|  |  |
| --- | --- |
|  | **UNIVERSIDADE FEDERAL DE LAVRAS**  **Departamento de Computação Aplicada** |

**Relatório para atividade de análise de desempenho - REO 6**

|  |
| --- |
| **Código e Nome: GCC177 – Programação Paralela e Concorrente** |
| **Nome do Aluno:** Lucas Antonio Lopes Neves |

**Ambiente de teste**

Processador: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz

Número de núcleos de processamento: 4

Número de threads: 8

**Resultados obtidos**

Os códigos (programa paralelizado, script de execução e script de análise) utilizados, os resultados colhidos e as imagens utilizadas podem ser consultados [neste repositório no GitHub](https://github.com/lucas54neves/sieve-eratosthenes-threads).

O teste em questão foi feito separando 7 quantidades de números distintas e 4 números de threads distintos. Para os números foram escolhidas as quantidades 100 (cem), 1.000 (mil), 10.000 (dez mil), 100.000 (cem mil), 1.000.000 (um milhão), 10.000.000 (10 milhões) e 100.000.000 (cem milhões). Para os threads, foram escolhidas: 1, 2, 4 e 8 threads. Para cada combinação entre números e threads, o programa foi executado 10 vezes. Totalizando 280 execuções.

Os tempos das execuções podem ser observados nas seguintes tabelas:

