COMPUTER ENGINEERING MASTER DEGREE

COMPUTER ARCHITECTURE

PROJECT DISCUSSION

INTEGER FACTORIZATION

PROFESSORS

ANTONIO COSIMO PRETE
ANTONIO DI TECCO

GROUP MEMBERS

GIOVANNI LIGATO
ALICE ORLANDINI

IN SUPREMÆ DIGNITATIS 1343

University of Pisa

SYLLABUS

ALGORITHM > GOALS > CPU > GPU

ALGORITHM

INTEGER FACTORIZATION

«Every positive integer can be written uniquely as a product of primes»

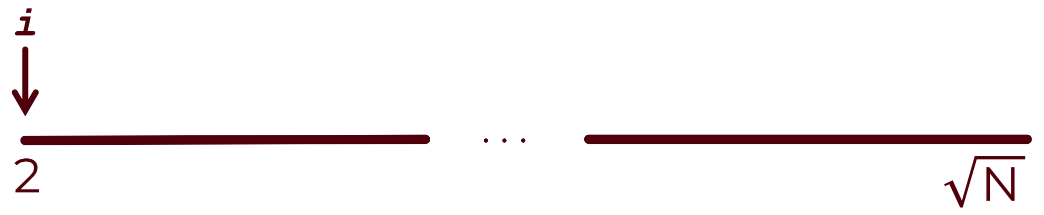
MATHEMATICAL FORMULATION

TRIAL DIVISION

[SEQUENTIAL]

INPUT

N : Number to be Factorized



```
    while N mod $i$ == 0
        N = N / $i$
        Add $i$ to Primes
if N ≠ 1
    Add N to Primes
```

OUTPUT

Primes : Product of primes

INTEGER FACTORIZATION

«Every positive integer can be written uniquely as a product of primes»

MATHEMATICAL FORMULATION

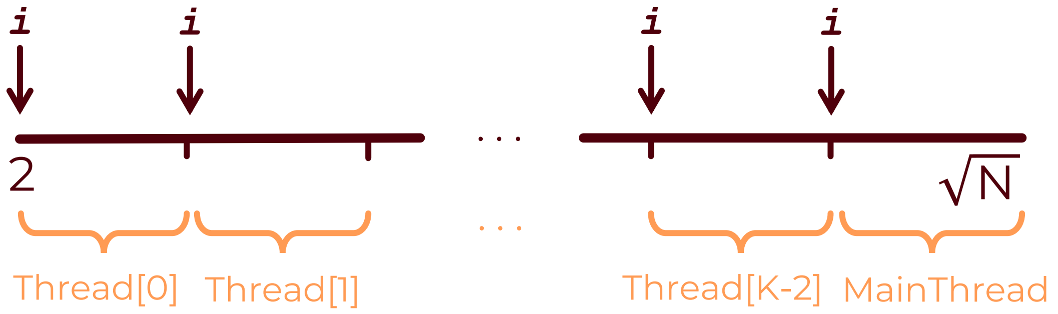
TRIAL DIVISION

[PARALLEL]

INPUT

N : Number to be Factorized

K : Number of Threads



OUTPUT

Primes : Product of primes 🔒 Mutex

GOALS

GOALS

END USER

Execution Time

≤ **1** second

For numbers up to

18

DIGITS

e.g. **975734686214396237**

DEVELOPER

>

Develop a GPU version that **outperforms** the CPU version

CPU

IMPLEMENTATION

11th Gen intel® Core(TM) **i5**-11400

x Cores per socket: **6**

= Threads per core: **2**

TOTAL # LOGICAL CORES: **12**

L1**d** Cache: **288** KiB

L1**i** Cache: **192** KiB

L2 Cache: **3** MiB

L3 Cache: **12** MiB

Base Frequency: **2.60** GHz

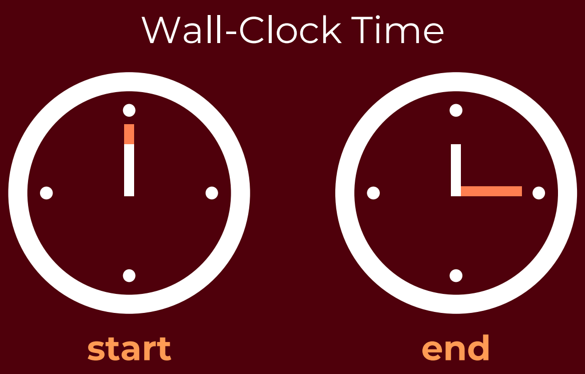
Max Turbo Frequency: **4.40** GHz

HARDWARE DETAILS

TOOLS

Execution Time

```
chrono::steady_clock::time_point start = chrono::steady_clock::now();
parallelTrialDivision(NUMBER, NUM_THREADS);
chrono::steady_clock::time_point end = chrono::steady_clock::now();

chrono::milliseconds duration = chrono::duration_cast<chrono::milliseconds>(end - start);
```



Intel **VTune** Profiler

Visual Studio Code

FIRST EXECUTION

#THREADS : **12**

$$\underbrace{975734686214396237}_{\textbf{18}\text{ DIGITS}} = 748609 * 1303396948493$$

EXECUTION TIME **568340** MILLISECONDS ≈ **9** MINUTES >> **1** SECOND

⚠ REQUIREMENT **NOT** SATISFIED

LET'S GO INTO **DETAIL!**

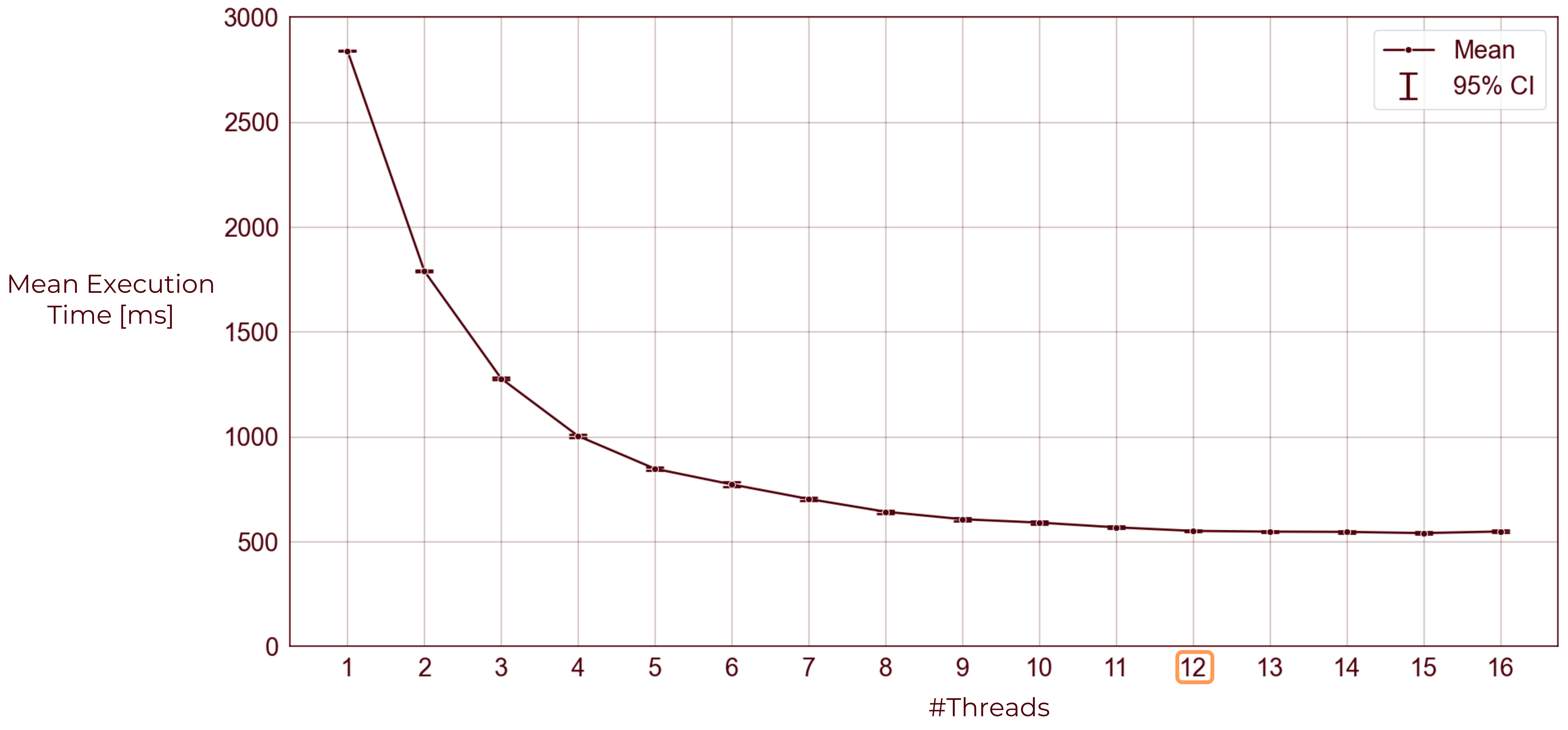
EXECUTION TIME

FIRST VERSION

#ITERATIONS : **30**

59999999999991 = 3 * 19999999999997

(**14** DIGITS)



