

Algorithms Homework 1 Report

2020. 4. 19.
20161349 Jaeho Kim

Intro

Matrix multiplication is a central operation requiring multiple numerical operations. The usual matrix multiplication we learn in high school is an iterative algorithms that takes a complexity¹ of $O(n^3)$ when multiplying size of $n \times n$ matrices. An alternative algorithm is the divide and conquer where it relies on partitioning the matrix into blocks. The divide and conquer algorithm divides the matrices into 4 sub matrices and performs 8 multiplications with matrix size $n/2$ and 4 additions. As the algorithm is recursive the time complexity would be $T(n) = 8T(n/2) + O(n^2)$. Unfortunately, this also takes an algorithm complexity of $O(n^3)$ which is the same as the usual matrix multiplication. And Strassen comes to the rescue. Strassen proposes to reduce the recursive calls from 8 to 7, leading to a time complexity of $T(n) = 7T(n/2) + O(n^2)$. This has an algorithm complexity of $O(n^{\log 7})$ leading to a faster implementation of matrix multiplication.

In our homework, we are given a input.txt file which consists of integer Matrix A and B. We are required to implement 4 matrix multiplication algorithms which are schoolbook multiplication, naïve divide and conquer, Strassen's method, and optimized Strassen's method. Matrix sizes will only be given with size that equals power of 2. Execution time should be compared between algorithms. Here is the detail of my implementation environment.

Language	Python3.6
OS	Ubuntu 16.04
CPU	Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz

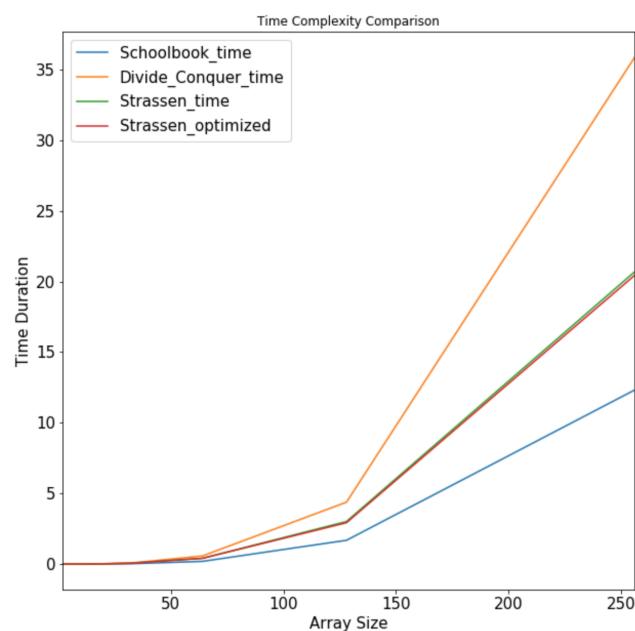
All the source code is in the appendix.

¹ https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm

Observation

The implementation of the algorithm itself was not difficult as the code was not much different with the pseudo-code provided in the text book. However, after implementation, I found out that the execution time of Naïve divide and conquer and Strassen took so much more time compared to the standard matrix multiplication. It took 20 times longer for divide and conquer and 8 times more for Strassen compared to schoolbook implementation. Below is the time comparison when I first implemented the algorithm.

Array_size	Schoolbook_time	Divide_Conquer_time	Strassen_time
2	4.7206878662109375e-05	0.0002162456512451172	0.0001704692840576172
4	0.00010967254638671875	0.0013222694396972656	0.0012149810791015625
8	0.0006349086761474609	0.01047968864440918	0.005822181701660156
16	0.002545595169067383	0.05387568473815918	0.03977251052856445
32	0.019907236099243164	0.5146727561950684	0.33355069160461426
64	0.22497129440307617	4.117962598800659	2.360898733139038
128	1.554166555404663	33.267693758010864	16.539644956588745
256	13.431137800216675	263.7457625865936	116.66768956184387
512	108.36793613433838	2108.810677051544	808.7790343761444



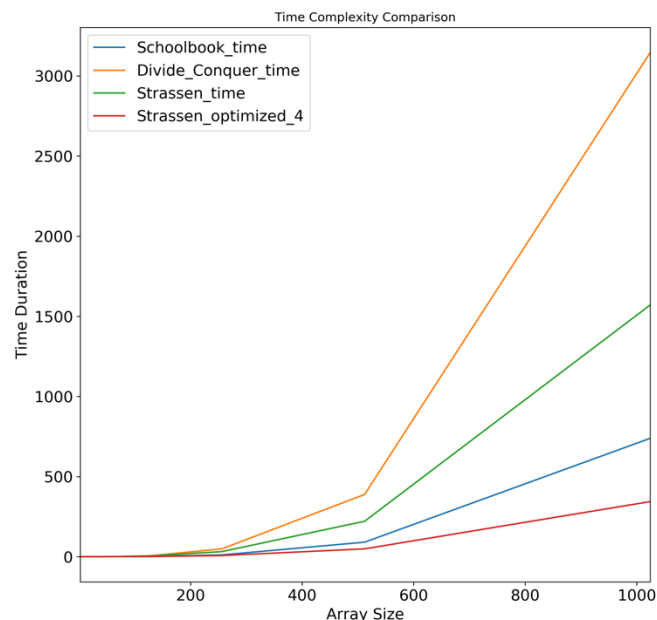
There were some comments in Google saying that Strassen will only perform better when the array size is big enough. However, even after increasing the size of the matrix to 1024, there wasn't much difference although the gap decreased slightly. I tried reducing the lines of code, removed numpy functions like "np.vstack" and made the codes neater hoping that it would result in difference. However, the results weren't much different from what I first observed.

