Relatório de Aceleração e Eficiência para Multiplicação de Matrizes em Concorrência

Tabelas comparativas entre médias de tempo, para 3 execuções, com cada tamanho de matriz e quantidade de threads

Inicialização	Sequencial	1	2	4	8
500x500	0.006	0.007	0.006	0.006	0.006
1000x1000	0.011	0.013	0.014	0.012	0.013
2000x2000	0.051	0.041	0.040	0.035	0.036

Processamento	Sequencial	1	2	4	8
500x500	0.387	0.442	0.240	0.141	0.093
1000x1000	3.394	4.028	2.129	1.190	0.842
2000x2000	33.659	53.767	28.266	14.866	9.649

Finalização	Sequencial	1	2	4	8
500x500	0.002	0.002	0.002	0.002	0.002
1000x1000	0.006	0.008	0.008	0.007	0.006
2000x2000	0.022	0.026	0.025	0.024	0.024

Considerações sobre a tabela gerada

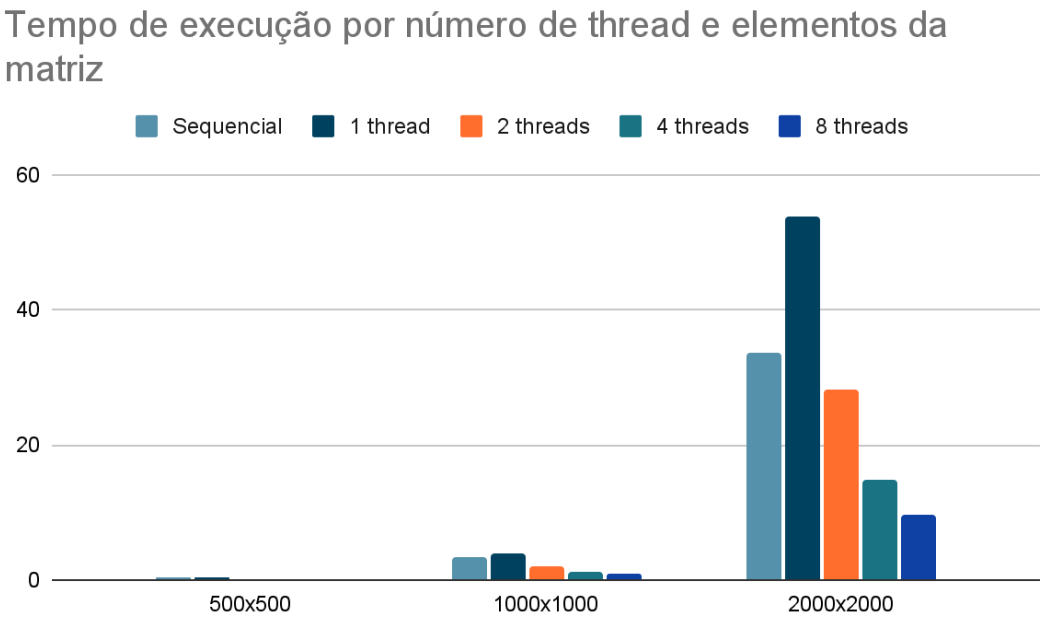
Podemos notar que, mesmo alterando o número de threads, quase não há diferença no tempo de inicialização e finalização de dados para um mesmo tamanho de matriz. Isso ocorre porque esse processo não possui concorrência nesse processo..

Analisando especificamente os dados de processamento, percebemos que com o aumento do tamanho da matriz, seu tempo aumenta de forma exponencial. Para cada dobro de tamanho, existe um aumento de cerca de 10x no tempo total de cálculo.

Além disso, percebemos que com o dobro de threads, existe um tempo que representa quase metade do tempo anterior, porém com limitações. Não há um ganho exato de “dobro de thread, metade do tempo”, mas se aproxima disso.

Por fim, podemos perceber que o método sequencial executa os cálculos mais rapidamente que o método concorrente com uma thread, já que existe perda no tempo de criação das threads.

