UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Trabalho Prático: Programação com Threads

Giovanne da Silva Santos
Noé Fernandes Carvalho Pessoa

5 de julho de 2020

1 Introdução

Neste relatório, faremos a análise dos experimentos que foram feitos na implementação de multiplicação entre matrizes utilizando um método sequencial (estratégia comum) e um método concorrente usando threads.

Na seção metodologia, dizemos o ambiente de testes, linguagem utilizada, o procedimento para gerar os resultados e como comparamos os dois métodos. Já em resultados, analisamos os resultados comparando a forma sequencial com a concorrente em relação ao tempo de execução. Por fim, na seção conclusão iremos concluir a análise dos resultados, comentando o que foi possível determinar a partir deles.

2 Metodologia

A linguagem de programação utilizada foi o Python em sua versão 3.7.3. Vale ressaltar que esta linguagem utiliza apenas um core do processador do computador. As bibliotecas usadas foram: a *threading*, para o uso das threads e a *numpy* para o uso de facilidades estatísticas.

No desenvolvimento do algoritmo, decidimos respeitar o modelo de pseudocódigo proposto no documento da tarefa tanto na forma sequencial quanto na forma concorrente, evitando inclusive usar estruturas e funções do numpy na função *multiplicar* (responsável por mutiplicar as matrizes), pois ouvimos que algumas ferramentas do numpy poderiam fazer uso do paralelismo quando o python em si não o faz. Ou seja, estaríamos usando funções que poderiam ou não estar executando em paralelo sem ter conhecimento disso (já que não temos acesso ao código da função da biblioteca) e isso poderia prejudicar a análise do desempenho real da linguagem.

Tendo isso em vista, também é importante informar que a configuração do computador em que foram feitos os experimentos é: 4 gb de memória RAM, Processador Intel(R) Pentium(R) CPU G4400 @ 3.30GHz, 3300 Mhz, com 2 núcleos e 2 processadores lógicos.

O método de comparação foi focado na análise das tabelas e gráfico de tempo de execução por método (sequencial ou threads) e na análise do *speed-up*. Sendo seu cálculo feito da seguinte forma:

$$speedup = \frac{T_{sequencial}}{T_{concorrente}}$$

Sendo $T_{sequencial}$ o tempo de execução do método sequencial e $T_{concorrente}$ o tempo de execução do método concorrente. Caso o *speedup* seja superior a 1, então de fato o método concorrente é melhor em temos de desempenho do que o sequencial.

Dessa forma, para gerar os 20 testes de cada matriz por método, criamos uma lista *tempo* em que foram inseridos os tempos de execução de cada teste, usando o pacote time. No final, usamos a lista *tempo* para calcular todas as outras informações necessárias para a análise tais como tempo mínimo, tempo médio, tempo máximo, desvio padrão, etc.

3 Resultados

As tabelas a seguir trazem os valores em tempo de execução obtidos para cada matriz quadrada analisada em um total de 20 testes. Foram realizados 20 testes para a versão sequencial e mais 20 para a versão com threads. Os dados coletados foram os seguintes:

Matriz 4x4	Sequencial	Threads
Tempo Mínimo (s)	0.00001073	0.00016642
Tempo Médio (s)	0.00001115	0.00019702
Tempo Máximo (s)	0.00001454	0.00065351

Figura 1: Tabela da matriz 4x4

Matriz 8x8	Sequencial	Threads
Tempo Mínimo (s)	0.00006604	0.00037694
Tempo Médio (s)	0.00006748	0.00041264
Tempo Máximo (s)	0.00007915	0.00095749

Figura 2: Tabela da matriz 8x8

Matriz 16x16	Sequencial	Threads
Tempo Mínimo (s)	0.00045466	0.00105476
Tempo Médio (s)	0.00046185	0.00108699
Tempo Máximo (s)	0.00056577	0.00144482

Figura 3: Tabela da matriz 16x16

Matriz 32x32	Sequencial	Threads
Tempo Mínimo (s)	0.00335479	0.00437474
Tempo Médio (s)	0.00336152	0.00454404
Tempo Máximo (s)	0.00339818	0.00556159

Figura 4: Tabela da matriz 32x32

Matriz 64x64	Sequencial	Threads
Tempo Mínimo (s)	0.02697372	0.02617502
Tempo Médio (s)	0.02701625	0.02775373
Tempo Máximo (s)	0.02712345	0.03066611

Figura 5: Tabela da matriz 64x64

Matriz 128x128	Sequencial	Threads
Tempo Mínimo (s)	0.20298147	0.2008357
Tempo Médio (s)	0.20485501	0.20371631
Tempo Máximo (s)	0.20554328	0.20595646

Figura 6: Tabela da matriz 128x128

Matriz 256x256	Sequencial	Threads
Tempo Mínimo (s)	1.59878254	1.49788713
Tempo Médio (s)	1.61471089	1.51745002
Tempo Máximo (s)	1.61695147	1.60785341

Figura 7: Tabela da matriz 256x256

Matriz 512x512	Sequencial	Threads
Tempo Mínimo (s)	13.65529847	12.68542266
Tempo Médio (s)	13.8528955	12.89457064
Tempo Máximo (s)	14.37008238	13.37885165

Figura 8: Tabela da matriz 512x512

Matriz 1024x1024	Sequencial	Threads
Tempo Mínimo (s)	115.95298171	106.99426126
Tempo Médio (s)	118.17975917	114.49046715
Tempo Máximo (s)	122.91936707	148.83837128

Figura 9: Tabela da matriz 1024x1024

Matriz 2048x2048	Sequencial	Threads
Tempo Mínimo (s)	991.96751285	928.58719277
Tempo Médio (s)	1005.1074438	953.85583936
Tempo Máximo (s)	1060.42918944	963.36278105

Figura 10: Tabela da matriz 2048x2048

Nas tabelas, temos uma melhora de desempenho com o uso das *threads*, principalmente em instâncias maiores, mesmo que não seja tão significativa com o uso de um núcleo apenas.

