Relatório

Foi feito no repositório mandado do github, na pasta Lab2, o programa matriz.c, onde realiza a multiplicação matricial entre duas matrizes aleatórias duas vezes, uma de forma concorrente e outra de forma sequencial, onde a dimensão das matrizes e o número de threads sao passadas pelo terminal, nessa ordem.

Eu primeiro realizei 3 testes fazendo a multiplicação com matrizes de dimensão 500 e apenas 1 thread na parte concorrente, o resultado foi esse:

```
caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 500 1
Tempo multiplicacao concorrente (dimensao 500) (nthreads 1): 0.748146
Tempo multiplicacao sequencial (dimensao 500) (nthreads 1): 0.633472
Aceleração: 0.846723
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 500 1
Tempo multiplicacao concorrente (dimensao 500) (nthreads 1): 0.749182
Tempo multiplicacao sequencial (dimensao 500) (nthreads 1): 0.621489
Aceleração: 0.829557
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 500 1
Tempo multiplicacao concorrente (dimensao 500) (nthreads 1): 0.741610
Tempo multiplicacao sequencial (dimensao 500) (nthreads 1): 0.643440
Aceleração: 0.867625
```

Como pode-se perceber, todos tiveram resultados semelhantes, em que a multiplicação concorrente teve uma velocidade de cerca de 0.8 vezes a multiplicação sequencial. Isso se deve pois resolvemos fazer a multiplicação concorrente com apenas uma única thread, o que anula toda a vantagem de fazer uma programação concorrente, pois termo apenas uma única thread seria o mesmo que o sequencial, com a diferença que ainda tem o tempo gasto para criar e deletar cada thread, o que fez a multiplicação demorar mais.

Agora realizarei as mesmas operações, com uma matriz com a mesma dimensão, mas agora com 2 threads:

```
caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 500 2
Tempo multiplicacao concorrente (dimensao 500) (nthreads 2): 0.388435
Tempo multiplicacao sequencial (dimensao 500) (nthreads 2): 0.617877
Aceleração: 1.590685
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 500 2
Tempo multiplicacao concorrente (dimensao 500) (nthreads 2): 0.397580
Tempo multiplicacao sequencial (dimensao 500) (nthreads 2): 0.621980
Aceleração: 1.564415
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 500 2
Tempo multiplicacao concorrente (dimensao 500) (nthreads 2): 0.387914
Tempo multiplicacao sequencial (dimensao 500) (nthreads 2): 0.622598
Aceleração: 1.604990
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$
```

Como pode-se ver, agora a multiplicação concorrente foi cerca de 1.6 mais rapido que a multiplicação sequencial. Considerando que o tempo da multiplicação sequencial não é afetada pelo numero de threads, podemos dizer que ao dobrarmos o numero de threads, acabamos diminuindo pela metade o tempo gasto. Isso é verdade na teoria, e agora na prática também se provou verdade.

Com matriz de dimensão 1000 agora:

```
caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 1000 1
Tempo multiplicacao concorrente (dimensao 1000) (nthreads 1): 7.213623
Tempo multiplicacao sequencial (dimensao 1000) (nthreads 1): 6.134381
Aceleração: 0.850388
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 1000 1
Tempo multiplicacao concorrente (dimensao 1000) (nthreads 1): 7.336430
Tempo multiplicacao sequencial (dimensao 1000) (nthreads 1): 6.346236
Aceleração: 0.865031
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 1000 1
Tempo multiplicacao concorrente (dimensao 1000) (nthreads 1): 7.364210
Tempo multiplicacao sequencial (dimensao 1000) (nthreads 1): 6.178886
Aceleração: 0.839043
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$
```

Neste caso, não saiu muito de acordo com o esperado. Uma vez que a dimensão da matriz é maior, o tempo gasto para a criação de threads deveria se tornar mais irrelevante, mas acabou que os resultados foram idênticos (em questão de aceleração) com uma matriz de dimensão 500.

Agora usando 2 threads, o resultado também não foi como o esperado, acabou que a aceleração média foi menor que no teste de dimensão 500, em teoria, deveria ser maior, visto que com o aumento da dimensão, o tempo gasto na criação deveria ser mais irrelevante.

```
caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 1000 2
Tempo multiplicacao concorrente (dimensao 1000) (nthreads 2): 4.177679
Tempo multiplicacao sequencial (dimensao 1000) (nthreads 2): 6.381718
Aceleração: 1.527575
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 1000 2
Tempo multiplicacao concorrente (dimensao 1000) (nthreads 2): 4.309887
Tempo multiplicacao sequencial (dimensao 1000) (nthreads 2): 6.456220
Aceleração: 1.498002
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 1000 2
Tempo multiplicacao concorrente (dimensao 1000) (nthreads 2): 4.379997
Tempo multiplicacao sequencial (dimensao 1000) (nthreads 2): 6.469437
Aceleração: 1.477041
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$
```

Agora, testando com dimensão 2000:

```
caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 2000 1
Tempo multiplicacao concorrente (dimensao 2000) (nthreads 1): 103.893005
Tempo multiplicacao sequencial (dimensao 2000) (nthreads 1): 86.090998
Aceleração: 0.828651
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 2000 1
Tempo multiplicacao concorrente (dimensao 2000) (nthreads 1): 106.550070
Tempo multiplicacao sequencial (dimensao 2000) (nthreads 1): 103.098930
Aceleração: 0.967610
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 2000 1
Tempo multiplicacao concorrente (dimensao 2000) (nthreads 1): 110.697054
Tempo multiplicacao sequencial (dimensao 2000) (nthreads 1): 90.075595
Aceleração: 0.813713
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$
```

Dessa vez, o melhor caso teve um velocidade de 0.96 em relação à multiplicação sequencial, que já é o esperado, uma vez que com o aumento da dimensão da matriz, o tempo da criação das threads se tornou mais irrelevante.

Agora com 2 threads:

```
caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 2000 2
Tempo multiplicacao concorrente (dimensao 2000) (nthreads 2): 60.044019
Tempo multiplicacao sequencial (dimensao 2000) (nthreads 2): 102.000307
Aceleração: 1.698759
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 2000 2
Tempo multiplicacao concorrente (dimensao 2000) (nthreads 2): 68.488180
Tempo multiplicacao sequencial (dimensao 2000) (nthreads 2): 104.308539
Aceleração: 1.523015
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$ ./matriz.out 2000 2
Tempo multiplicacao concorrente (dimensao 2000) (nthreads 2): 67.009448
Tempo multiplicacao sequencial (dimensao 2000) (nthreads 2): 94.319683
Aceleração: 1.407558
matrizes batem!caiocohen@DESKTOP-G46MNSP:/mnt/d/VSCode/compConc/lab2$
```

Como pode-se ver, no melhor caso teve uma aceleração de 1.7 aproximadamente. Como nos outros dois casos, devido ao tempo que levou para cada processo terminar, eu esperei vendo videos no youtube, talvez isso tenha influenciado.

Segue abaixo as informações de núcleos e processadores do computador usado para os testes: