Análise de tempo de execução ao performar multiplicação de matrizes variando a quantidade de threads utilizadas

Sobre o software

O programa desenvolvido possui 5 arquivos:

- Matrix.java
- MatrixTest.java
- MatrixMultiThread/MatrixThreadTest.java
- MatrixMultiThread/ParallelThreadsCreator.java
- MatrixMultiThread/RowMultiplyWorker.java

O primeiro arquivo, Matrix.java, se refere a um java bean que define uma classe Matrix, possuindo atributos e funções de uma matriz. Essa classe é utilizada para a execução sequencial, que é feita pelo arquivo MatrixTest.java através da função *main*.

Dentro da pasta MatrixMultiThread, há os arquivos necessários à execução com threads. O arquivo MatrixThreadTest.java, dentro dessa pasta, é o responsável pela execução, possuindo o método *main*, o qual chama a função *multiply* do arquivo ParallelThreadsCreator.java. Esta função é a responsável por instanciar as threads e dividir as linhas da matriz entre elas para que o cálculo seja realizado em paralelo. Cada thread recebe uma instância da classe RowMultiplyWorker, que é onde ocorre a multiplicação de uma linha da matriz, desse modo, cada thread instanciada irá calcular a multiplicação de apenas uma linha da matriz por vez.

Testes

Para a análise de performance do programa desenvolvido, utilizou-se um computador possuindo um processador AMD Ryzen 2700 com 8 núcleos físicos e 16 threads. Desse modo, espera-se que o tempo de execução decaia conforme mais threads forem utilizadas, até um limite de 16 threads. Para averiguar tal suposição, o programa foi executado 21 vezes: uma execução sem usar threads (representada pelo número de threads 0 na tabela e no gráfico, onde foi utilizada a matriz sequencial) e demais execuções utilizando de 1 a 20 threads. Para uma análise mais detalhada, esse cenário de testes foi avaliado com uma matriz 2000x2000 e com uma matriz 4000x4000, totalizando 42 execuções. Com valores maiores para o tamanho da matriz, o software leva um tempo consideravelmente grande para a execução de todos os testes, mas é uma possibilidade caso haja a disponibilidade do tempo. Os resultados dos testes descritos estão detalhados nas tabelas e gráficos a seguir:

Tabela 1 – Resultados da execução com matriz 2000x2000.

Número de threads	Tempo (segundos)
0	35
1	52
2	26
3	18
4	14
5	12
6	11
7	12
8	11
9	10
10	10
11	9
12	8
13	7
14	7
15	7
16	7
17	7
18	7
19	8
20	7

Gráfico 1 – Resultados da execução com matriz 2000x2000.