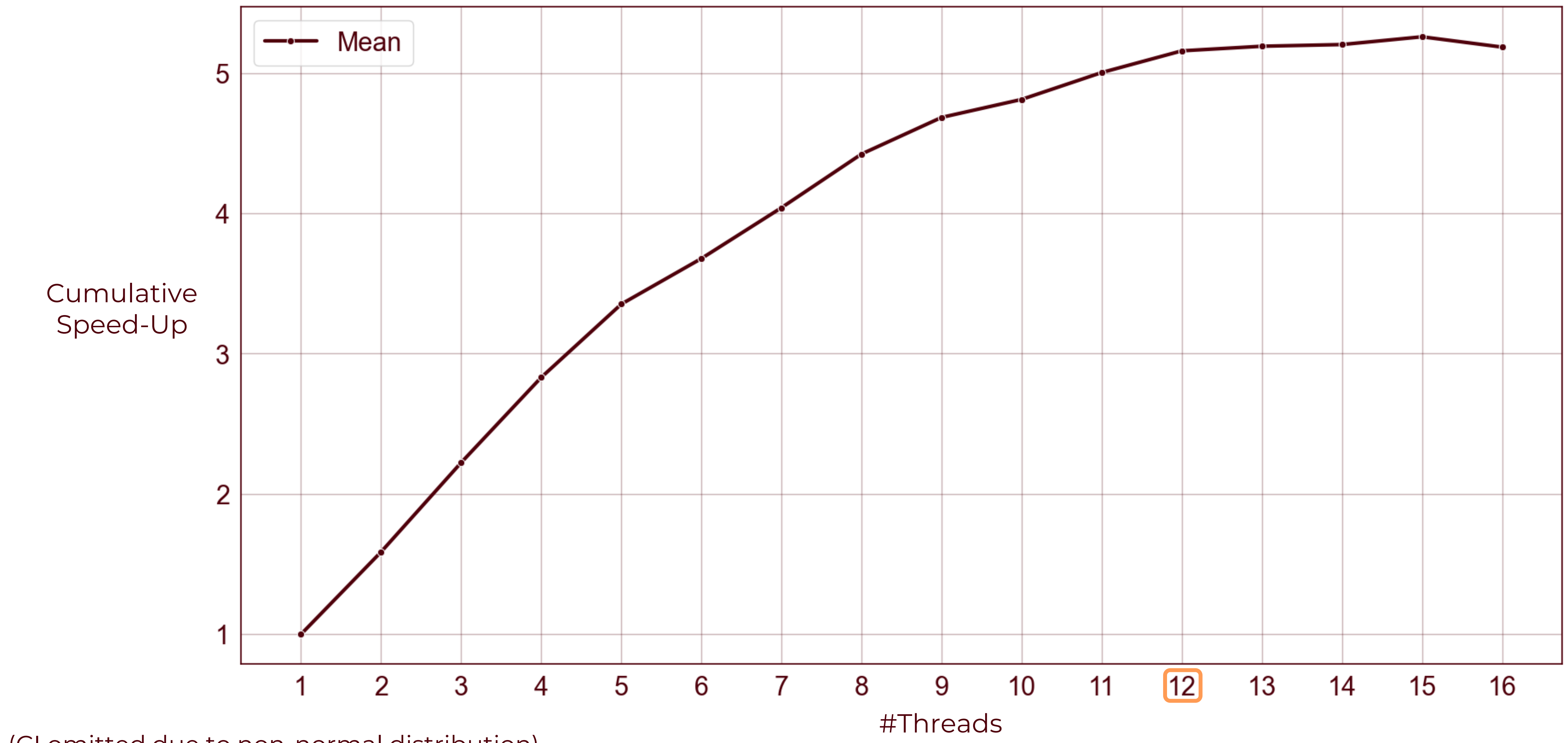
SPEED-UP

FIRST VERSION

#ITERATIONS : **30**

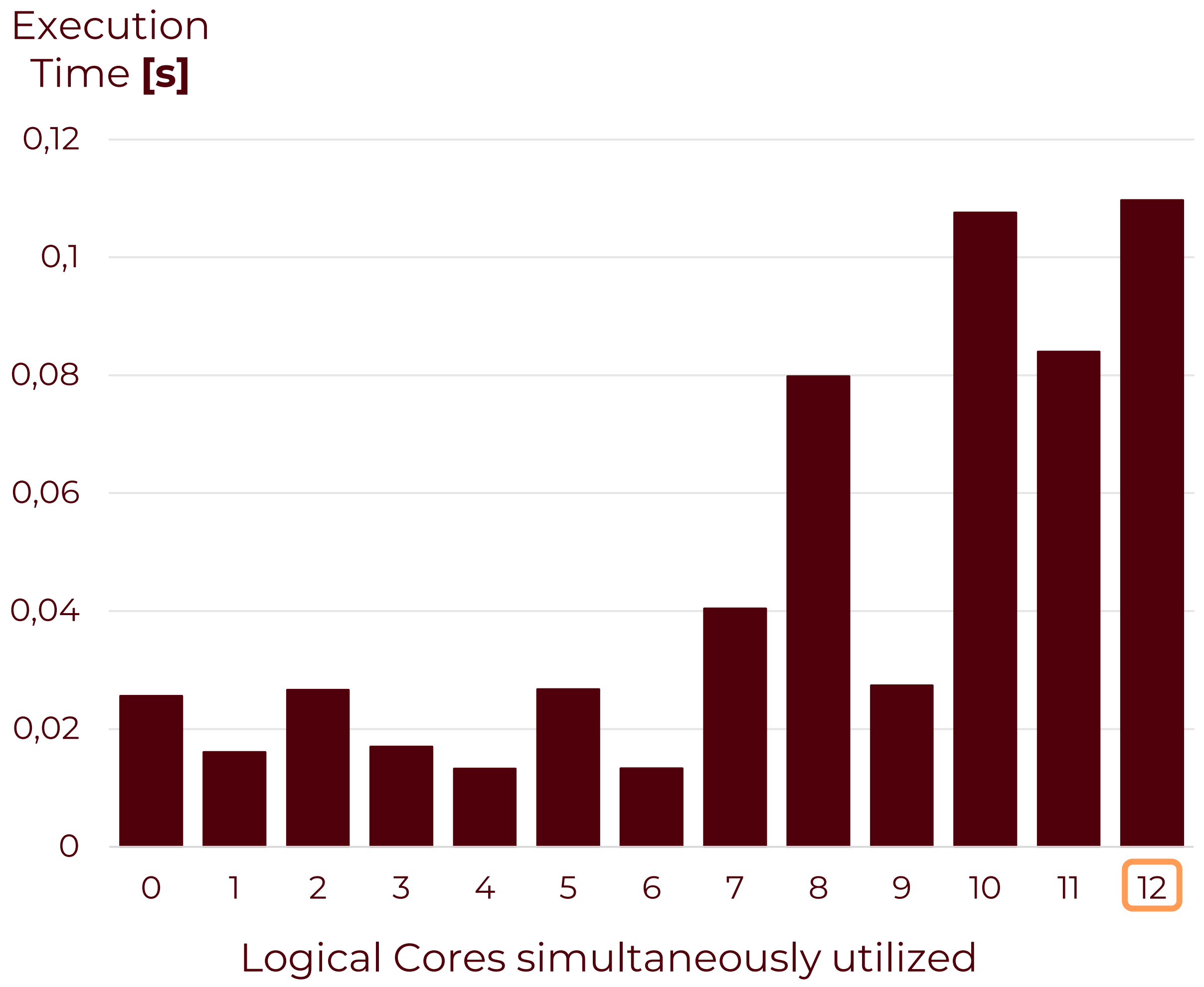
59999999999991 = 3 * 19999999999997

(**14** DIGITS)



(CI omitted due to non-normal distribution)

EFFECTIVE CPU UTILIZATION



TOP **HOTSPOTS**

#THREADS : **12**

0.588s
EXECUTION TIME

4.829s
CPU TIME

0s
PAUSED TIME

0s (0.0% of CPU Time)
THREAD OVERSUBSCRIPTION

Function	**CPU Time**	**% CPU Time**
isPrime	4.808s	99.6%
findPrimesInRange	0.013s	0.3%
others	0.008s	0.2%

FUNCTION	CPU Time	% CPU Time
→ isPrime	4.808s	99.6%

```
bool isPrime(unsigned long long n) {

    …

    for(unsigned long long i = 2; i * i <= n; ++i) {
            if(n % i == 0) {
                    is_prime = false;
                    break;
            }
    }

    …
}
```

CPU TIME

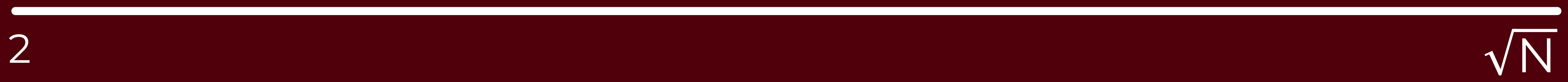
Total (%)	Self (s)
92,8%	4,483s

HOTSPOT

LET’S

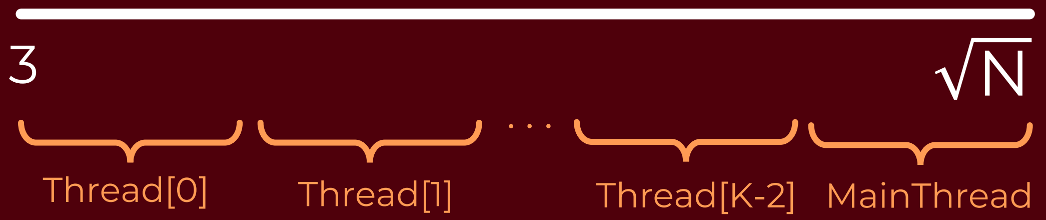
OPTIMIZE!

OPTIMIZATIONS **#1**



MainThread: N mod 2 == 0

OPTIMIZATIONS **#1**



OPTIMIZATIONS **#2**

findPrimesInRange

```
for (unsigned long long i = start; i <= end;  ++i ) {
     if (isPrime(i) && num % i == 0) {

          int exponent = 0;
          while (num % i == 0) {
               exponent++;
               num /= i;
          }

          {
               lock_guard<mutex> lock(mtx);
               primes.push_back({i, exponent});
          }
     }
}
```

OPTIMIZATIONS **#2**

findPrimesInRange

```
for (unsigned long long i = start; i <= end; i += 2) {
    if (isPrime(i) && num % i == 0) {

        int exponent = 0;
        while (num % i == 0) {
            exponent++;
            num /= i;
        }

        {
            lock_guard<mutex> lock(mtx);
            primes.push_back({i, exponent});
        }
    }
}
```

OPTIMIZATIONS **#2**

findPrimesInRange

```
for (unsigned long long i = start; i <= end; i += 2) {
    if (isPrime(i) && num % i == 0) {

        int exponent = 0;
        while (num % i == 0) {
            exponent++;
            num /= i;
        }

        {
            lock_guard<mutex> lock(mtx);
            primes.push_back({i, exponent});
        }
    }
}
```

