CS 124 Programming Assignment 2: Spring 2023

Your name(s) Yuting Liu:

Collaborators: (You shouldn't have any collaborators but the up-to-two of you, but tell us if you did.)

No. of late days used on previous psets: 4
No. of late days used after including this pset: 4

1.Cross-over point analysis

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. In this case, the conventional matrix multiplication takes n^3 of multiplication operations and $n^2(n-1)$ addition steps, so the cost of conventional method is $n^3+n^2(n-1)$. For Strassen's algorithm, each step takes 7 recursions and 18 adding or substracting operations, so $T(n) = 7T(\frac{n}{2}) + 18(\frac{n}{2})^2 = 7T(\frac{n}{2}) + \frac{9}{2}n^2$. Let n_0 be the cross-over point, when $\frac{n}{2} < n_0 < n$, the base case for two $\frac{n}{2} * \frac{n}{2}$ matrices takes conventional multiplication, and for n*n size multiplication it takes steps as:

$$\begin{aligned} T(n) &= 7T(\tfrac{n}{2}) + \tfrac{9}{2}n^2 \\ &= 7(2(\tfrac{n}{2})^3 - (\tfrac{n}{2})^2) + \tfrac{9}{2}n^2 \\ &= 7(2\tfrac{n^3}{8} - \tfrac{n^2}{4}) + \tfrac{9}{2}n^2 \\ &= \tfrac{7n^3}{4} + \tfrac{11n^2}{4} \end{aligned}$$

Recursively, the number of arithmetic steps that Strassen's algorithm takes will have the polynomial increase, and the lowest cost curve we can make is to let the smallest value of the Strassen's part (this function is not right-continuous but we assume it is) equal with largest value of the conventional part. Therefore, $\frac{7n^3}{4} + \frac{11n^2}{4} = n^3 + n^2(n-1)$, and $n = 15$.

However, 15 is not practically useful as not a power of 2, so we may choose either 8 or 16. The result of cost subtraction using $n_0 = 8$ minus $n_0 = 16$ gets a negative result. Therefore, I choose the cross-over point as 8.

$T(2^{m-3} * 2^3) - T(2^{m-4} * 2^4)$
$= [7^{m-3}T(8) + 7^{m-4}18 * 8^2 + ... + 18(2^{m-4}8)^2] - [7^{m-4}T(16) + 7^{m-5}18 * 16^2 + ... + 18(2^{m-5}16)^2]$
$= 7^{m-3}T(8) + 7^{m-4}18 * 8^2 - 7^{m-4}T(16)$
$= 7^{m-4} * (-80) < 0$

2. Strassen implementation

At first, I constructed the base case for this algorithm: the simple conventional matrix multiplication taking the cross-over point as a parameter of the function. In order to make it work for any size matrices, I paddles the matrices to matrices whose size are the closest but not smaller power of 2. When the length of the matrices is less than or equal with the cross-over point, the conventional method is implemented, otherwise it applies the Strassen's recursively.

To test the experimental cross-over point of the algorithm, I need to select the values of n_0 to run the algorithm. Since the length of paddles matrices that needed to be compared with n_0 always to be a power of 2, setting n_0 not a power of 2 is not practically useful. Therefore I only test the power of 2 ranging from 1 to 512.

The factors that may affect the running time of the algorithm include the matrices to be multiplied, their component values and their size. I tried two types of matrices where each entry is randomly selected to be 0 or 1, or randomly selected to be 0, 1 or 2. As for the size of the matrices, I use 700, 1000, 2000. The paddling time is also taken into account. The running time (ms) as the function of n_0 in these 6 experiments are demonstrated as following:

	700,2	700,3	1000,2	1000,3	2000,2	2000,3
0	1.221177e+06	1.211036e+06	1.322441e+06	1.335698e+06	9.079782e+06	9.002107e+06
1	3.744712e+05	3.728625e+05	4.213652e+05	4.168839e+05	2.922192e+06	3.602215e+06
2	1.698046e+05	1.896578e+05	1.928738e+05	1.935799e+05	1.357913e+06	1.193224e+06
3	1.100465e+05	1.262335e+05	1.251259e+05	1.249159e+05	8.812336e+05	7.773303e+05
4	9.185822e+04	1.037234e+05	1.033935e+05	1.039347e+05	7.314946e+05	6.511628e+05
5	8.796708e+04	9.765265e+04	9.799558e+04	9.833481e+04	6.849641e+05	6.154588e+05
6	1.441282e+05	1.017644e+05	1.030728e+05	1.028991e+05	7.238819e+05	6.387931e+05
7	2.105402e+06	1.129725e+05	1.115752e+05	1.116700e+05	7.805384e+05	6.978577e+05
8	2.097904e+06	1.289480e+05	1.269238e+05	1.260858e+05	8.818682e+05	7.917919e+05

Figure 1: Running time for optimal base case

According to this chart, the optimal cross-over point n_0 is $2^5 = 32$, regardless of the components and the size of matrices. We can observe that with same matrices component and size of matrices, the running time decrease at first then increase, and the smallest value occurs all at 32.

