UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
DIM0124 – PROGRAMAÇÃO CONCORRENTE

Gabriel Estevam Narciso
Júlia Ferreira de Souza Glória

Trabalho Prático: Programação com Threads

1. Introdução

A programação concorrente, assim como a programação sequencial, possui suas vantagens e desvantagens, e para determinar qual delas é mais adequada ao seu sistema é importante avaliar diversos aspectos. Um dos pontos que devemos levar em consideração quando comparamos essas duas formas de programação é o desempenho com relação ao tempo de execução. Espera-se que ocorra uma melhora de performance ao utilizar um programa concorrente, mas isso varia de acordo com muitos fatores e dependendo do problema que seu programa busca solucionar, esse ganho de desempenho vai ser mais ou menos significativo.

A multiplicação de matrizes é um problema que têm diversas aplicações não só na área computacional mas em diferentes áreas da ciência. Isso porque, as matrizes são capazes de representar diferentes fenômenos e objetos, e assim muitos problemas teóricos ou práticos podem ser moldados por meio de operações entre matrizes. A operação de multiplicação em si ainda é bastante complexa do ponto de vista computacional e por isso um dos grandes desafios da Ciência da Computação é a criação de algoritmos mais eficientes para esse problema, que executem com um menor tempo de execução e não consumam tanto recurso computacional.

Diante desse contexto, esse projeto tem o propósito geral de realizar uma comparação entre soluções concorrentes e soluções sequenciais para o problema de multiplicação de matrizes. Para realizar essa análise, iremos apresentar e avaliar tempos de execução de cada solução em diferentes cenários, para finalmente validar a hipótese de que os programas concorrentes oferecem uma performance maior em relação as suas versões sequenciais.

2. Metodologia

2.1. Caracterização Técnica

Para a realização deste projeto foi utilizado um computador com processador Intel® Core™ i7-7500U CPU @ 2.70GHz×4, com 8 GB de memória RAM e com o Sistema Operacional Ubuntu 18.04.2. Optou-se pela implementação do programa com a linguagem Java, e com o compilador JDK na versão 11.0.7. O processo de desenvolvimento, teste e depuração do código, ocorreu por meio da IDE Eclipse e o versionamento foi feito através da plataforma do GitHub.

2.2. Obtenção dos Resultados

Como conjunto de dados/amostra, foram utilizadas matrizes quadradas com dimensões de 4x4 até 2048x2048 todas como potências de base 2. Para cada dimensão existiam duas matrizes, A e B, e a multiplicação A X B deveria ser realizada. Essa multiplicação foi realizada 20 vezes de forma sequencial e 20 vezes de forma concorrente para cada uma das dimensões e o tempo para realizar cada uma das operações foi computado e armazenado.

O tempo de execução das soluções foi calculado considerando unicamente o intervalo do início da multiplicação das matrizes até a finalização da operação, ou seja, o tempo gasto com a leitura das amostras e escrita do resultado não foi contabilizado. Isso porque, as entradas e saídas do programa estão sendo processadas da mesma forma, de maneira sequencial. Além disso, os períodos de tempo computados foram armazenados e exportados no formato de arquivo .csv pelo próprio programa permitindo a criação das tabelas e gráficos que serão apresentados e analisados posteriormente.

2.3. Análise dos Resultados

A diferença no desempenho do programa de forma sequencial e de forma concorrente, foi avaliada levando em consideração o tempo mínimo de execução, o tempo máximo de execução, e o desvio padrão desses valores. E o conjunto de dados obtidos foi examinado de acordo com a quantidade de elementos da matriz, ou seja, procurou-se entender como a dimensão da matriz afeta o tempo de execução do programa sequencial e como a mesma afeta o tempo de execução do programa concorrente.

Para uma comparação mais direta entre as duas soluções, foi determinado um ganho de desempenho (speed-up) que pode ser calculado como a razão entre o tempo de execução do programa concorrente e o tempo de execução do programa sequencial. Quando a razão entre esses dois valores é maior que 1, a performance do programa concorrente foi de fato superior ao programa sequencial.

3. Implementação

3.1. Entrada e Saída

A entrada deve receber exatamente dois argumentos, o primeiro argumento é a dimensão da matriz e o segundo argumento é a forma que deverá ser feita a multiplicação das matrizes. O programa verifica se os dois argumentos estão sendo passados de maneira correta e quando um deles for inválido uma exceção será lançada informando o problema.

Foi criada uma classe apenas para a manipulação dos arquivos, nela os valores passados como argumento são processados e os arquivos de texto contendo as matrizes com a dimensão informada são devidamente lidos e guardados em uma variável global. Da mesma forma, ao finalizar a multiplicação das matrizes, seja pelo método sequencial seja pelo método concorrente, os valores da matriz resultante são escritos em um novo arquivo de texto seguindo os mesmos padrões.