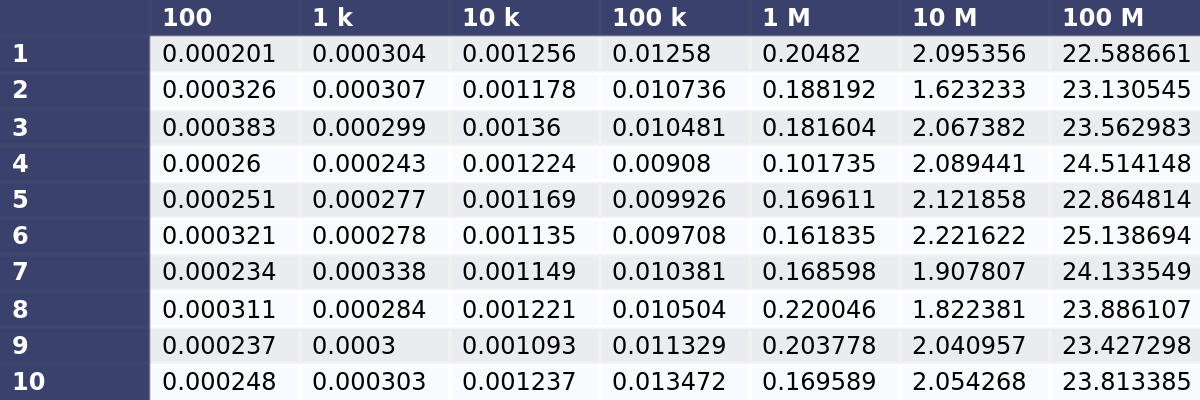


Tabela - Tempos em segundos das execuções com um thread

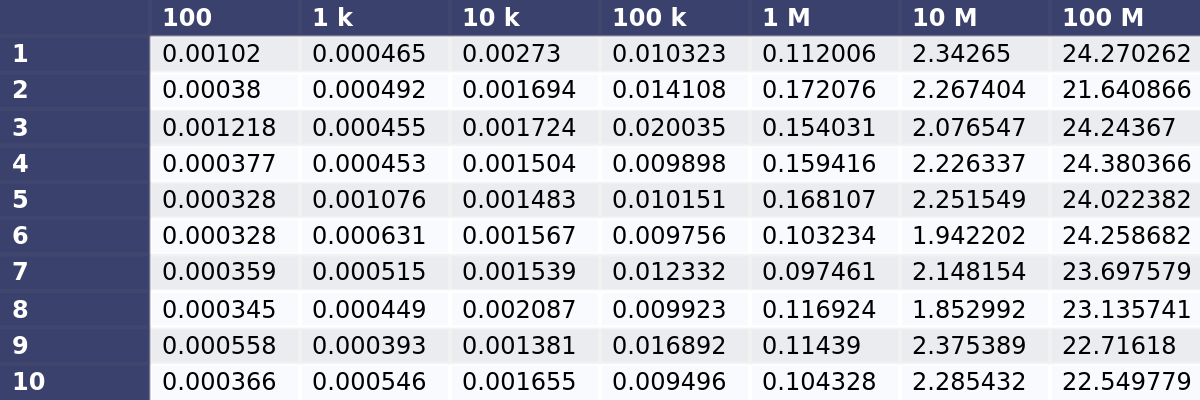


Tabela - Tempos em segundos das execuções com dois threads

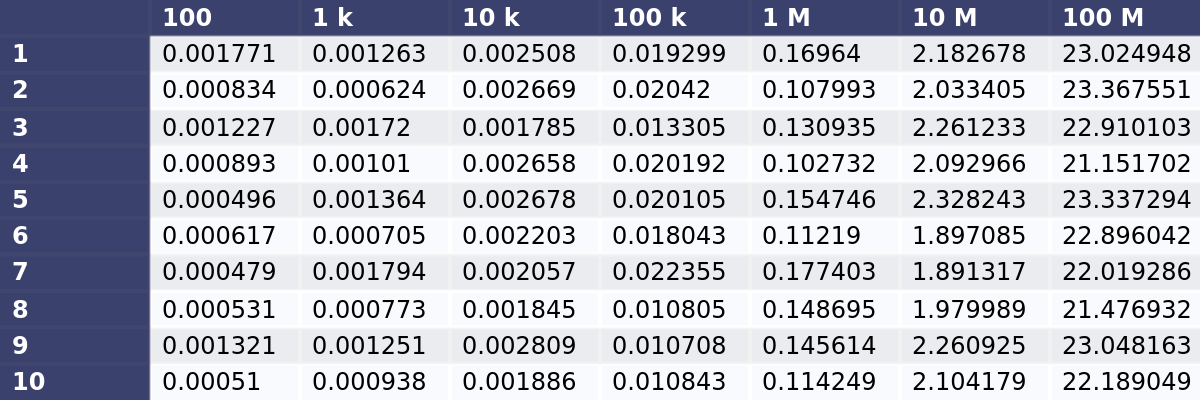


Tabela - Tempos em segundos das execuções com quatro threads

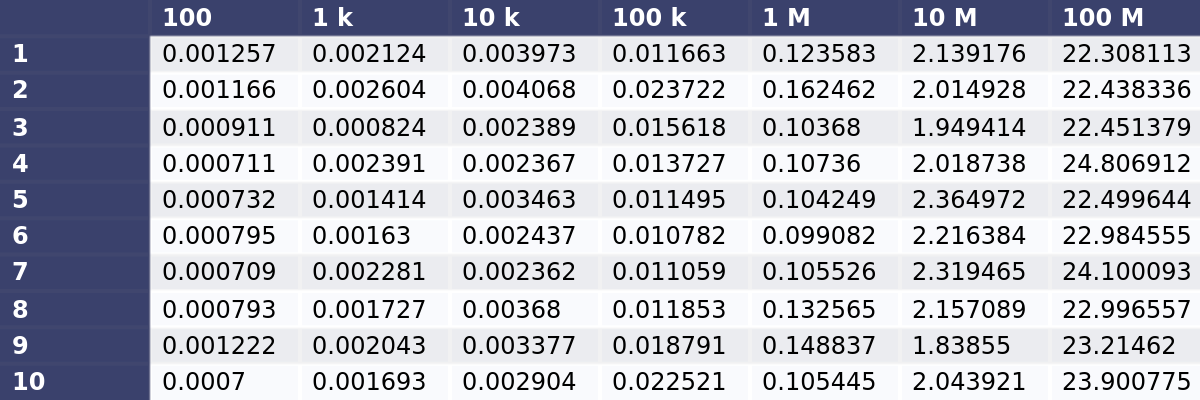


Tabela - Tempos em segundos das execuções com oito threads

O tempo médio e o desvio padrão podem ser observados nas tabelas a seguir:

