STRASSEN'S ALGORITHM

MAXFIELD MA

1. Introduction

We currently live in a world full of data and to be able to use this data we must be able to analyze it. Matrices are a great way to store vast amounts of information neatly and concisely. Being able to perform operations on our data set allows us to manipulate and extract components we want to analyze. One of the most basic operations on matrices, matrix multiplication, is simple to perform but time consuming. Multiplying two $n \times n$ matrices requires a total of n^3 scalar multiplications and $n^3 - n^2$ scalar additions making the standard method of matrix multiplication have a time complexity of $O(n^3)$. As data sets only continue to grow in size, the time complexity starts to become a bottleneck to many algorithms. In 1969, Volker Strassen published a paper that demonstrated an innovative way to perform square matrix multiplication in $O(n^{2.81})$ time [2]. Over the years, his method and this area of research have continued to advance allowing us to perform faster matrix multiplication [1].

This paper takes an in depth look at the first of these algorithms, Strassen's Algorithm. The goal is to prove why this method of square matrix multiplication provides the correct result and why its runtime is faster than the standard method.

2. Strassen's Algorithm [1]

2.1. Implementation to $2^k \times 2^k$ matrices.

Strassen's Algorithm can only be directly applied to $n \times n$ matrices where n is a power of 2. There exists a fix for other n that we discuss later 2.2.

Let's start with the case $n = 2$. Let A, B, and C be 2×2 matrices defined by:

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}.$$

Date: 2025-11-10.

We first compute the follow values:

$$\begin{aligned}
x_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\
x_2 &= (a_{21} + a_{22})b_{11} \\
x_3 &= a_{11}(b_{12} - b_{22}) \\
x_4 &= a_{22}(-b_{11} + b_{21}) \\
x_5 &= (a_{11} + a_{12})b_{22} \\
x_6 &= (-a_{11} + a_{21})(b_{11} + b_{12}) \\
x_7 &= (a_{12} - a_{22})(b_{21} + b_{22}).
\end{aligned}$$

Notice that we only perform 7 scalar multiplications. Using these values, we can compute matrix C as follows (3.3 provides a proof):

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} x_1 + x_4 - x_5 + x_7 & x_3 + x_5 \\ x_2 + x_4 & x_1 + x_3 - x_2 + x_6 \end{pmatrix}.$$

Example 2.1. *Let* $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ *and* $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$. *We can use this method to compute* AB.

$$\begin{aligned}
x_1 &= (1 + 4)(5 + 8) = 65 \\
x_2 &= (3 + 4)5 = 35 \\
x_3 &= 1(6 - 8) = -2 \\
x_4 &= 4(-5 + 7) = 8 \\
x_5 &= (1 + 2)8 = 24 \\
x_6 &= (-1 + 3)(5 + 6) = 22 \\
x_7 &= (2 - 4)(7 + 8) = -30.
\end{aligned}$$

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} 65 + 8 - 24 - 30 & -2 + 24 \\ 35 + 8 & 65 - 2 - 35 + 22 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

One can quickly verify that this does provide the correct answer.

This algorithm can be generalized to larger matrices through recursion. Notice that each of our entries in C were obtained through addition and multiplication where we did not need to use the fact that scalar multiplication is commutative. Thus, Strassen's algorithm works over rings which means our entries of B and C, (a_{ij}, b_{ij}), could also be matrices. Consider 4×4 matrices X, Y, and Z such that $Z = XY$. Let us denote the entries of X, Y, and Z in block form:

$$Z = \begin{pmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{pmatrix}, X = \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix}, Y = \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix}$$

The entries Z_{ij}, X_{ij}, and Y_{ij} are themselves 2×2 matrices. Using matrix addition and matrix multiplication (2×2 matrix multiplication using Strassen's algorithm), we can compute values x_{1-7} and therefore the entries of Z.

