Alternatives to the Naive Algorithm for Matrix Multiplication

Strassen's, Triangular Matrices, and Inversion

Jack Ruder

Semester Project for
Advanced Linear Algebra (Math 390)

University of Puget Sound
Tacoma, WA
April 25, 2021

Alternatives to the Naive Algorithm for Matrix Multiplication

Jack Ruder

Introduction

Matrix operations are extremely important computations used in different fields of computing, notably in fields such as graphics and optimization. For this reason many look to speed up matrix operations in order to save computing resources in all types of programs. One of the most common computations performed is matrix multiplication, a computation that runs in $O(n^3)$ time. Cubic time is undesirable, and is enough of a motivation to seek an improvement. This paper will show that through Strassen's algorithm this can be slightly improved, and that the algorithm's implementation plays a large role in its effectiveness. We will also look at some of the other modern algorithms to gain a larger picture of the available options.

This paper will also examine the special case of triangular matrices, and how Strassen's algorithm can affect their multiplication. It is known that the multiplicaion of triangular matrices occurs quicker than full matrices since the existence of zeroes under/over the diagonals in both factors will guarantee zeroes under/over the diagonals in the product. We can extend this idea to Strassen's multiplication algorithm, and allow us to perform fewer recursions in Strassen's, which will potentially lead to speedups vesus the naive triangular multiplication algorithm.

Next, this paper will examine one of the implications of reducing the complexity of matrix multiplication. Solving a system of linear equations is a very common task, and is often performed by computing an inverse. Inverting a matrix is typically a slow task, and in programs that rely on solving linear systems of equations, speeding up inversion is a useful thing. As we will show in the paper, speedups in matrix multiplication translate directly to speedups in inversion.

Lastly, we will briefly visit some of the other algorithms for matrix multiplication, and explain their relevance in the real world.

1 The Naive Method

The naive method for matrix multiplication relies on calculating each entry independently. For the multiplication $C = AB$ where A is an $m \times n$ matrix and B is an $n \times p$ matrix, the entries in C may be defined as

$$[C]_{ik} = \sum_{j=1}^{n} [A]_{ij} [B]_{jk}.$$

Using this method, for every entry in C, n multiplications are needed, and there are $m \times p$ entries in C, yielding a total of mnp multiplications. Similarly, $n-1$ additions are needed per entry, for a total of $m(n-1)p$ additions for the entire matrix. For square matrices of size n, we can write the number of multiplications as n^3 and the number of additions as $n^3 - n^2$. This yields a total of $2n^3 - n^2$ arithmetic operations.

Another method that will act as a warm up to Strassen's is to compute the product by factoring the matrix into blocks, and recurse down until the blocks degrade to scalars. We have

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

We see that 8 multiplications and 4 matrix additions are needed at each recursion, and for a matrix of size n, $\log_2 n$ recursions are needed. This allows us to write the number of aritmetic operations as

$$\begin{aligned} M(2n_{\text{min}}) &= 8M(n_{\text{min}}) + 4n_{\text{min}}^2 \\ M(4n_{\text{min}}) &= 8M(2n_{\text{min}}) + 4(2n_{\text{min}})^2 \\ &= 8^2 M(n_{\text{min}}) + (4+8)4n_{\text{min}}^2 \\ M(8n_{\text{min}}) &= 8^3 M(n_{\text{min}}) + (4^2 + 4\cdot 8 + 8^2)4n_{\text{min}}^2 \\ M(16n_{\text{min}}) &= 8^i M(n_{\text{min}}) + (4^3 + 4^2\cdot 8 + 4\cdot 8^2 + 8^3)4n_{\text{min}}^2 \\ M(2^i n) &= 8^i M(n_{\text{min}}) + (4^{i-1} + 4^{i-2}\cdot 8 + \cdots + 4\cdot 8^{i-2} + 8^{i-1})4n_{\text{min}}^2 \\ &= 8^i M(n_{\text{min}}) + 8^{i-1}(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{i-1}})4n_{\text{min}}^2 \\ &= 8^i M(n_{\text{min}}) + 8^i(\frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots + \frac{1}{2^{i-4}})4n_{\text{min}}^2 \\ &= 8^i M(n_{\text{min}}) + 8^i \left(\frac{1}{8}(2 - \frac{1}{2^{i-1}})\right) 4n_{\text{min}}^2. \end{aligned}$$

