Relatório de Avaliação
Aluno: Henrique Lima Cardoso
Professora: Silvana Rossetto
Disciplina: Programação Concorrente (ICP 361)
Laboratório: 3
Configuração:

AMD Ryzen 7 5700, 3700 Mhz, 8 cores, 16 Processadores Lógicos
16 GB RAM
SO: Linux Debian 12

Método:

Como pedido na atividade, escrevi os dois programas. Após testar, percebi que a diferença de tempo entre o programa sequencial e o programa concorrente rodando com uma thread era negligente. Para simplificar as coisas então, tomei a liberdade de considerar Ts(n) como Tp(n,1) nas contas.

O programa concorrente imprime na tela os seguintes valores (sem especificá-los):

```
(Num de Threads), (Tamanho da matriz), (Tempo Total), (Tempo de Inicialização), (Tempo de Processamento), (Tempo de finalização)
```

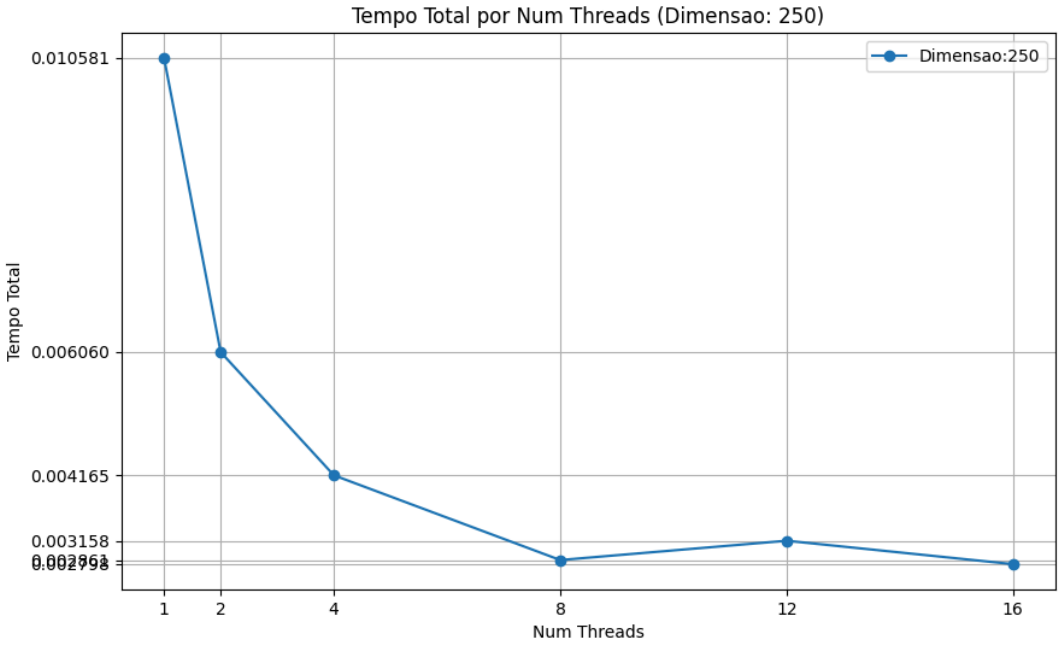
Então, por exemplo se chamamos:
`./mult_mat_conc testes/mat1000-1.bin testes/mat1000-2.bin testes/mult1000.bin 10`
Temos como saída:
`10, 1000, 0.105268, 0.004896, 0.096497, 0.003875`

