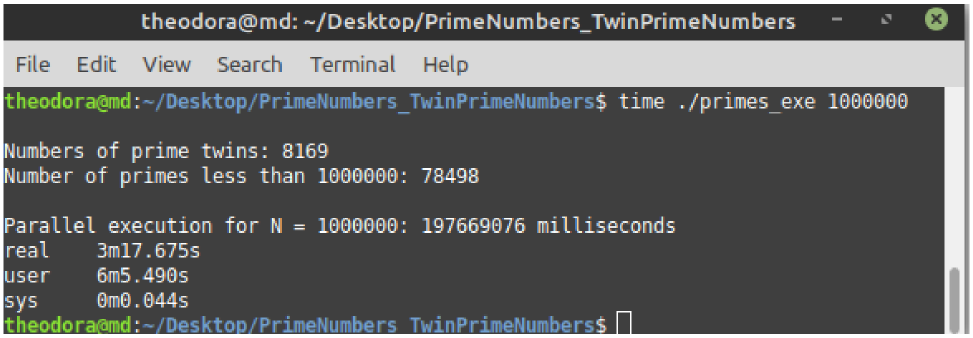


Figure 14

Multiple Runs (real time):

- 3m17.675s
- 3m16.900s
- 3m17.088s

Average: 3m17.221s

CONCURRENT – 64 THREADS

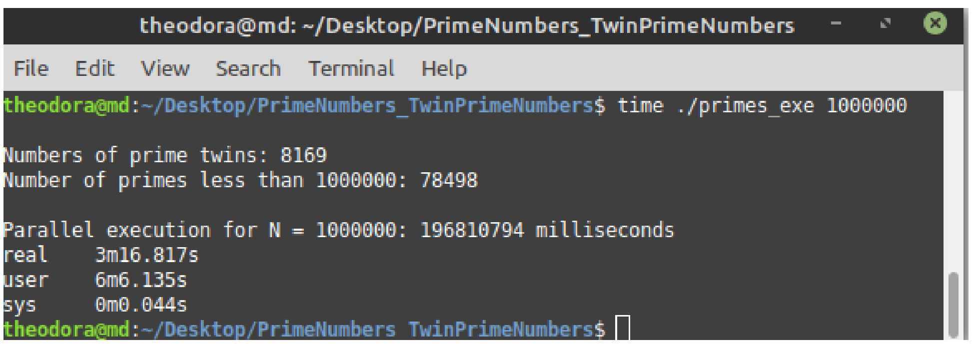


Figure 15

Multiple Runs (real time):

- 3m16.817s
- 3m16.704s
- 3m17.003s

Average: 3m16.841s

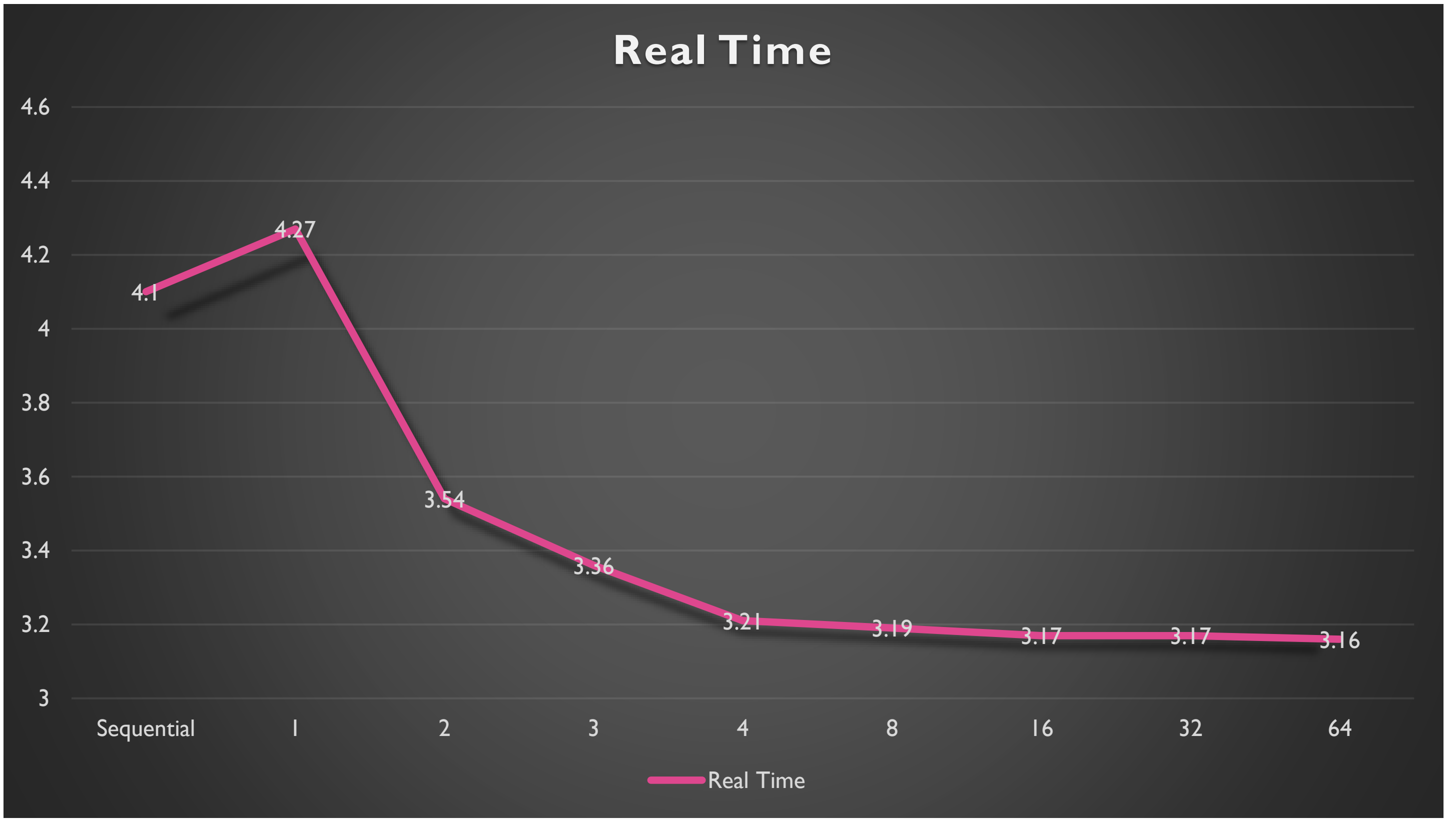


Figure 16

As seen in **Figure 16**, as the number of threads increases the execution time decreases, resulting that the number of threads has a positive impact on the execution time of the program.

More than 16 threads do not seem to benefit the execution.

ABSOLUTE AND RELATIVE SPEED

Formula:

ABSOLUTE SPEED = TIME SEQUENTIAL / TIME PARALLEL

RELATIVE SPEED = SINGLE_THREAD/TIME_PARALLEL

Sequential Time

4 minutes and 10 seconds = 250 seconds

1 Threads

4 minutes and 27 seconds = 267 seconds

Absolute Speed: 250/267 = 0.936

2 Threads

3 minutes and 54 seconds = 234 seconds

Absolute Speed: 250/234 = 1.068

Relative Speed: 267/234 = 1.141

3 Threads

3 minutes and 36 seconds = 216 seconds

Absolute Speed: 250/216 = 1.157

Relative Speed: 267/216 = 1.236

4 Threads

3 minutes and 21 seconds = 201 seconds

Absolute Speed: 250/201 = 1.243

Relative Speed: 267/201 = 1.328

8 Threads

3 minutes and 19 seconds = 199 seconds

Absolute Speed: 250/199 = 1.256

Relative Speed: 267/199 = 1.341

16 Threads

3 minutes and 17 seconds = 197 seconds

Absolute Speed: 250/197 = 1.269

Relative Speed: 267/197 = 1.355

32 Threads

3 minutes and 17 seconds = 197 seconds

Absolute Speed: 250/197 = 1.269

Relative Speed: 267/197 = 1.355

64 Threads

3 minutes and 16 seconds = 196 seconds

Absolute Speed: 250/196 = 1.275

Relative Speed: 267/196 = 1.362

TWINS(1000000)

For this experiment, the twin() method executes in parallel and concurrently in some cases, while numbersPrimeLessThen() executes sequentially.