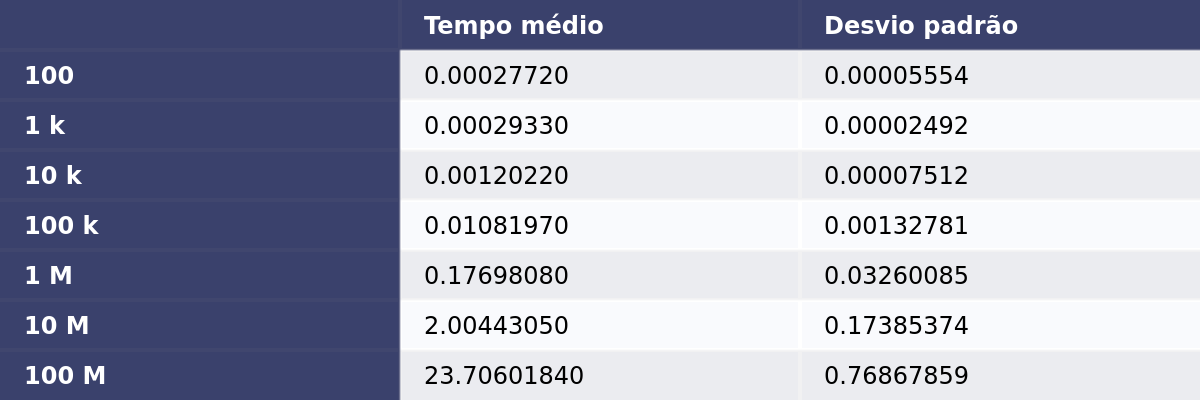


Tabela - Tempo médio e desvio padrão das execuções com um thread

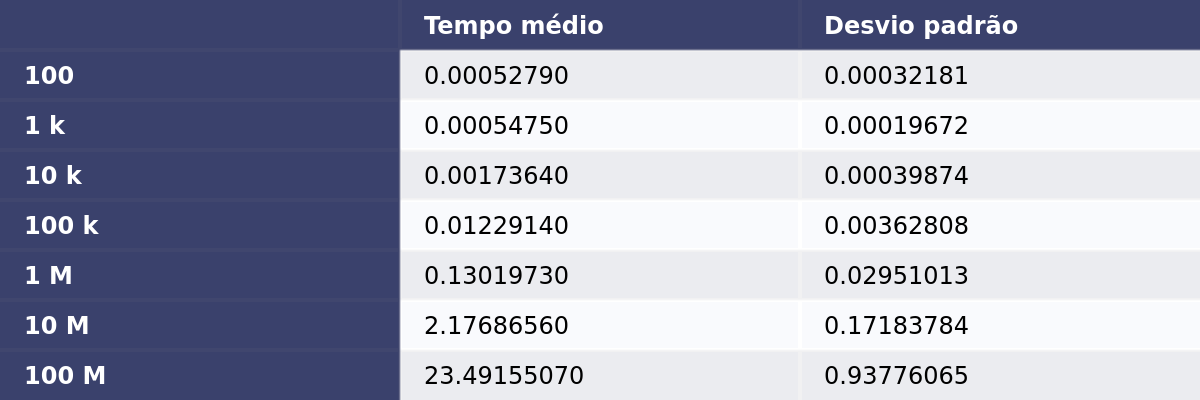


Tabela - Tempo médio e desvio padrão das execuções com dois threads

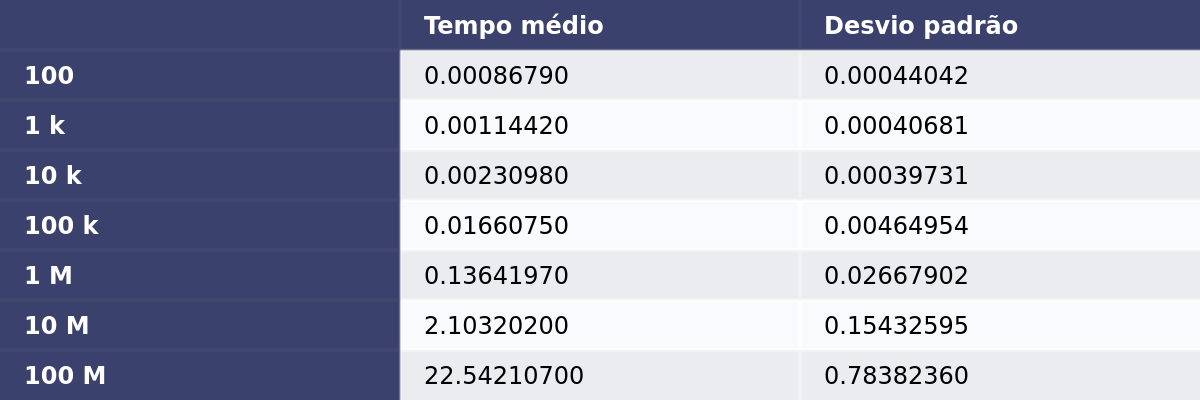


