TESTE DE PERFORMANCE DE OPERAÇÕES COM MATRIZES DA BIBLIOTECA NUMPY

Mauricio Silva Soares[1]
Brauliro Gonçalves Leal[2]

RESUMO: Foi feita uma avaliação de performance de matrizes geradas pela biblioteca NumPy, da ordem de 100 colunas por 100 linhas, ao realizar operações de transposição, soma e multiplicação de matrizes, ao executar 100 blocos de 1 milhões operações. A duração média das operações unitárias (DOU) de transposição, soma e multiplicação, foram de 7.69800e-06, 4.57590e-05, 4.57589e-05 segundos respectivamente, mas esses resultados apresentam grande variância para apenas uma operação. Então, para melhor avaliar o desempenho, decidiu-se utilizar da duração por bloco de operações (DBO), que foi em média 9,67614, 56,33278 e 115,2248464 segundos, para transposição, soma e multiplicação das matrizes.

PALAVRAS-CHAVE: banco de dados, operações com matrizes, NumPy

INTRODUÇÃO:

Matrizes são estruturas de dados capazes de armazenar informações de forma dimensional. São estruturas recorrentes em problemas de matemática, engenharia, economia e outras áreas, por facilitar o tratamento de dados e operações sobre os mesmos.

De uma forma simplificada, matrizes são apenas múltiplos vetores organizados de maneira associativa. Em geral, linguagens de programação possibilitam sua declaração com muita facilidade, mas dependendo da aplicação a matriz pode possuir grandes dimensões, assim bibliotecas específicas podem ser utilizadas para facilitar a criação e manipulação dessas estruturas.

Ao trabalhar com uma grande quantidade de dados, exige-se mais recursos do *hardware* utilizado. Operações simples como soma e multiplicação, que atualmente levam o mesmo tempo de execução, podem se tornar mais custosas em tempo, ao serem aplicadas em matrizes muito grandes.

Analisar o comportamento de um computador diante desse cenário, é a melhor maneira de avaliar se o desempenho do mesmo é satisfatório diante da necessidade do operador.

MATERIAIS E MÉTODOS:

A aplicação foi desenvolvida em Python 3.x, juntamente com os pacotes NumPy e CSV. Foram utilizadas matrizes de dimensão 100x100, preenchidas com valores aleatórios entre 0 e 9, para a aplicação das operações de transposição,

soma e multiplicação. A biblioteca utilizada para criação dessas matrizes foi a NumPy, facilmente instalada através do seguinte comando:

```
pip install numpy
```

A biblioteca CSV foi utilizada para a criação de uma planilha no formato .csv, que armazena os tempos obtidos na operações. Essa biblioteca é nativa do Python 3.x.

Para obter uma noção mais realista, foram realizadas 100 milhões de operações de cada tipo, divididas em 100 blocos de 1 milhão de operações.

O computador utilizado possui as seguintes especificações: laptop Dell Inspiron 5547, com Windows 10 Home 64bits, Intel Core i5-4210U 1.70GHz (4 núcleos) e 8182MB de memória RAM.

RESULTADOS E DISCUSSÃO:

A Tabela 2 contém os dados obtidos durante a análise de desempenho. Está apresentado os valores para DBO(Duração por Bloco de Operações) e DAB (Duração Acumulada por Bloco de operações), e como a DOU (Duração Unitária por Operação), possui alta variância e valor muito baixo com relação aos dados obtidos por bloco, de 7.69800e-06, 4.57590e-05, 4.57589e-05 segundos para as operações de transposição, soma e multiplicação respectivamente, esse dado não está presente na Tabela 2.

As operações de DAB variaram entre 9,44170 e 967,18349 s; 55,80305 e 5.633,27833 s; 114,15852 e 11.516,52504 s, para as operações de transposição, soma e multiplicação respectivamente. Por outro lado, os valores médios para DBO foram 9,67614, 56,33278 e 115,22484 segundos, respectivamente. As figuras 1.a, 2.a e 3.a mostram os gráficos da variação dos valores da DBO, tempo(em segundos) em função do número de bloco de operações de transposição, soma e multiplicação, respectivamente. As figuras 1.b, 2.b, 3.b mostram os gráficos da variação dos valores do DAB, tempo acumulado (em segundos) em função do número de blocos operações transposição, soma e multiplicação, respectivamente.