OPTIMIZATIONS **#2**

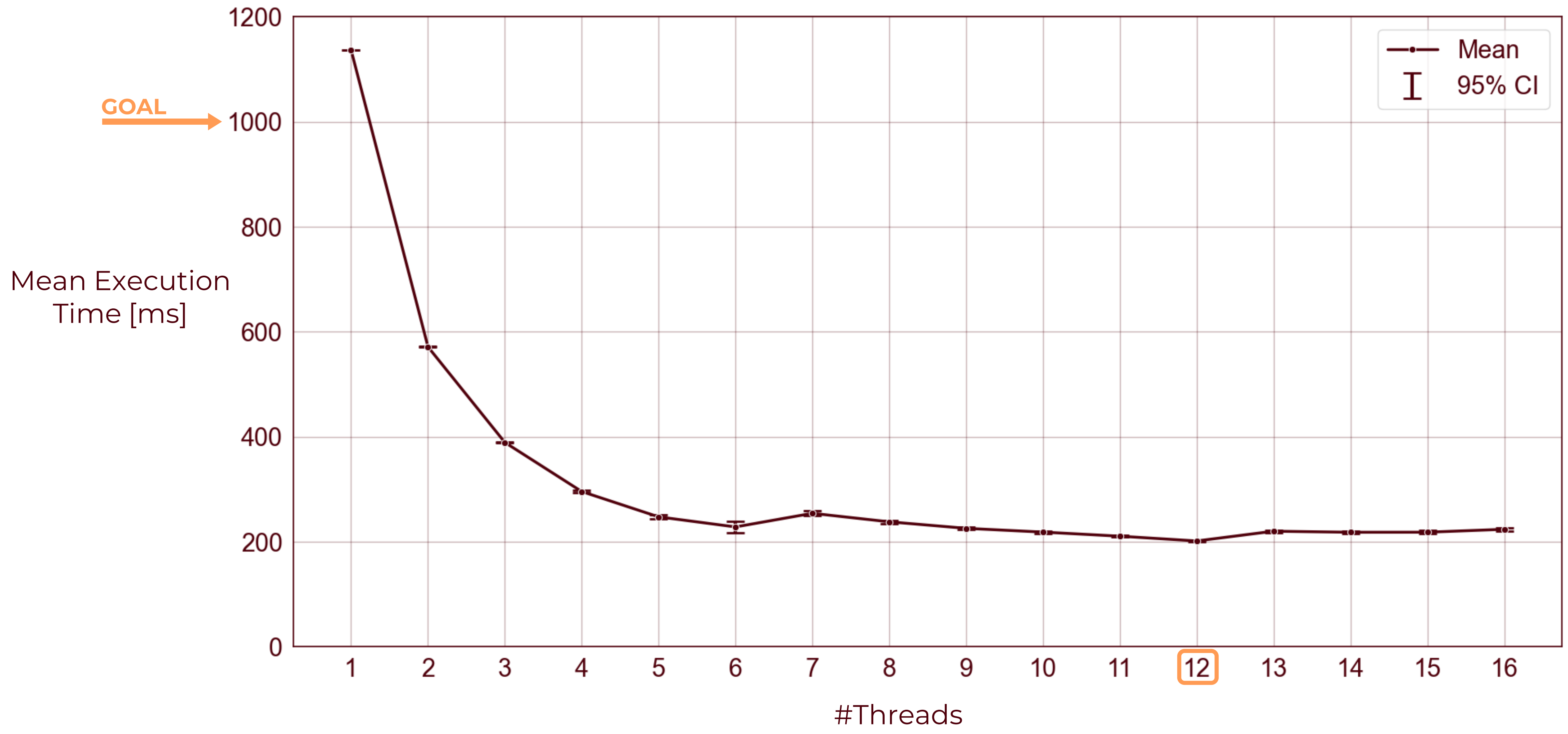
findPrimesInRange

```
for (unsigned long long i = start; i <= end; i += 2) {
        if (num % i == 0) {

                int exponent = 0;
                while (num % i == 0) {
                        exponent++;
                        num /= i;
                }

                if(isPrime(i)){
                        lock_guard<mutex> lock(mtx);
                        primes.push_back({i, exponent});
                }
        }
}
```

EXECUTION TIME
OPTIMIZED VERSION

#ITERATIONS : **30**

975734686214396237 = 748609 * 1303396948493
(**18** DIGITS)



FIRST VERSION

EXECUTION TIME
568340
MILLISECONDS

≈

9
MINUTES

OPTIMIZED VERSION

MAX (OVER 30 ITERATIONS)
EXECUTION TIME
211
MILLISECONDS

<

1
SECOND

x2500
SPEED-UP

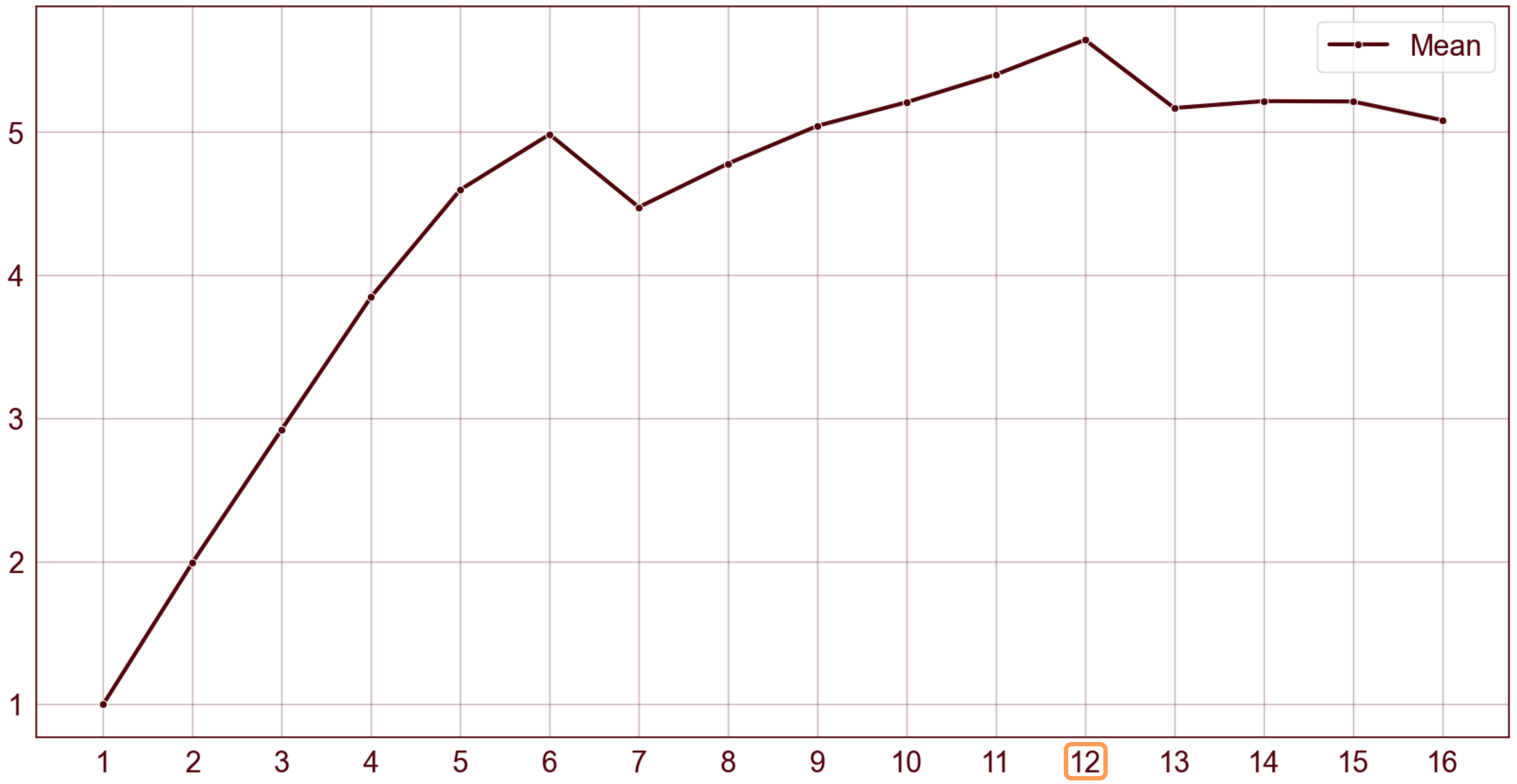
#THREADS : **12**

SPEED-UP
OPTIMIZED VERSION

#ITERATIONS : **30**

975734686214396237 = 748609 * 1303396948493
(**18** DIGITS)

Cumulative Speed-Up



#Threads

(CI omitted due to non-normal distribution)

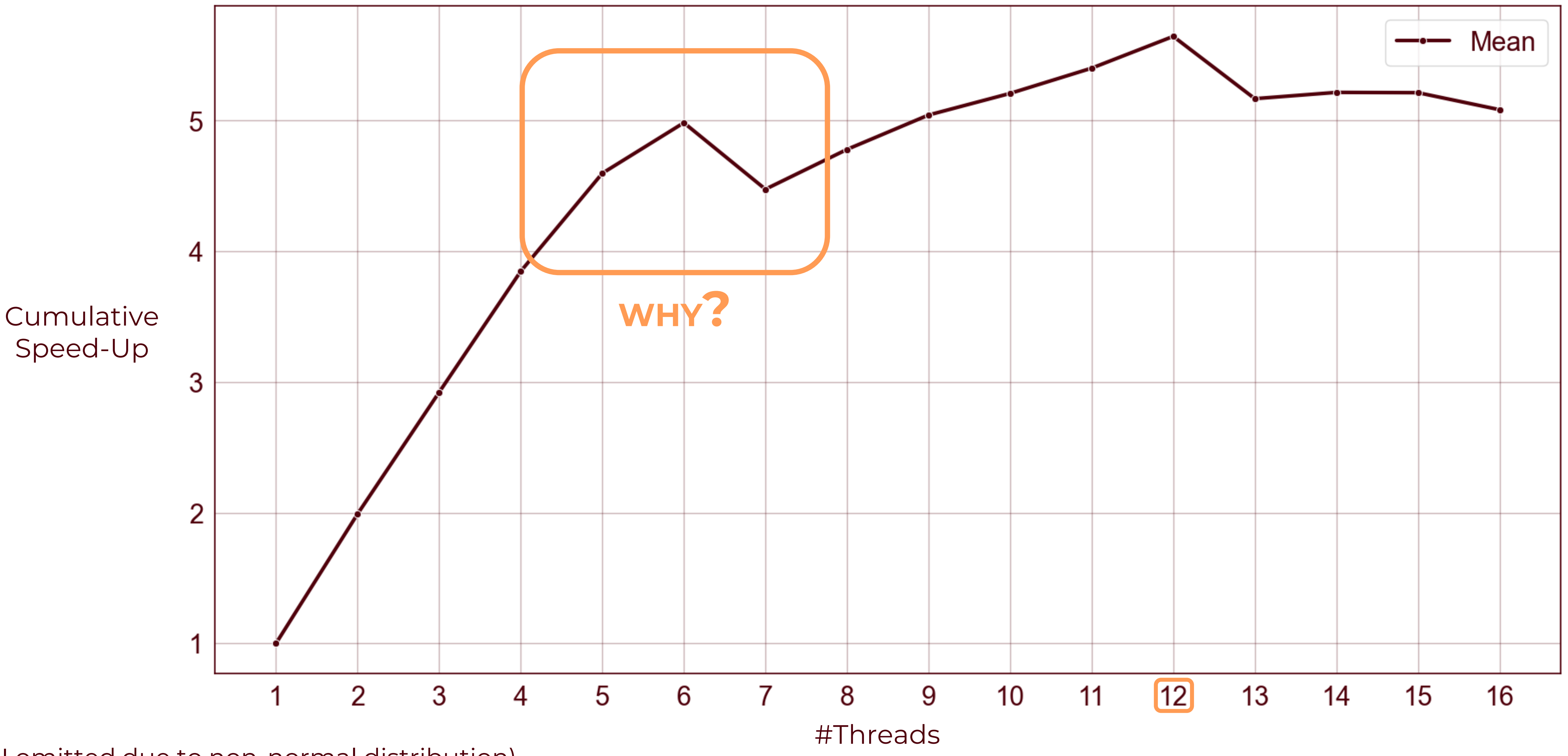
SPEED-UP

OPTIMIZED VERSION

#ITERATIONS : **30**

975734686214396237 = 748609 * 1303396948493

(**18** DIGITS)



