# Relatório

## Objetivo do laboratório:

- O objetivo desse lab3 é demonstrar como a capacidade de processamento de dados de forma paralela pode possuir um tempo de execução menor em comparação à execução de atividades sequenciais em um programa CPU Bound.

## Lista de programas usados:

Esta secção mostra como executar de maneira correta os programas para observar os resultados.

- **geramatriz.c** -> Programa usado para gerar um arquivo binário com uma **matriz** de números aleatórios as dimensões desejadas pelo usuário.

```
gcc geramatriz.c -o geramatriz <numero_de_linhas> <numero_de_colunas>
```

  - **Execução**:

```
./geramatriz 500 500 <nome_do_binario>
```

- **lematriz.c** -> Programa usado para ler um arquivo binário e apresentar seu conteúdo na tela.

```
gcc lematriz.c -o leamatriz
```

  - **Execução**:

```
./lematriz <nome_do_binario>
```

- **sequencial.c** -> Programa usado para multiplicar duas matrizes (contidas em arquivos binários) com dimensões M por N de forma sequencial e escrever o resultado e um arquivo binário. O programa também impreme o tempo que

```
gcc sequencial.c -o sequencial <nome_matriz1> <nome_matriz2> <nome_matriz_do_resultado>
```

  - **Execução**:

```
./sequencial <matriz1> <matriz2> <matriz_do_resultado>
```

- **concorrente.c** -> Programa semelhante ao **sequencial.c** porém realiza a operação de multiplicação de matrizes de forma concorrente

```
gcc sequencial.c -o sequencial <nome_matriz1> <nome_matriz2> <nome_matriz2>
```

- **Execução**:

```
./concorrente <matriz1> <matriz2> <matriz3> <numero_de_threads>
```

## Configuração da Máquina

- Todos os programas foram executados em uma máquina virtual com Kali Linux.

CPU: Intel Core i3 10105F CPU 3.70GHz (4 núcleos)
GPU: NVIDIA GeForce GTX 1650
Memory: 9500MiB

## Análise de tempo de execução

- Todos os tempos abaixo são um média aritmética de três execuções dos programas.

### Sequencial

| | Tempo de execução total (segundos) | Tempo de preparação (segundos) | Tempo de processamento (segundos) | Tempo de finalização (segundos) |
|---|---|---|---|---|
| matriz1_500x500 . matriz2_500x500 | 0.39629466 | 0.0015023 | 0.39225 | 0.0025393 |
| matriz1_1000x1000 . matriz2_1000x1000 | 3.4989293 | 0.0047546 | 3.48200233 | 0.012172333 |
| matriz1_2000x2000 . matriz2_2000x2000 | 48.06427199 | 0.011933 | 47.99428366 | 0.05805466 |

## Concorrente 1 thread

| | Tempo de execução total (segundos) | Tempo de preparação (segundos) | Tempo de processamento (segundos) | Tempo de finalização (segundos) |
|---|---|---|---|---|
| matriz1_500x500 . matriz2_500x500 | 0.56381166666 | 0.001154333333 | 0.561533666 | 0.001123666666 |
| matriz1_1000x1000 . matriz2_1000x1000 | 6.0628103333 | 0.00376 | 6.055683666666667 | 0.0033666666 |
| matriz1_2000x2000 . matriz2_2000x2000 | 68.017932 | 0.0094466666666 | 67.997206666 | 0.0112786666666 |

## Concorrente 2 threads

| | Tempo de execução total (segundos) | Tempo de preparação (segundos) | Tempo de processamento (segundos) | Tempo de finalização (segundos) |
|---|---|---|---|---|
| matriz1_500x500 . matriz2_500x500 | 0.317127 | 0.00176033333 | 0.31394766666 | 0.001419000000 |
| matriz1_1000x1000 . matriz2_1000x1000 | 3.16676433 | 0.003401333 | 3.158213 | 0.005150 |
| matriz1_2000x2000 . matriz2_2000x2000 | 34.651016 | 0.0801966666 | 34.556617 | 0.01420233333333 |

