UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

RELATÓRIO TÉCNICO

NATAL/RN
2021

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

ARTURO FONSECA DE SOUZA
DIEGO FILGUEIRAS BALDERRAMA

RELATÓRIO TÉCNICO

Trabalho referente à nota da Unidade I da disciplina DIM0124 - Programação Concorrente - T01, orientado pelo Prof. Dr. Everton Ranielly de Sousa Cavalcante.

NATAL/RN
2021

SUMÁRIO

1. INTRODUÇÃO

Este trabalho consiste basicamente em implementar o algoritmo de multiplicação de matrizes de diferentes ordens seguindo dois paradigmas distintos: sequencial e concorrente. Após implementados, os algoritmos devem passar por várias medições de tempo de execução para verificar seus respectivos desempenhos com as variadas cargas fornecidas. Por fim, os resultados devem ser analisados a fim de verificar se a hipótese de que programas concorrentes são sempre mais eficientes que programas sequenciais em relação ao tempo de execução.

Portanto, este relatório tem como objetivo descrever a metodologia, o processo de execução, apresentar e discutir os resultados e expor uma conclusão do trabalho descrito.

2. METODOLOGIA

Para a realização do trabalho proposto, um computador com as seguintes características técnicas foi utilizado para a etapa de testes de velocidade das execuções:

- Sistema Operacional: Linux Mint 20.2 (64bits)
- Kernel: 5.11.0-43-generic
- Processador: AMD Ryzen 3400G
- Memória RAM: 8GB

Em relação à linguagem de programação e compilador utilizados, o ambiente foi o seguinte:

- Linguagem de programação: C++11
- Versão do compilador G++: 9.3.0
- Versão do GNU Make: 4.2.1

Para executar cada versão da implementação do algoritmo de multiplicação de matrizes vinte vezes para cada dimensão das matrizes dadas, foram utilizados *shell scripts* compostos por dois laços de repetição, um deles varia a dimensão da matriz e o outro laço controla a quantidade de execuções para essa dimensão.

O contexto em que o computador se encontrou durante todos os testes efetuados foi o que trazia a maior disponibilidade possível dos seus recursos: nenhuma aplicação aberta além do terminal para executar os *scripts*.

Para gerar os resultados dos tempos de execução das duas soluções implementadas, foi utilizada a biblioteca *chrono* do C++, que permite marcar e armazenar um ponto específico no tempo durante a execução. No programa, são armazenados dois pontos no tempo: (i) imediatamente antes da chamada do método que multiplica as matrizes e (ii) imediatamente após a finalização do mesmo método. Com esses dados salvos, subtraímos (i) de (ii) e obtemos o tempo total de execução da multiplicação (em milissegundos). Feito isso, esse tempo é anexado em um arquivo do tipo *comma-separated values* (CSV) para análises comparativas posteriores.

A partir do arquivo gerado, foi possível calcular o tempo mínimo, médio e máximo de execução de cada uma das soluções para os diferentes tamanhos de matrizes, bem como os valores de desvio padrão.

As análises comparativas foram realizadas através do cálculo de ganho de desempenho (*speedup*), que consiste em dividir o tempo médio de execução da solução sequencial da multiplicação de matrizes pelo tempo médio de execução da solução concorrente. Caso o valor do *speedup* resulte um valor maior que 1, significa que a solução concorrente finalizou a multiplicação em menos tempo, portanto é mais eficiente que a sequencial nesse aspecto, de forma análoga a ideia também segue para o caso contrário.

3. RESULTADOS

Inicialmente, serão apresentadas as tabelas contendo os resultados obtidos através dos testes realizados para a multiplicação de matrizes de diferentes dimensões utilizando as diferentes soluções implementadas.

3.1. Matrizes 4x4

Resultados de tempo de execução mínimo, médio, máximo e desvio padrão para matrizes 4x4:

	Mínimo (ms)	***Médio (ms)***	***Máximo (ms)***	***Desvio Padrão (ms)***
Sequencial	0.000852	0.00111715	0.001783	0.0002666961333
Concorrente	0.094379	0.17757555	0.360984	0.06820620262

Tabela 1: Comparação para matrizes 4x4.