CONCURRENT – 1 THREAD

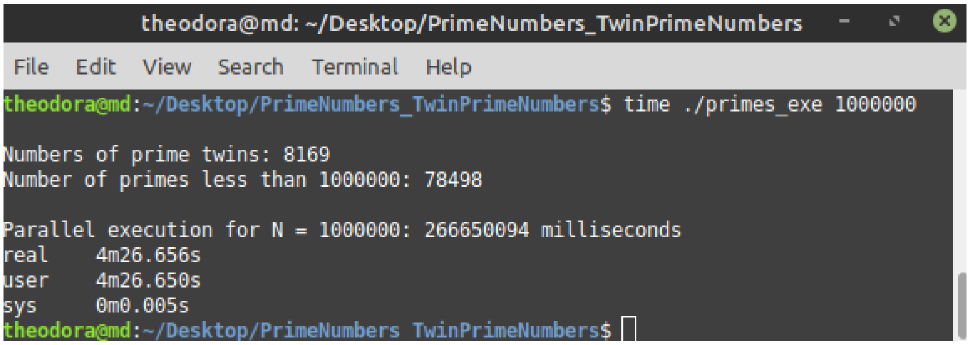


Figure 17

Multiple Runs (real time):

- 4m26.656s
- 4m27.394s
- 4m26.717s

Average: 4m26.922s

PARALLEL – 2 THREADS

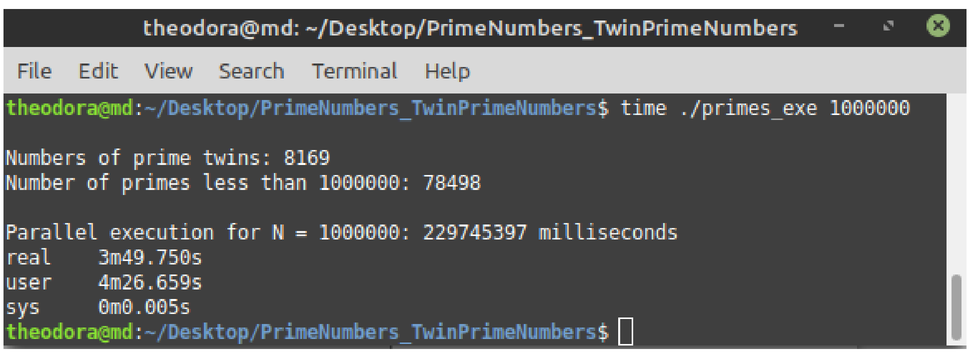


Figure 18

Multiple Runs (real time):

- 3m49.750s
- 3m50.116s
- 3m49.353s

Average: 3m49.739s

PARALLEL – 3 THREADS

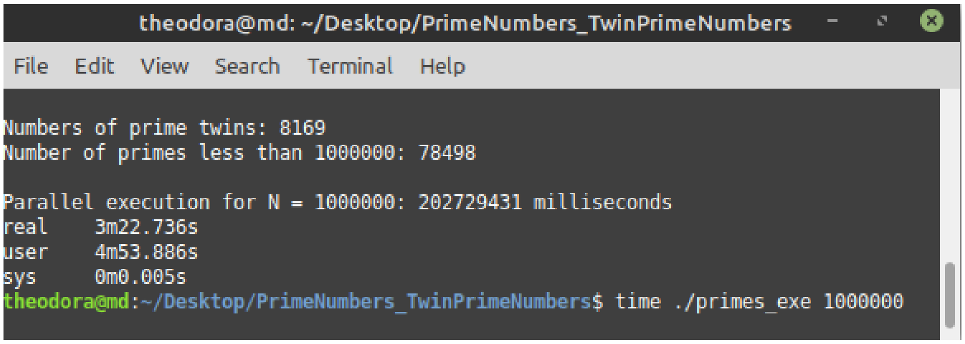


Figure 19

Multiple Runs (real time):

- 3m22.736s
- 3m22.769s
- 3m22.464s

Average: 3m22.656s

PARALLEL – 4 THREADS

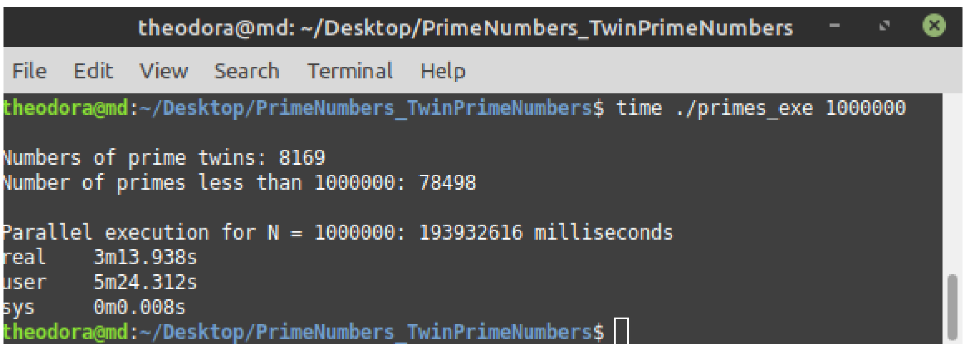


Figure 20

Multiple Runs (real time):

- 3m13.938s
- 3m13.629s
- 3m13.820s

Average: 3m13.795s

CONCURRENT – 8 THREADS

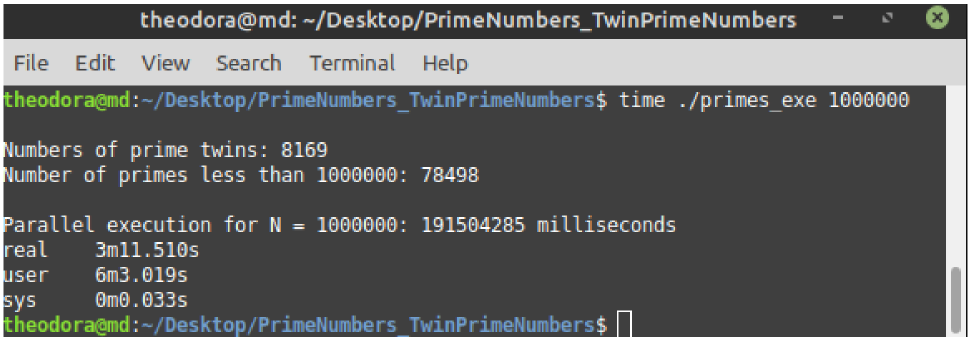


Figure 21

Multiple Runs (real time):

- 3m11.510s
- 3m11.183s
- 3m11.319s

Average: 3m11.337s

CONCURRENT – 16 THREADS

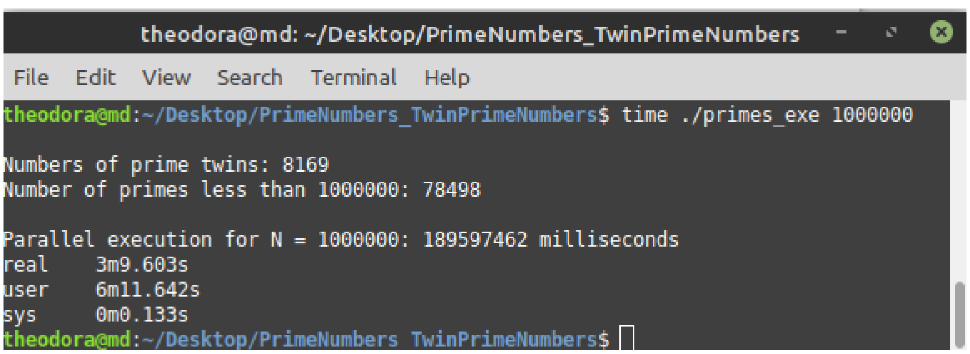


Figure 22

Multiple Runs (real time):