Tabela - Tempo médio e desvio padrão das execuções com quatro threads

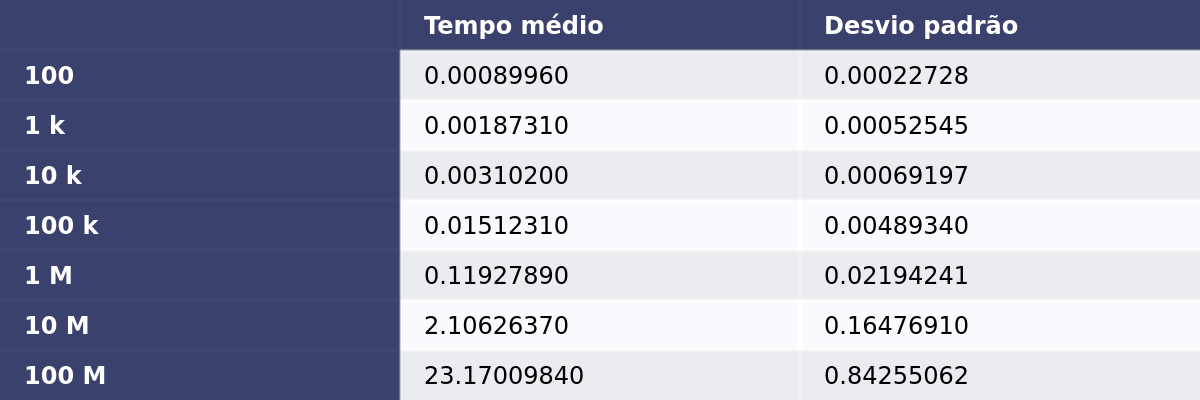


Tabela - Tempo médio e desvio padrão das execuções com oito threads

A seguir, os gráficos das médias e dos desvios padrões são apresentados.

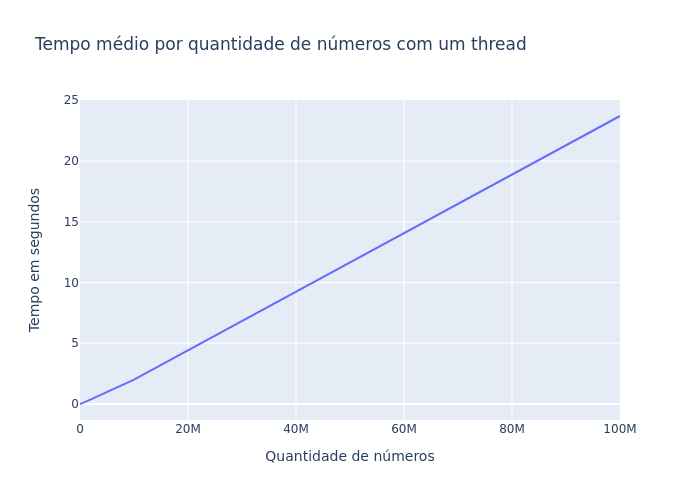


Figura - Gráfico da quantidade de números pelo tempo em segundos para um thread

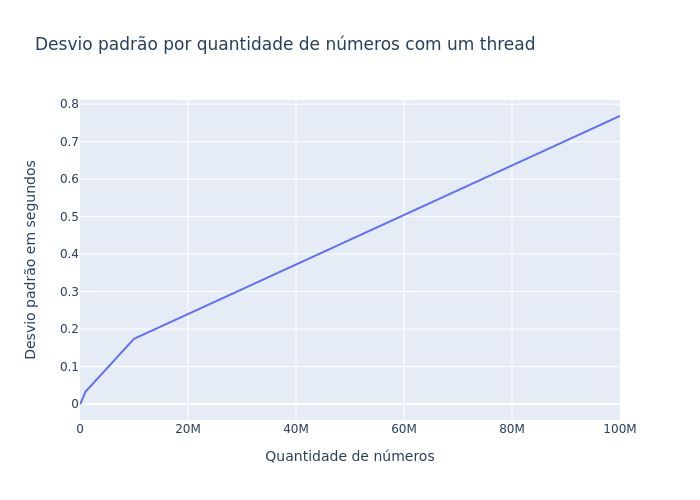


Figura - Gráfico da quantidade de números pelo desvio padrão para um thread

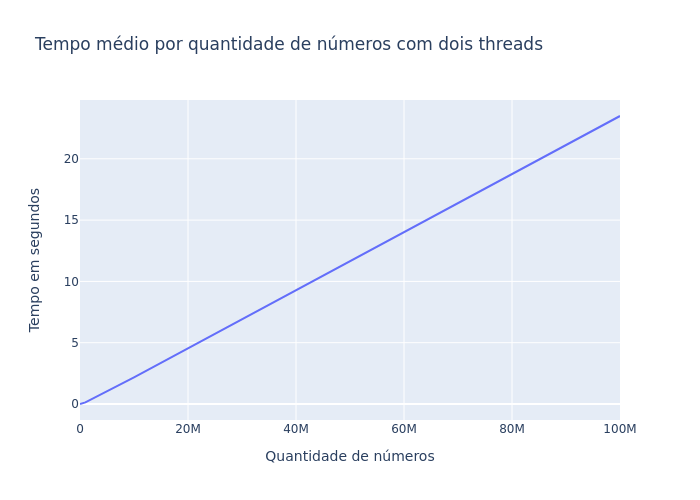


Figura - Gráfico da quantidade de números pelo tempo em segundos para dois threads

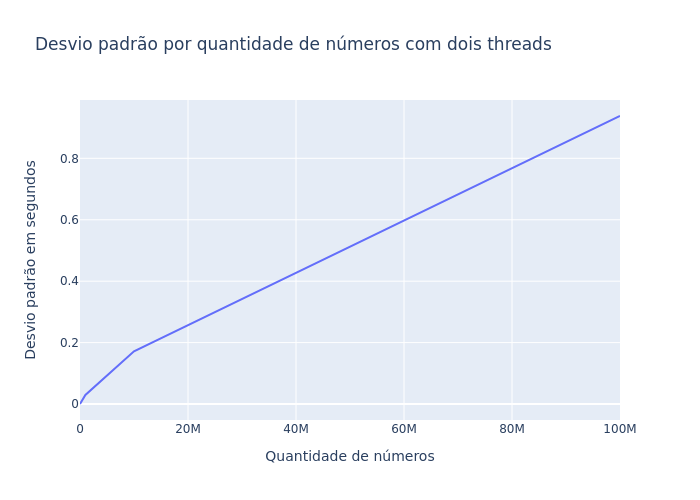


Figura - Gráfico da quantidade de números pelo desvio padrão para dois threads

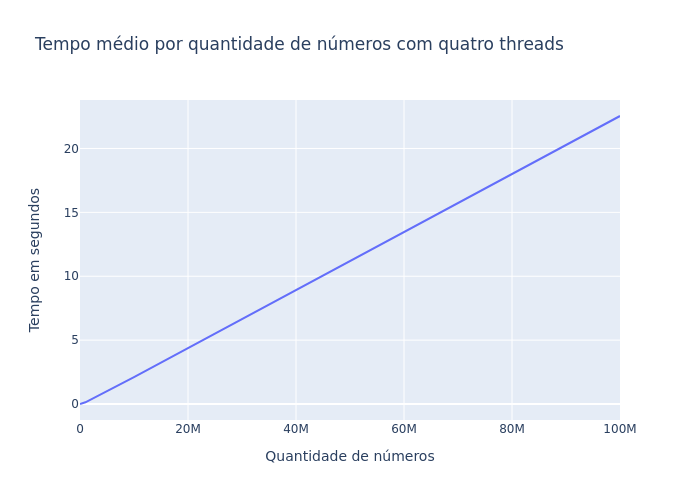


Figura - Gráfico da quantidade de números pelo tempo em segundos para quatro threads