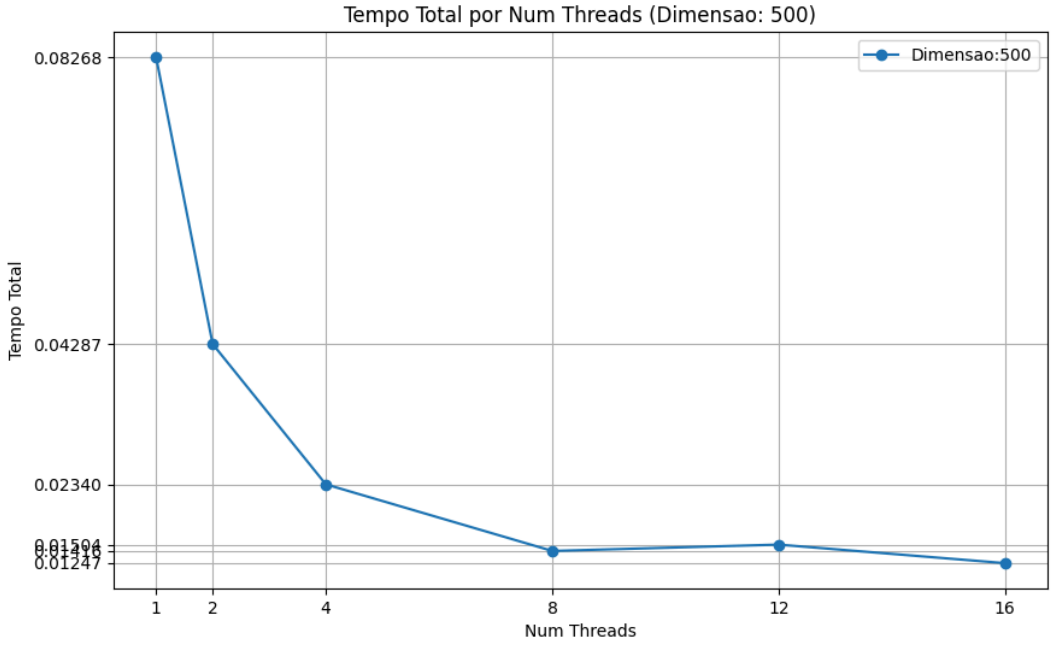
Isso é útil porque meu objetivo era criar um .CSV com os valores que queríamos. Essa é a função de script.sh, roda diversas vezes os programas com os inputs esperados e gera um CSV com as colunas:
`NumThreads, Dimensao, TempoTotal, TempoInit, TempoMult, TempoFim`

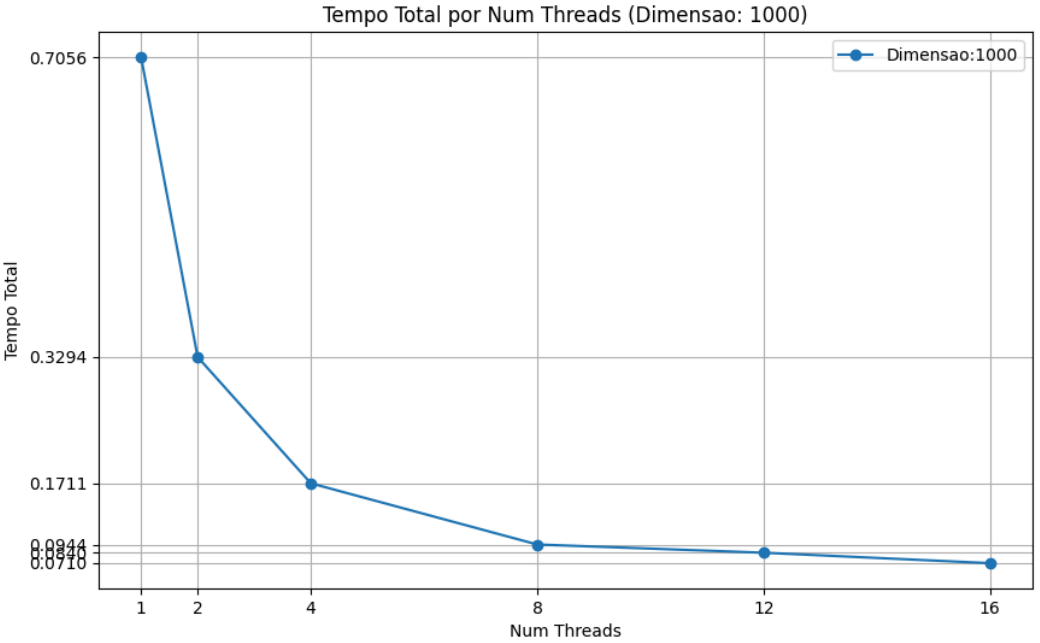
Após gerar o CSV, o programa em python graf.py, lê o arquivo e gera os gráficos que gostaríamos de obter. (Para não ficar muito ruim de ler os valores, nos gráficos de aceleração e eficiência, eu não mostro todos os valores tomados - somente metade deles).