3.2. Método Sequencial

A solução sequencial foi implementada em uma classe na qual o construtor deverá receber como parâmetro as matrizes a serem multiplicadas e as suas dimensões que serão guardadas com um atributo estático da classe. Há também um atributo para armazenar a matriz resultante da operação e é nessa matriz que o método de multiplicar as matrizes dessa classe fará suas alterações.

3.3. Método Concorrente

A solução concorrente foi implementada em uma classe separada, assim, como no método sequencial e essa classe deve também ser instanciada por meio do construtor. Essa classe estende a classe Thread e os recursos que deverão ser compartilhados entre as Threads são definidos como atributos estáticos. O método de multiplicar as matrizes dessa classe vai criar várias threads dessa mesma classe passando um índice no seu construtor. Cada thread criada pelo método de multiplicação irá ,no seu método run(), calcular as multiplicações de acordo com este índice e somar esse valor aos elementos correspondentes na matriz resultante.

3.4. Execução

O programa foi desenvolvido com duas classes principais MainThread e MainExperimento. Para executar o programa padrão é preciso executar a MainThread, já para gerar os resultados ou seja, realizar o método de multiplicação 20 vezes para cada uma das dimensões é preciso executar o MainExperimento como main.

4. Resultados

Com a obtenção dos tempos de execução de cada metodologia utilizada, que são a sequencial e a concorrente, foram criadas tabelas para representar esses tempos. No qual, essas tabelas estão representando as dimensões das matrizes utilizadas, o valor do menor e maior tempo de execução medido, a média desses tempos, o seu desvio padrão, pois com o desvio padrão é possível verificar se os tempos estão espalhados ou não em relação a média, e na tabela com a metodologia sequencial possui a coluna speed up, que demonstra o desempenho da metodologia sequencial em relação a concorrente.

4.1. Tabela da metodologia de solução concorrente

Com a tabela de resultado dos tempos de execução da metodologia concorrente abaixo é percebido que possui uma certa variação de tempo, pois os seus valores de tempo minimo e maximo possui uma desigualdade na maioria das vezes e seu desvio padrão possui uma certa elevação. Isso acontece devido a forma que é processada cada threads em cada uma das execuções do programa. E, isto deve-se ao modo de como cada execução realiza a organização e processamento das threads, sendo assim, algumas vezes ficar mais lentas ou rápidas em relação às demais.

Dimensão da matriz	Valor do tempo mínimo (ms)	Valor do tempo máximo (ms)	Média dos tempos (ms)	Desvio padrão
4x4	0	3	0.7	~0.80
8x8	0	5	~1.10	~1
16x16	0	6	~1.80	~1.50
32x32	1	19	3.5	~3.80
64x64	3	13	5	~2.10
128x128	10	17	~12	~1.90
256x256	31	39	~34.10	~2
512x512	120	224	159	~33.30
1024x1024	808	1212	~918.10	~97.60
2048x2048	7586	8919	8121.95	~458.53

Tabela 1 - Metodologia de solução concorrente.

4.2. Tabela da metodologia de solução sequencial

Com a tabela de resultado dos tempos de execução da metodologia sequencial abaixo é observado que possui uma constância de tempo até a chegada das maiores matrizes testadas que seriam as de dimensões 512, 1024 e 2048, nos quais elas possuem um tempo de execução muito alta e variação de tempo, como é verificado pelos valores de tempo mínimo e máximo e o seu desvio padrão.

Nesta tabela é mostrada também a questão de desempenho da metodologia sequencial em relação a concorrente, o speed up, no qual é percebido que a solução sequencial leva a vantagem nas menores matrizes com uma grande diferença e acaba perdendo essa vantagem nas matrizes maiores, que a concorrente leva a vantagem.

Dimensão da matriz	Valor do tempo mínimo (ms)	Valor do tempo máximo (ms)	Média dos tempos (ms)	Desvio padrão	Speed up (Tempo médio sequencial / Tempo médio concorrente)
4x4	0	0	0	0	0
8x8	0	0	0	0	0
16x16	0	1	0.05	~0.20	~0.28
32x32	0	1	0.10	0.30	~0.02
64x64	0	1	0.35	~0.48	0.07
128x128	2	4	2.55	~0.59	~0.21
256x256	20	25	21.85	~1.39	~0.64
512x512	178	262	200.70	~26.08	~1.26
1024x1024	1736	2176	1815.35	~114.79	~1.98
2048x2048	130185	153002	143018.3	~7421.64	~17.61

Tabela 2 - Metodologia de solução sequencial.

4.3. Gráficos comparativos dos tempos de execuções

Para se possuir uma melhor representação visual em relação aos dados das tabelas apresentadas acima e realizar uma comparação entre as metodologias foram criados gráficos, nos quais mostram as médias do tempo de execução de cada metodologia adotada e de cada dimensão e sua evolução.

O gráfico abaixo mostra como a programação concorrente, em relação ao tempo de execução e quantidade de dados a serem processados, pode ser tão destoante em relação a programação sequencial, deixando está diferença imensa, dependendo da máquina a ser usada.

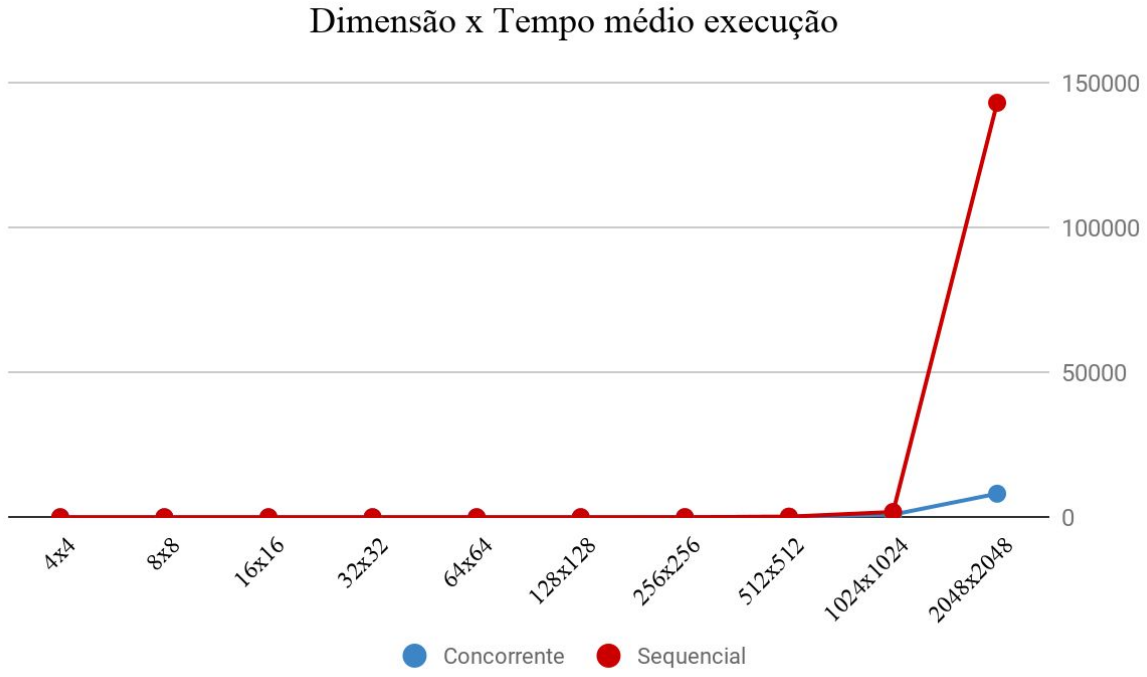

Gráfico 1 - Tempo de execução por dimensão.

Como o gráfico anterior tomou uma proporção muito grande acabou ocultando os dados anteriores a 2048, então foi-se gerado um outro gráfico de barra, no qual foi retirado os dados de 2048 e deixado o restante. Nesse novo gráfico abaixo foi percebido conforme foi notado anteriormente, que a metodologia concorrente possui vantagens apenas quando a uma grande quantia de dados a serem processadas. E, foi notado que as diferenças de tempo entre a sequencial e concorrente com as matrizes menores não são tão distintos quanto às diferenças da concorrente e sequencial com as matrizes maiores.