Resultado de *speedup* para matrizes 4x4:

$$speedup_{4x4} = \frac{0.00111715}{0.17757555} = 0.006291125$$

Equação 1: *Speedup* para matrizes 4x4.

3.2. Matrizes 8x8

Resultados de tempo de execução mínimo, médio, máximo e desvio padrão para matrizes 8x8:

	Mínimo (ms)	***Médio (ms)***	***Máximo (ms)***	***Desvio Padrão (ms)***
Sequencial	0.002054	0.0038158	0.006182	0.001264391814
Concorrente	0.187596	0.2942208	0.519054	0.08198454182

Tabela 2: Comparação para matrizes 8x8.

Resultado de *speedup* para matrizes 8x8:

$$speedup_{8x8} = \frac{0.0038158}{0.2942208} = 0.012969171$$

Equação 2: *Speedup* para matrizes 8x8.

3.3. Matrizes 16x16

Resultados de tempo de execução mínimo, médio, máximo e desvio padrão para matrizes 16x16:

	Mínimo (ms)	***Médio (ms)***	***Máximo (ms)***	***Desvio Padrão (ms)***

Sequencial	0.018455	0.0225307	0.03682	0.006272586278
Concorrente	0.330937	0.54255055	0.7859	0.1138252207

Tabela 3: Comparação para matrizes 16x16.

Resultado de *speedup* para matrizes 16x16:

$$speedup_{16x16} = \frac{0.0225307}{0.54255055} = 0.041527375$$

Equação 3: *Speedup* para matrizes 16x16.

3.4. Matrizes 32x32

Resultados de tempo de execução mínimo, médio, máximo e desvio padrão para matrizes 32x32:

	Mínimo (ms)	***Médio (ms)***	***Máximo (ms)***	***Desvio Padrão (ms)***
Sequencial	0.138496	0.15058725	0.271961	0.02870711451
Concorrente	0.620306	0.86595455	1.244469	0.1872375151

Tabela 4: Comparação para matrizes 32x32.

Resultado de *speedup* para matrizes 32x32:

$$speedup_{32x32} = \frac{0.15058725}{0.86595455} = 0.173897406$$

Equação 4: *Speedup* para matrizes 32x32.

3.5. Matrizes 64x64

Resultados de tempo de execução mínimo, médio, máximo e desvio padrão para matrizes 64x64:

	Mínimo (ms)	***Médio (ms)***	***Máximo (ms)***	***Desvio Padrão (ms)***
Sequencial	0.623014	1.28654015	2.121045	0.3394617954
Concorrente	1.305104	1.9535653	2.700299	0.3822910366

Tabela 5: Comparação para matrizes 64x64.

Resultado de *speedup* para matrizes 64x64:

$$speedup_{64x64} = \frac{1.28654015}{1.9535653} = 0.658560095$$

Equação 5: *Speedup* para matrizes 64x64.

3.6. Matrizes 128x128

Resultados de tempo de execução mínimo, médio, máximo e desvio padrão para matrizes 128x128:

	Mínimo (ms)	***Médio (ms)***	***Máximo (ms)***	***Desvio Padrão (ms)***
Sequencial	4.964332	8.19462375	11.507192	2.078375315
Concorrente	3.920503	4.5869222	5.777854	0.5235343119

Tabela 6: Comparação para matrizes 128x128.

Resultado de *speedup* para matrizes 128x128:

$$speedup_{128x128} = \frac{8.19462375}{4.5869222} = 1.786519019$$

Equação 6: *Speedup* para matrizes 128x128.

3.7. Matrizes 256x256

Resultados de tempo de execução mínimo, médio, máximo e desvio padrão para matrizes 256x256:

	Mínimo (ms)	***Médio (ms)***	***Máximo (ms)***	***Desvio Padrão (ms)***
Sequencial	40.317056	42.5544155	48.952886	2.619470762
Concorrente	13.48793	15.95441865	18.472507	1.394453401

Tabela 7: Comparação para matrizes 256x256.