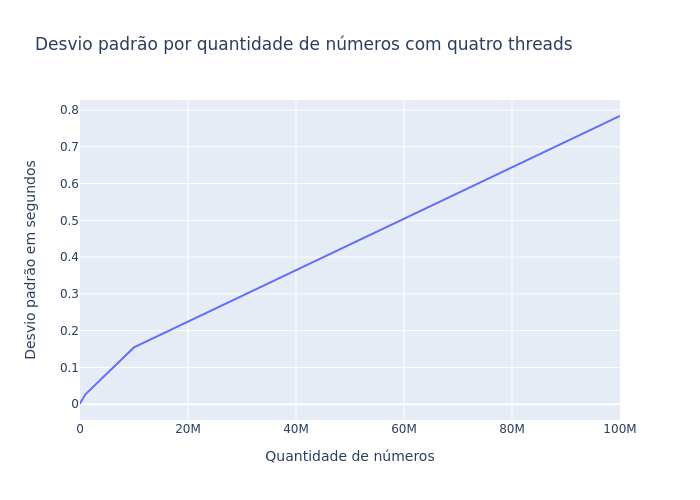


Figura - Gráfico da quantidade de números pelo desvio padrão para quatro threads

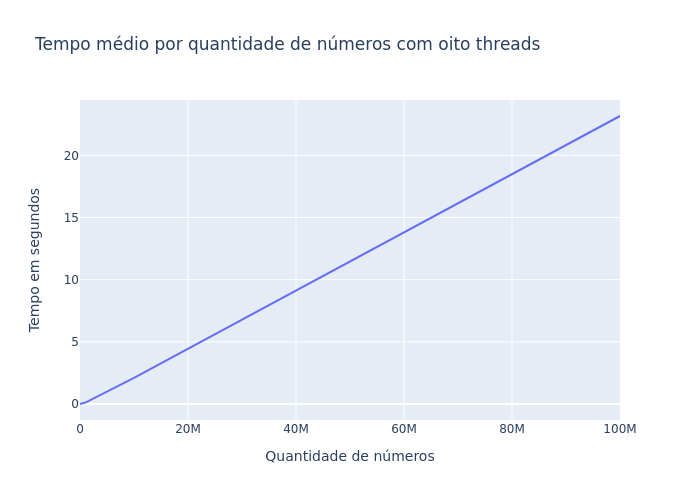


Figura - Gráfico da quantidade de números pelo tempo em segundos para oito threads

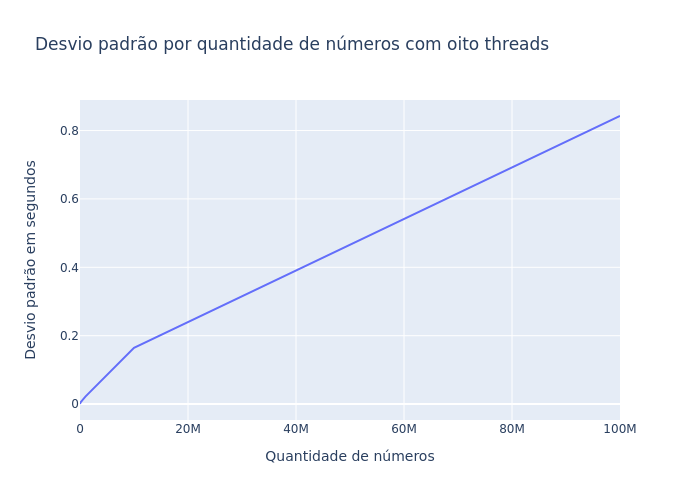


Figura - Gráfico da quantidade de números pelo desvio padrão para oito threads

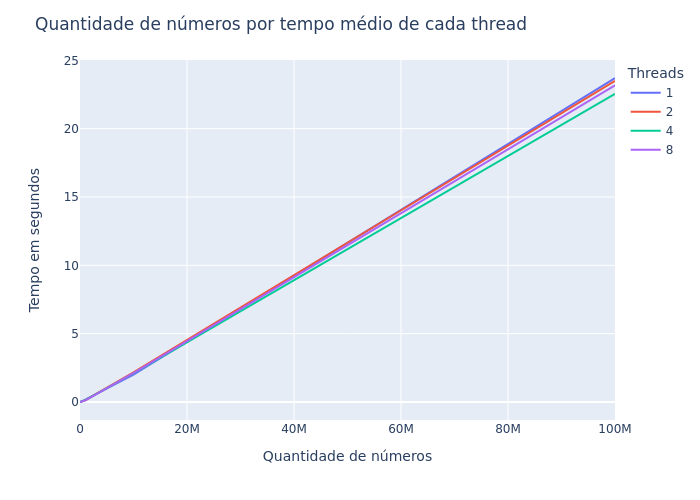


Figura - Gráfico da quantidade de números pelo tempo médio para cada thread

Observando principalmente a imagem 13, nota-se que o aumento do tempo de execução é proporcional ao aumento da quantidade de números, ou seja, quanto maior a quantidade de números, maior será o tempo gasto para calcular todos os números.

