Untitled

March 29, 2019

1. Analytical:
   Assume that the cost of any single arithmetic operation (adding, subtracting, multiplying, or dividing two real numbers) is 1, and that all other operations are free. Consider the following variant of Strassen's algorithm: to multiply two n by n matrices, starting using Strassen's algorithm, but stop the recursino at some size n_0 - the crossover point. Analytically determine the value of n_0 that optimizes the running time of this algorithm in this model. This gives a crude estimate for the cross-over point between Strassen's algorithm and the standard matrix multiplication algorithm.

   The exact equation for the conventional algorithm is $T(n) = 2n^3 - n^2$. To compute each entry in the matrix product, there are n multiplication steps and $n-1$ addition or subtraction steps, where n is size of the matrix. The computed matrix there are n^2 entries. Therefore, the number of steps is $n^2[(n) + (n-1)]$.
   The exact equation for strassen algorithm is $S(n) = 7S(n/2) + 18(n^2/4)$. There are 18 addition or subtraction steps in the post processing step. Each of these 18 additions or subtractions are on submatrices each with $n^2/4$ entries in each matrix. Assuming addition and subtraction can be done in 1 step. It is known that to compute a matrix addition or matrix subtraction, each entry in the final matrix is computed using 1 addition or subtraction. Since there are $n^2/4$ entries in the final array, we know there must be $n^2/4$ additions or subtractions. There are 7 recursive calls.
   Lets say that n is equal the the size of the matrix that is at the crossover point. Crossover point is defined as the point where matrices with size n that is less than the crossover point will have their matrix product be computed using conventional algorithm. Since we're at the crossover point and not below it, we will call strassen algorithm 1 time. At the recursive calls, the size of the submatrices are $n/2$. Therefore, the matrix product of each of these submatrices will be computed using conventional algorithm.
   Therefore, $S(n) = 7S(n/2) + 18(n^2/4) = 7(2(n/2)^3 - (n/2)^2) + 18(n^2/4)$. This equation represents the modified Strassen algorithm that uses conventional algorithm to compute matrix multiplication when n is less than the crossover point.
   $2n^3 - n^2 = 7(2(n/2)^3 - (n/2)^2) + 18(n^2/4)$ is solved for n. Setting the two equations equal to each other is done because that corresponds to when the number of steps in conventional algorithm is equal to the number of steps in the modified Strassen algorithm. And we know that conventional algorithm performs faster on smaller n. This implies that the conventional algorithm is faster for n smaller than crossover point.
   Solving the above equation will give $n = 15$. Any matrix size below this point will have its matrix product computed using only conventional algorithm because it has less number of steps. This results in the modified strassen algorithm being faster than a pure strassen algorithm and faster than the pure conventional algorithm.

If n is not a power of 2, then the optimal crossover point really depends on the implementation of the padding procedure.

2. Experimental:
Implement your variant of Strassen's algorithm and the standard matrix multiplication algorithm to find the crossover point experimentally. Experimentally optimize for n_0 and compare the experimental results with your estimate from above. Make both implementations as efficient as possible. The actual crossover point, which you would like to make as small as possible, will depend on how efficiently you implement strassens algorithm. Your implementation should work for any matrices, not just those whose dimensions are a power of 2. To test your algorithm, you might try matrices where each entry is randomly selected to be 0 or 1; similarly, you might try matrices where each entry is randomy selected to be 0, 1, 2 or instead 0, or 1, or -1. You might also try matrices where each entry is a randomly selected real number in the range $[0, 1]$. You need not try all of these, but do test your algorithm adequately.
> Strassen.py contains all the code necessary for the implementation of strassen algorithm with modified crossover point. The code is also modified to support non power of 2 matrices. The algorithm will output all the diagonal entries of the final matrix product. Originally, I had set the padding to turn any matrix that is not a power of 2 into a matrix with size that is a power of 2. However, I realized that this padding function would add a lot of 0's, and make the matrix considerably bigger. I believed that this would of had a significant impact on the runtime of the algorithm. Instead, I decided to do multiple padding steps whenever a matrix is not divisible by 2. In other words, if the matrix is not even, I made it even. Although there would be multiple times in which the matrix would have 0's added, I believed that this would lead to less 0's when n is large. Also, in python, list comprehensions are significantly faster than regular for loops. Therefore, list comprehensions were used whenever possible to optimize performance. The results correspond to the runtime of the modified strassen algorithm with different crossover points. It makes no sense to compute the pure conventional runtime and then find the crossover point for the modified strassen algorithm that gives the equivalent runtime. The reason we set the conventional runtime equal to the modified strassen runtime is because the n, the size of the input matrix, used at this situation gives the runtime the same for modified strassen and pure conventional algorithm. In other words, the size of the input matrix will have the same runtime for conventional algo and modified strassen when used to compute matrix products. However, the experimental procedure involves fixing the size of the input matrix n. The runtime computed for the modified strassen algorithm includes the runtime for recursing the input matrix of size n down to a matrix of size that is equal to the crossover point. As a result, the crossover point is the point at which the modified strassen algorithm gives the lowest runtime.
> The following is the time data corresponding to a crossover point for a matrix with size 100. Each matrix contains a repeating sequence of numbers from 0, 1, 2, 3, ..., 1000 until all 10,000 entries are filled.