(CI omitted due to non-normal distribution)

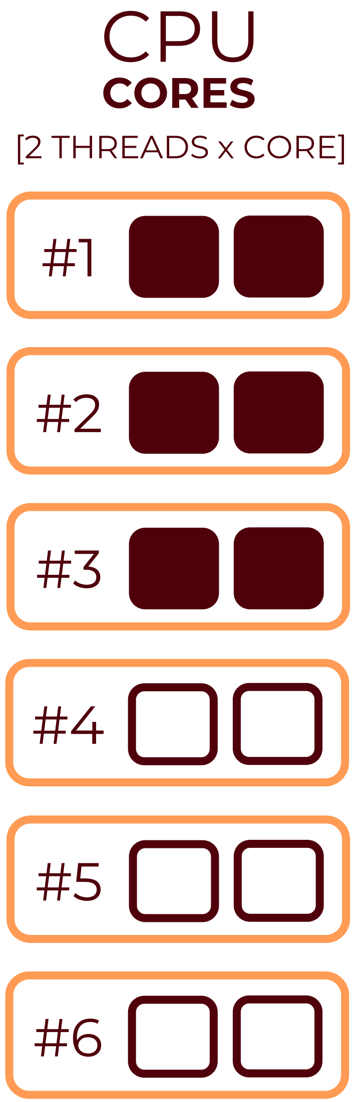
SPEED-UP + AFFINITY

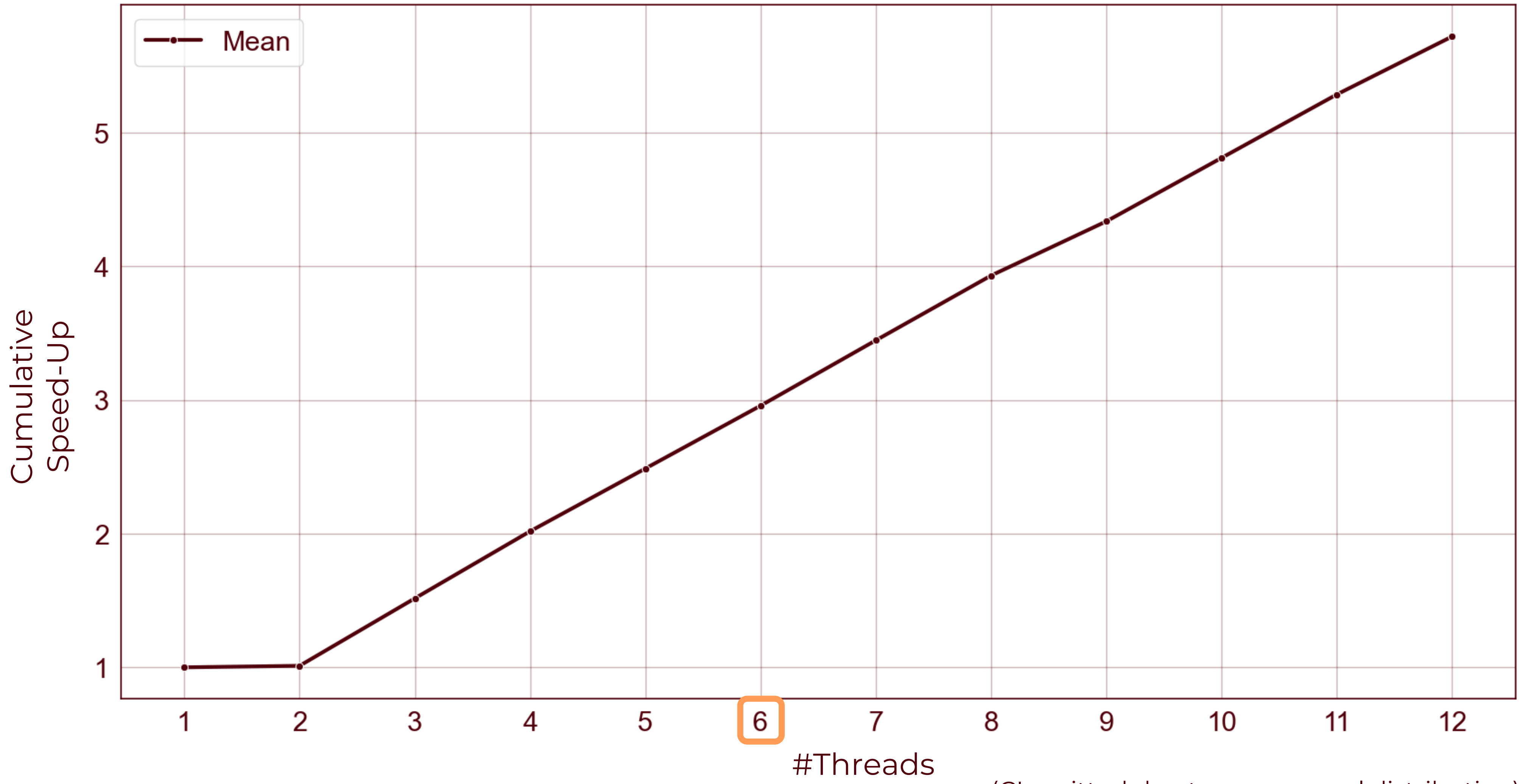
OPTIMIZED VERSION

#ITERATIONS : **30**

975734686214396237 = 748609 * 1303396948493

(**18** DIGITS)





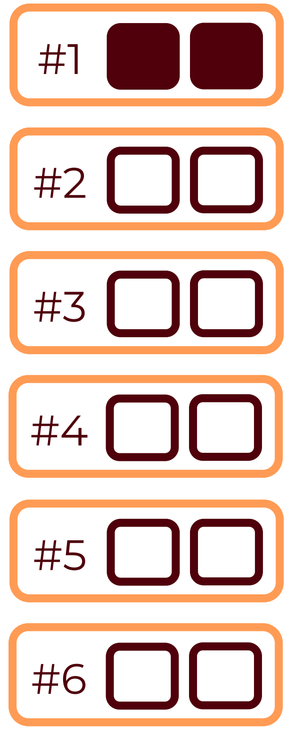
(CI omitted due to non-normal distribution)

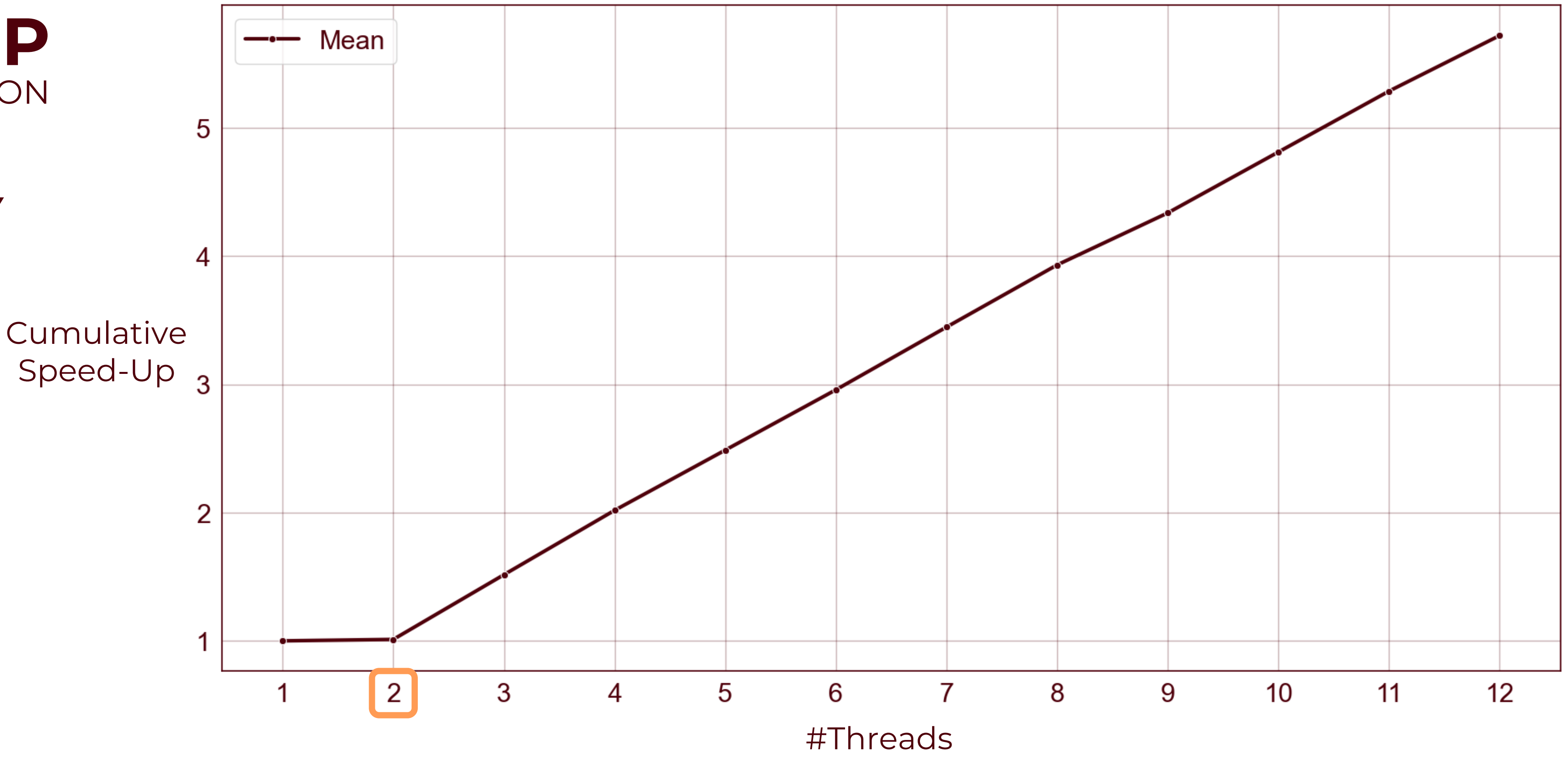
SPEED-UP
OPTIMIZED VERSION

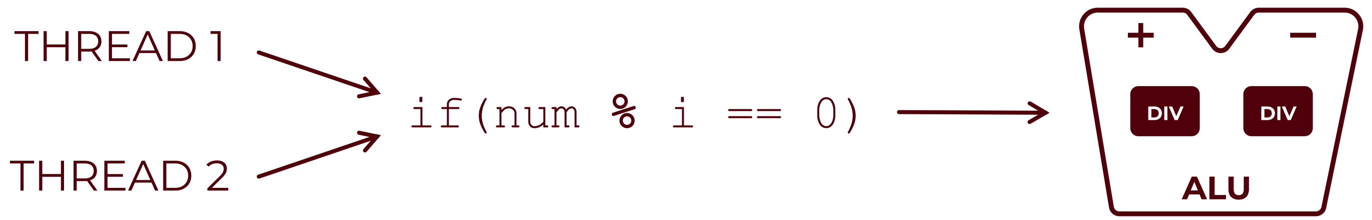
+

AFFINITY

CPU
CORES
[2 THREADS x CORE]