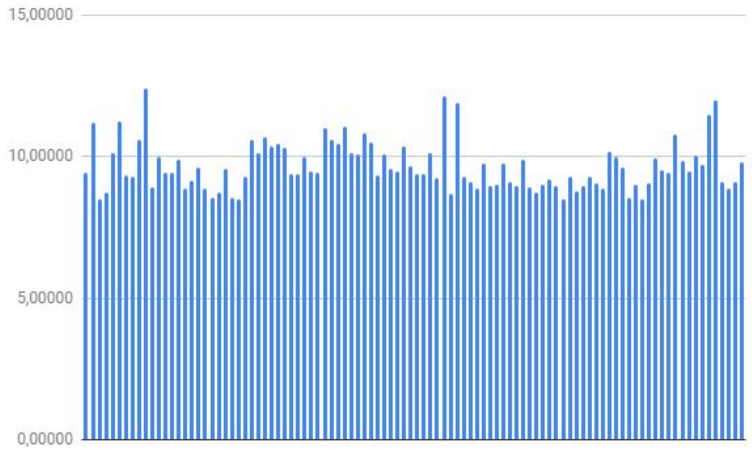
NOP (x1milhão)	Transposição		Soma		Multiplicação	
	DBO (s/bloco)	DAB (s)	DBO (s/bloco)	DAB (s)	DBO (s/bloco)	DAB (s)
1	9,44170	9,44170	55,80305	55,80305	114,15852	114,15852
2	11,22214	20,66385	55,68500	111,48805	114,01868	228,17720
3	8,52286	29,18670	56,49107	167,97912	113,81601	341,99321
4	8,72846	37,91517	55,56022	223,53934	113,75007	455,74328
5	10,14946	48,06462	58,08107	281,62041	113,44733	569,19061

6	11,25309	59,31772	55,15178	336,77219	113,38132	682,57193
7	9,33966	68,65738	55,10611	391,87830	113,18556	795,75748
8	9,30908	77,96646	55,22322	447,10152	113,50994	909,26743
9	10,59803	88,56449	55,37315	502,47467	113,87284	1.023,14026
10	12,42059	100,98509	56,06757	558,54225	113,56922	1.136,70948
11	8,91137	109,89646	55,72366	614,26591	113,89265	1.250,60214
12	9,99271	119,88917	54,93714	669,20305	114,27161	1.364,87374
13	9,42422	129,31339	55,80064	725,00368	116,55516	1.481,42890
14	9,44907	138,76246	56,07908	781,08276	119,28047	1.600,70937
15	9,90375	148,66621	55,20438	836,28714	119,65785	1.720,36721
16	8,87479	157,54100	55,08280	891,36994	118,52199	1.838,88920
17	9,15164	166,69264	55,70029	947,07023	118,49308	1.957,38228
18	9,59784	176,29048	56,02609	1.003,09632	135,28510	2.092,66738
19	8,88629	185,17678	57,58714	1.060,68346	136,45019	2.229,11757
20	8,54324	193,72001	56,64878	1.117,33224	125,01781	2.354,13537
21	8,73544	202,45545	56,33292	1.173,66516	120,44111	2.474,57649
22	9,58099	212,03644	58,39621	1.232,06136	126,18482	2.600,76131
23	8,53029	220,56673	56,68867	1.288,75004	127,97518	2.728,73649
24	8,51586	229,08259	58,13721	1.346,88725	121,97264	2.850,70913
25	9,28486	238,36746	56,89619	1.403,78344	121,01007	2.971,71920
26	10,59388	248,96134	56,34078	1.460,12421	119,85483	3.091,57404
27	10,13343	259,09477	56,42489	1.516,54911	117,84456	3.209,41860
28	10,71497	269,80974	56,64440	1.573,19351	115,98224	3.325,40083
29	10,38811	280,19785	56,46187	1.629,65538	116,04768	3.441,44852
30	10,46944	290,66729	56,78605	1.686,44143	115,98448	3.557,43299
31	10,31623	300,98352	56,70269	1.743,14412	115,98778	3.673,42078
32	9,39080	310,37432	56,59446	1.799,73859	115,66219	3.789,08296
33	9,39295	319,76727	56,97526	1.856,71385	115,19768	3.904,28064
34	9,99284	329,76011	59,68555	1.916,39940	115,93324	4.020,21388
35	9,45829	339,21840	57,04479	1.973,44419	117,26300	4.137,47688
36	9,43539	348,65379	57,86998	2.031,31418	115,21019	4.252,68707
37	11,00400	359,65780	56,48541	2.087,79959	116,12716	4.368,81423
38	10,57699	370,23479	56,89144	2.144,69103	116,73787	4.485,55211
39	10,45703	380,69183	55,91165	2.200,60268	115,00175	4.600,55386
40	11,07742	391,76925	61,22628	2.261,82896	116,72701	4.717,28087
41	10,13562	401,90487	57,73048	2.319,55945	116,54692	4.833,82779
42	10,08668	411,99154	56,56389	2.376,12334	115,19242	4.949,02022