- 3m9.603s
- 3m9.194s
- 3m9.525s

Average: 3m9.440s

CONCURRENT – 32 THREADS

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File  Edit  View  Search  Terminal  Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 1000000

Numbers of prime twins: 8169
Number of primes less than 1000000: 78498

Parallel execution for N = 1000000: 188907652 milliseconds
real    3m8.914s
user    6m14.440s
sys     0m0.013s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 23

Multiple Runs (real time):

- 3m8.914s
- 3m8.781s
- 3m9.103s

Average: 3m8.932s

CONCURRENT – 64 THREADS

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File  Edit  View  Search  Terminal  Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 1000000

Numbers of prime twins: 8169
Number of primes less than 1000000: 78498

Parallel execution for N = 1000000: 189251973 milliseconds
real    3m9.258s
user    6m15.207s
sys     0m0.040s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 1000000
```

Figure 24

Multiple Runs (real time):

- 3m9.258s
- 3m8.830s
- 3m8.639s

Average: 3m8.909s

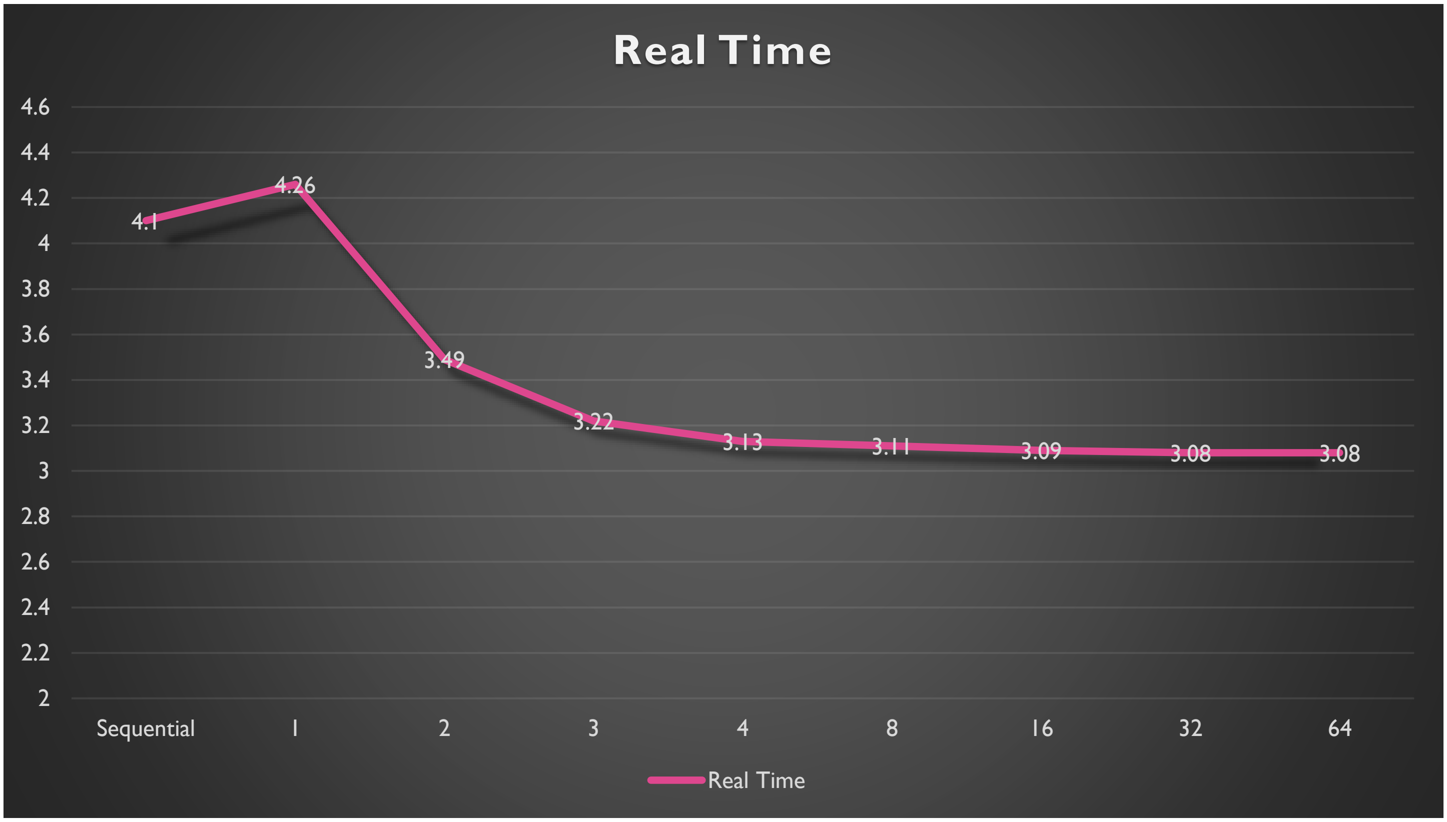


Figure 25

As seen in **Figure 25**, as the number of threads increases, the execution time decreases. More than four threads used for the **for** loop do not seem to impact the execution significantly.

ABSOLUTE AND RELATIVE SPEED

Formula:

ABSOLUTE SPEED = TIME SEQUENTIAL / TIME PARALLEL

RELATIVE SPEED = SINGLE_THREAD/TIME_PARALLEL

Sequential Time

4 minutes and 10 seconds = 250 seconds

1 Threads

4 minutes and 26 seconds = 266 seconds

Absolute Speed: 250/266 = 0.939

2 Threads

3 minutes and 49 seconds = 229 seconds

Absolute Speed: 250/229 = 1.091

Relative Speed: 266/229 = 1.161

3 Threads

3 minutes and 22 seconds = 202 seconds

Absolute Speed: 250/202 = 1.237

Relative Speed: 266/202 = 1.316

4 Threads

3 minutes and 13 seconds = 193 seconds

Absolute Speed: 250/193 = 1.295

Relative Speed: 266/193 = 1.378

8 Threads

3 minutes and 11 seconds = 191 seconds

Absolute Speed: 250/191 = 1.308

Relative Speed: 266/191 = 1.392

16 Threads

3 minutes and 9 seconds = 189 seconds

Absolute Speed: 250/189 = 1.322

Relative Speed: 266/189 = 1.407

