Programming Assignment 2 Writeup

Piyush Kathuria

April 3, 2020

1 PART I - Crossover point via analysis

We will first try to get the crossover point via analysis. Let's first consider the standard algorithm for matrix multiplication.

In the conventional matrix multiplication of two nxn square matrix A and B, for each entry $c(i,j)$ in the resultant matrix C, we perform two sets of operations :

1. n multiplication operations of $a(i,k) * b(k,j)$ from k = 1 to n.
2. $n-1$ summation operations for the products we retrieve from step 1.

Since we have n^2 entries in the matrix, the total work done is equal to $n^2 * (n+n-1) = 2n^3 - n^2$.

Now let's look at Strassen's algorithm. We can extend the recurrence equation taught in lecture in the following way:

$T(n) = 7 \cdot T(\lceil \frac{n}{2} \rceil) + 18 \cdot (\lceil \frac{n}{2} \rceil)^2$

The $\lceil \frac{n}{2} \rceil$ is to extend the recurrence equation to handle odd and even matrix dimension values, and the $18 \cdot (\lceil \frac{n}{2} \rceil)^2$ accounts for the total number of sum operations required in Strassen's algorithm.

Now, we require a point n_0 such that

$$T(n) = \begin{cases} 7 \cdot T(\lceil \frac{n}{2} \rceil) + 18 \cdot (\lceil \frac{n}{2} \rceil)^2 & n > n_0 \\ 2n^3 - n^2 & n \leq n_0 \end{cases}$$

In order to solve for n_0, we will need to solve for two cases, even and odd matrix dimension.

1.1 Solving for even n

For even values of n, we can simplify our recurrence relation to the following equation:

$$14(\frac{n}{2})^2 + 11(\frac{n}{2})^2$$

Thus, we would like to get a value of n such that :

$$\frac{14n^3}{8} + \frac{11n^2}{4} < 2n^3 - n^2$$

Solving the above equation, we get the value as :

$$\frac{15n^2}{4} < \frac{n^3}{4}$$

$$15 < n$$

Thus, for all even $n_0 > \mathbf{16}$, our Strassen's algorithm is going to be faster, and below this value, for all even $n_0 < 16$ we can switch to our traditional matrix multiplication method.

1.2 Solving for odd n

For even values of n, we can simplify our recurrence relation to the following equation:

$$14(\frac{n+1}{2})^3 + 11(\frac{n+1}{2})^2$$

Thus, we would like to get a value of n such that :

$$14(\frac{n+1}{2})^3 + 11(\frac{n+1}{2})^2 < 2n^3 - n^2$$

Solving for n_0, we get the odd value of 37. Thus, for odd values of $n_0 < \mathbf{37}$ we should use the traditional matrix multiplication algorithm.

2 PART II - Experimental evidence for crossover point

2.1 Traditional Matrix Multiplication Implementation

As we know, our traditional matrix multiplication is a $O(n^3)$ algorithm. However, to make it much more memory efficient, I have tried to make the algorithm much more cache efficient. In this implementation, the entries of the matrices that are closer to memory are multiplied first. As it turns out, this trick proves to perform significantly better as you increase the value of n. You can find more about the implementation on this link : https://sites.cs.ucsb.edu/ tyang/class/240a17/slides/Cache3.pdf. You can find the benchmarks here :

Size	Optimized	Non Optimized
1	0.000002	0.000002
2	0.000002	0.000002
4	0.00003	0.000003
8	0.000010	0.000007
16	0.000049	0.000039
32	0.000373	0.000291
64	0.002910	0.001028
128	0.009404	0.009483
256	0.071018	0.069688
512	0.552080	0.641357
1024	4.409454	9.780876
2048	36.223057	155.373367
4096	481.939255	1300.16409186

2.2 Strassen's Algorithm

There are three major optimization that I have done to improve the memory efficiency of the Strassen's algorithm.