## Concorrente 4 threads

| | Tempo de execução total (segundos) | Tempo de preparação (segundos) | Tempo de processamento (segundos) | Tempo de finalização (segundos) |
|---|---|---|---|---|
| matriz1_500x500 . matriz2_500x500 | 0.2334376665 | 0.0009976666 | 0.230909 | 0.001531 |
| matriz1_1000x1000 . matriz2_1000x1000 | 2.07622066666 | 0.002862999 | 2.0680176666 | 0.00534 |
| matriz1_2000x2000 . matriz2_2000x2000 | 22.371884666 | 0.00992100 | 22.35016066 | 0.011803 |

### Concorrente 8 threads

| | Tempo de execução total (segundos) | Tempo de preparação (segundos) | Tempo de processamento (segundos) | Tempo de finalização (segundos) |
|---|---|---|---|---|
| matriz1_500x500 . matriz2_500x500 | 0.235331666666 | 0.00149233333333 | 0.231527999999 | 0.0023113333333 |
| matriz1_1000x1000 . matriz2_1000x1000 | 2.17059233 | 0.0048623333333 | 2.16176066 | 0.00396933 |
| matriz1_2000x2000 . matriz2_2000x2000 | 23.41714533 | 0.0111756 | 23.39367133 | 0.0122333 |

## Gráficos de Aceleração e Eficiência

### Considerações e Legenda

- Os gráficos abaixo comparam a Aceleração e a Eficência da abordagem **sequencial** em relação a abordagem **concorrente**, para as dimensões 500x500, 1000x1000 e 2000x2000 das matrizes.
- Os gráficos possuem os eixos: y = Aceleração / Eficiência e x = Número de Threads.
- Cálculo **Aceleração**:

$$A(n,p) = T_s(n)/T_p(n,p)$$

- Cálculo **Eficiência**

$$E(n,p) = A/p$$

Sendo A = **aceleração** e p = **número de núcleos do processador**

## Matriz 500x500

- **Tempo Sequencial** = 0.39629466
- **Tempo concorrente 1 Thread** = 0.56381166666
- **Tempo concorrente 2 Thread** = 0.317127
- **Tempo concorrente 4 Thread** = 0.2334376665
- **Tempo concorrente 8 Thread** = 0.235331666666