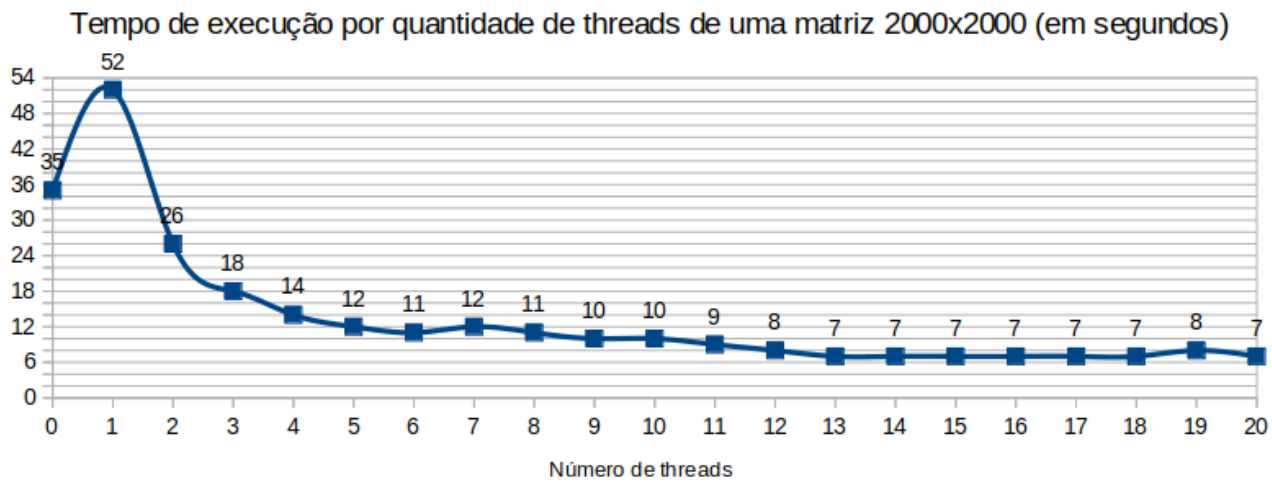
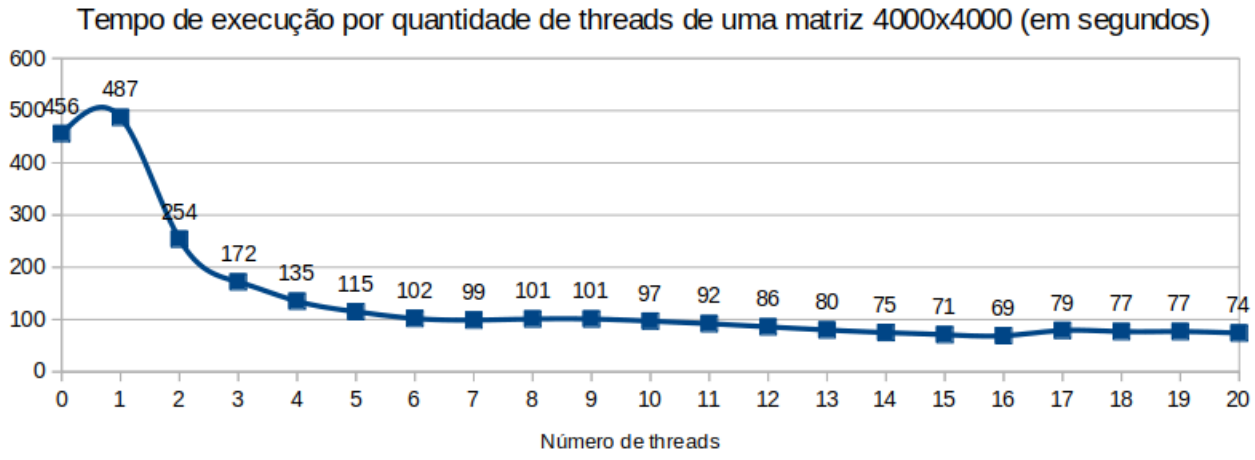


Tabela 2 – Resultados da execução com matriz 4000x4000.

Número de threads	Tempo (segundos)
0	456
1	487
2	254
3	172
4	135
5	115
6	102
7	99
8	101
9	101
10	97
11	92
12	86
13	80
14	75
15	71
16	69
17	79
18	77
19	77
20	74

Gráfico 2 – Resultados da execução com matriz 4000x4000.



Conclusão

Tendo em vista os resultados apresentados nas tabelas e gráficos, pode-se concluir que a execução do programa utilizando múltiplas threads é significantemente mais veloz que a execução sequencial. É interessante notar que os gráficos seguem um padrão: inicia-se com a execução sequencial (número de threads igual a 0) e obtêm-se um valor relativamente alto, em seguida, com 1 thread, o tempo de execução acaba sendo maior que o sequencial, entretanto, ao utilizar 2 ou mais threads o tempo de execução decai, passando a ter um comportamento similar ao logarítmico, até um limite mínimo alcançado quando o número de threads é 16, que é a quantidade disponível na máquina em que os testes foram realizados.