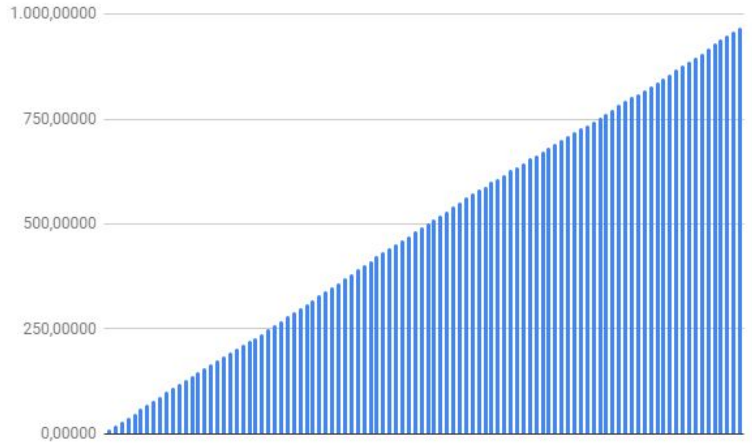
43	10,84907	422,84062	58,66310	2.434,78644	116,86673	5.065,88695
44	10,48694	433,32756	56,85309	2.491,63954	116,50537	5.182,39232
45	9,33905	442,66661	56,04301	2.547,68255	120,40532	5.302,79764
46	10,08310	452,74970	57,78502	2.605,46757	118,50236	5.421,30000
47	9,55356	462,30326	56,61903	2.662,08660	113,02579	5.534,32578
48	9,46786	471,77112	56,34218	2.718,42878	113,41520	5.647,74099
49	10,34415	482,11528	58,14434	2.776,57311	113,43084	5.761,17183
50	9,64657	491,76185	57,34951	2.833,92262	113,40187	5.874,57369
51	9,40691	501,16876	56,83405	2.890,75667	113,41284	5.987,98653
52	9,38704	510,55580	55,92097	2.946,67763	115,63804	6.103,62458
53	10,12955	520,68536	55,60221	3.002,27984	118,08989	6.221,71446
54	9,26414	529,94950	56,00017	3.058,28001	115,41002	6.337,12448
55	12,12703	542,07653	55,48360	3.113,76361	112,56870	6.449,69318
56	8,68644	550,76297	56,40403	3.170,16764	112,64766	6.562,34084
57	11,91976	562,68272	55,61744	3.225,78508	112,46393	6.674,80476
58	9,30850	571,99122	57,70947	3.283,49455	112,66197	6.787,46674
59	9,10996	581,10118	56,57958	3.340,07413	112,66939	6.900,13612
60	8,86483	589,96601	56,39134	3.396,46547	112,53244	7.012,66856
61	9,77650	599,74251	55,71749	3.452,18296	114,78656	7.127,45512
62	8,98568	608,72820	56,80939	3.508,99235	114,59094	7.242,04606
63	9,02071	617,74891	56,60500	3.565,59735	112,41316	7.354,45921
64	9,75844	627,50735	56,74236	3.622,33971	112,68056	7.467,13977
65	9,09545	636,60280	59,63788	3.681,97759	112,64869	7.579,78846
66	8,98275	645,58555	57,72641	3.739,70400	112,53193	7.692,32039
67	9,87687	655,46242	57,37392	3.797,07792	112,66813	7.804,98851
68	8,92501	664,38743	56,41842	3.853,49634	112,65149	7.917,64001
69	8,75041	673,13784	56,53833	3.910,03467	112,57437	8.030,21438
70	8,99891	682,13675	56,66340	3.966,69807	112,49971	8.142,71409
71	9,20706	691,34381	55,17958	4.021,87765	112,70957	8.255,42367
72	8,98716	700,33097	56,92779	4.078,80545	112,44899	8.367,87266
73	8,51224	708,84322	56,23938	4.135,04483	112,44533	8.480,31799
74	9,27851	718,12173	56,85806	4.191,90289	112,75714	8.593,07512
75	8,79102	726,91275	56,21640	4.248,11929	112,78731	8.705,86243
76	8,98921	735,90196	55,86061	4.303,97990	112,43430	8.818,29673
77	9,31485	745,21681	55,52329	4.359,50319	112,46045	8.930,75719
78	9,05377	754,27058	56,00315	4.415,50633	112,32535	9.043,08253
79	8,88888	763,15946	55,58036	4.471,08669	112,38859	9.155,47113