**Gráfico Aceleração**

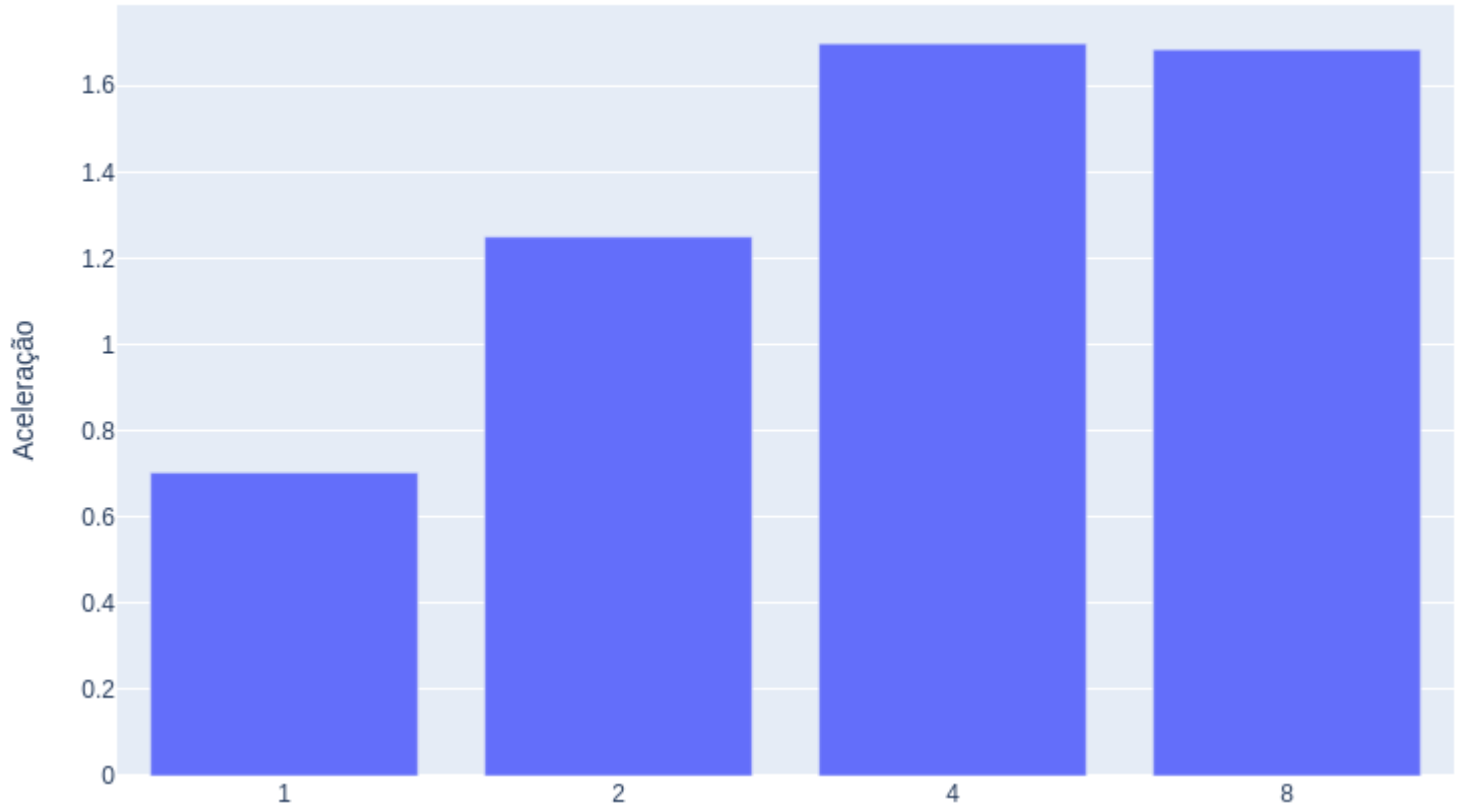