2.2. **Padding.** Since Strassen's algorithm provides both a correct procedure for multiplying 2×2 matrices and a recursive method for multiplying larger matrices, it follows that the algorithm applies directly to all $n \times n$ matrices where n is a power of two. To apply the method to arbitrary dimensions $n \in \mathbb{Z}^+$, we can pad the input matrices with rows and columns of zeros until their size reaches the next power of two. This padding step does not change the value of the final matrix product, and the additional zero rows and columns can be removed after the computation. For example, below is the 3×3 matrix A padding to 4×4 matrix $\hat{A}$:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \to \hat{A} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

3. Analysis of Strassen's Algorithm

3.1. **Correctness of Strassen's Algorithm.** My proof of the correctness of Strassen's Algorithm.

Proposition 3.1 (Block Addition)**.** *Given $n \times n$ matrices A, B, and C such that $C = A + B$, then $\begin{pmatrix} C & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} B & 0 \\ 0 & 0 \end{pmatrix}$ also holds.*

Proof. Let A', B', and C' be our augmented matrices, where

$$A' = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix}, \quad B' = \begin{pmatrix} B & 0 \\ 0 & 0 \end{pmatrix}, \quad C' = \begin{pmatrix} C & 0 \\ 0 & 0 \end{pmatrix}.$$

Note that A', B', and C' are $(n+1) \times (n+1)$ matrices. We claim that

$$C' = A' + B'.$$

For $1 \leq i, j \leq n+1$, matrix addition is entrywise, so

$$(A' + B')_{ij} = A'_{ij} + B'_{ij}.$$

If $1 \leq i, j \leq n$, then by construction $A'_{ij} = A_{ij}$ and $B'_{ij} = B_{ij}$, hence

$$(A' + B')_{ij} = A_{ij} + B_{ij} = C_{ij} = C'_{ij}.$$

If $i = n+1$ or $j = n+1$, then $A'_{ij} = 0$ and $B'_{ij} = 0$ (since the last row and last column of each augmented matrix are zero), so

$$(A' + B')_{ij} = 0 = C'_{ij}.$$

Therefore $(A' + B')_{ij} = C'_{ij}$ for all $1 \leq i, j \leq n+1$, which implies

$$\begin{pmatrix} C & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} B & 0 \\ 0 & 0 \end{pmatrix}.$$

□

Proposition 3.2 (Block Multiplication)**.** *Given $n \times n$ matrices A, B, and C such that $C = AB$, then $\begin{pmatrix} C & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} B & 0 \\ 0 & 0 \end{pmatrix}$ also holds.*

Proof. Let A', B', and C' be our augmented matrices. Note that A', B', and C' are $(n+1) \times (n+1)$ matrices. For $1 \leq i, j \leq n+1$, the product $C' = A'B'$ satisfies

$$C'_{ij} = \sum_{k=1}^{n+1} A'_{ik}\, B'_{kj}.$$

If $1 \leq i, j \leq n$, then $A'_{ik} = A_{ik}$ and $B'_{kj} = B_{kj}$ for $1 \leq k \leq n$, and $A'_{ik} = B'_{kj} = 0$ otherwise. Hence

$$C'_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} = C_{ij}.$$

If $i = n+1$, the entire i-th row of A' is 0, so $C'_{ij} = 0$ for all j. If $j > n$, the j-th column of B' is 0, so $C'_{ij} = 0$ for all i. Thus C' is equal to

$$C' = \begin{pmatrix} C & 0 \\ 0 & 0 \end{pmatrix},$$

which proves the proposition. □

Lemma 3.3. *Strassen's algorithm holds for 2×2 matrices.*

Proof. Let $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ and $B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$. We compute AB via Strassen's Algorithm, then simplify the result.