As execuções com quatro threads são as execuções mais rápidas. As execuções com apenas um thread são as execuções mais lentas. Por fim, as execuções com oito threads são as execuções mais rápidas entre as intermediárias e as execuções com dois threads são as execuções mais lentas.

Foram calculados o speedup pada cada thread e quantidade de números. Os resultados são apresentados na Tabela 9 e na Figura 10.

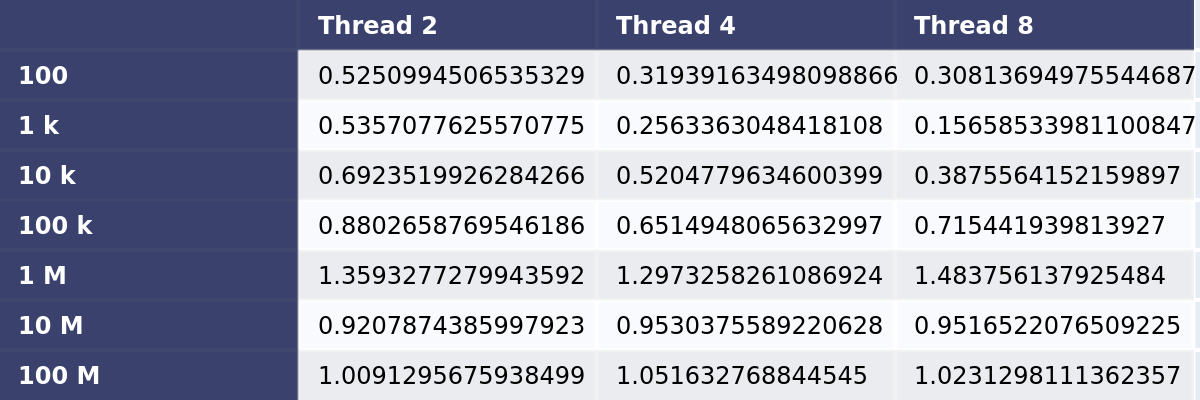


Tabela - Tabela de speedup por quantidade de números e threads

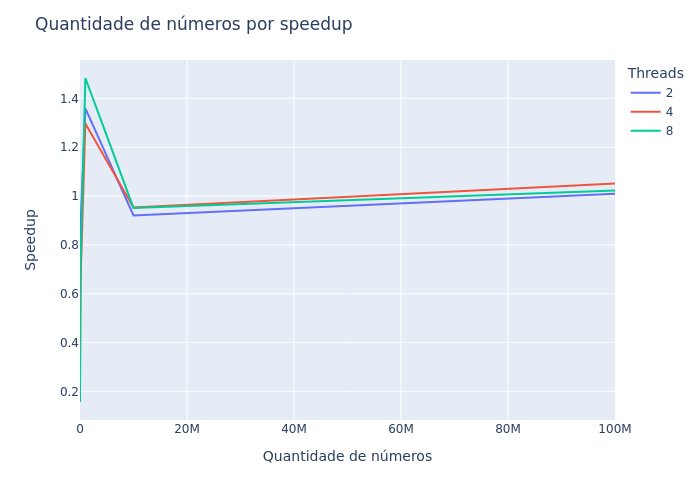


Figura - Gráfico de quantidade de números por speedup para cada thread

A partir dos resultados da Tabela 9 e da Figura 10 é possível é possível verificar que, até uma quantidade de dez milhões de números, o speedup reduz proporcionalmente ao aumento da quantidade de números. A partir de dez milhões de números, o speedup aumenta proporcionalmente ao aumento da quantidade de número.

Analisando a Figura 10, é possível verificar que um aumento na quantidade de threads não significa, necessariamente, um ganho de desempenho, visto que as execuções com quatro threads possuem um speedup maior a partir de trinta milhões de números. Isso ocorre pois os threads precisam ser criados e sincronizados.