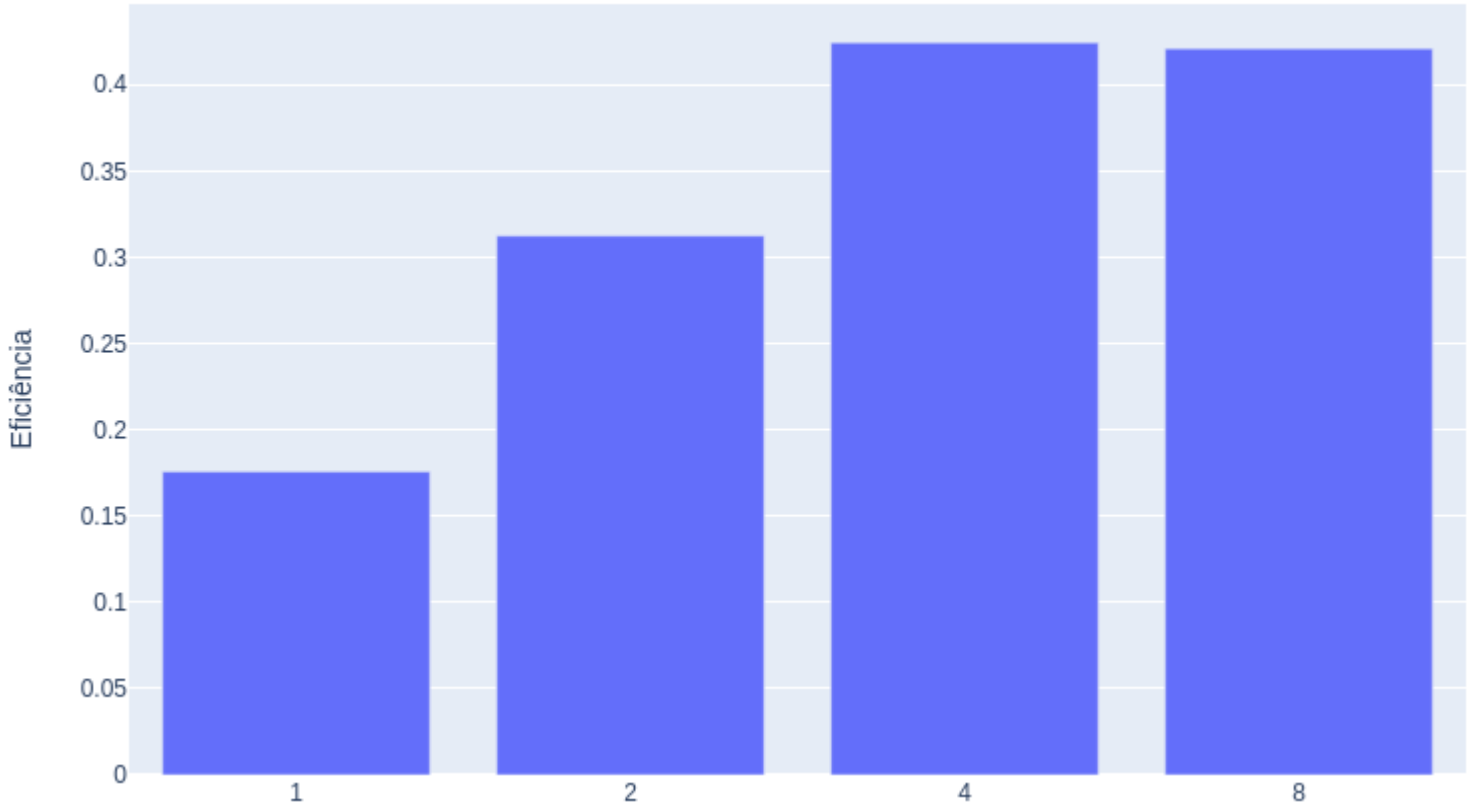