Matriz	Speed-up
4x4	0.0566
8x8	0.1635
16x16	0.4249
32x32	0.7398
64x64	0.9734
128x128	1.0056
256x256	1.0641
512x512	1.0743
1024x1024	1.0322
2048x2048	1.0537

Figura 11: Cálculo do speed-up das matrizes

Na tabela de *speed-up*, é visto que houve *speed-up* nas instâncias 128x128, 256x256, 1024x1024 e 2048x2048, ou seja, nas cinco maiores instâncias, embora que quase houve na instância 64x64. Talvez se fosse utilizado mais de um core, tivesse ocorrido *speed-up* em mais instâncias. Vale repetir que, quando o valor dá 1 ou mais, significa que houve *speed-up* no uso do método concorrente em relação ao sequencial.

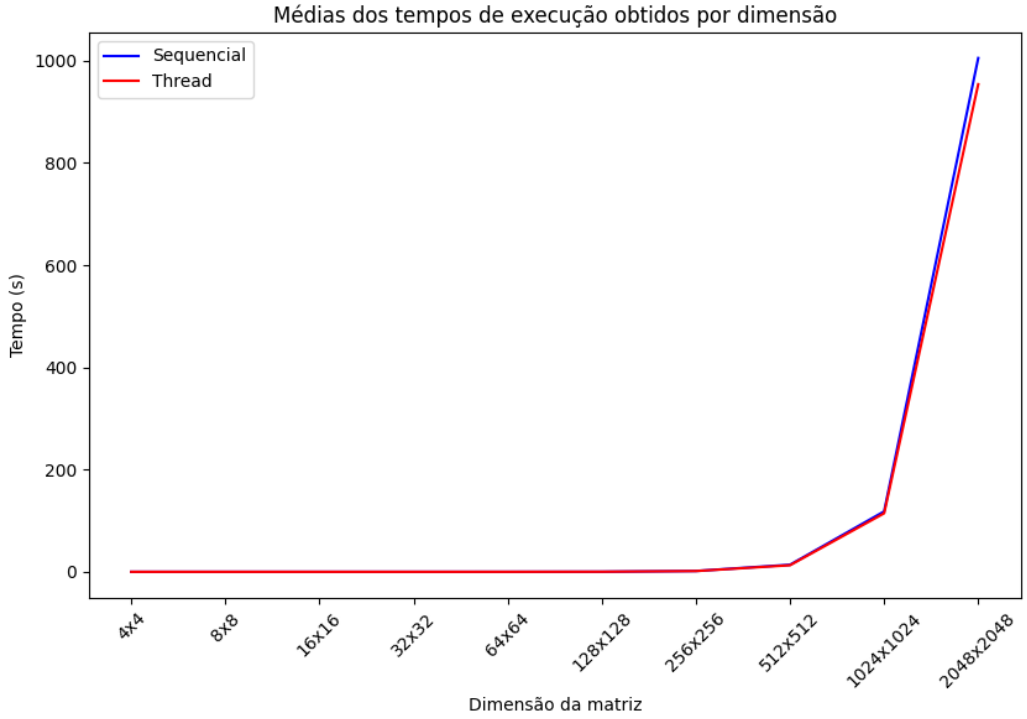

Figura 12: Gráfico de linhas comparando os métodos de execução

No gráfico da média do desempenho dos dois métodos é evidenciado que houve uma diferença de performance nas maiores instâncias, visto que as linhas de sequencia e thread (concorrete) começam a se separar a partir da instância de matriz de tamanho 128x128 com a linha do método sequencial mais acima da linha do método concorrente.

4 Conclusão

Como, para threads, python pode apenas fazer uso de um núcleo do processador, os resultados se refletem na pouca diferença entre os tempos de execução apresentados nas tabelas 1-11. É interessante notar que o valor de speed-up das matrizes (figura 11) só ultrapassou 1.0 a partir da matriz de dimensão 128. Ou seja, em todas as matrizes de dimensão menor o uso de threads torna a solução menos eficiente do que a sequencial.

Já no gráfico que apresenta os tempos médios de execução para cada dimensão de matriz (figura 12), podemos notar visualmente o que os valores das tabelas apresentam: em termos de performance, o ganho foi muito pequeno. As duas linhas do gráfico só se descolam visualmente a partir da matriz de dimensão 1024.

Concluímos, então, que o uso de threads no contexto analisado, utilizando python, não representa ganho considerável em tempo de execução do programa. Provavelmente, utilizar processos com o módulo multiprocessing representaria melhores resultados. No final, pode-se dizer que python não é a melhor escolha se há interesse em explorar o paralelismo da máquina.