crossover point	time
1	6.316427827
2	1.724118221
3	1.739031339
4	0.668495476

crossover point	time
5	0.662989366
6	0.682287991
7	0.278555024
8	0.286055529
9	0.281349885
10	0.278797996
11	0.28150456
12	0.277634454
13	0.164631593
14	0.161835217
15	0.161939144
16	0.166419232
17	0.165259814
18	0.164722049
19	0.16384238
20	0.164868593
21	0.162635636
22	0.163420498
23	0.165752017
24	0.162554145
25	0.127250075
26	0.125394773
27	0.126759017
28	0.12487489
29	0.127231991
30	0.126024151
31	0.127437341
32	0.124930716
33	0.126116538
34	0.124398947
35	0.126703739
36	0.12715596
37	0.126036048
38	0.125642741
39	0.127703965
40	0.125235009
41	0.126585829
42	0.128329158
43	0.126985216
44	0.126248753
45	0.125291264
46	0.128246784
47	0.125582647
48	0.126106036
49	0.126421559
50	0.129061043

crossover point	time
51	0.129822648
52	0.126638591
53	0.128419697
54	0.12707634
55	0.126157761
56	0.129650307
57	0.127713227
58	0.129436052
59	0.126315498
60	0.128389966
61	0.128243613
62	0.126826191
63	0.126277232
64	0.127773154
65	0.127836823
66	0.128049874
67	0.128130245
68	0.127086639
69	0.126479399
70	0.127691543
71	0.128954029
72	0.13180325
73	0.126891243
74	0.128448021
75	0.132727325
76	0.128327465
77	0.129361963
78	0.127563226
79	0.128408837
80	0.130402732
81	0.127879751
82	0.126959729
83	0.130120659
84	0.126330447
85	0.130506039
86	0.128681397
87	0.127860272
88	0.126510513
89	0.129546666
90	0.128906882
91	0.128992057
92	0.128566766
93	0.128008819
94	0.130066109
95	0.128299844
96	0.128150272

crossover point	time
97	0.128347278
98	0.130738091
99	0.127097428
100	0.153787363

At crossover point of 100, the algorithm will use conventional matrix product to compute the final result because the starting matrix has size 100. Therefore, the .153 runtime is the runtime of pure conventional algorithm. The results indicates that the optimal crossover point is at 25 because the runtime for the next 30+ crossover points are roughly equal to the runtime of this crossover point. The following is the runtime data corresponding to a crossover point for a matrix with size 400. Each matrix contains a repeating sequence of numbers from 0, 1, 2, 3, . . . , 99 until all 25,000 entries are filled.

crossover point	time
2	85.25
4	32.47
6	32.07
8	13.32
10	13.43
12	13.3
14	7.91
16	7.85
18	7.86
20	7.92
22	7.96
24	7.94
26	6.09
28	6.02
30	6.07
32	6.13
34	6.13
36	6.06
38	6.03
40	6.07
42	6.06
44	6.04
46	6.08
48	6.02
50	6.05
52	6.21
54	6.09
56	6.08
58	6.12
60	6.06
62	6.1

crossover point	time
64	6.09
66	6.16
68	6.09
70	6.1
72	6.14
74	6.05
76	6.09
78	6.12
80	6.12
82	6.15
84	6.07
86	6.09
88	6.1
90	6.09
92	6.12
94	6.11
96	6.11
98	6.74
100	6.74
102	6.67
104	6.64
106	6.65
108	6.67
110	6.68
112	6.66
114	6.64
116	6.66
118	6.67
120	6.73
122	6.68
150	6.65
200	7.71
300	7.8
400	10.17

At crossover point of 400, the runtime of conventional algorithm is 10.17. This table shows that the first lowest runtime is 6.09, which corresponds to crossover point of 26. The next 50+ runtimes are roughly equal to the runtime of this crossover point. Also, the runtime of crossover point 24 is significantly higher than the runtime at 26.

Notice that matrices of size 100 and 400 are not powers of 2. This means that somewhere in the recursive calls, padding will be used to turn odd sizes into even sizes. It is of interest to see how the algorithm performs when padding is not utilized.
The following runtime results are done on a matrix of size 256. Each matrix contains the repeated sequence of numbers from 0, 1, 2, 3, . . . , 127 until reached 65,536 entries.

crossover point	time
1	45.38928246
2	11.56899333
4	4.171915054
6	4.278640032
8	2.356656551
10	2.377503872
12	2.378200531
14	2.403790474
16	1.834965467
18	1.824825525
20	1.786560297
22	1.810774565
24	1.834287167
26	1.804689168
28	1.802449942
30	1.791874886
32	1.684848308
34	1.716532946
36	1.712934494
38	1.6973207
40	1.726763487
42	1.718386412
44	1.688958168
46	1.6940341
48	1.706473589
50	1.721262932
52	1.696356792
54	1.690901756
56	1.698409796
58	1.715611696
60	1.717405319
62	1.737286568
64	1.776877642
66	1.803012371
68	1.773481131
70	1.80351758
72	1.797300816
74	1.788123846
76	1.796143055
78	1.783159018
80	1.812639952
82	1.767787457
84	1.798843622
100	1.779609442
150	2.019761086
200	2.033017874

crossover point	time
256	2.439218283

The results indicate that the best crossover point is at 32. This is odd because no padding of 0's is utilized to calculate the matrix product, so would expect the crossover point to be lower than the others.
The experimental crossover point is larger than the analytical one. The reason is probably because of the implementation methods, hardware, and programming language.