Resultado de *speedup* para matrizes 256x256:

$$speedup_{256x256} = \frac{42.5544155}{15.95441865} = 2.667249521$$

Equação 7: *Speedup* para matrizes 256x256.

3.8. Matrizes 512x512

Resultados de tempo de execução mínimo, médio, máximo e desvio padrão para matrizes 512x512:

	Mínimo (ms)	***Médio (ms)***	***Máximo (ms)***	***Desvio Padrão (ms)***
Sequencial	322.411833	324.2278294	328.392725	1.860051402
Concorrente	74.138871	76.9007075	83.036209	2.298049251

Tabela 8: Comparação para matrizes 512x512.

Resultado de *speedup* para matrizes 512x512:

$$speedup_{512x512} = \frac{324.2278294}{76.9007075} = 4.21618786$$

Equação 8: *Speedup* para matrizes 512x512.

3.9. Matrizes 1024x1024

Resultados de tempo de execução mínimo, médio, máximo e desvio padrão para matrizes 1024x1024:

	Mínimo (ms)	***Médio (ms)***	***Máximo (ms)***	***Desvio Padrão (ms)***
Sequencial	3127.373512	3166.711849	3264.09058	35.38365283
Concorrente	632.911588	701.8831042	787.004665	62.94949258

Tabela 9: Comparação para matrizes 1024x1024.

Resultado de *speedup* para matrizes 1024x1024:

$$speedup_{1024x1024} = \frac{3166.711849}{701.8831042} = 4.511736826$$

Equação 9: *Speedup* para matrizes 1024x1024.

3.10. Matrizes 2048x2048

Resultados de tempo de execução mínimo, médio, máximo e desvio padrão para matrizes 2048x2048:

	Mínimo (ms)	***Médio (ms)***	***Máximo (ms)***	***Desvio Padrão (ms)***
Sequencial	43480.05104	43697.07735	44016.6317	130.1380954
Concorrente	7869.440799	8232.021223	8588.515501	188.5789647

Tabela 10: Comparação para matrizes 2048x2048.

Resultado de *speedup* para matrizes 2048x2048:

$$speedup_{2048x2048} = \frac{43697.07735}{8232.021223} = 5.308183272$$

Equação 10: *Speedup* para matrizes 2048x2048.

Em seguida, é possível observar o gráfico de linhas gerado a partir dos tempos médios de execução de cada solução implementada:

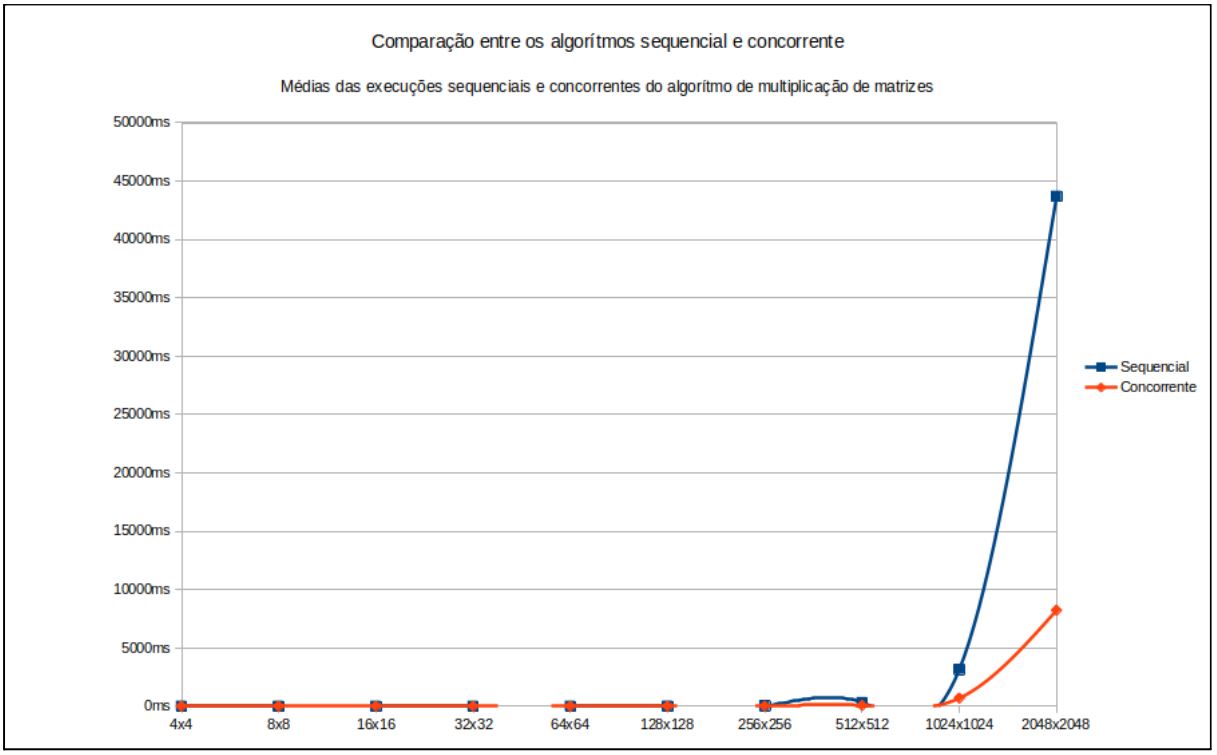

Gráfico 1: Comparação entre tempos de execução (escala linear).

Entretanto, a grande diferença entre os tempos médios de cada execução torna a comparação praticamente impossível se o gráfico estiver em escala linear, portanto, encontra-se abaixo o gráfico em escala logarítmica para melhor visualização dos resultados:

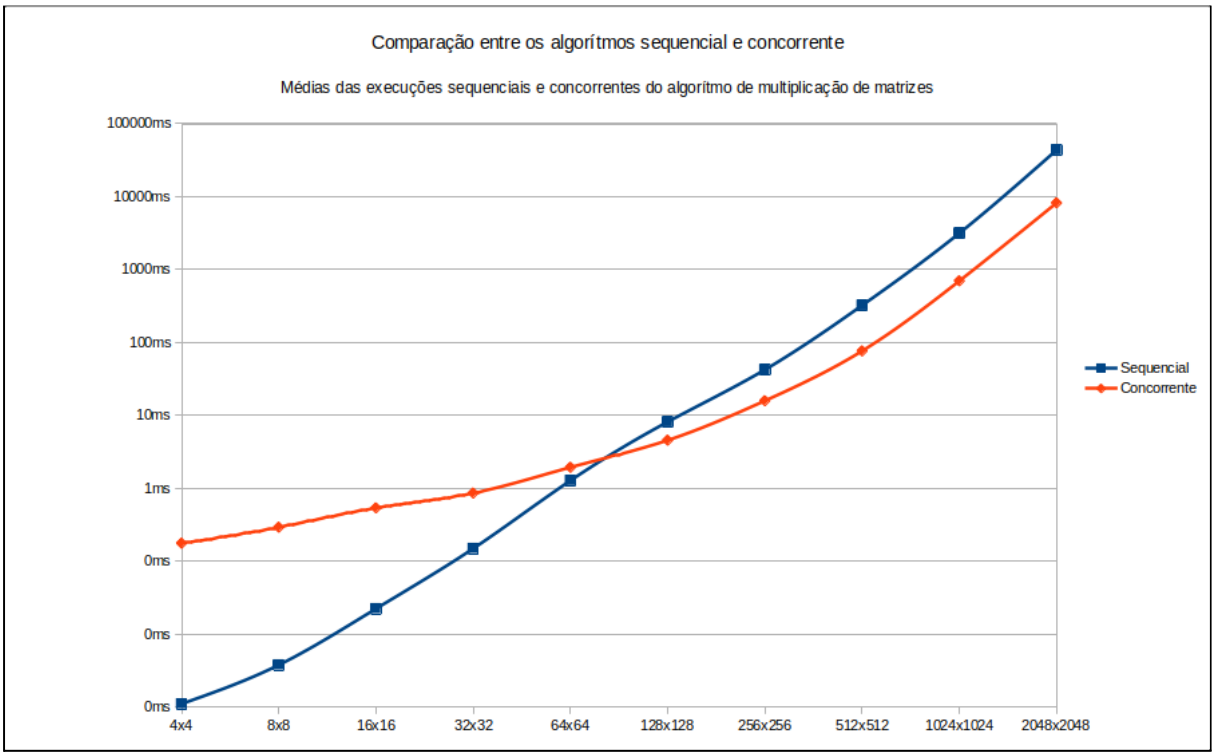

Gráfico 2: Comparação entre tempos de execução (escala logarítmica).