80	10,16554	773,32500	56,00716	4.527,09385	112,58760	9.268,05873
81	9,98666	783,31166	56,00750	4.583,10135	112,41142	9.380,47015
82	9,62342	792,93508	55,26918	4.638,37053	112,47592	9.492,94607
83	8,56106	801,49614	55,25667	4.693,62719	112,42629	9.605,37236
84	9,01566	810,51180	55,39206	4.749,01925	112,37198	9.717,74434
85	8,49947	819,01127	55,21231	4.804,23156	112,36280	9.830,10714
86	9,04443	828,05570	55,25194	4.859,48350	112,42578	9.942,53292
87	9,95073	838,00643	55,32632	4.914,80982	112,43516	10.054,96808
88	9,52740	847,53383	55,33333	4.970,14314	112,44635	10.167,41444
89	9,45586	856,98969	55,24941	5.025,39256	112,42842	10.279,84286
90	10,80190	867,79159	55,30128	5.080,69384	112,42751	10.392,27036
91	9,84331	877,63490	55,19307	5.135,88691	112,58253	10.504,85290
92	9,47860	887,11350	55,32064	5.191,20755	112,35569	10.617,20859
93	10,01895	897,13245	55,27756	5.246,48511	112,31363	10.729,52222
94	9,69226	906,82471	55,24649	5.301,73160	112,31718	10.841,83940
95	11,46288	918,28760	55,27594	5.357,00754	112,43147	10.954,27088
96	11,99627	930,28386	55,19148	5.412,19902	112,65312	11.066,92400
97	9,12711	939,41097	55,24641	5.467,44543	112,54321	11.179,46721
98	8,85156	948,26253	55,24692	5.522,69234	112,56774	11.292,03495
99	9,09300	957,35553	55,22262	5.577,91497	112,50372	11.404,53866
100	9,82796	967,18349	55,36336	5.633,27833	111,98637	11.516,52504
Média	9,676148268	--------	56,33278	--------	115,2248464	--------

Tabela 2 - Duração por Bloco de Operações (DBO), Duração Acumulada por Bloco de operações, para transposição, soma e multiplicação respectivamente.

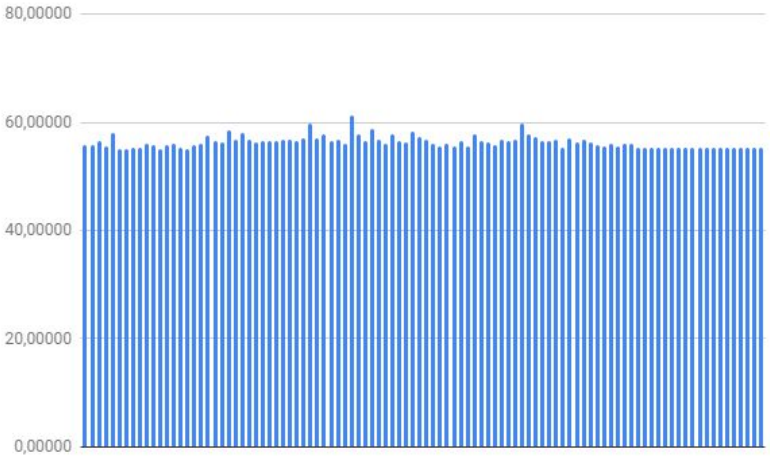


(a)