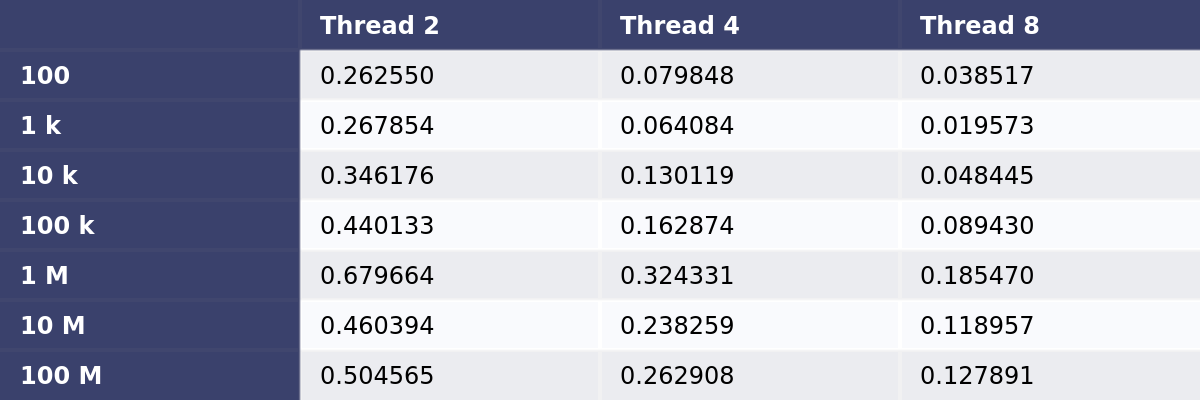


Figura – Resultados do cálculo da eficiência

Chart, line chart

Description automatically generated

Figura - Gráfico de quantidade de números por eficiência para cada thread

Ao se utilizar oito threads, a eficiência é baixa pois o tempo de ociosidade e o desperdício de energia são maiores. A eficiência ao se utilizar quatro threads é média e, utilizando dois threads, a eficiência é maior.

Em seguida foi calculado a fração sequencial definida experimentalmente utilizando a métrica de Karp-Flatt. Essa métrica define qual é a fração sequencial presente no programa levando em conta o overhead gerado ao se paralelizar um programa. Quanto menor o valor de e melhor é a paralelização de um dado programa.

A fórmula para o cálculo da métrica Karp-Flatt, onde S é o valor de speedup e p é o número de threads, é a seguinte:

A picture containing text, clock

Description automatically generated

Figura - Fórmula da Métrica de Karp-Flatt

Os resultados da métrica de Karp-Flatt estão a seguir na Figura 13:

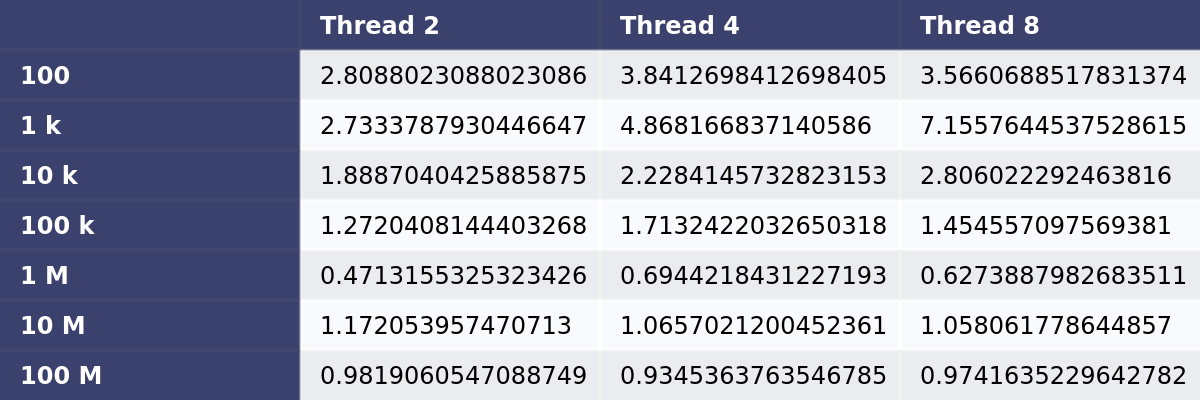


Figura - Resultado da métrica de Karp-Flatt

O gráfico da quantidade de números pela métrica de Karp-Flatt pode ser observado a seguir na Figura 13:

Chart

Description automatically generated

Figura - Quantidade de números pela métrica de Karp-Flatt

**Conclusão**

A partir desta análise, é possível concluir que aumentar a quantidade de threads não significa, necessariamente, mais desempenho. Como os threads precisam de tempo para serem criados e sincronizados é preciso saber se compensa utilizá-los.

Considerando todas as análises feitas, o desempenho máximo ocorre quando utilizado quatro threads. A utilização de oito threads e dois threads tiveram um desempenho intermediário, mas, nesse caso, oito threads tiveram um desempenho melhor.