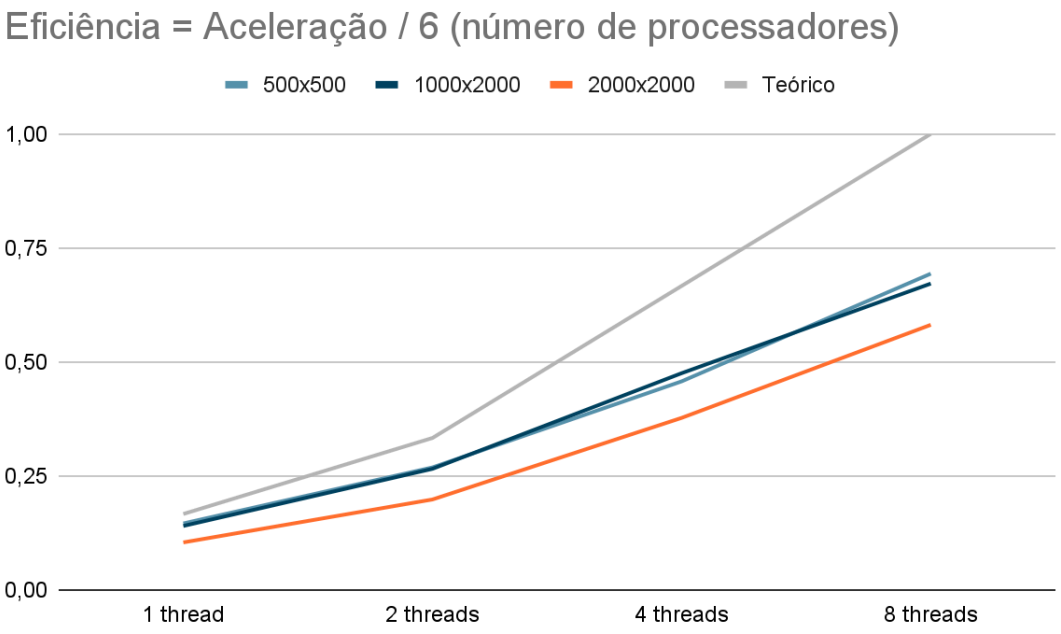
Gráfico que correlaciona os tempos de processamento com a quantidade de threads utilizada

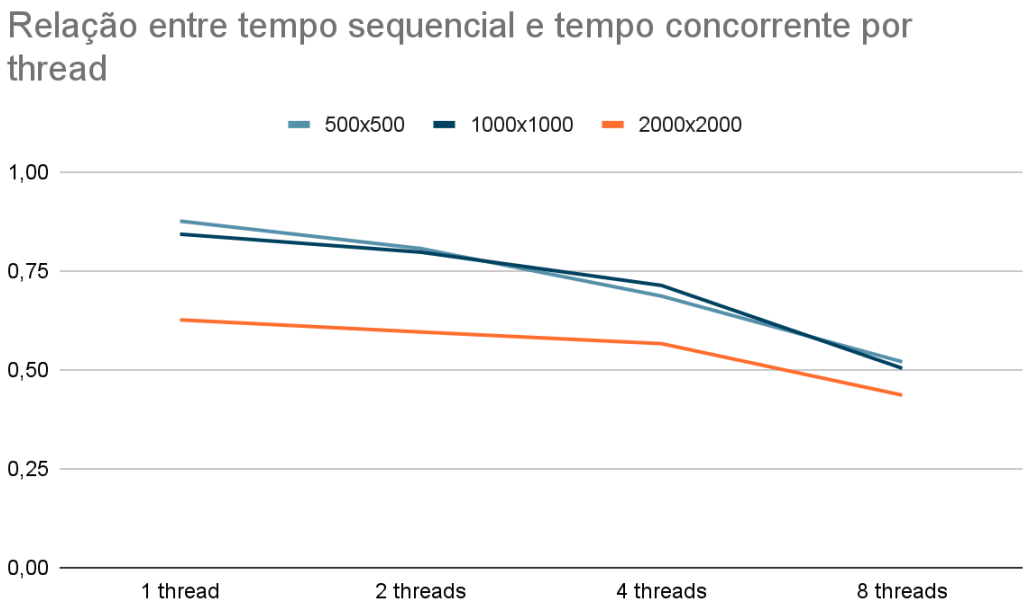


Considerações sobre o gráfico gerado

Percebemos uma diminuição de cerca de 50% para cada vez que dobramos o número de threads, embora não seja exato. Isso se correlaciona com o explicado nas considerações sobre a tabela.

Gráficos de eficiência e aceleração





Considerações sobre esses gráficos

No primeiro, temos a eficiência, calculada pela fórmula $E = \frac{A}{N}$, sendo A a aceleração (tempo sequencial / tempo concorrente) e N o número de processadores da máquina. Percebemos que existe uma diferença clara entre o esperado teoricamente e o efetivamente testado, afinal existem perdas de trocas de contexto para um número maior de threads.

Já no segundo gráfico, temos uma relação entre o tempo sequencial e o tempo concorrente para o número de thread. Era esperado que mantivessem um número perto de um, caso realmente existisse a relação “dobrar as threads, metade do tempo”.

- Caso sequencial levasse X segundos, duas threads deveriam demorar $\frac{X}{2}$ segundos, o que resultaria em $\frac{X}{2\frac{X}{2}} = 1.$

Porém, como existem perdas de performance, o tempo em concorrência não é exatamente metade, resultando em um valor menor que 1.

Configurações da máquina utilizada

- Rodando a compilação e execução em WSL (Debian) no Windows 11
- Processador: Intel Core i5-10400 2.90 GHz - 6 Cores - 12 Logical Processors
- Memória: 16GB DDR4 2333MHz
- Placa-Mãe: ASRock H510M-HVS
- Armazenamento: SSD Sata 2

Link para arquivo com anotações sobre cada execução

Anotações sobre Tempo de Execução de Programa Concorrente