We can also observe that it did not seem to matter significantly whether the inputs were 0/1 matrices or 0/1/2 matrices, as there is no consistent relationship between these two settings: the 0/1 matrices can take longer time than 0/1/2, but can also take shorter time than 0/1/2 with the same matrices size and crossover point, and the experiments were not perfected controlled due to the machine dependences.

However, cross-over point = 32 does not mean that when the size of matrices larger than 32 the optimized strassen's is practically better than naive method. I therefore designed experiments solely on naive method and compare it with optimized strassen's. When the matrice size less than 2048, the conventional method is always faster than Strassen's. In this case I use the own-written simple method code, if use library method it could perform even better. On the other side, there are things need to be improved for Strassen's, such as memory access strategy.

3. Triangle in random graphs

The chart showing your results compared to the expectation is as follows:

The expected number is [178.43302400000002, 1427.4641920000001, 4817.691647999999, 11419.713536000001, 22304.128000000004], and the actual count is [152, 1483, 4702, 11542, 21548]. The actual number is very close to the expected number. The difference of the actual minus the expected can be positive or negative, without a clear pattern.

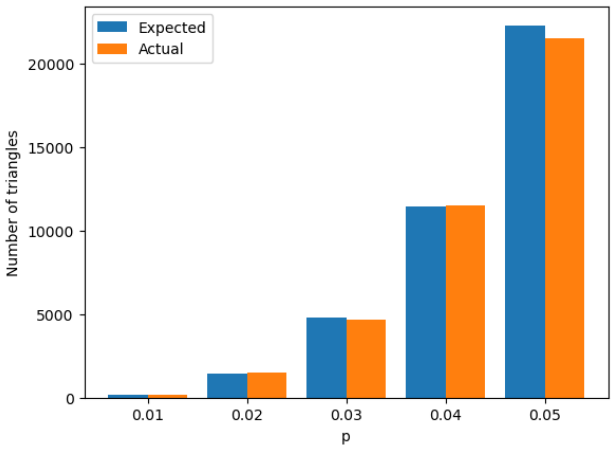

Figure 2: The actual and expected number of triangles

Discussion

Cross-over point

According to the analytical part, the cross-over point is 8, while based on the implementation part, the crossover point is 32. The reason of this difference is that we assume that the time cost of any single arithmetic operation (adding, subtracting, multiplying, or dividing two real numbers) is 1, and that all other operations are free. This is unrealistic, as each arithmetic operations takes different running time (multiplication is more expensive than addition), and non-arithmetic operations also need time for running (memory allocation is much more expensive than the assumption as 0).

Difficulties and optimization

One of the difficulties is how to implement Strassen's algorithm on non-power of 2 size of matrices. The solution is to pad with 0's to make the size of the matrices N become the closest and larger power of 2 than the original size n. $N < 2n$, and this padding doesn't affect the asymptotic complexity. This implementation is the simplest. However, static padding introduce enormous sparse elements into the matrices, almost three times the number of original matrix elements in the worst case. This takes more memory for loading, and the arithmetic done on these additional zero elements is pure overhead.

There are still bottlenecks for the naive and strassen's algorithm. The way to improve conventional matrix multiplicatoin is use vector arithmetic rather than process it on elements. Using numpy dot operation would be faster. Also the strassen's will be improved if using better cache performance.

Another difficulties is to create a random graph on 1024 vertices where each edge is included with probability p. This is implemented by create a random matrix with values uniformly range from 0 to 1, and choose elements less than p and label as 1, with the remaining labelled as 0. Choose the lower triangle of this matrix and combine it with its transpose to form an un-directed adjacency matrix. The randomization function applied here is numpy.random.rand(), generating pesudo random number from 0 to 1 under the uniform distribution.

Matrix types

In our implementation, I chose type of elements in the matrices as int64 (to be consistent across architectures). This choice does not matter. I took an experiment to change type from int to float, and the output of the algorithm is consistent as before. As for the running time, using cross-over point as 32, the running using float64 is slower than using int64.