Optimization

With some more digging in to Google, I found this comment² where it states *“Strassen's algorithm has a better Big O complexity, but constants do matter in reality, which means in reality you're better off with a standard n^3 matrix multiplication for smaller problem sizes.”*

To decrease the constant time I figured out Both Naïve divide and conquer and Strassen were making a recursive call until when $n == 1$. After trying out with different n , Strassen shows way much better performance than the standard matrix multiplication. The experimental results are shown as below.

	Array_size	Schoolbook_time	Divide_Conquer_time	Strassen_time	Strassen_optimized_4
0	2	1.8358230590820312e-05	4.220008850097656e-05	7.653236389160156e-05	1.1444091796875e-05
1	4	9.5367431640625e-05	0.00027322769165039057	0.0005576610565185547	5.507469177246094e-05
2	8	0.00037288665771484375	0.0018463134765625	0.002040386199951172	0.00043487548828125
3	16	0.002859354019165039	0.012175559997558594	0.013447046279907228	0.003189563751220703
4	32	0.022125244140625	0.09489798545837402	0.09223532676696776	0.02053999900817871
5	64	0.1759471893310547	0.7579932212829591	0.6472883224487305	0.14138078689575195
6	128	1.3935551643371582	6.051497459411621	4.538825750350952	1.0051209926605225
7	256	11.085448026657104	48.66181397438049	31.73547124862671	7.021599531173706
8	512	90.54794383049013	388.38110184669495	221.06101369857788	48.91199851036072
9	1024	738.2962219715117	3145.115173816681	1570.400215625763	343.60464906692505



We can see that Strassen now performs better than any other algorithms.

² <https://stackoverflow.com/questions/11495723/why-is-strassen-matrix-multiplication-so-much-slower-than-standard-matrix-multip>

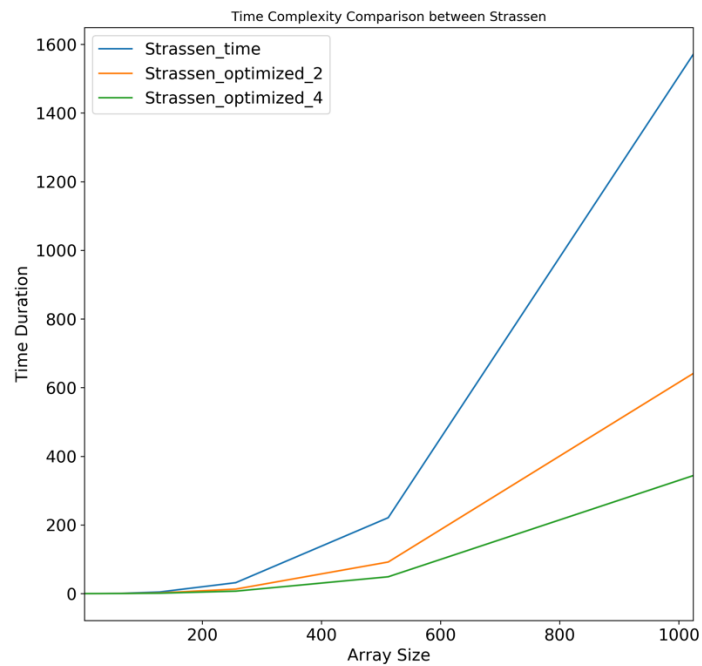
Strassen Comparison

	Array_size	Strassen_time	Strassen_optimized_2	Strassen_optimized_4
0	2	7.653236389160156e-05	1.33514404296875e-05	1.1444091796875e-05
1	4	0.0005576610565185547	0.00012946128845214844	5.507469177246094e-05
2	8	0.002040386199951172	0.0008156299591064452	0.00043487548828125
3	16	0.013447046279907228	0.005401372909545898	0.003189563751220703
4	32	0.09223532676696776	0.03788375854492188	0.02053999900817871
5	64	0.6472883224487305	0.2641913890838623	0.14138078689575195
6	128	4.538825750350952	1.859738111495972	1.0051209926605225
7	256	31.73547124862671	12.974822998046875	7.021599531173706
8	512	221.06101369857788	91.91951942443848	48.91199851036072
9	1024	1570.400215625763	641.0124650001526	343.60464906692505

Strassen Time : Ordinary Strassen with base recursion as 1

Strassen_optimized_2 : Strassen with base recursion as 2

Strassen_optimized_4 : Strassen with base recursion as 4



As we can see by increasing the base case size, the Strassen algorithm takes shorter time.