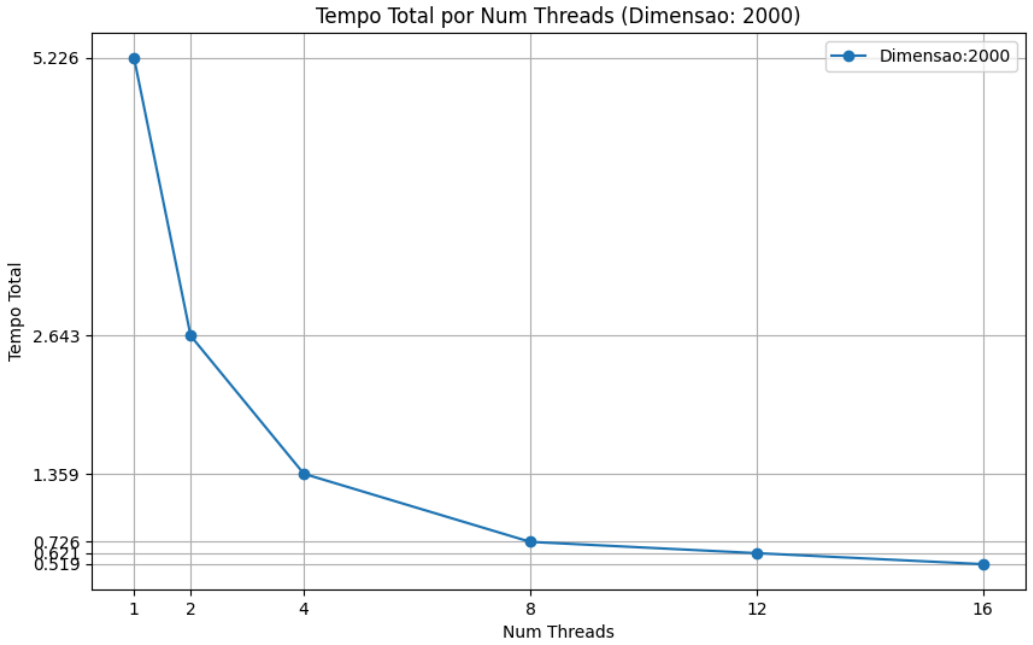
Observações: Inicialmente, eu tinha esquecido de compilar o código com a flag de otimização. Eu percebi uma melhora significativa no desempenho do programa, e pela disponibilidade de tempo, eu fiz os mesmos testes para:
Dimensões: 250, 500, 1000, 2000
Número de Threads: 1, 2, 4, 8, 12, 16