DIVIDER
UTILIZATION
~ **56**%

SPEED-UP
OPTIMIZED VERSION

+ **AFFINITY**

#ITERATIONS : **30**

975734686214396237 = 748609 * 1303396948493

(**18** DIGITS)

CPU
CORES
[2 THREADS x CORE]

#1 ■ □

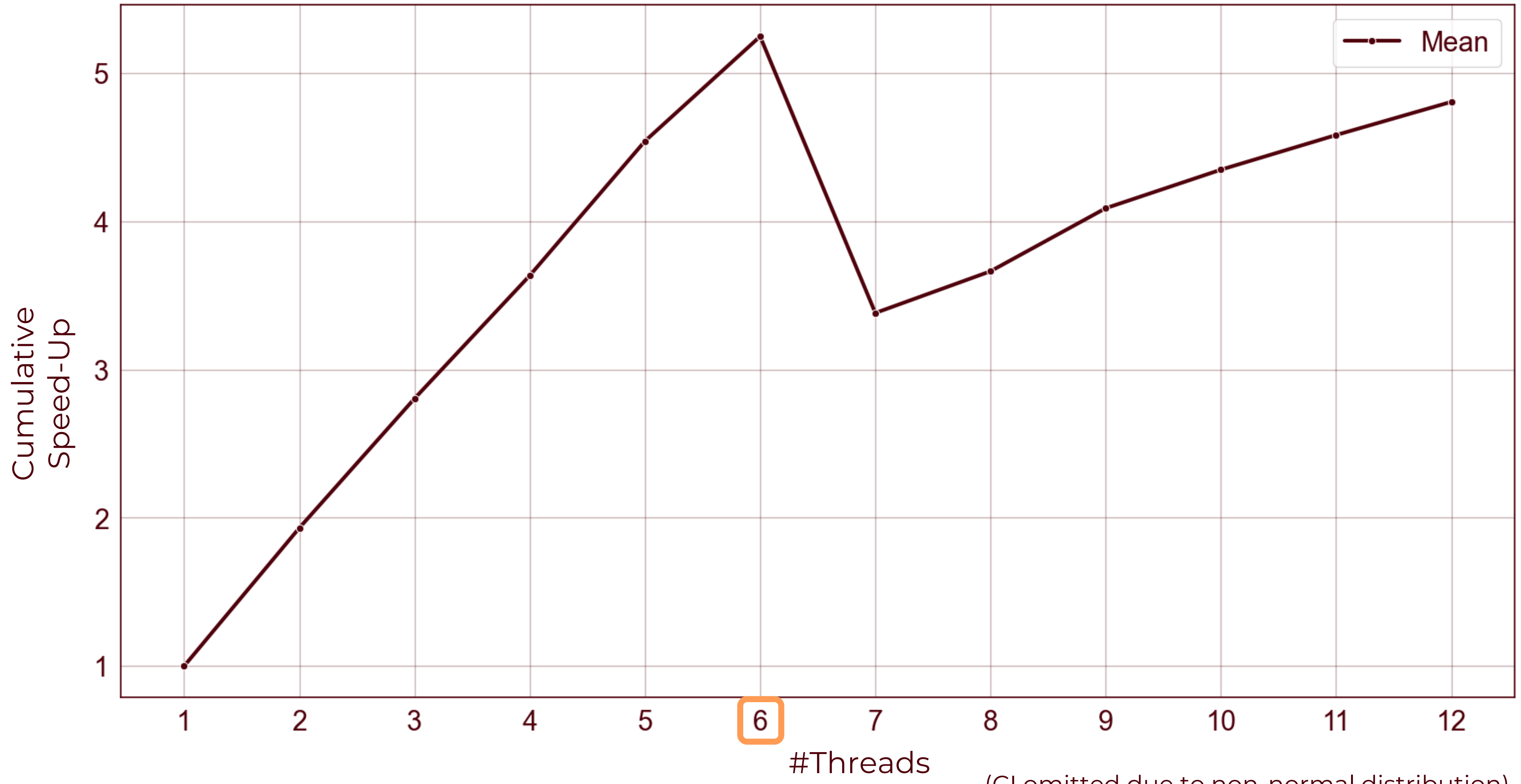
#2 ■ □

#3 ■ □

#4 ■ □

#5 ■ □

#6 ■ □



(CI omitted due to non-normal distribution)

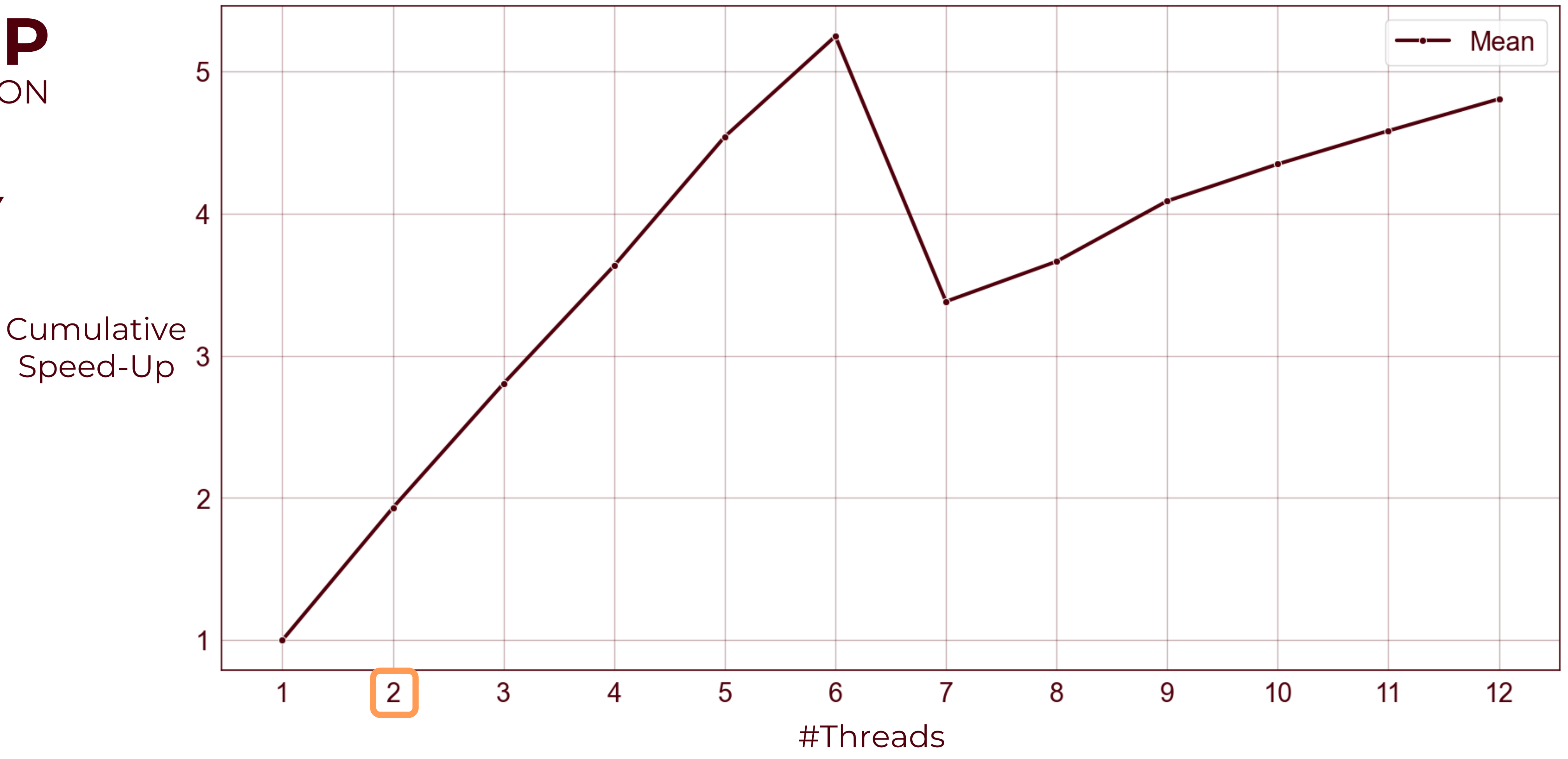
SPEED-UP
OPTIMIZED VERSION

+

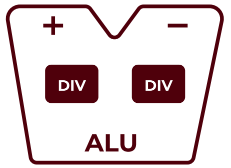
AFFINITY

CPU
CORES
[2 THREADS x CORE]

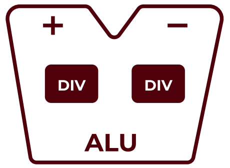
#1	■	□
#2	■	□
#3	□	□
#4	□	□
#5	□	□
#6	□	□



THREAD 1 → `if(num % i == 0)` →



THREAD 2 → `if(num % i == 0)` →



DIVIDER
UTILIZATION
~ **92**%

EXECUTION TIME - SCALABILITY

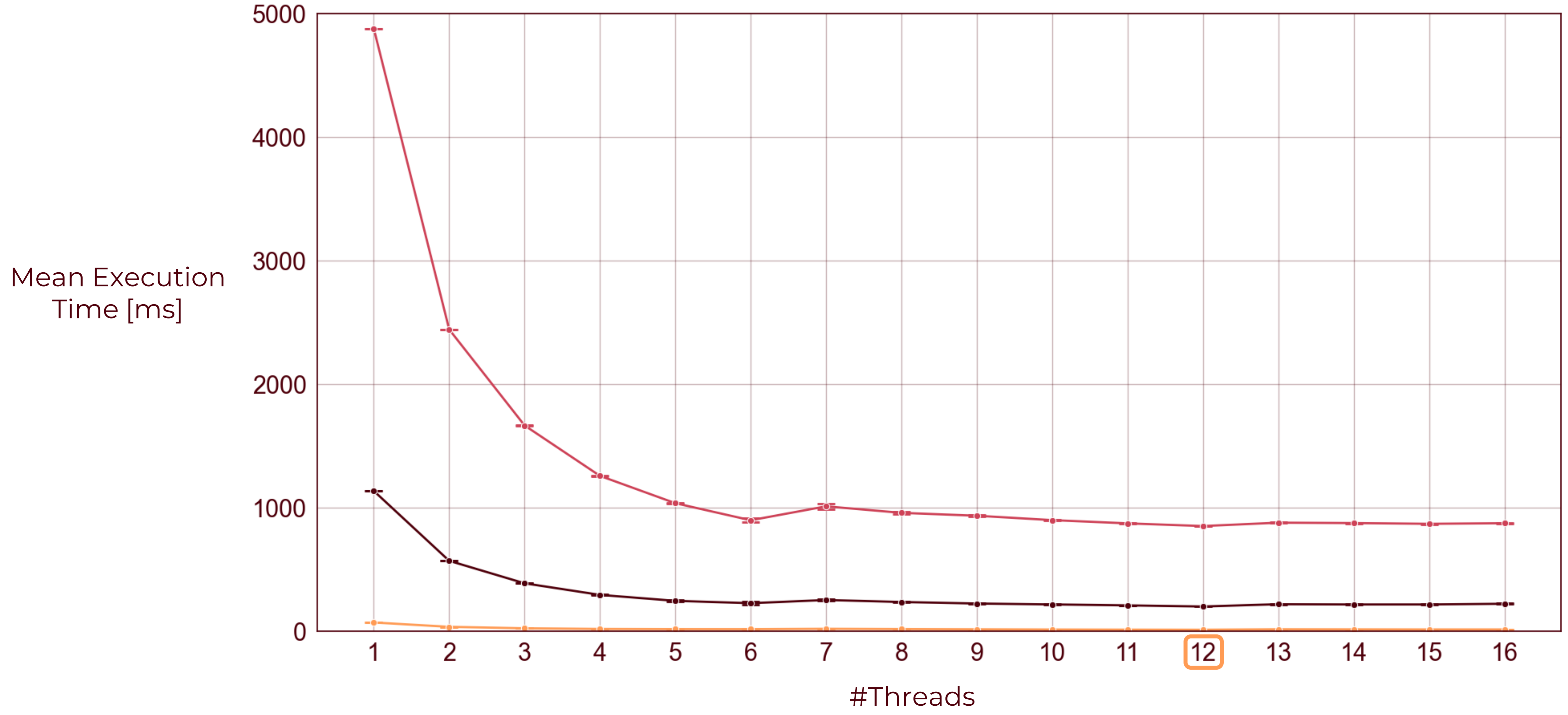
OPTIMIZED VERSION

#ITERATIONS : **30**

20 DIGITS **17975734686214396237**

18 DIGITS **975734686214396237**

16 DIGITS **3934686214396237**



SPEED-UP - SCALABILITY

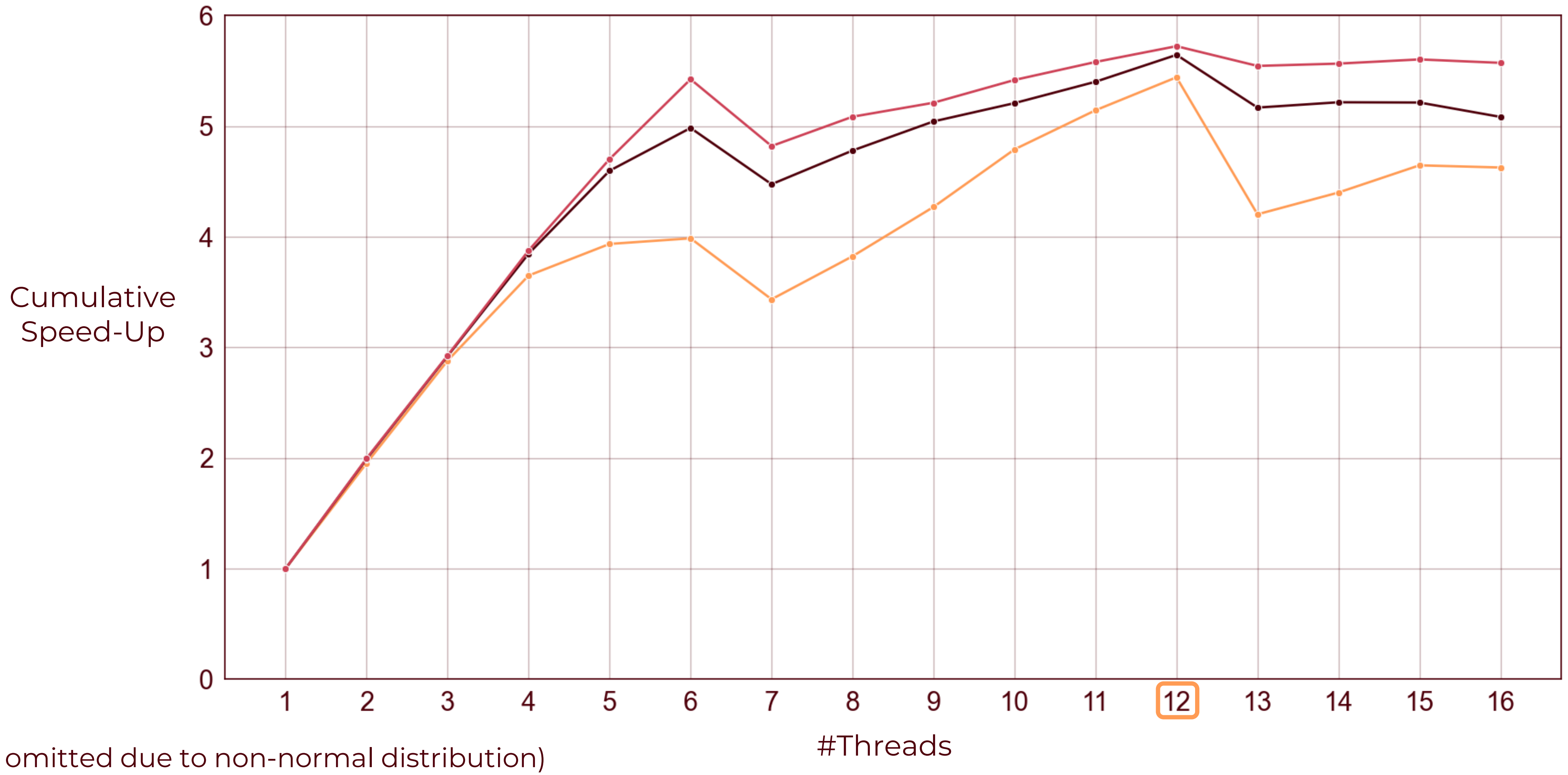
OPTIMIZED VERSION

#ITERATIONS : **30**

20 DIGITS **17975734686214396237**

18 DIGITS **975734686214396237**

16 DIGITS **3934686214396237**



(CI omitted due to non-normal distribution)