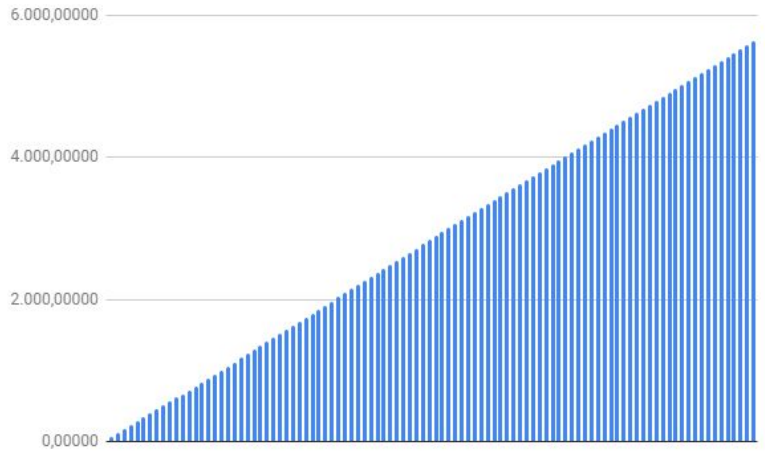


(b)

Figura 1 - (a)Duração por Bloco de Operações (DBO) para transposição (b) Duração Acumulada por Bloco de operações para transposição

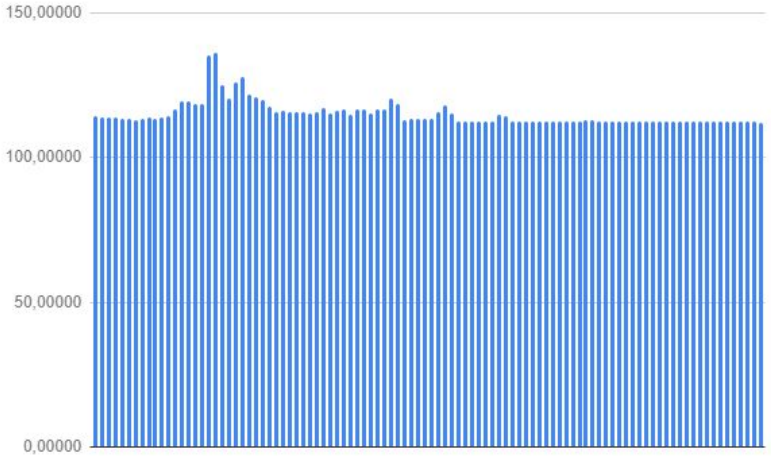


(a)

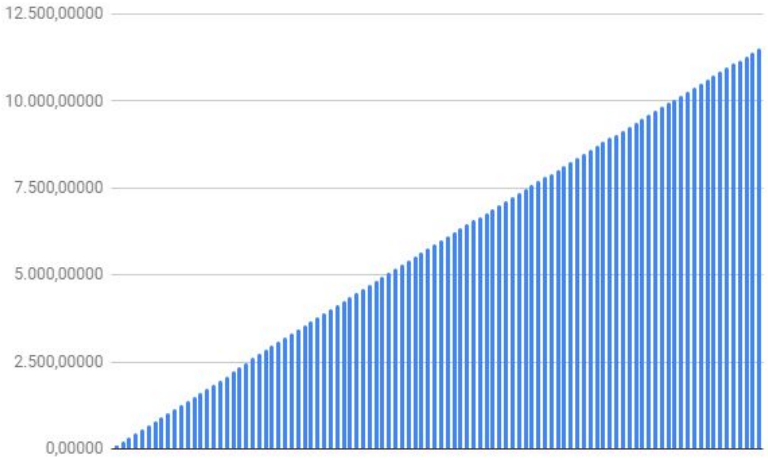


(b)

Figura 2 - (a)Duração por Bloco de Operações (DBO) para soma (b) Duração Acumulada por Bloco de operações para soma



(a)



(b)

Figura 3 - (a)Duração por Bloco de Operações (DBO) para multiplicação (b) Duração Acumulada por Bloco de operações para multiplicação

Das três operações avaliadas a que apresentou maior tempo de execução foram as de multiplicação, seguida pelas de soma que, por sua vez exigiram mais tempo que as de transposição. As operações de soma e transposição requereram, respectivamente, 48,89 e 8,39% do tempo que as de multiplicação.