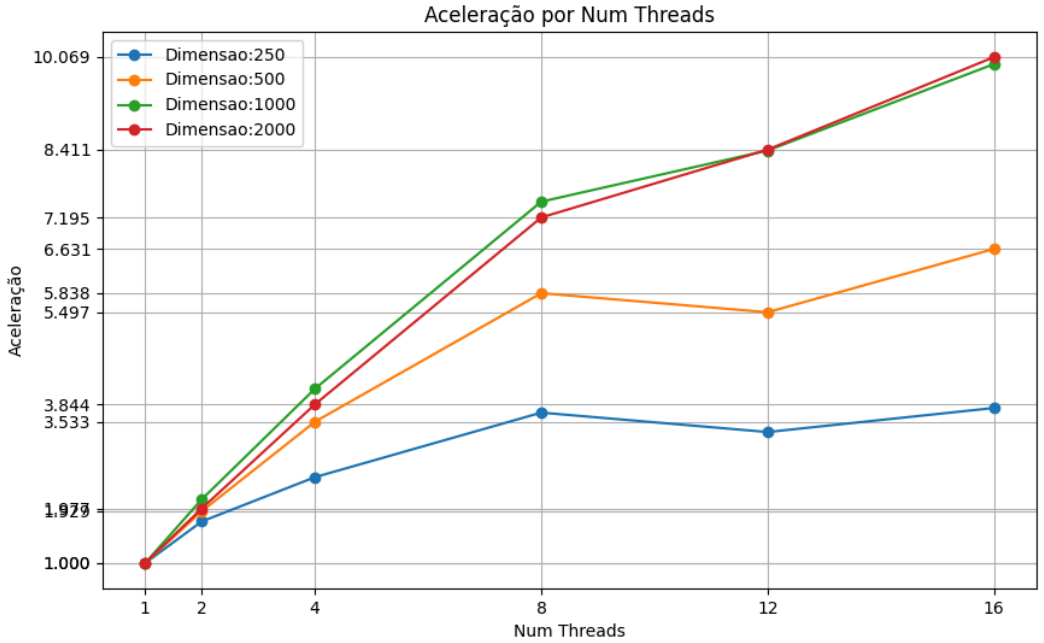
Resultados (Compilando com gcc -Wall -Wextra -O3) :

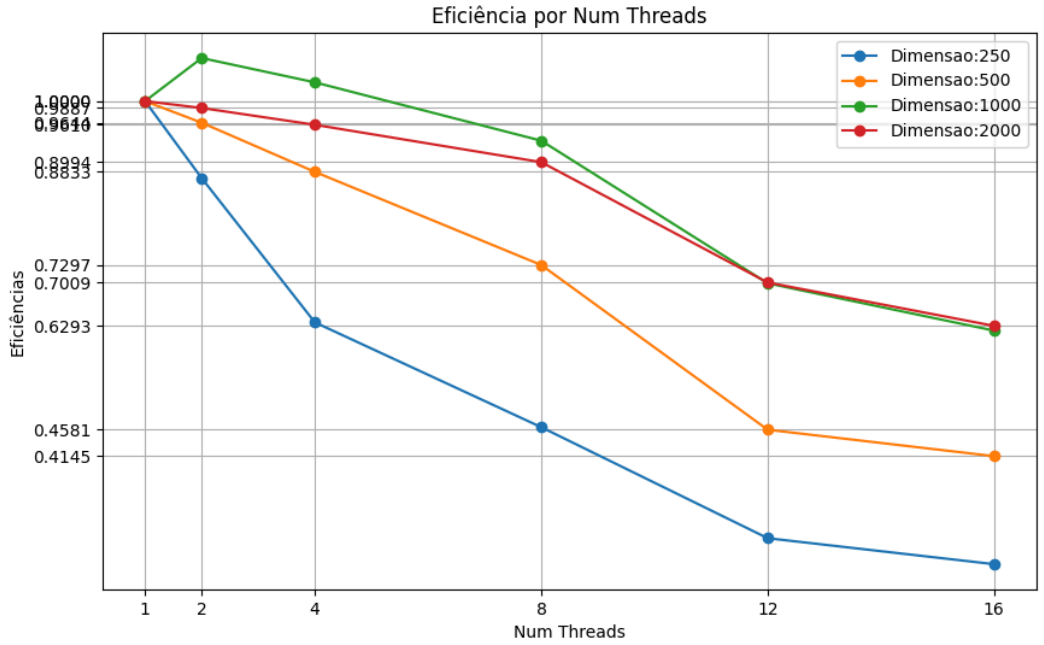





Tempo Total por Num Threads (Dimensao: 1000)
Dimensao:1000
0.7056
0.3294
0.1711
0.0944
0.0840
0.0710
Tempo Total
1
2
4
8
12
16
Num Threads