GPU
IMPLEMENTATION

NVIDIA Tesla T4

Streaming Multiprocessors (SM): **40** x
CUDA Cores per SM: **64** =

TOTAL # CUDA Cores: **2560**

Warp Size: **32**

Memory: **16 GB GDDR6 300 GB/s**

Interconnection bandwidth: **32 GB/s**

Max Threads per Block: **1024**

GPU Max Clock Rate: **1590 MHz**

Memory Max Clock Rate: **5001 MHz**

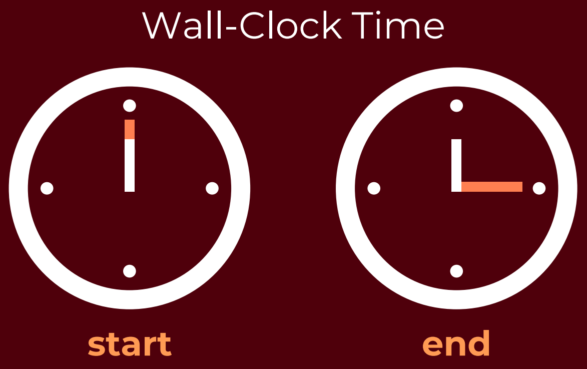
HARDWARE DETAILS

TOOLS

Execution Time

TOTAL
CPU + GPU → chrono : : steady_clock

GPU
TIME → cudaEvent_t



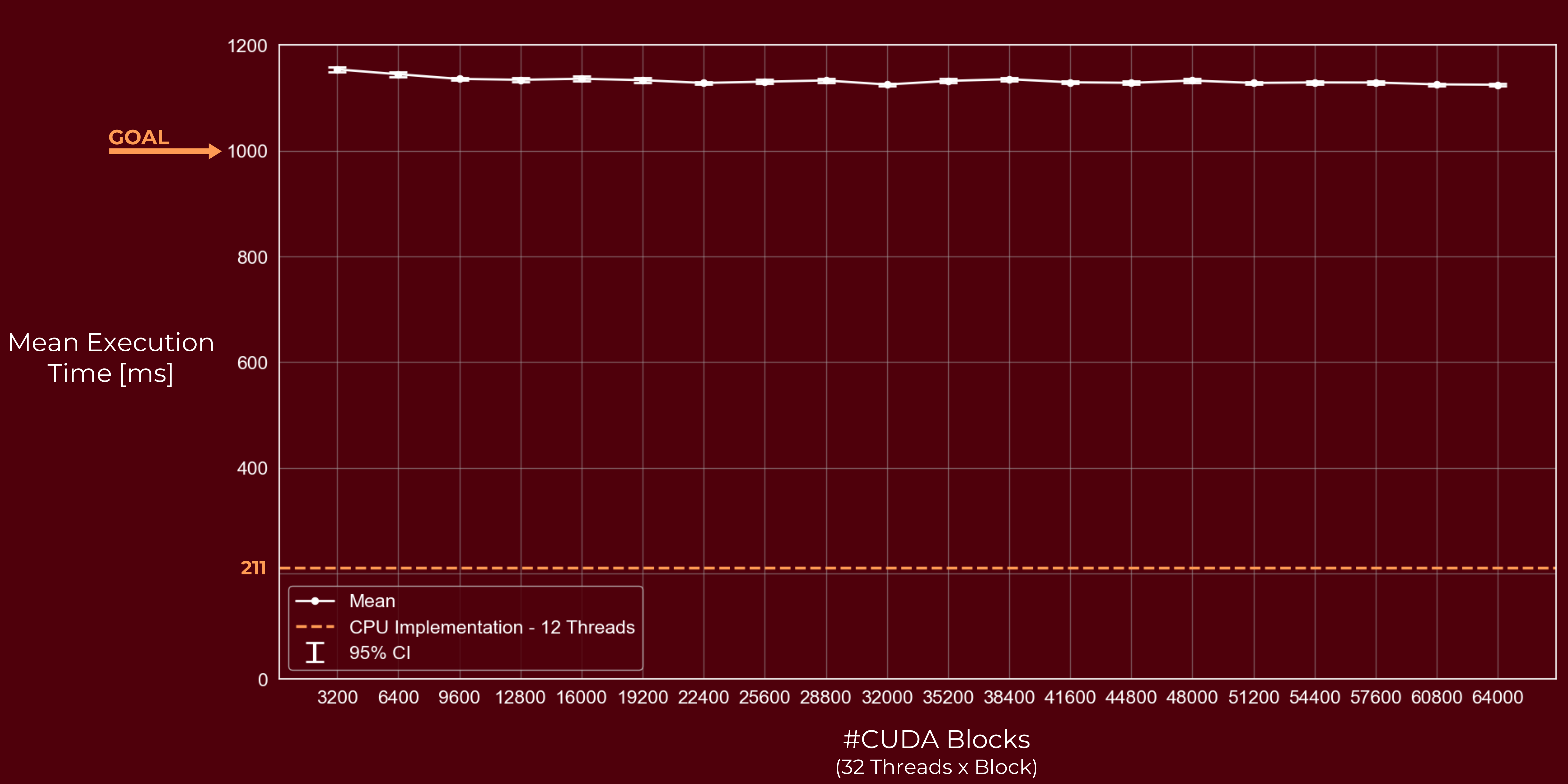
NVIDIA **Nsight Compute** | Visual Studio Code

EXECUTION TIME

FIRST VERSION ON GPU

#ITERATIONS : **30**

18 DIGITS **975734686214396237**



OPTIMIZATIONS

KERNEL

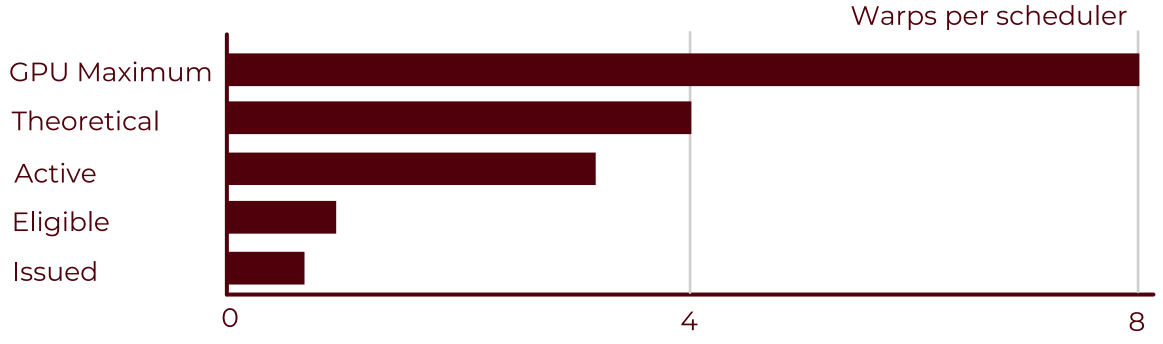
```
__global__ void findPrimesInRange(...)
```

JUST **1 THREAD** PER WARP WAS WORKING

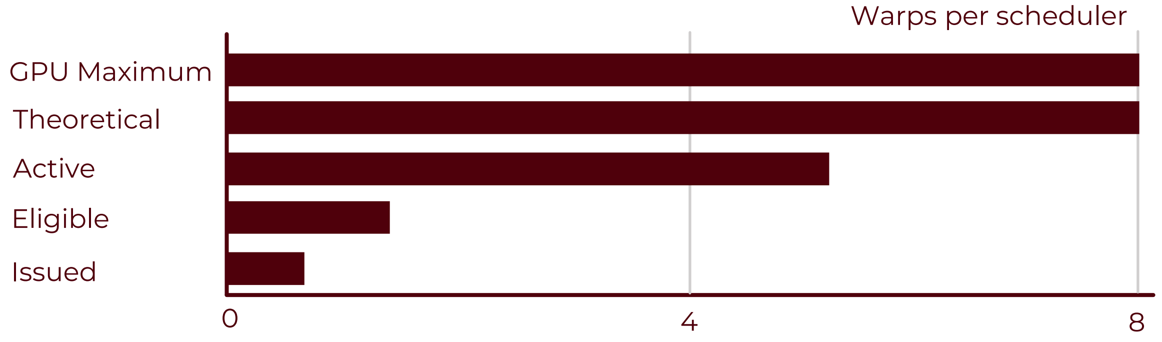
32 THREADS PER WARP*

*DIVERGENCE CAN HAPPEN BUT ITS EFFECT IS NEGLIGIBLE AS THE AVERAGE NUMBER OF NOT PREDICATED OFF THREADS PER WARP IS **31.64**