$$\begin{aligned} x_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ x_2 &= (a_{21} + a_{22})b_{11} \\ x_3 &= a_{11}(b_{12} - b_{22}) \\ x_4 &= a_{22}(-b_{11} + b_{21}) \\ x_5 &= (a_{11} + a_{12})b_{22} \\ x_6 &= (-a_{11} + a_{21})(b_{11} + b_{12}) \\ x_7 &= (a_{12} - a_{22})(b_{21} + b_{22}). \end{aligned}$$

$$AB = \begin{pmatrix} x_1 + x_4 - x_5 + x_7 & x_3 + x_5 \\ x_2 + x_4 & x_1 + x_3 - x_2 + x_6 \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}.$$

We can see that Strassen's algorithm calculates the dot products between the rows of A and columns of B just like standard matrix multiplication. Thus, the algorithm holds for 2×2 matrices. □

Proposition 3.4 (Block Multiplication)**.** *For any matrices, A and B, of size $2^k \times 2^k$ we can divide A and B into four smaller matrices, each of size $n \times n$:*

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

Then the product AB can be calculated recursively as

$$AB = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}.$$

Proof. We compute the product AB entrywise. For $1 \leq i, j \leq 2n$,

$$(AB)_{ij} = \sum_{\ell=1}^{2n} a_{i\ell} b_{\ell j}.$$

Consider four cases depending on whether i and j lie in the first or second block of indices.

Case 1: $1 \leq i, j \leq n$ **(upper–left block).** Then the sum splits into two parts:

$$(AB)_{ij} = \sum_{\ell=1}^{n} a_{i\ell} b_{\ell j} + \sum_{\ell=n+1}^{2n} a_{i\ell} b_{\ell j}.$$

The first sum equals $(A_{11}B_{11})_{ij}$, and the second equals $(A_{12}B_{21})_{ij}$. Thus

$$(AB)_{ij} = (A_{11}B_{11} + A_{12}B_{21})_{ij}.$$

Case 2: $1 \leq i \leq n$ **and** $n < j \leq 2n$ **(upper–right block).** A similar decomposition yields

$$(AB)_{ij} = (A_{11}B_{12} + A_{12}B_{22})_{ij}.$$

Case 3: $n < i \leq 2n$ **and** $1 \leq j \leq n$ **(lower–left block).** Again,

$$(AB)_{ij} = (A_{21}B_{11} + A_{22}B_{21})_{ij}.$$

Case 4: $n < i, j \leq 2n$ **(lower–right block).** We obtain

$$(AB)_{ij} = (A_{21}B_{12} + A_{22}B_{22})_{ij}.$$

Since matrix addition and multiplication in each block use only the ring operations, all computations are valid. We can combine all four cases to reconstruct the product matrix. □

Theorem 3.5. *Strassen's algorithm correctly computes AB for any $n \times n$ matrices A and B.*

Proof. By Lemma 3.3, Strassen's algorithm is correct for the base case of 2×2 matrices. By Theorem 3.4, the recursive block formulation of matrix multiplication is correct: if the algorithm correctly multiplies $(m/2) \times (m/2)$ matrices, then it correctly multiplies $m \times m$ matrices.

Combining these statements, Strassen's algorithm is correct for all $m \times m$ matrices with m a power of 2, by induction on m using Lemma 3.3 and Theorem 3.4. Since any $n \times n$ matrices can be padded to such an $m \times m$ size without affecting the final product (by Theorem 3.1 and Theorem 3.2), the algorithm also computes AB correctly for arbitrary $n \times n$ matrices. □

3.2. **Time Complexity of Strassen's Algorithm.** From Section 2.1, we can see that in the 2×2 core, Strassen's Algorithm computes 7 products and 18 additions. To be more specific, when it is applied recursively, the algorithm computes 7 $(n/2 \times n/2)$ matrix products and 18 $(n/2 \times n/2)$ matrix additions. We can analyze the time complexity of Strassen's Algorithm using the Master Theorem [4]. For a problem of size n, we must solve 7 smaller problems of size $n/2$ in addition to performing 18 additions. Notice that when adding matrices, we simply add each element to its corresponding element in the 2nd matrix. As there are $\left(\frac{n}{2}\right)^2$ elements in each sub-matrix, we perform a total of $O(n^2)$ scalar additions. Let $T(n)$ denote the time complexity of a problem of size n. We can write:

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2).$$

By applying Master Theorem [4] we get the runtime of our algorithm to be $O(n^{\log_2 7}) \approx O(n^{2.81})$.