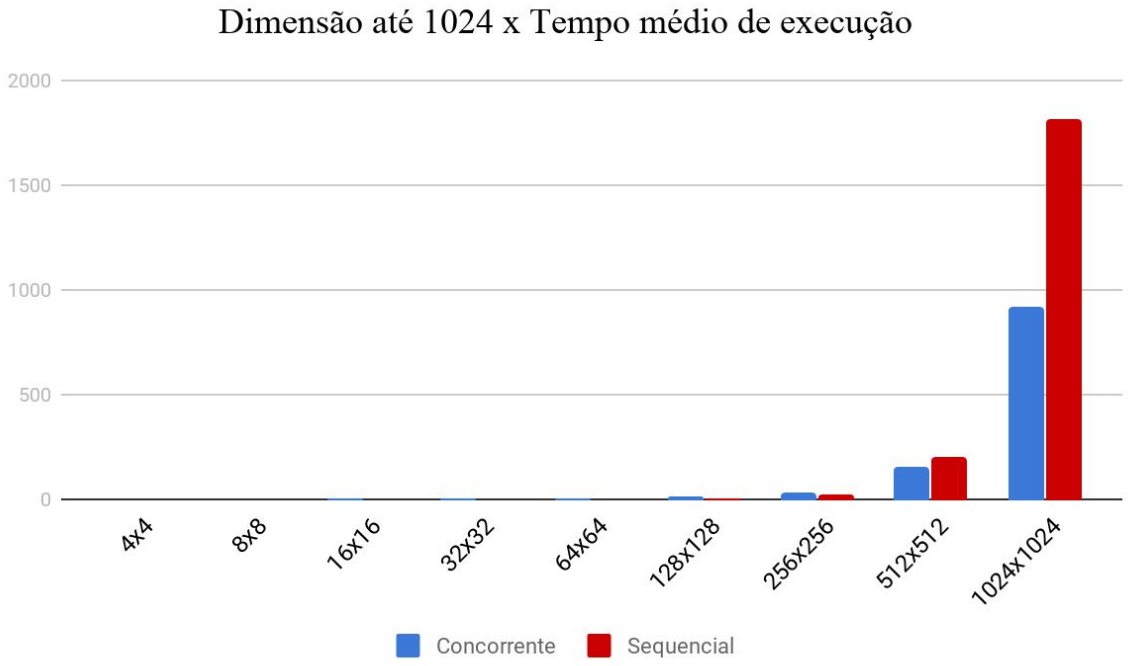

Gráfico 2 - Tempo de execução por dimensão sem 2048.

4.4. Gráfico comparativo de desempenho

Com o gráfico abaixo é notado a evolução do desempenho (speed up) da programação sequencial em relação a concorrente. Então é notado que ao decorrer do aumento das dimensões das matrizes o desempenho da sequencial vai caindo em relação a concorrente, pois o valor de speed up vai aumentando gradativamente.

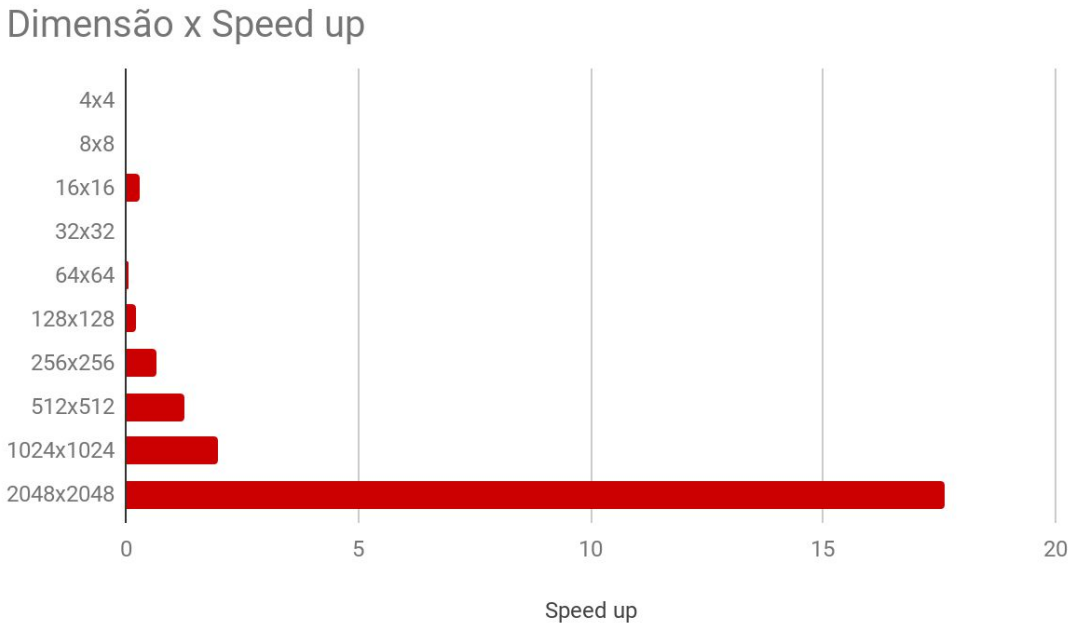

Gráfico 3 - Speedup por dimensão.

5. Conclusões

Tendo em vista os resultados obtidos, pode-se concluir que a programação concorrente possui um desempenho semelhante ou muitas das vezes inferior ao desempenho da programação sequencial, quando não se possui uma demanda grande de processos. Devido a programação sequencial não precisar realizar um gerenciamento das threads e já possuir uma ordem certa dos comandos, então ela possui uma constância e rapidez em relação a concorrência.

Porém, quando parte-se para situações que demandam mais desempenho para os processos a concorrência toma-se a frente e acaba obtendo maior vantagem que a sequencial. Devido ela conseguir dividir seus processos em múltiplas threads e com isso processá-los mais rápidos, pois as threads vão sendo processadas de forma concorrente, não tendo uma ordem, enquanto que o método sequencial está tentando executar seus comandos em um único processo e um comando após o outro. E com isso o desempenho da concorrência é melhorada significativamente.