1. Passing a pointer storing the address of the first element of the square matrix instead of the matrix itself. This saves me precious memory which proves to be very important as the value of n increases.

2. If the dimension of the matrix is not a power of 2, I padded the square matrix such that the new matrix would have a dimension that is a power of 2. For example, if we pass a 13x13 matrix, the padded matrix would become 16x16.

3. When performing addition of subtraction of two matrices A and B, instead of creating a new matrix C and filling it's element $c(i,j)$ with $a(i,j) \pm b(i,j)$, I update the $a(i,j)$ memory address with the result using the pointer variable I passed to the sum and add function. This saves me the memory to store 18 new matrices at the cost of a single extra sum operation. This there would be 19 add operations instead of 18. In my opinion, however, the memory I save far outweighs the one addition operation.

Here is a proof that, even after step 3, we can get the same expressions as our original Strassen algorithm.

$F' = F - H$
$P_1 = A \cdot F'$
$A' = A + B$
$P_2 = A' \cdot H$
$C' = C + D$
$P_3 = C' \cdot E$
$G' = G - E$
$P_4 = D \cdot G'$
$P_5 = ((A' - B) + D) \cdot (E + H) = (A + B - B + D) \cdot (E + H) = (A + D) \cdot (E + H)$
The $(A' - B)$ adds one extra subtraction.
$A'' = A + D$
$E' = E + H$
$P_6 = (B - D) \cdot (G' + E') = (B - D) \cdot (G - E + E + H) = (B - D) \cdot (G + H)$
$B' = B - D$
$G'' = G + H$
$P_7 = (A'' - C') \cdot (E' + F') = (A + D - C - D) \cdot (E + H + F - H) = (A - C) \cdot (E + F)$

Top Right = $AF + BH = P1' = P1 + P2$
Bottom Left = $CE + DG = P3' = P3 + P4$
Top Left = $AE + BG = P6 + P5'$, where $P5' = P5 + (P4 - P2)$
Bottom Right = $CF + DH = (P5' + P1') - (P7 + P3') = P5 + P4 - P2 + P1 + P2 - P3 - P4 - P7 = P5 + P1 - P3 - P7$

2.3 Findings

In general, the crossover point was not as clean as I had expected. After a lot of experiments and trials, I determined that the strassen's algorithm became almost always faster than traditional algorithm when between some value between 70 80. After looking through the data, I realized that 75 was the value after which strassen's algorithm was faster for the majority of the time. Thus, the crossover point $n0$ that I discovered was **75**.

As we can see from the results, the crossover point in practice is far larger than the crossover point theoretically (meaning the mixed Strassen's algorithm is not faster until we reach a higher value of n_0).Strassen's algorithm, although having much better time efficiency, still requires extra memory for all the recursive calls we are making. We also made our traditional matrix multiplication as efficient as possible which made the crossover point a little more blurry. This exercise reinforces the idea tht for smaller matrices, Strassen's algorithm may not be worth the extra memory and we are better off with the traditional optimized matrix multiplication algorithm

3 PART III - No. of triangles in random graphs

3.1 Creating a random graph

For representing a random graph, we created an adjacency matrix A with the same method as previous section. To include the element of randomness in selecting the edges, we used the Bernoulli Distribution. Given a probability value p between 0 and 1, we check if p is greater than a randomly generated value. We generate this value via the statement $((double)rand())/RANDMAX)$. For every edge, if the above observations turn out to be true, we assign element a_{ij} as 1, signifying an undirected edge between vertices i and j.

3.2 No. triangles vs expected no of triangles

For each value of p, I performed 10 trials of creating a random graph and calculating the no. of triangles present in the graph by calculating the A^3 matrix using our implementation of the Strassen's algorithm. I then take an average of the number of triangles we get from the 10 trials. Here are the findings :

Probability	Calculated Avg. Value	Expected Value(Rounded off)
0.01	178	178
0.02	1425	1427
0.03	4843	4817
0.04	11469	11419
0.05	22320	22304