4. Practical Performance of Strassen's Algorithm

Matrix multiplication is commonly found in scientific computing, computer graphics, and machine learning, so even a minor improvement in matrix multiplication can have a huge impact. Asymptotically, Strassen's algorithm improves the running time of multiplying two $n \times n$ matrices from $O(n^3)$ to $O(n^{\log_2 7}) \approx O(n^{2.81})$. However, big-$O$ notation hides constant factors. In practice, the recursive form of Strassen's algorithm performs surprisingly poorly on realistic matrix sizes.

4.1. **Why Pure Strassen is Slow in Practice.** For very small matrices, the disadvantage of Strassen is obvious. When $n = 2$,

- the standard dot-product formulation uses 8 scalar multiplications and 4 additions (12 scalar operations), while
- Strassen's scheme uses 7 scalar multiplications but requires 18 additions and subtractions (25 scalar operations).

Thus on the level of scalar operations, Strassen has a significantly larger constant factor.

To demonstrate this effect, I implemented both algorithms in C[1] (optimizing the code to the best of my ability). For each size n, we generate random $n \times n$ matrices, pad them up to the next power of two (n_{pad}), and then benchmark both standard and Strassen multiplication on the same $n_{\text{pad}} \times n_{\text{pad}}$ matrices over 100 trials[2]. Table 1 summarizes the results for several sizes.

n	n_{pad}	Standard avg (s)	Pure Strassen avg (s)	Factor
32	32	0.000069	0.000465	6.74
64	64	0.000556	0.003201	5.76
128	128	0.004267	0.022708	5.32
256	256	0.034772	0.160049	4.60
512	512	0.277440	1.122273	4.04
1000	1024	2.203858	7.839432	3.56

TABLE 1. Average running time of standard and pure Strassen multiplication over 100 trials. Factor indicates slowdown relative to standard multiplication.

I will refer to Strassen multiplication as pure strassen in contrast to hybrid strassen, which is introduced in the next section. Even for $n = 1000$ (padded to 1024), pure Strassen is about 3.6 times slower than the standard algorithm, with an average speedup of roughly -256% (Strassen takes about 2.56 times longer than standard). This is roughly on par with what the theoretical constants suggest: while Strassen saves one matrix multiplication per recursion level (7 instead of 8), it introduces a large number of extra matrix additions and subtractions.

The small-n experiments make this extremely clear. For $n = 32$ and $n = 64$, Strassen is between roughly 5 times and 7 times slower than the standard method. For these dimensions, the asymptotic advantage of $O(n^{2.81})$ is completely overwhelmed by constant factors.

4.2. **Hybrid Strassen: Combining Recursive and Standard Multiplication [3].** In practice, high-performance implementations do not use pure Strassen. Intuitively, Strassen is profitable only once submatrices are large enough that saving one multiplication per level outweighs the cost of all the extra additions and subtractions. Below that scale, it is often better

[1]I originally attempted this experiment in python but python is not very recursion friendly. C provided better times overall, not to mention the further optimizations that could be made regarding dynamic memory and pointers. My code can be found here: https://github.com/mma2027/strassen

[2]All data was collected on my computer, runtime may be drastically different on different computers but will most likely follow the same trend.

to perform standard matrix multiplication. Even on a theoretical level, it is apparent that for smaller dimensions, standard matrix multiplication runs faster. On a practical level, the standard method runs even faster compared to Strassen, because Strassen uses recursion which is resource intensive. It repeatedly splits matrices into four blocks, performs computations and then reassembles the original matrix. This requires a computer to create many different matrices and manipulate them which requires extra time and memory, both of which affect the duration the algorithm takes to physically run. The standard method eliminates all of these factors by avoiding recursion and only modifying one matrix. Therefore, Strassen is only advantageous when time reduction is able to outweigh both the theoretical runtime advantage and the practical runtime advantage of the standard method.

