Strassen Divide Conquer

Swati Goel[1] and Lev Kruglyak[2]

sgoel@college.harvard.edu[1], levkruglyak@college.harvard.edu[2]

1 Introduction

In this report, we would like to both analytically and experimentally determine

- The crossover point n_0 after which an optimized strassen algorithm is always more efficient than the standard matrix multiplication algorithm.

The optimized strassen algorithm performs strassen until some small m, at which point strassen becomes inefficent and we switch to the linear/standard algorithm.

1.1 Strassen Definition

To multiply two matrices P and Q, break each matrix into four submatrices like so:
$\begin{pmatrix} a & b \\ c & d \end{pmatrix} * \begin{pmatrix} e & f \\ g & h \end{pmatrix}$

Then perform the following 7 matrix multiplications (recursively using strassen).

$m1 = (a+d)(e+h)$
$m2 = (c+d)e$
$m3 = a(f-h)$
$m4 = d(g-e)$
$m5 = (a+b)h$
$m6 = (c-a)(e+g)$
$m7 = (b-d)(g+h)$

With these definitions, $PQ = M$ can be written:

$M = \begin{pmatrix} m1 + m4 - m5 + m7 & m3 + m5 \\ m2 + m4 & m1 - m2 + m3 + m6 \end{pmatrix}$

Now, we only need 7 multiplications of $n/2$ x $n/2$ matrices to calculate M (as opposed to 8)!

2 Analytical Approach

We determine that the runtime of strassen can be approximated by the recurrence equation $T(n) = 7T(n/2) + 18/4(n^2)$. We modify the algorithm such that all odd n are rounded up to be even. The updated recurrence is $T(n) = 7T(\lceil n/2 \rceil) + 18/4(n^2)$.

However, the problem statement asks us to compare optimized strassen to the standard algorithm. In optimized strassen, we switch to the standard algorithm for small n. Thus, the equation becomes $T(n) = 7\min(T(\lceil n/2 \rceil), \text{std alg for}\lceil n/2 \rceil) + 18/4(n^2)$.

The standard algorithm recurrence we find is $2n^3 - n^2$. Thus our final optimized strassen equation is:

$$T(n) = 7\min(T(\lceil n/2 \rceil), 2\lceil n/2 \rceil^3 - \lceil n/2 \rceil^2) + 18/4(n^2)$$

In the following subsection, we will explain how we determined the nonoptimized strassen and standard recurrence equations.

2.1 Finding the recurrence equations

The nonoptimized strassen recurrence has form $T(n) = 7T(n/2) + c(n/2)^2$, where c represents the number of additions of $n/2$ size matrices. There are 10 such additions needed to calculate $m1$ through $m7$ and 8 such additions needed to calculate M. Thus, $c = 18$ and the recurrence becomes $T(n) = 7T(n/2) + 18/4(n^2)$.

Similarly we see the standard matrix mult algorithm can be written as ((n mults + $n-1$ additions to multiply a row with a column)$*n$ columns)$*n$ rows. This simplifies to $2n^3 - n^2$ steps.

2.2 Solving the recurrence equations

We plug in inputs and binary search to find the crossover point. The table of our calculations is as follows:

Table 1. Crossover Points (Even vs Odd n)

i	strassen	standard
32	59,712	64,512
33	71,961	70,785
16	7,872	7,936
65	515,097	545,025
8	1,072	960
49	225,625	232,897
12	3,420	3,312
41	134,505	136,161
14	5,341	5,292
37	99,997	99,937
39	112,900	117,117

2.3 Numerical results

By calculating the strassen recurrence at every n from 1 to 256 and comparing to the standard algorithm runtime at that n, we determine a crossover point of 16 for even n and

39 for odd n. Figures 1 and 2 show our results. Strassen's algorithm (red) and the standard algorithm (blue) are plotted vs N.