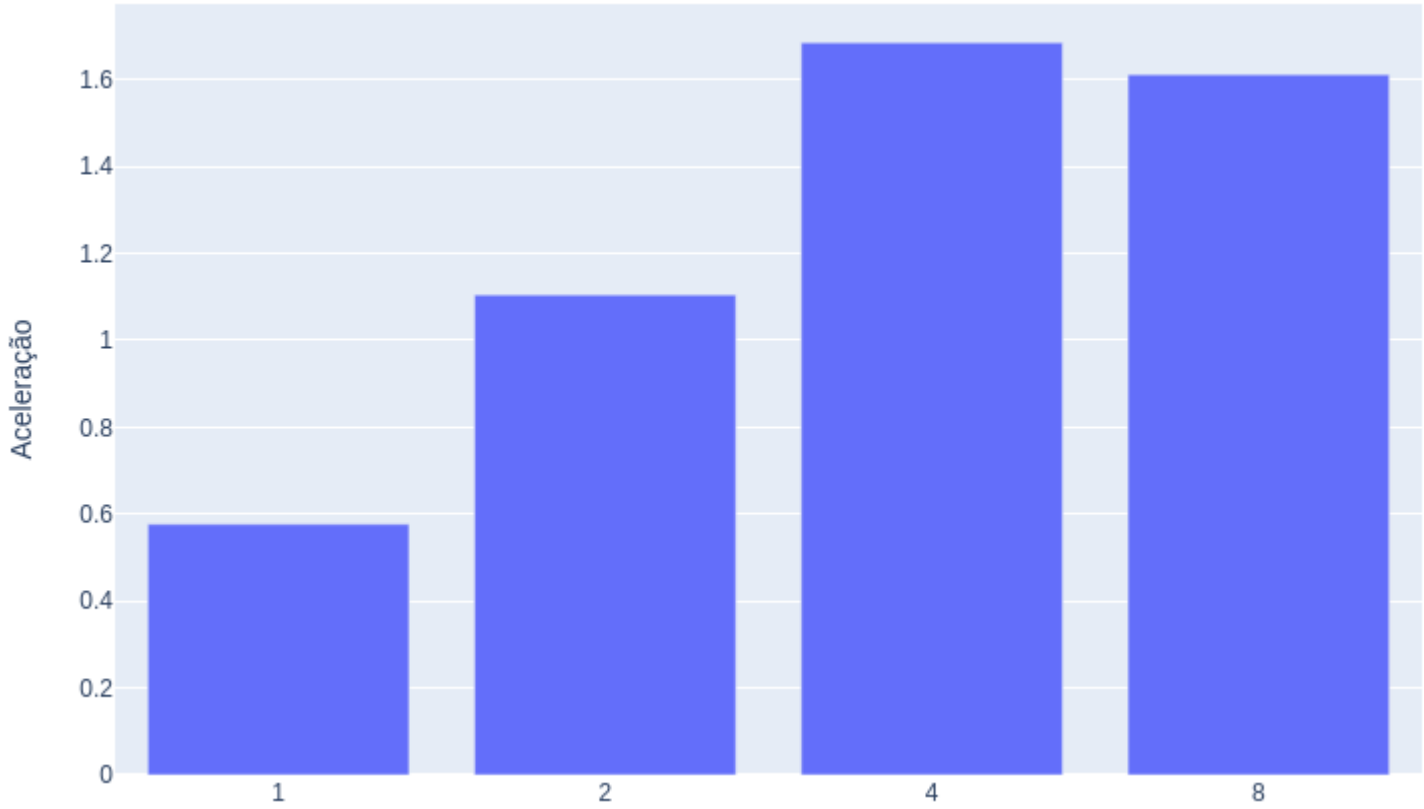

Gráfico Eficiência



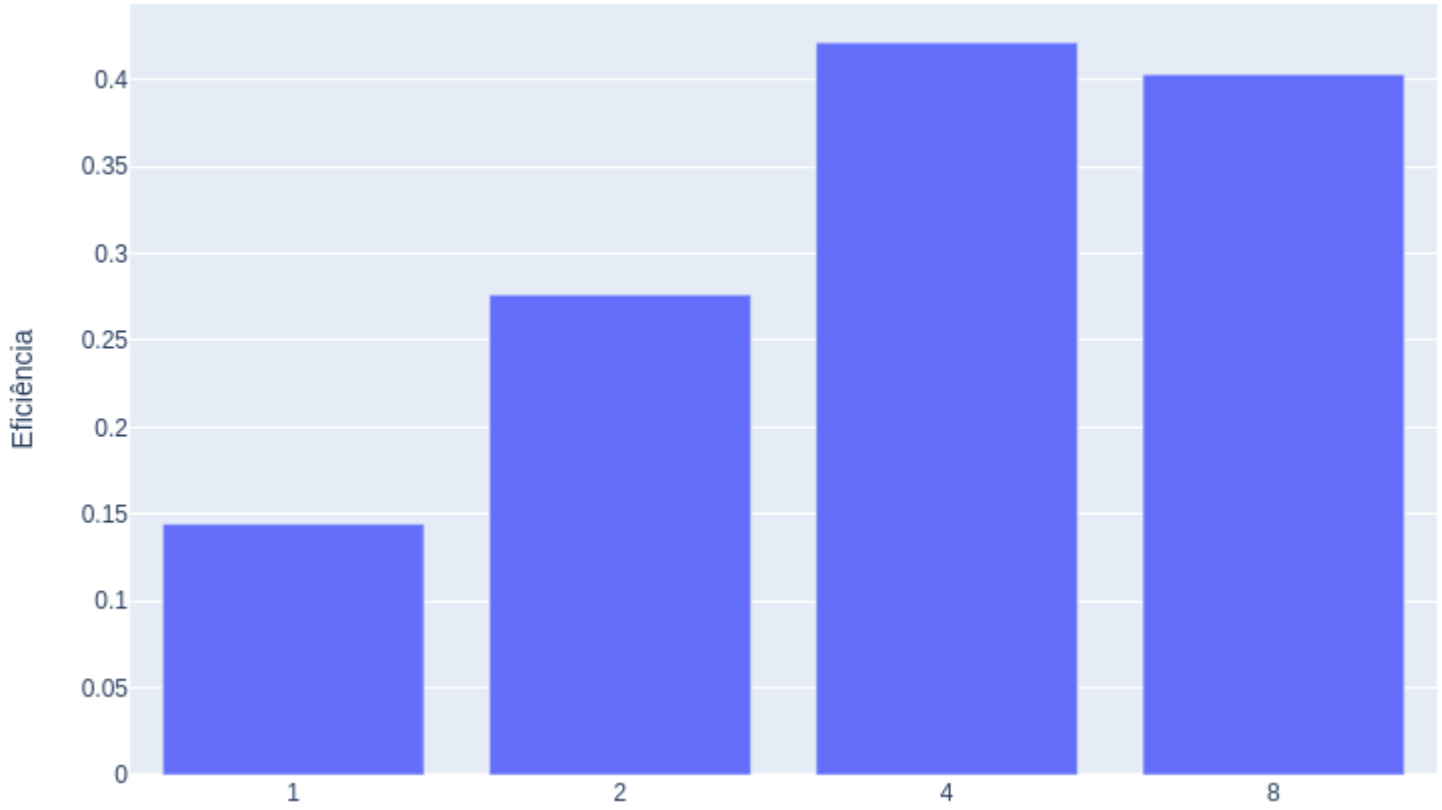

## Matriz 1000x1000

- **Tempo Sequencial** = 3.4989293
- **Tempo concorrente 1 Thread** = 6.0628103333
- **Tempo concorrente 2 Thread** = 3.16676433
- **Tempo concorrente 4 Thread** = 2.07622066666
- **Tempo concorrente 8 Thread** = 2.17059233

### Gráfico Aceleração




### Gráfico Eficiência




## Matriz 2000x2000

- **Tempo Sequencial** = 48.06427199
- **Tempo concorrente 1 Thread** = 68.017932
- **Tempo concorrente 2 Thread** = 34.651016
- **Tempo concorrente 4 Thread** = 22.371884
- **Tempo concorrente 8 Thread** = 23.41714533