32 Threads

3 minutes and 8 seconds = 188 seconds

Absolute Speed: 250/188 = 1.329

Relative Speed: 266/188 = 1.414

64 Threads

3 minutes and 8 seconds = 173 seconds

Absolute Speed: 250/188 = 1.329

Relative Speed: 266/188 = 1.414

PARALLELIZING TWINS() AND NUMBERSPRIMELESSTHEN()

For this section, both methods (***twins() and numbersPrimeLessTen***) were parallelized, calculating the number of primes and their twins less than 1,000,000. Each number of threads is tested with dynamic and static scheduling.

1 THREAD

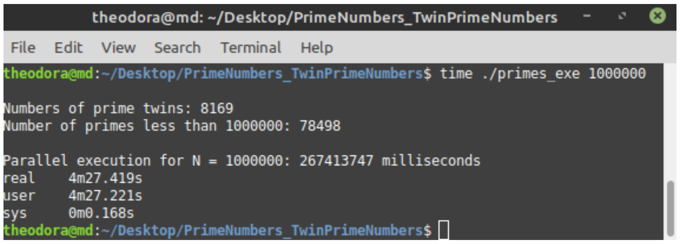


Figure 27 "Static scheduling

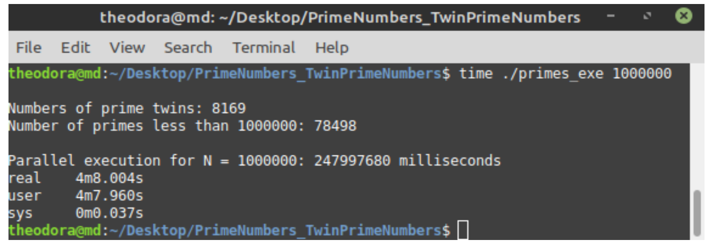


Figure 26 "Dynamic scheduling

2 THREADS

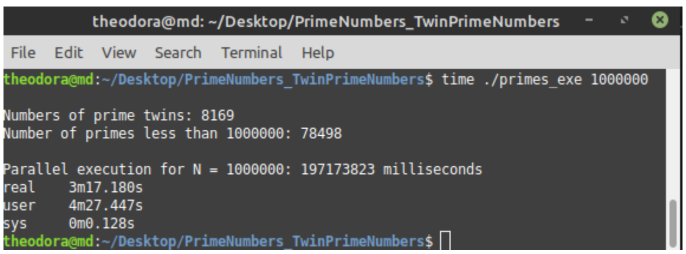


Figure 29 "Static scheduling"

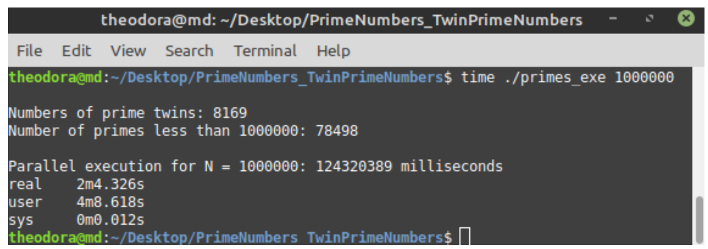


Figure 28 "Dynamic scheduling"

3 THREADS

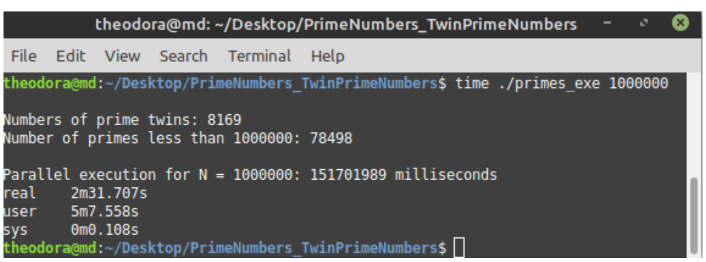


Figure 30 "Static scheduling"

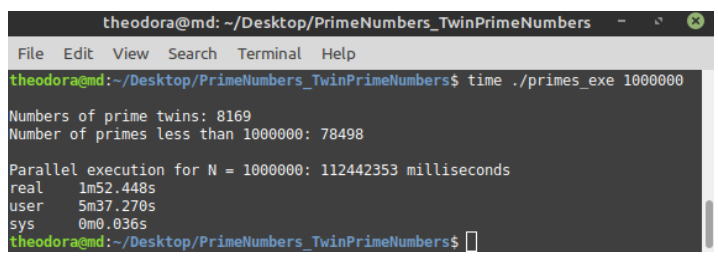


Figure 31 "Dynamic Scheduling"

4 THREADS

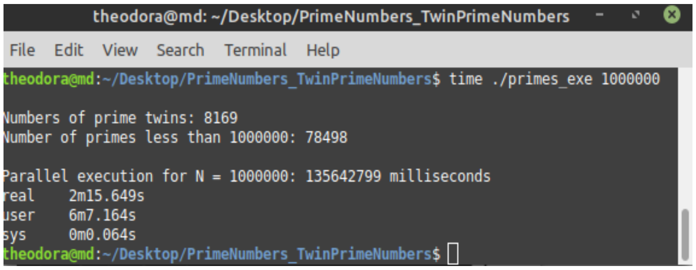


Figure 33 "Static scheduling"

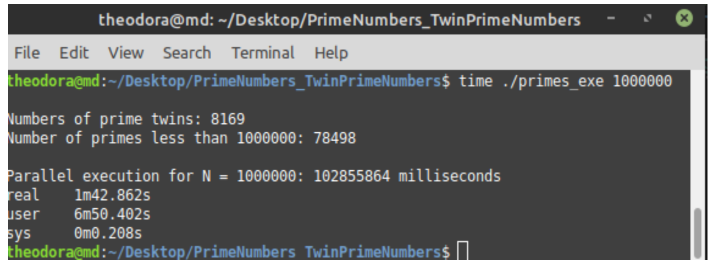


Figure 32 "Dynamic scheduling"

8 THREADS

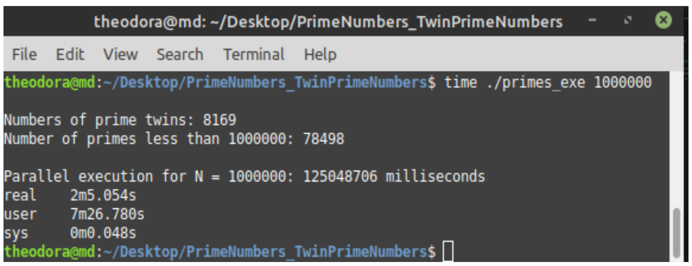


Figure 35 "Static scheduling"

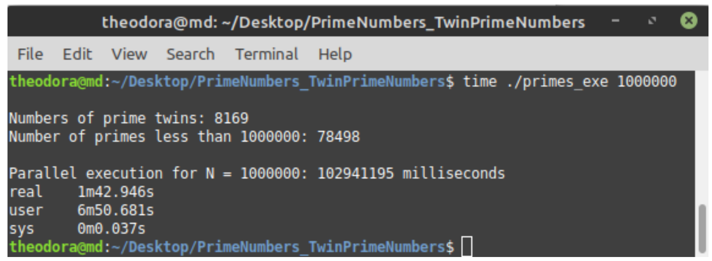


Figure 34 "Dynamic scheduling"

16 THREADS

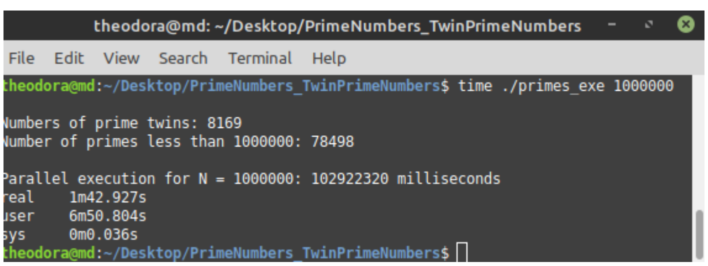


Figure 37 "Static scheduling"

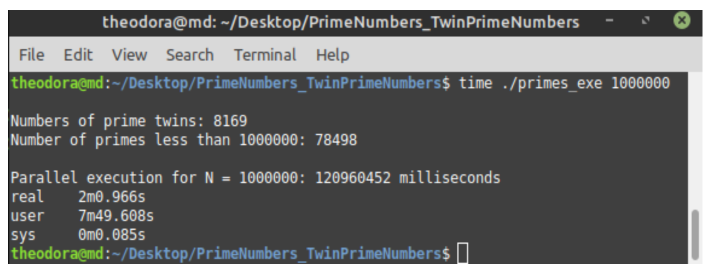


Figure 36 "Dynamic scheduling"

32 THREADS

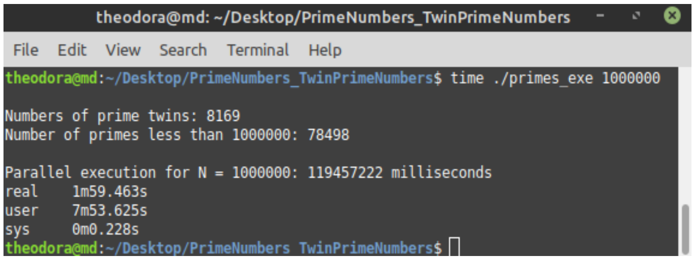


Figure 39 "Static scheduling"

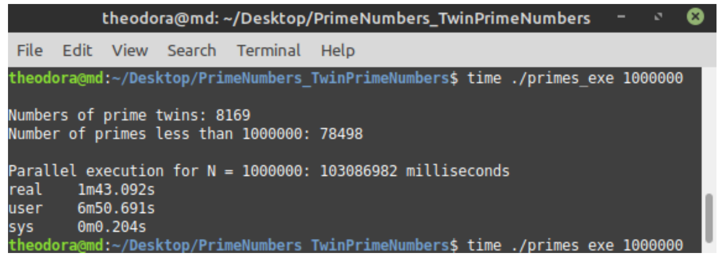


Figure 38 "Dynamic scheduling"

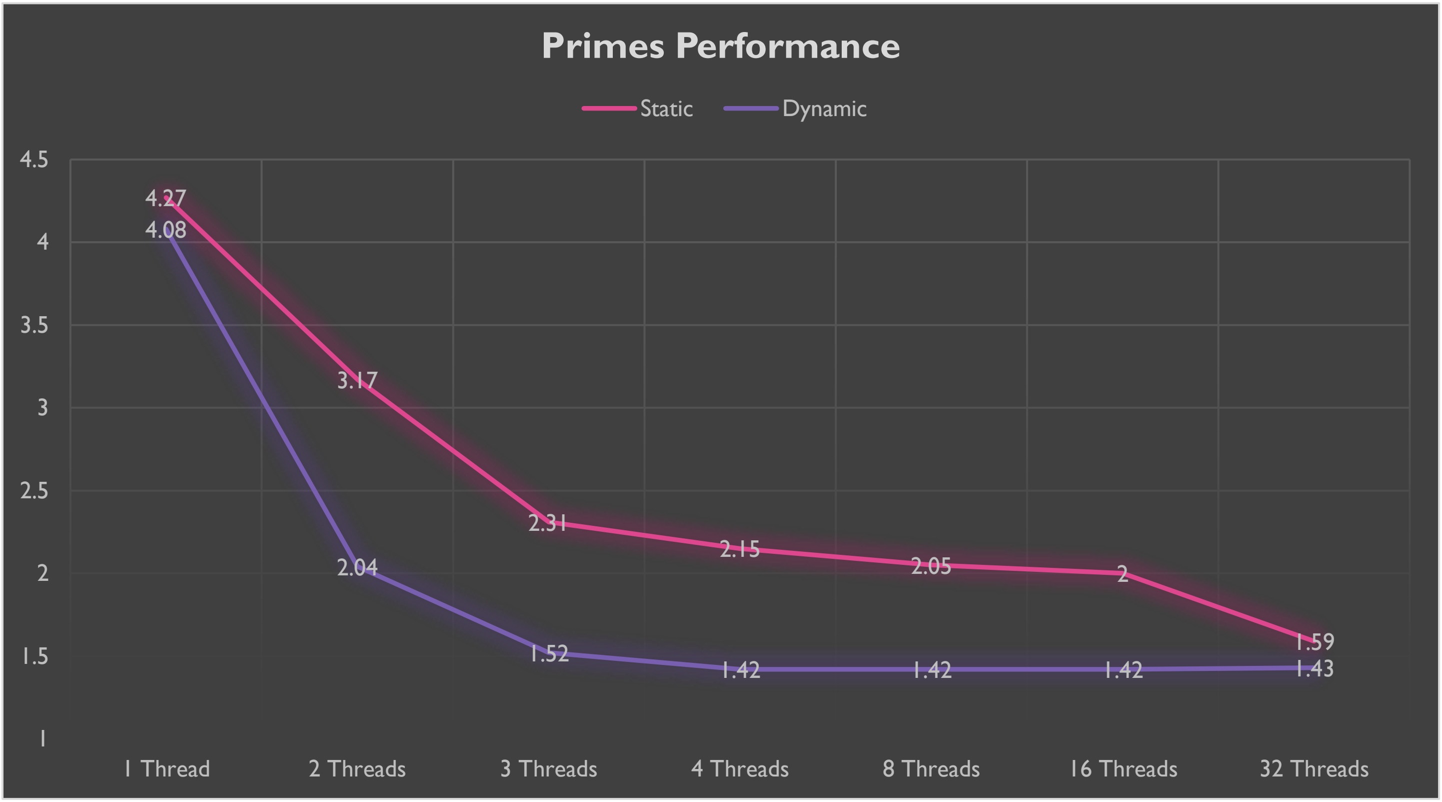


Figure 40

Comparing the static and the dynamic scheduling can be seen in **Figure 40**. The dynamic scheduling benefits the execution of the "Primes" significantly, compared with the static scheduling.

ABSOLUTE AND RELATIVE SPEED

Formula:

ABSOLUTE SPEED = TIME SEQUENTIAL / TIME PARALLEL

RELATIVE SPEED = SINGLE_THREAD/TIME_PARALLEL

	Static	**Dynamic**
Sequential Time	4 minutes and 10 seconds = 250 seconds	4 minutes and 10 seconds = 250 seconds