Foi gerado também um gráfico para facilitar a comparação entre os valores de *speedups*, exibido abaixo:

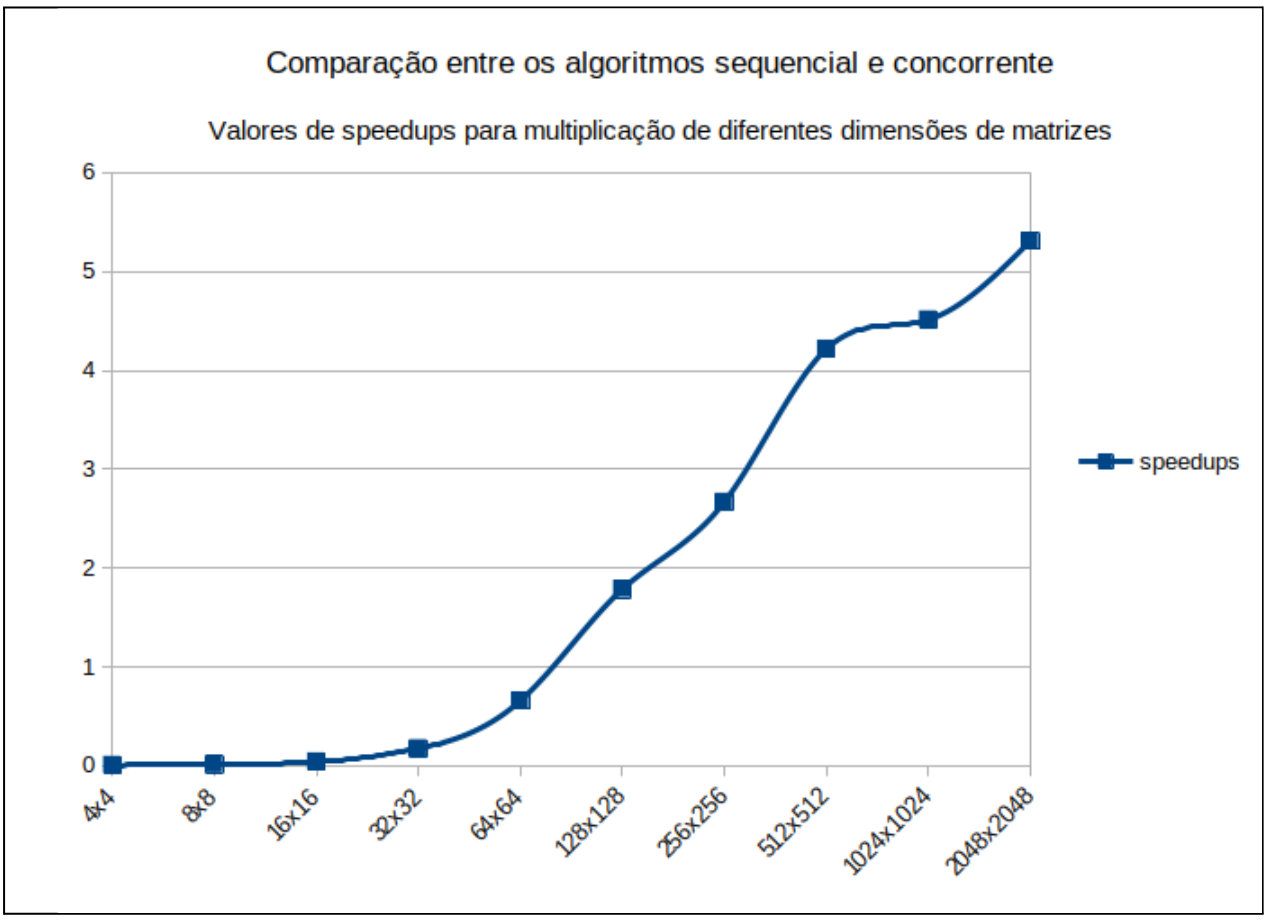

Gráfico 3: Comparação entre os valores de *speedups*.