Table 2. Crossover Points (Even vs Odd n)

i	strassen	standard
16	$7,872$	$7,936$
39	$112,900$	$117,117$

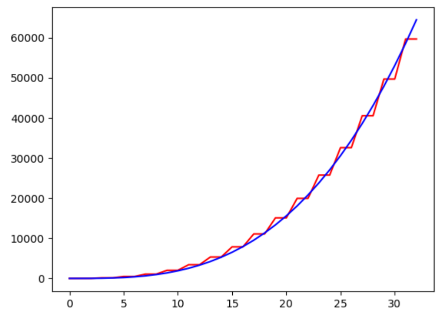

Figure 1. Strassen vs Standard (N=32)

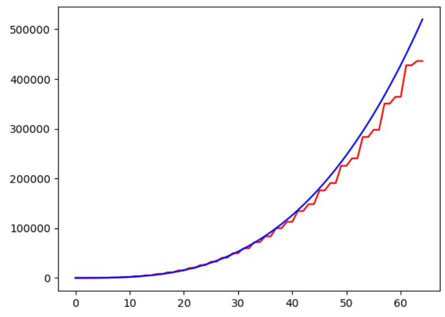

Figure 2. Strassen vs Standard (N=64)

3 Experimental Approach

Experimentally, we find $n_0 = 32$ to be the optimal crossover point.

3.1 Overview of the algorithm

We implement the optimized strassen algorithm with the following tricks for efficiency:

1. Recursive storage
2. Padding
3. Optimized cache reading/writing

Recursive storage allows us to ensure storage grows as some linear function of N, not more. We define four matrices, our inputs A and B, a matrix C for the answer, and a matrix D for scratch space. For each computation of a submatrix m_i, we store m_i in the top left quadrant of D. We store the two factor matrices in the bottom quandrants. This leaves one quadrant of D to be our scratch space matrix in the recursive step. Once m_i is calculated, we can add and subtract from the relevant quadrants of C and start anew with m_{i+1}.

Our padded matrix size = $k * 2^p$, where p is the smallest exponent such that $2^p * k > N$ and $k <$ cutoff= 32. We fill all pad space with 0s. We use this method because it simplifies the recursive step without hugely increasing runtime (as padding up to the nearest power of 2 would). We do not need to pad to the nearest power of 2 as we will switch to the standard algorithm at some point anyways, and thus we do not need to be able to divide N by 2 at every step. Our initial idea was to pad to the nearest even number at each recursive step, however this led to incorrect answers in some cases!

Optimized cache reading/writing simply means we are careful with the order of our for loops such that we access contiguous memory. Ex, we loop through matrices by row, and store them as one contiguous array.

The crossover point we experimentally determined was $n_0 = 32$, using randomized matrices of size $n = 1024$. Each matrix had entries of 0 or 1, with equal probability. We then tested over randomized matrices of increasing size.

4 Triangles

We use the same strassen procedure as in the previous section. We find the cube of a symmetric matrix of 0s and 1s, where each entry is a 1 with probability p. We find that our strassen results (red) are similar to our results in expectation (blue), as shown in Figure 3, which plots the number of triangles against p. We did not expect the results to match exactly, as randomness never matches expectation. The growth as a function of p is very similar, however, which indicates the validity of our approach.

5 Discussion

Our crossover point was low (32). This indicates the huge increase in efficiency between strassen and the standard multiplication algorithm. Some difficulties we ran into centered especially around efficient allocation of space for calculations in strassen and also an efficient padding scheme. Our initial padding scheme led to wrong answers in some cases! Ultimately, we multiplied matrices of even

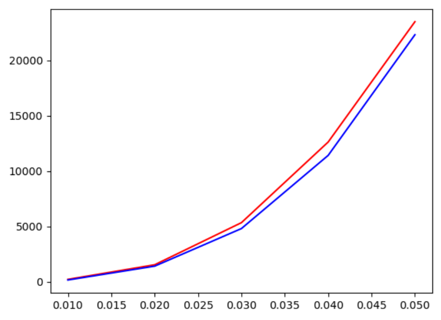

Figure 3. Triangles (Experimental vs Expectation)

dimension, though not quite powers of 2, which greatly simplified our recursive step.