1 Threads	4 minutes and 27 seconds = 267 seconds Absolute Speed: 250/267 = 0.936	4 minutes and 08 seconds = 248 seconds Absolute Speed: 250/248= 1.008
2 Threads	3 minutes and 17 seconds = 197 seconds Absolute Speed: 250/197 = 1.267 Relative Speed: 267/197 = 1.355	2 minutes and 4 seconds = 124 seconds Absolute Speed: 250/124 = 2.016 seconds Relative Speed: 248/124 = 2.000 seconds
3 Threads	2 minutes and 31 seconds = 151 seconds Absolute Speed: 250/151 = 1.655 Relative Speed: 267/151 = 1.768	1 minute and 52 seconds = 112 Absolute Speed: 250/112 = 2.232 Relative Speed: 248/112 = 2.214
4 Threads	2 minutes and 15 seconds = 135 seconds Absolute Speed: 250/135 = 1.851 Relative Speed: 267/135 = 1.977	1 minute and 42 seconds = 102 seconds Absolute Speed: 250/102 = 2.450 Relative Speed: 248/102 = 2.431
8 Threads	2 minutes and 5 seconds = 125 seconds Absolute Speed: 250/125 = 2.000 Relative Speed: 267/125 = 2.136	1 minute and 42 seconds = 102 seconds Absolute Speed: 250/102 = 2.450 Relative Speed: 248/102 = 2.431
16 Threads	2 minutes and 0 seconds = 120 seconds Absolute Speed: 250/120 = 2.083 Relative Speed: 267/120 = 2.225	1 minute and 42 seconds = 102 seconds Absolute Speed: 250/102 = 2.450 Relative Speed: 248/102 = 2.431
32 Threads	1 minutes and 59 seconds = 119 seconds Absolute Speed: 250/119 = 2.100 Relative Speed: 267/119 = 2.243	1 minute and 43 seconds = 103 seconds Absolute Speed: 250/103 = 2.427 Relative Speed: 248/103 = 2.407

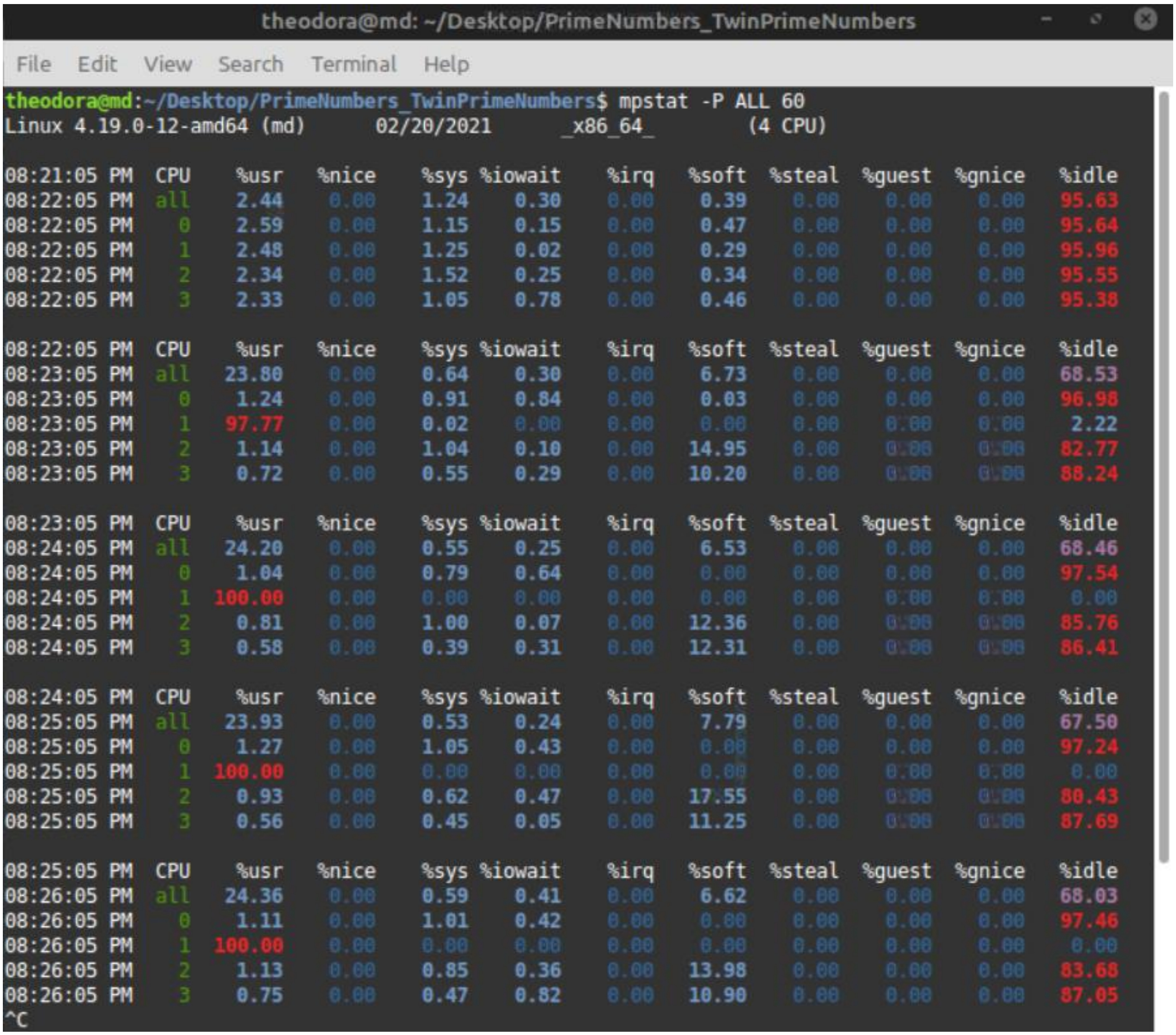
SCALABILITY

Considering that the "Primes" program's best performance was achieved with 8 threads and dynamic scheduling, a further analysis was performed using **mpstat -P ALL 60**. This action was performed once for the sequential execution and once for the 8 threads dynamic scheduling execution.

The measurement tool was kept open simultaneously with the program in execution to monitor the CPUs usage.

The first group of rows in each figure relates to the system in standby, and the following groups are related to the CPUs usage during "Primes" execution. In both cases, the program was finding the prime numbers and their twins for all the numbers less than 1 million.

SEQUENTIAL