4. CONCLUSÕES

Analisando primeiramente apenas os tempos médios de execução das multiplicações de matrizes com as diferentes soluções implementadas (disponíveis nas tabelas 1 a 10 e nos gráficos 1 e 2), é possível concluir que a operação com matrizes de ordem 4, 8, 16, 32 e 64 obtiveram um tempo de execução menor com a solução sequencial do que com a concorrente. Entretanto, o mesmo não ocorreu para as matrizes de ordem 128, 256, 512, 1024 e 2048, visto que a solução concorrente conseguiu efetuar as multiplicações com mais rapidez.

Dessa forma, para avaliar de fato quão melhor ou pior foi cada uma das soluções nos diferentes casos, analisemos mais discriminadamente os valores de ganho de desempenho obtidos (disponíveis nas equações 1 a 10 e no gráfico 3):

- Para as matrizes 4x4, o *speedup* foi de 0.006291125, isso indica que a versão concorrente teve um péssimo desempenho, realizando a multiplicação aproximadamente 159 vezes mais devagar do que a sequencial, entretanto isso passa despercebido para nós porque que as execuções não duraram mais que 0.4ms;
- Em seguida, para as matrizes 8x8, o *speedup* foi de 0.012969171, ou seja, a solução concorrente ainda teve um péssimo desempenho, pois efetuou a operação cerca de 77 vezes mais devagar;
- Já para as matrizes 16x16, apesar de ter apresentado um *speedup* maior, de 0.041527375, o desempenho continuou ruim, visto que a multiplicação levou em torno de 24 vezes mais tempo para ser concluída quando comparada à sequencial;
- Para as matrizes 32x32, o *speedup* foi de 0.173897406, com um desempenho da versão concorrente ainda ruim, executando a operação aproximadamente 5 vezes mais lentamente;
- Seguidamente, para as matrizes 64x64, o *speedup* foi de 0.173897406, ou seja, o desempenho da solução concorrente foi cerca de 5 vezes menor quando comparado ao da sequencial, sendo ainda considerado baixo;
- Já para as matrizes 128x128, o desempenho da implementação concorrente da multiplicação conseguiu superar o da sequencial, executando quase 2 vezes mais rápido, com o *speedup* de 1.786519019;

- Para as matrizes 256x256, o *speedup* foi maior ainda, com um valor de 2.667249521, ou seja, o desempenho da versão concorrente foi mais de duas vezes melhor que o da sequencial;
- Em seguida, para as matrizes 512x512, o desempenho continuou crescendo e a solução concorrente realizou a multiplicação cerca de 4 vezes mais rápido, com o *speedup* atingindo 4.21618786;
- Já para as matrizes 1024x1024, o *speedup* foi de 4.511736826, isto é, comparativamente à implementação sequencial, o desempenho da concorrente foi mais que 4 vezes melhor;
- E enfim, para as matrizes 2048x2048, o *speedup* foi de 5.308183272, o maior de todos, isso significa que a versão concorrente foi capaz de realizar as multiplicações em um tempo 5 vezes menor do que a versão sequencial.

Por fim, a conclusão final a qual chegamos após o cumprimento deste trabalho prático em relação ao desempenho de algoritmos concorrentes com *threads* é que não é válido o senso comum de que todo programa dessa natureza seja sempre mais eficiente do que a solução sequencial em relação ao tempo de execução. Isso ocorre pois o uso de *threads* demanda certas etapas a mais durante a sua execução, como trocas de contextos. Nessas trocas, novas variáveis são criadas, o que acaba gastando um tempo a mais de processamento. Esse tempo chega a ser mais relevante em matrizes menores, onde a proporção entre trocas de contexto e número de cálculos efetivados é menor, tornando a versão sequencial mais vantajosa nesses casos.