### Gráfico Aceleração

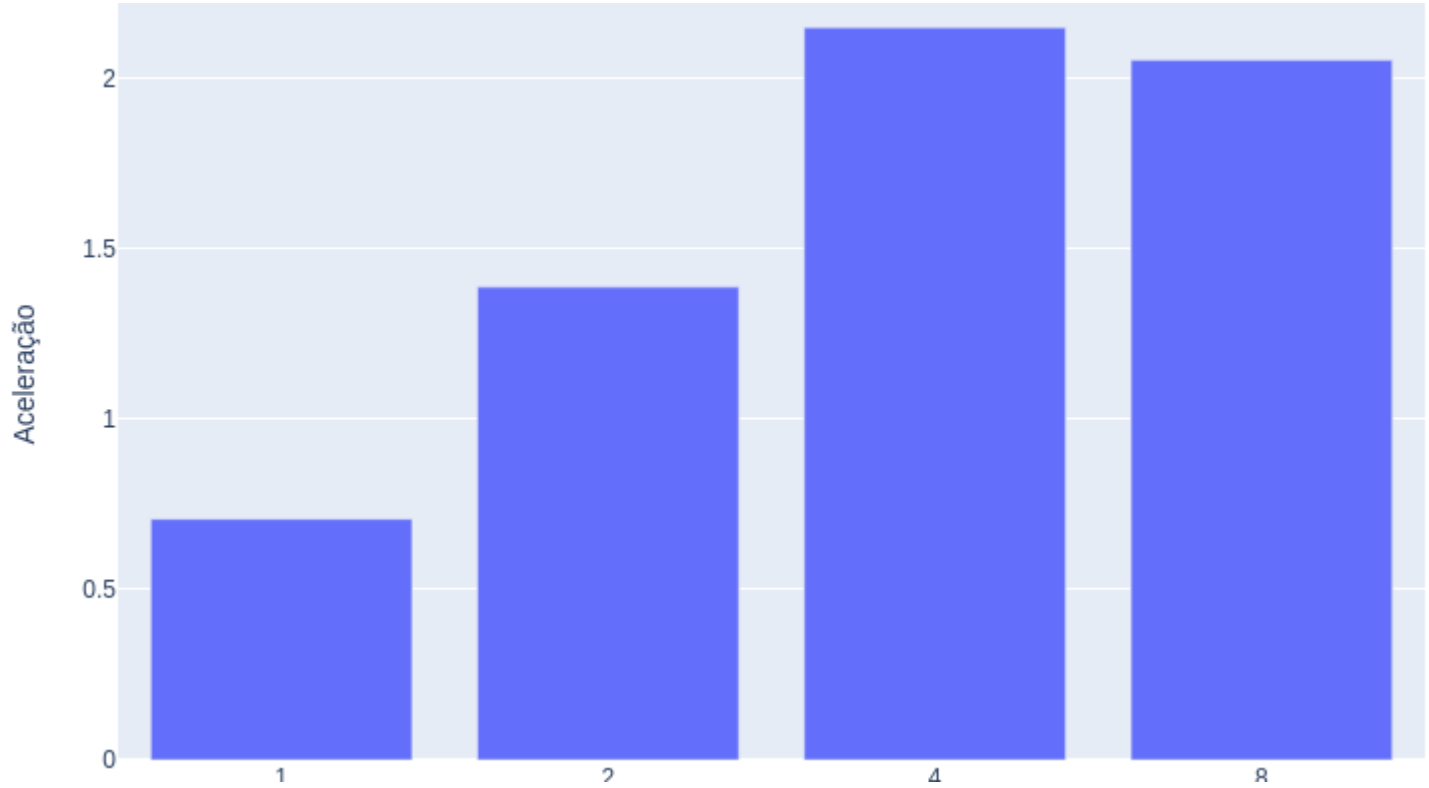
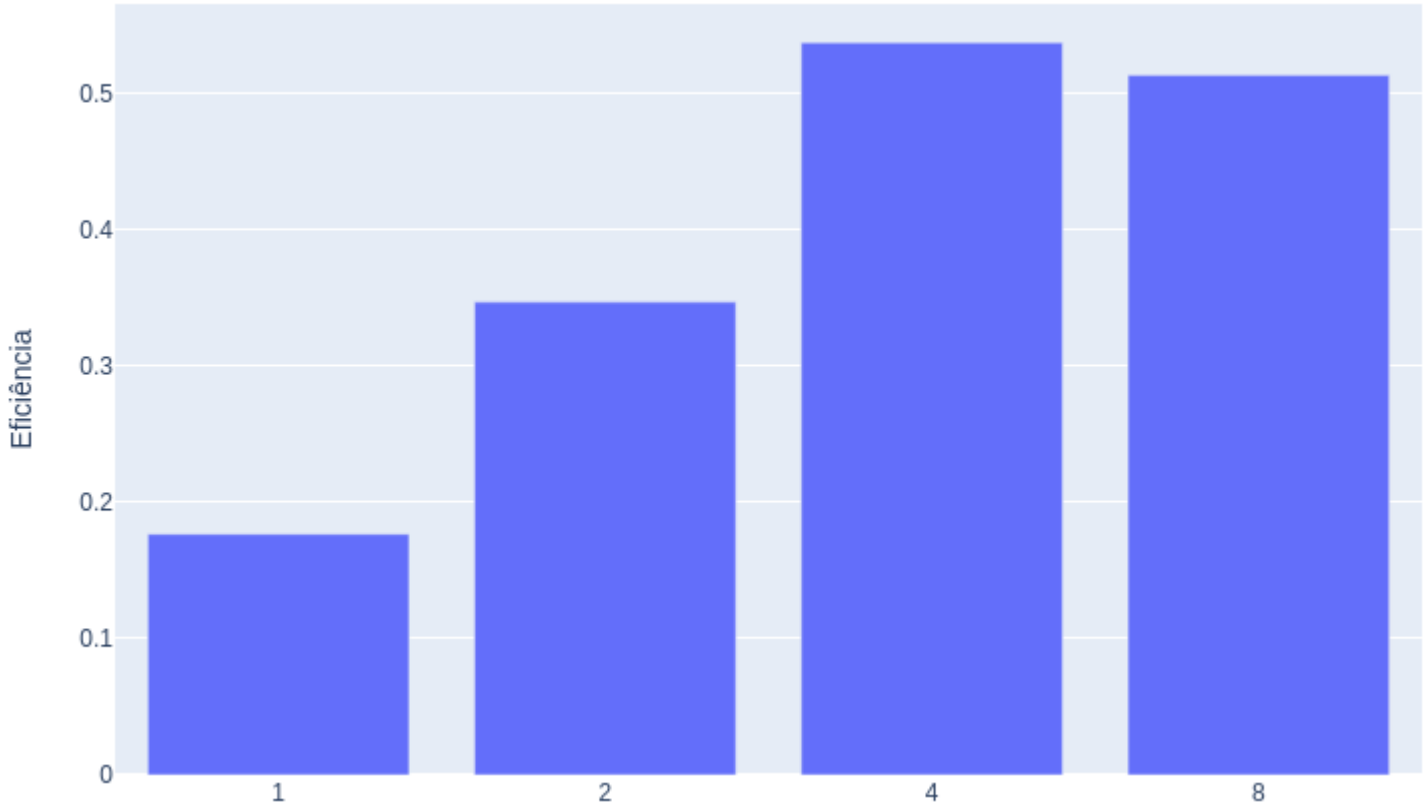



Gráfico Eficiência




## Conclusão

Conclusões que podem ser tiradas:

- Como vimos, ao executar a tarefa de multiplicar duas matrizes possuímos muitos ganhos de aceleração quando fazemos uso das técnicas de concorrência.

### Por que isso acontece?

- Com mais de duas threads, ou seja, mais de dois fluxos de execução diferentes, começamos a ter ganhos significativos em aceleração por conta do **paralelismo** das atividades. Isso significa que uma linha pode estar sendo multiplicada e no **MESMO** instante de tempo outra linha também está sendo multiplicada em outro núcleo! O que faz com que o programa execute mais coisas em **MENOS** tempo.

### Pequena diferença entre 4 threads e 8 threads.

- Minha máquina possui apenas 4 núcleos, apesar de possuirmos mais threads não podemos abusar do parelelismo, ficamos limitados a apenas 4 threads sendo executadas ao mesmo tempo.
- Algo que notei é: Crescendo até um ponto bem alto de threads, tive um pouco mais de aceleração. Acredito que isso se deve pelo fato de termos menos linhas sendo executadas por threads,então é possível liberar os núcleos mais rapidamente.
- Porém se esse número cresce demais, acabamos por PERDER aceleração por conta de muitas trocas de contextos entre os processos.