```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File  Edit  View  Search  Terminal  Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ mpstat -P ALL 60
Linux 4.19.0-12-amd64 (md)      02/20/2021      _x86_64_        (4 CPU)

08:21:05 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:22:05 PM  all    2.44    0.00    1.24    0.30    0.00    0.39    0.00    0.00    0.00   95.63
08:22:05 PM    0    2.59    0.00    1.15    0.15    0.00    0.47    0.00    0.00    0.00   95.64
08:22:05 PM    1    2.48    0.00    1.25    0.02    0.00    0.29    0.00    0.00    0.00   95.96
08:22:05 PM    2    2.34    0.00    1.52    0.25    0.00    0.34    0.00    0.00    0.00   95.55
08:22:05 PM    3    2.33    0.00    1.05    0.78    0.00    0.46    0.00    0.00    0.00   95.38

08:22:05 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:23:05 PM  all   23.80    0.00    0.64    0.30    0.00    6.73    0.00    0.00    0.00   68.53
08:23:05 PM    0    1.24    0.00    0.91    0.84    0.00    0.03    0.00    0.00    0.00   96.98
08:23:05 PM    1   97.77    0.00    0.02    0.00    0.00    0.00    0.00    0.00    0.00    2.22
08:23:05 PM    2    1.14    0.00    1.04    0.10    0.00   14.95    0.00    0.00    0.00   82.77
08:23:05 PM    3    0.72    0.00    0.55    0.29    0.00   10.20    0.00    0.00    0.00   88.24

08:23:05 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:24:05 PM  all   24.20    0.00    0.55    0.25    0.00    6.53    0.00    0.00    0.00   68.46
08:24:05 PM    0    1.04    0.00    0.79    0.64    0.00    0.00    0.00    0.00    0.00   97.54
08:24:05 PM    1  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
08:24:05 PM    2    0.81    0.00    1.00    0.07    0.00   12.36    0.00    0.00    0.00   85.76
08:24:05 PM    3    0.58    0.00    0.39    0.31    0.00   12.31    0.00    0.00    0.00   86.41

08:24:05 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:25:05 PM  all   23.93    0.00    0.53    0.24    0.00    7.79    0.00    0.00    0.00   67.50
08:25:05 PM    0    1.27    0.00    1.05    0.43    0.00    0.00    0.00    0.00    0.00   97.24
08:25:05 PM    1  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
08:25:05 PM    2    0.93    0.00    0.62    0.47    0.00   17.55    0.00    0.00    0.00   80.43
08:25:05 PM    3    0.56    0.00    0.45    0.05    0.00   11.25    0.00    0.00    0.00   87.69