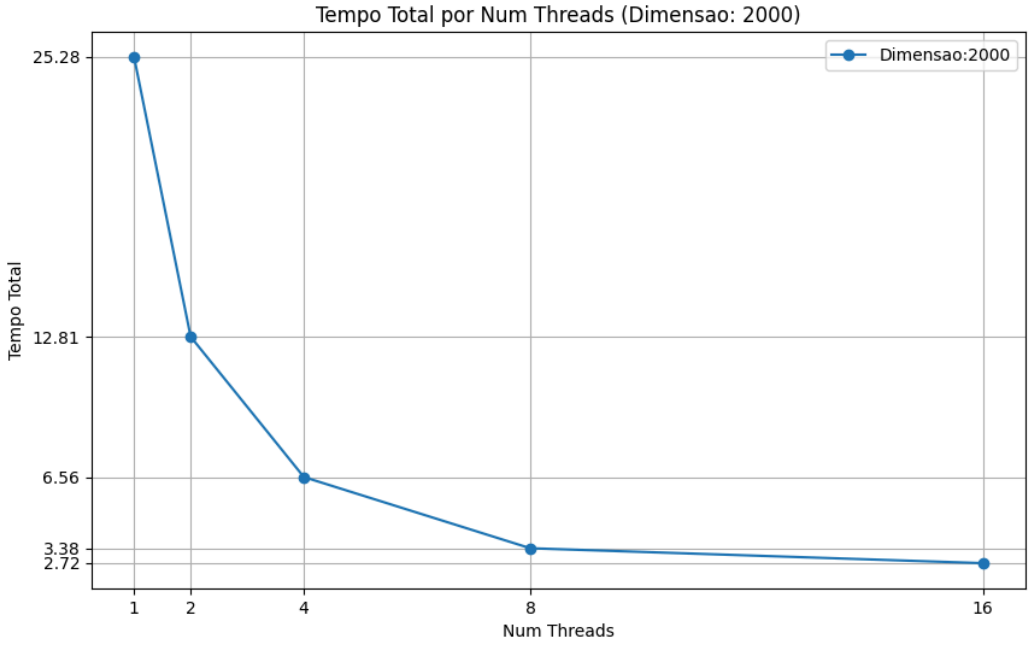

Tempo Total por Num Threads (Dimensao: 2000)
Dimensao:2000
5.226
2.643
1.359
0.726
0.621
0.519
Tempo Total
1
2
4
8
12
16
Num Threads


Aceleração por Num Threads
Dimensao:250
Dimensao:500
Dimensao:1000
Dimensao:2000
10.069
8.411
7.195
6.631
5.838
5.497
3.844
3.533
1.000
Aceleração
1
2
4
8
12
16
Num Threads


Eficiência por Num Threads
Dimensao:250
Dimensao:500
Dimensao:1000
Dimensao:2000
1.0000
0.8833
0.7297
0.7009
0.6293
0.4581
0.4145
Eficiências
1
2
4
8
12
16
Num Threads

A efeito de comparação, sem colocar a flag -O3 olhemos para o gráfico da velocidade do mesmo programa.

Tempo Total n = 2000 (SEM FLAG DE OTIMIZAÇÃO)



(É quase 5 vezes mais lento, e isso é consistente para as outras dimensões da matriz).

Observações especiais:

- Compilar o programa com a flag de otimização diminuiu significativamente o tempo de execução do programa.
- Quando N = 1000, usando entre 2 e 4 threads, tivemos eficiência > 1! Isso ocorreu provavelmente por conta da diminuição de cache miss. (O programa tenta diminuir cache miss ao alocar os vetores de coluna sequencialmente ao utilizá-los)
- Os gráficos de Tempo Total por Num Threads são visualmente muito similares, (a diferença é mais perceptível com valores de N mais baixos, onde o overhead é mais significativo).

Dificuldades:

Pensar em como levantar esses dados e organizá-los foi completamente novo para mim, inicialmente não sabia se usaria ferramentas tipo excel (tentei e falhei miseravelmente). Após algum esforço decidi usar Python, mas ainda não estou satisfeito com o código. Fora isso, adorei usar shell script e misturar várias funcionalidades!