32 THREADS x BLOCK



64 THREADS x BLOCK



max_blocks_per_multiprocessor **16**

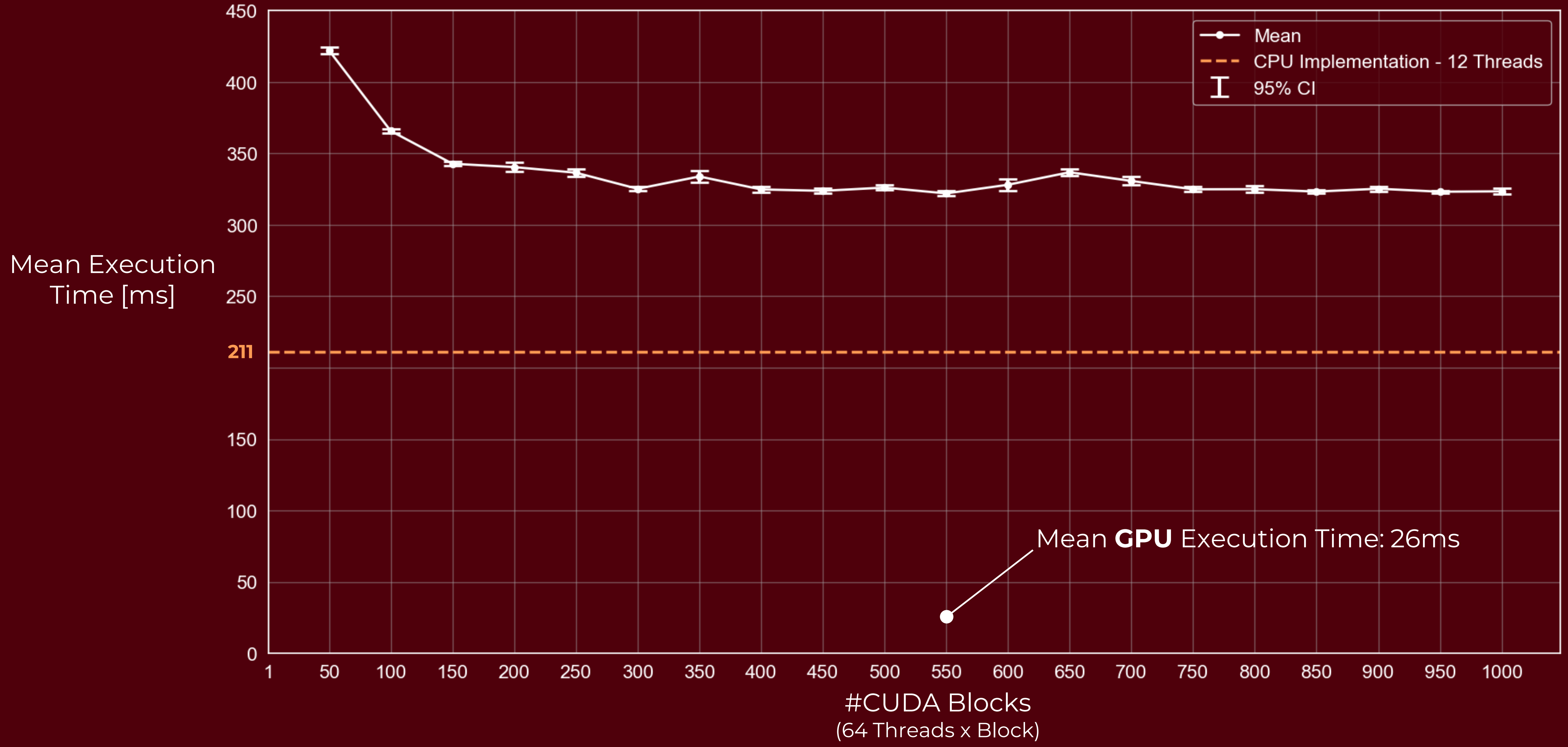
max_warps_per_multiprocessor **32**

max_warps_per_scheduler **8**

NVIDIA Nsight™ Compute

EXECUTION TIME
OPTIMIZED VERSION ON GPU

#ITERATIONS : **30**
18 DIGITS **975734686214396237**

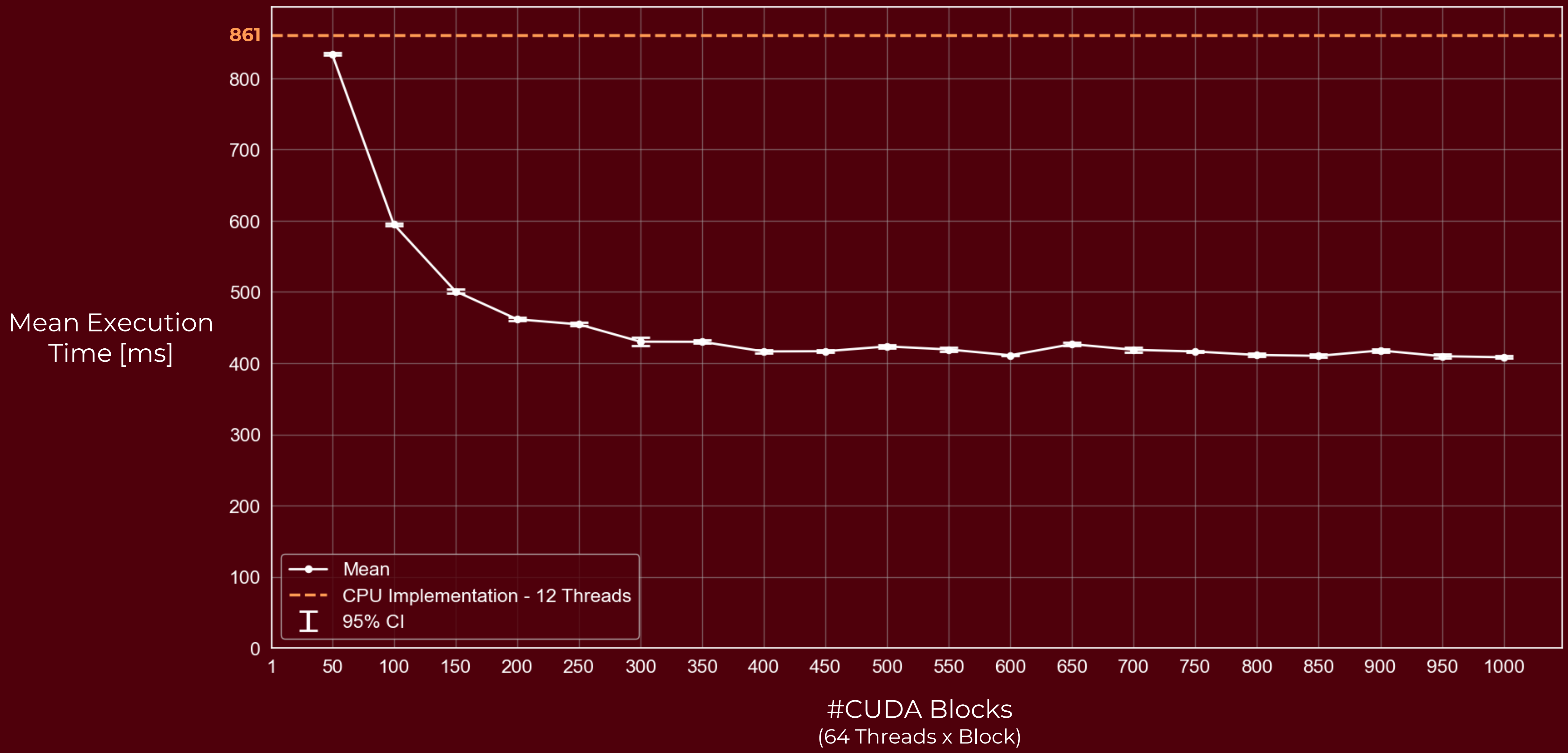


EXECUTION TIME

OPTIMIZED VERSION ON GPU

#ITERATIONS : **30**

20 DIGITS **17975734686214396237**

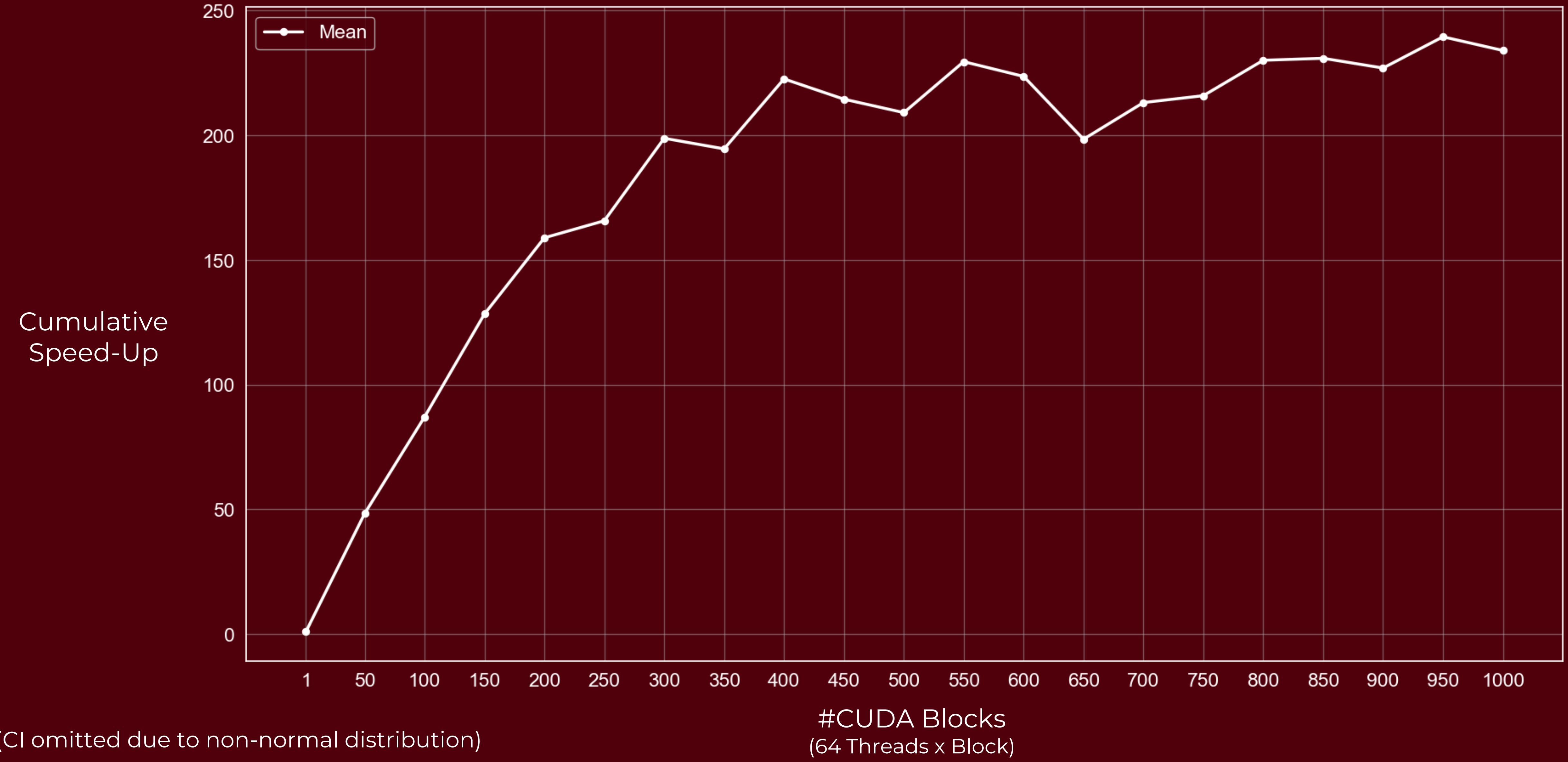


SPEED-UP

OPTIMIZED VERSION ON GPU

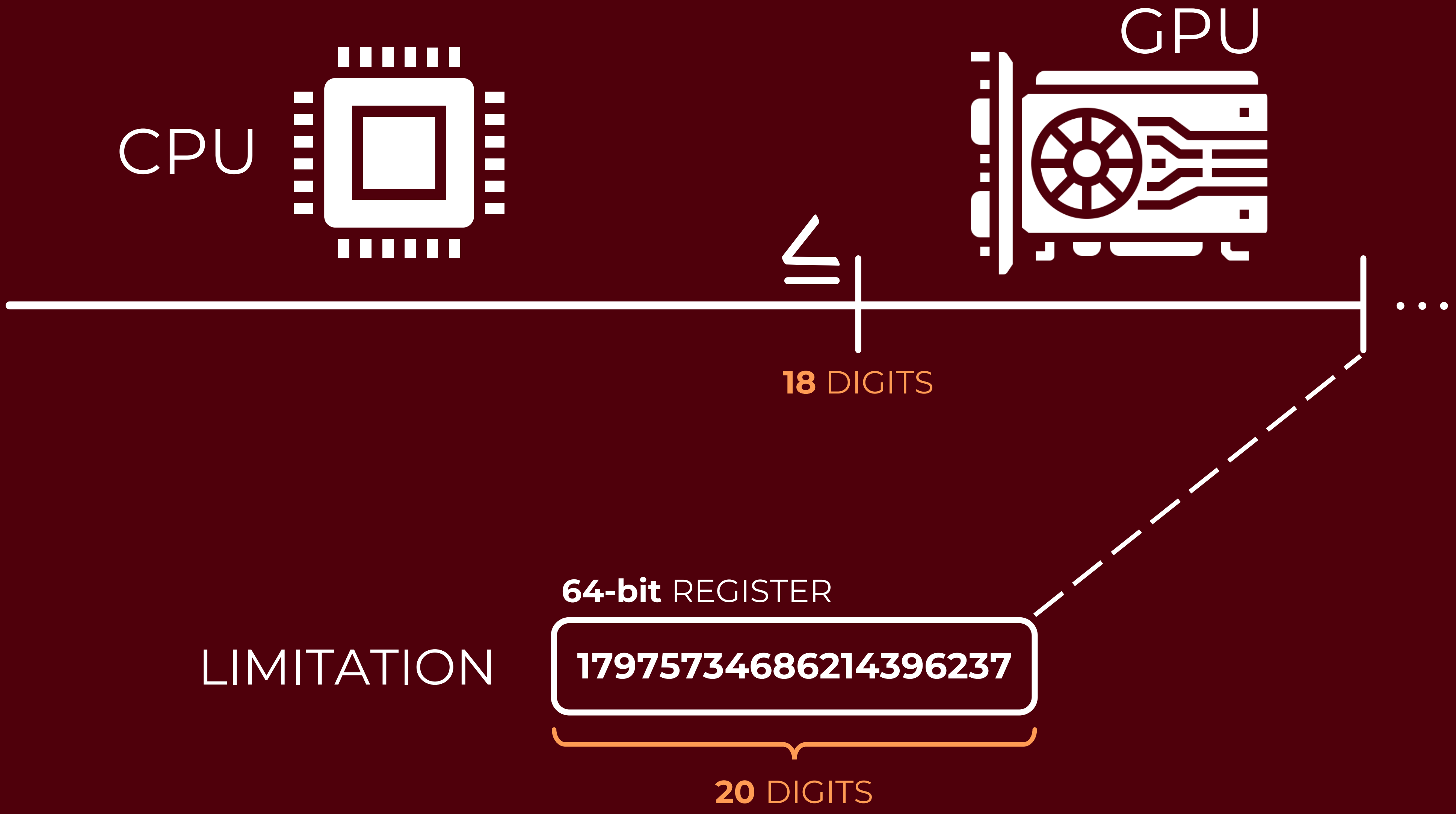
#ITERATIONS : **30**

20 DIGITS **17975734686214396237**



(CI omitted due to non-normal distribution)

CONCLUSIONS



THANKS
FOR THE
ATTENTION