This observation—that Strassen's asymptotic improvement only becomes beneficial beyond a certain problem size and that a cutoff should be used in practice—is a central conclusion of Huss-Lederman et al. [3].

However, we can design a combination of standard methods of matrix multiplications with Strassen to form a hybrid method [5]:

> Recurse using Strassen while n is above some threshold T, and once $n \leq T$, switch to the standard $O(n^3)$ matrix multiplication on the subblocks.

To find a good threshold T on my machine, I implemented a trial and error autotuning algorithm. For a fixed matrix size n, we:

(1) pad to n_{pad},
(2) fix random matrices A and B,
(3) try hybrid Strassen with thresholds $T \in \{1, 2, 4, 8, \ldots, n_{\text{pad}}\}$,
(4) measure the average time for each T, and
(5) select the threshold with the smallest average time.

Table 2 shows the autotuned thresholds and the resulting speedups of hybrid Strassen over standard multiplication for three representative sizes.

n	n_{pad}	Best T	Standard avg (s)	Hybrid Strassen avg (s)	Speedup
32	32	32	0.000069	0.000050	1.38
64	64	32	0.000556	0.000377	1.47
128	128	32	0.004267	0.002413	1.77
256	256	32	0.034258	0.018326	1.87
512	512	16	0.278633	0.135234	2.06
1000	1024	32	2.211756	0.922053	2.40

TABLE 2. Autotuned hybrid Strassen thresholds and resulting average times over 100 trials. Speedup is relative to standard multiplication.

These results illustrate two important points:

- **Pure Strassen is a bad idea.** Recursing all the way down (threshold $T = 1$) is consistently slower than standard multiplication, often by large factors, across all tested sizes. The extra recursive structure, temporary matrices, and additions surpass the asymptotic savings.
- **Hybrid Strassen is extremely effective with an appropriate threshold.** Once we stop recursing when submatrices reach a moderate size (here between 16 and 32), Strassen's asymptotic advantage can really shine. The upper levels of the recursion tree reduce the number of large matrix multiplications, while the lower levels are handled by the more efficient standard method.

From an applications perspective, this means Strassen's algorithm should not be viewed as a substitution for standard matrix multiplication, but as a tool for reducing the cost of large matrix multiplications.

References

[1] Markus Bläser. "Fast Matrix Multiplication". In: *Theory of Computing Library — Graduate Surveys* 5 (2013), pp. 1–60. DOI: 10.4086/toc.gs.2013.005.

[2] Shmuel Friedland. "Strassen's Algorithm and the Asymptotic Complexity of Matrix Multiplication". In: *The American Mathematical Monthly* 102.10 (1995), pp. 894–903. DOI: 10.1080/00029890.1995.12004600.

[3] Steven Huss-Lederman et al. "Implementation of Strassen's Algorithm for Matrix Multiplication". In: *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing (Supercomputing '96)*. ACM/IEEE, 1996. DOI: 10.1145/369028.369096.

[4] Salvador Roura. "Improved Master Theorems for Divide-and-Conquer Recurrences". In: *Journal of the ACM* 48.2 (2001). Free PDF: https://www.cs.upc.edu/~diaz/RouraMT.pdf (accessed 2025-12-18), pp. 170–205. DOI: 10.1145/375827.375837.

[5] Lorenzo De Stefani. "The I/O complexity of hybrid algorithms for square matrix multiplication". In: *arXiv preprint arXiv:1904.12804* (2019). Preprint.