Percebe-se que a duração dos blocos de operação, mantém-se relativamente constante, sem apresentar muitas variações durante o tempo, para todas as operações. As variações que ocorreram, mostram que existe uma sensibilidade ao uso concorrente do computador para outras atividades paralelas, o que exige mais do poder de processamento.

Em geral, os valores de operações de soma e multiplicação simples, possuem tempos similares, graças a capacidade atual dos processadores. Mas, quando se trata de grandes matrizes, os valores mostram uma diferença considerável nos tempos de execução das operações em grandes matrizes, onde a multiplicação leva aproximadamente o dobro do tempo da soma. A operação de transposição é a mais simples, pois não necessita a realização de cálculos, apenas uma reorganização da matriz.

CONCLUSÃO:

É fácil perceber que atualmente as operações em matrizes são triviais, e o processamento das mesmas já não implicam num grande esforço computacional. Quando tomam-se grandes matrizes, torna-se possível observar que algum esforço por parte do computador passa a ser necessário.

As operações de transposição, soma e multiplicação, apresentaram duração de blocos de operação em ordem crescente. A multiplicação, por se tratar computacionalmente de somas subsequentes, mostrou-se mais lenta que a soma, que se tratou apenas de uma operação direta. Assim, evidencia-se que a complexidade matemática do problema é diretamente proporcional ao esforço computacional para o processamento da operação.

BIBLIOGRAFIA:

LEAL, Brauliro G. **Avaliação de Desempenho de Sistemas**. UNIVASF, Junho de 2016.

ANEXO 1:

```
import numpy as np
import time
import csv

#----------------------DEFINES AND VARIABLES_INIT----------------------------

block = 10000000
```

```
numBlocks = 100
transposeTimes = []
productTimes = []
sumTimes = []
totalTimetranspose = 0
totalTimeProduct = 0
totalTimeSum = 0

#create csv
w = csv.writer(open("times.csv", "w", newline=''))

#matrix 100x100 filled with random values(0-9)
m = np.random.randint(10, size=(100, 100))

#----------FUNCTIONS----------------------------------

#transpose matrix
def transposeMatrix(matrix):
    for j in range(numBlocks):
        initialTime = time.perf_counter()
        for i in range(block):
            c = np.transpose(matrix)
        transposeTimes.append(time.perf_counter() - initialTime)
        #print(transposeTimes[j])

#matricial product
def matricialProduct(matrix):
    for j in range(numBlocks):
        initialTime = time.perf_counter()
        for i in range(block):
            c = matrix*matrix
        productTimes.append(time.perf_counter() - initialTime)
        #print(productTimes[j])

#matricial sum
def matricialSum(matrix):
    for j in range(numBlocks):
        initialTime = time.perf_counter()
        for i in range(block):
            c = m+m
        sumTimes.append(time.perf_counter() - initialTime)
        #print(sumTimes[j])
```

```
#insert in csv
def insertCsv(array,totalTime):

    w.writerow(["Block process times"])

    for i in range(numBlocks):
        w.writerow([i+1, array[i]])

    w.writerow(["....."])
    w.writerow(["TOTAL",totalTime])

#---------------------------MAIN CODE-------------------------------------

##transpose operations
# transposeMatrix(m)
# for i in range(numBlocks):
#     totalTimetranspose = totalTimetranspose + transposeTimes[i]
# print(totalTimetranspose)
# insertCsv(transposeTimes, totalTimetranspose)

##product operations
# matricialProduct(m)
# for i in range(numBlocks):
#     totalTimeProduct = totalTimeProduct + productTimes[i]
# print(totalTimeProduct)
# insertCsv(productTimes, totalTimeProduct)

#sum operations
matricialSum(numBlocks)
for i in range(100):
    totalTimeSum = totalTimeSum + sumTimes[i]
print(totalTimeSum)
insertCsv(sumTimes, totalTimeSum)
```