08:25:05 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:26:05 PM  all   24.36    0.00    0.59    0.41    0.00    6.62    0.00    0.00    0.00   68.03
08:26:05 PM    0    1.11    0.00    1.01    0.42    0.00    0.00    0.00    0.00    0.00   97.46
08:26:05 PM    1  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
08:26:05 PM    2    1.13    0.00    0.85    0.36    0.00   13.98    0.00    0.00    0.00   83.68
08:26:05 PM    3    0.75    0.00    0.47    0.82    0.00   10.90    0.00    0.00    0.00   87.05
^C
```

Figure 41 "CPU utilization for sequential execution"

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File  Edit  View  Search  Terminal  Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ mpstat -P ALL 30
Linux 4.19.0-12-amd64 (md)      02/20/2021      _x86_64_        (4 CPU)

08:31:17 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:31:47 PM  all    1.78    0.00    1.53    0.19    0.00    1.08    0.00    0.00    0.00   95.43
08:31:47 PM    0    1.96    0.00    1.65    0.57    0.00    0.61    0.00    0.00    0.00   95.21
08:31:47 PM    1    1.82    0.00    1.31    0.13    0.00    0.67    0.00    0.00    0.00   96.07
08:31:47 PM    2    2.29    0.00    2.12    0.03    0.00    0.34    0.00    0.00    0.00   95.23
08:31:47 PM    3    1.05    0.00    1.02    0.00    0.00    2.71    0.00    0.00    0.00   95.22

08:31:47 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:32:17 PM  all   95.65    0.00    0.98    0.00    0.00    0.53    0.00    0.00    0.00    2.84
08:32:17 PM    0   96.70    0.00    0.50    0.00    0.00    0.00    0.00    0.00    0.00    2.80
08:32:17 PM    1   96.83    0.00    0.30    0.00    0.00    0.00    0.00    0.00    0.00    2.87
08:32:17 PM    2   96.46    0.00    0.63    0.00    0.00    0.00    0.00    0.00    0.00    2.90
08:32:17 PM    3   92.60    0.00    2.50    0.00    0.00    2.10    0.00    0.00    0.00    2.80

08:32:17 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:32:47 PM  all   99.06    0.00    0.66    0.00    0.00    0.28    0.00    0.00    0.00    0.00
08:32:47 PM    0   99.50    0.00    0.50    0.00    0.00    0.00    0.00    0.00    0.00    0.00
08:32:47 PM    1   99.90    0.00    0.10    0.00    0.00    0.00    0.00    0.00    0.00    0.00
08:32:47 PM    2   99.73    0.00    0.27    0.00    0.00    0.00    0.00    0.00    0.00    0.00
08:32:47 PM    3   97.10    0.00    1.77    0.00    0.00    1.13    0.00    0.00    0.00    0.00

08:32:47 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:33:17 PM  all   99.34    0.00    0.48    0.00    0.00    0.18    0.00    0.00    0.00    0.00
08:33:17 PM    0   99.37    0.00    0.63    0.00    0.00    0.00    0.00    0.00    0.00    0.00
08:33:17 PM    1   99.97    0.00    0.03    0.00    0.00    0.00    0.00    0.00    0.00    0.00
08:33:17 PM    2   99.93    0.00    0.07    0.00    0.00    0.00    0.00    0.00    0.00    0.00
08:33:17 PM    3   98.10    0.00    1.20    0.00    0.00    0.70    0.00    0.00    0.00    0.00
^C

Average:     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
Average:     all   74.16    0.00    0.91    0.05    0.00    0.51    0.00    0.00    0.00   24.37
Average:       0   74.59    0.00    0.82    0.14    0.00    0.15    0.00    0.00    0.00   24.30
Average:       1   74.78    0.00    0.43    0.03    0.00    0.17    0.00    0.00    0.00   24.58
Average:       2   74.76    0.00    0.77    0.01    0.00    0.08    0.00    0.00    0.00   24.38
Average:       3   72.50    0.00    1.62    0.00    0.00    1.66    0.00    0.00    0.00   24.22
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 42 "CPU utilization for concurrent – 8 threads execution"

SEQUENTIAL

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File Edit View Search Terminal Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 100000

Numbers of prime twins: 1224
Number of primes less than 100000: 9592

Parallel execution for N = 100000: 3046816 milliseconds
real    0m3.052s
user    0m3.051s
sys     0m0.000s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 44

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File Edit View Search Terminal Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 200000

Numbers of prime twins: 2160
Number of primes less than 200000: 17984

Parallel execution for N = 200000: 11392453 milliseconds
real    0m11.397s
user    0m11.393s
sys     0m0.004s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 46

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File Edit View Search Terminal Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 300000

Numbers of prime twins: 2994
Number of primes less than 300000: 25997

Parallel execution for N = 300000: 24614151 milliseconds
real    0m24.618s
user    0m24.613s
sys     0m0.004s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 49

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File Edit View Search Terminal Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 400000

Numbers of prime twins: 3804
Number of primes less than 400000: 33860

Parallel execution for N = 400000: 42941029 milliseconds
real    0m42.947s
user    0m42.877s
sys     0m0.068s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 50

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File Edit View Search Terminal Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 500000

Numbers of prime twins: 4565
Number of primes less than 500000: 41538

Parallel execution for N = 500000: 65672731 milliseconds
real    1m5.677s
user    1m5.677s
sys     0m0.000s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 52

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File Edit View Search Terminal Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 600000

Numbers of prime twins: 5331
Number of primes less than 600000: 49098

Parallel execution for N = 600000: 93058242 milliseconds
real    1m33.064s
user    1m33.063s
sys     0m0.000s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 43

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File Edit View Search Terminal Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 700000

Numbers of prime twins: 6061
Number of primes less than 700000: 56543

Parallel execution for N = 700000: 125051273 milliseconds
real    2m5.057s
user    2m4.948s
sys     0m0.104s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 45

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File Edit View Search Terminal Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 800000

Numbers of prime twins: 6766
Number of primes less than 800000: 63951

Parallel execution for N = 800000: 161685011 milliseconds
real    2m41.689s
user    2m41.576s
sys     0m0.096s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 47

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File Edit View Search Terminal Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 900000

Numbers of prime twins: 7472
Number of primes less than 900000: 71274

Parallel execution for N = 900000: 202639987 milliseconds
real    3m22.645s
user    3m22.497s
sys     0m0.136s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 48

```
theodora@md: ~/Desktop/PrimeNumbers_TwinPrimeNumbers
File Edit View Search Terminal Help
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$ time ./primes_exe 1000000

Numbers of prime twins: 8169
Number of primes less than 1000000: 78498

Parallel execution for N = 1000000: 247886274 milliseconds
real    4m7.892s
user    4m7.805s
sys     0m0.072s
theodora@md:~/Desktop/PrimeNumbers_TwinPrimeNumbers$
```

Figure 51

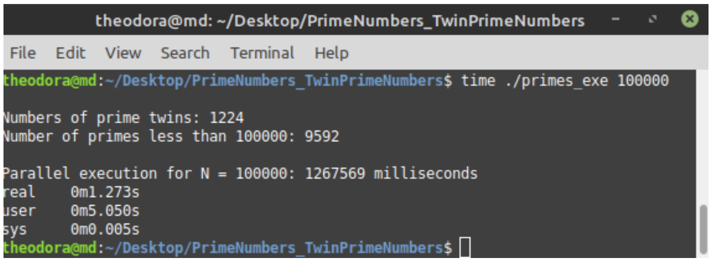


Figure 54

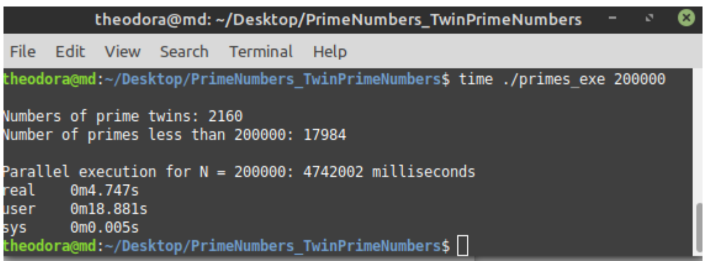


Figure 56

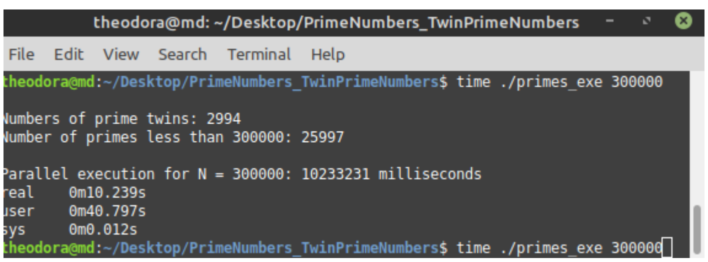


Figure 58

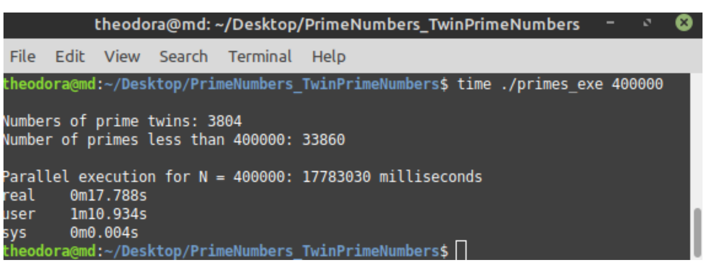


Figure 60

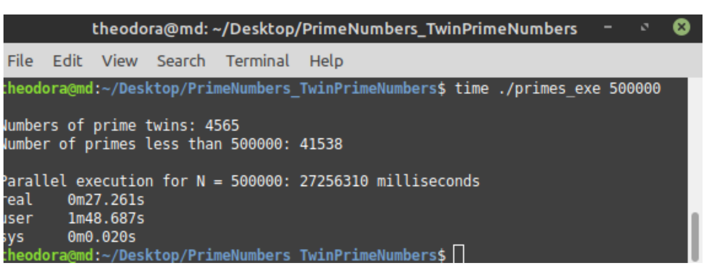


Figure 62

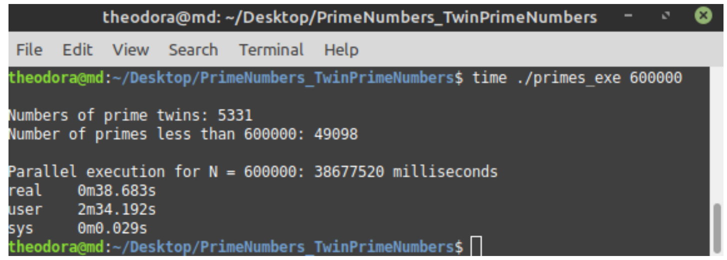


Figure 53

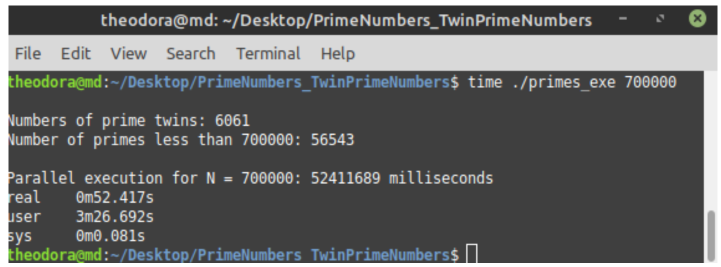


Figure 55

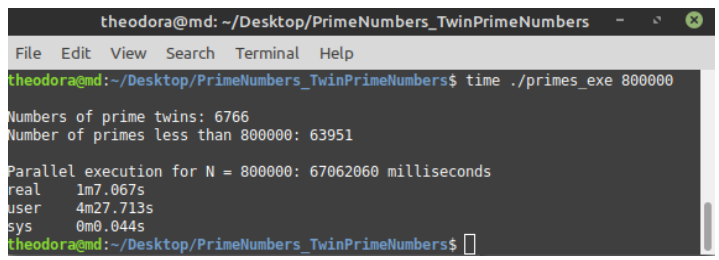


Figure 57

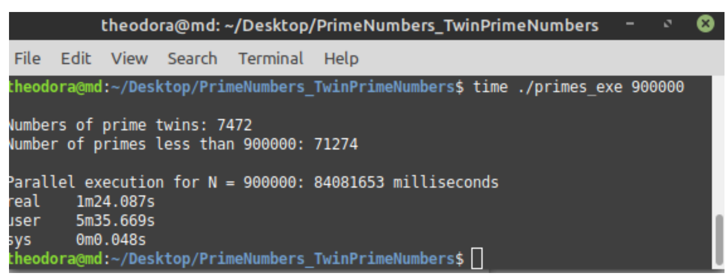


Figure 59

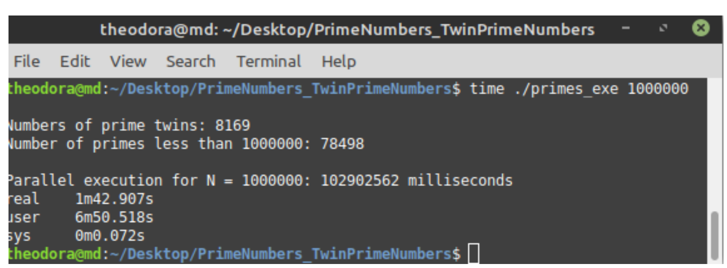


Figure 61

SCALABILITY GRAPH

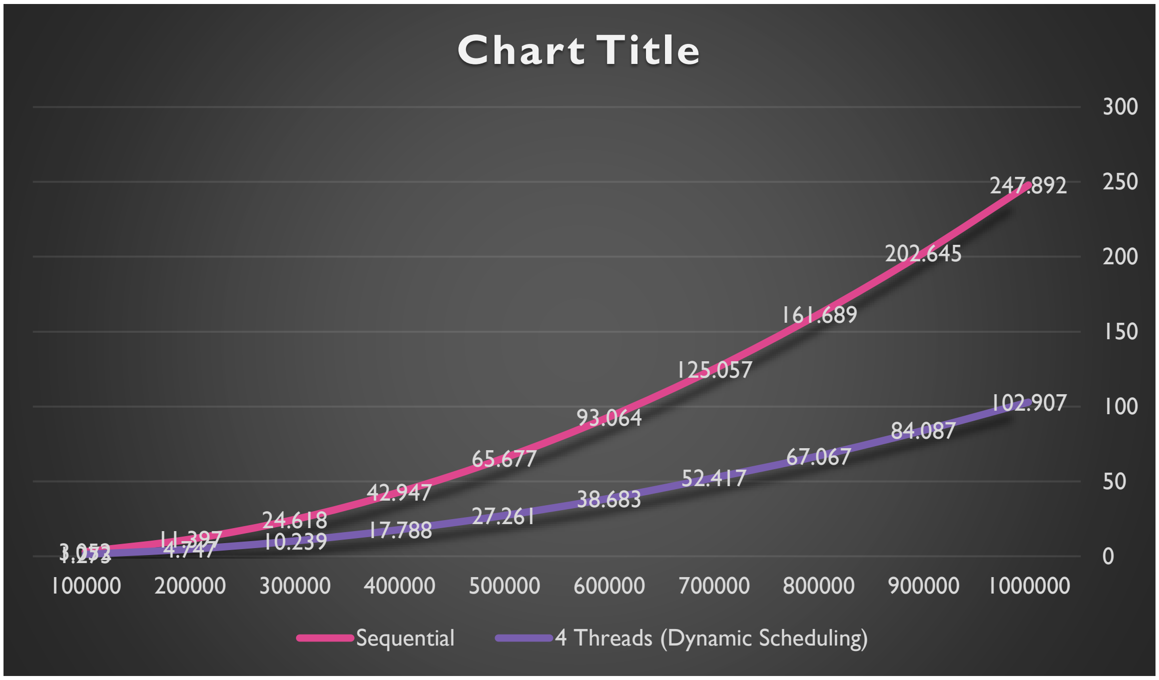


Figure 63

As seen in **Figure 63**, the program takes advantage of multiple threads scaling more efficiently than the program's sequential version.

Both versions were tested, starting with the number100,000 increasing with 100,000 for every iteration until 1,000,000.

The sequential version time of execution is increasing rapidly, where each iteration doubles the execution time when the input number is increased by 100,000.

The 16 threads version starts at the same pace as the sequential version, but as the input number increases, the time difference between the executions decreases.

Bottom line, a code that can run on multiple cores, programmed correctly, takes advantage of the parallel and concurrent execution, scaling better than a sequential code.

CONCLUSION

A program in C++ was developed to return the number of primes and their twins less than a certain number. The program was tested with different input numbers in sequential, parallel, and concurrent execution to observe the program's performance.

The concurrent and parallel version of the program was achieved with the OpenMP library.

The performance of the program in sequential, parallel and concurrent was analyzed by graphs and by calculating the absolute and relative speed up.

Various strategies were implemented to obtain the best performance, and the 16 threads version with dynamic scheduling was observed to be most efficient on the system where the program was tested.

The program was also scaling in the most efficient way with the 8 threads dynamic scheduling version.

BIBLIOGRAPHY

Ferreira, C., 2020. *8 Ways to Measure Execution Time in C/C++.* [Online]
Available at: https://levelup.gitconnected.com/8-ways-to-measure-execution-time-in-c-c-48634458d0f9
[Accessed 16 February 2021].

Levon, 2012. *Why real time can be lower than user time.* [Online]
Available at: https://unix.stackexchange.com/questions/40694/why-real-time-can-be-lower-than-user-time
[Accessed 16 February 2021].

Mattson, T., n.d. *OpenMP in a nutshell.* [Online]
Available at: http://www.bowdoin.edu/~ltoma/teaching/cs3225-GIS/fall17/Lectures/openmp.html#:~:text=OpenMP%20in%20a%20nutshell,supports%20C%2C%20C%2B%2B%20and%20Fortran.
[Accessed 10 February 2021].

PANDE, N. A., 2016. ANALYSIS OF TWIN PRIMES LESS THAN A TRILLION. *Journal of Science and Arts* , p. 280.

Pandit, H. J., 2021. *OpenMP - MultiProcessing Library/Framework.* [Online]
Available at: https://www.scss.tcd.ie/~pandith/CS3014/slides/l2-openmp.pdf
[Accessed 15 February 2021].

PrimePages, 2020. *How Many Primes are There?.* [Online]
Available at: https://primes.utm.edu/howmany.html
[Accessed 10 February 2021].