We want to manually compute the products when we reach a block size of 2. Using $i = \log_2 n - \log_2 n_{\text{min}}$,

$$\begin{aligned} M(n) &= \frac{8^{\log_2 n}}{8^{\log_2 2}} M(2) + \frac{8^{\log_2 n}}{8^{\log_2 2}} \left(\frac{1}{8}\left(2 - \frac{1}{\frac{2^{\log_2 n}}{2^{\log_2 2 + 1}}}\right)\right) 4\cdot 2^2 \\ &= n^{\log_2 8} \frac{12 + 4 - \frac{8}{n}}{8} \\ &= 2n^3 - n^2, \end{aligned}$$

where $M(n)$ is the number of arithmetic operations to compute the product for matrices of size n. We recognize that this is the same number of operations as in the naive method, meaning the product of matrices of size $2^q \times 2^q$ can be computed in the same time.

2 Strassen's

Similarly to block matrix multiplication, Strassen's algorithm is a recursive algorithm that relies on partitioning $2n \times 2n$ matrices into $n \times n$ blocks. The expression $AB = C$ when partitioned is written as

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

To begin, Strassen's algortihm [GL13] recursively computes 7 multiplications and 10 additions to obtain

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) & M_2 &= (A_{21} + A_{22})B_{11} \\ M_3 &= A_{11}(B_{12} - B_{22}) & M_4 &= A_{22}(B_{21} - B_{11}) \\ M_5 &= (A_{11} + A_{12})B_{22} & M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}). \end{aligned}$$

There are then 8 more additions (for a total of 18 additions) required to obtain C as

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7 & C_{12} &= M_3 + M_5 \\ C_{21} &= M_2 + M_4 & C_{22} &= M_1 + M_3 - M_2 + M_6. \end{aligned}$$

It is difficult to understand the motivation of Strassen's proposed algorithm, however it is easy to verify that it is correct. We have

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7 \\ &= (A_{11} + A_{22})(B_{11} + B_{22}) + A_{22}(B_{21} - B_{11}) - (A_{11} + A_{12})B_{22} + (A_{12} - A_{22})(B_{21} + B_{22}) \\ &= A_{11}B_{11} + A_{12}B_{21}, \\ C_{12} &= M_3 + M_5 \\ &= A_{11}(B_{12} - B_{22}) + (A_{11} + A_{12})B_{22} \\ &= A_{11}B_{12} + A_{12}B_{22}, \\ C_{21} &= M_2 + M_4 \\ &= (A_{21} + A_{22})B_{11} + A_{22}(B_{21} - B_{11}) \\ &= A_{21}B_{11} + A_{22}B_{21}, \\ C_{22} &= M_1 + M_3 - M_2 + M_6 \\ &= (A_{11} + A_{22})(B_{11} + B_{22}) + A_{11}(B_{12} - B_{22}) - (A_{21} + A_{22})B_{11} + (A_{21} - A_{11})(B_{11} + B_{12}) \\ &= A_{21}B_{12} + A_{22}B_{22}, \end{aligned}$$

and find that we end up with our definition of block matrix multiplication, verifying that Strassen's algorithm is accurate.

In the basic implementation of Strassen's algorithm, the factors are assumed to be of size $2n$. The algorithm recurses until the size of the blocks is $n_{\min}$ or smaller, and then computes the multiplication of the blocks using another method. Similarly to naive block multiplication, the total number of arithmetic operations can then be written as

$$\begin{aligned}
M(2n) &= 7M(n) + 18n^2 \\
M(4n) &= 7^2M(n) + (4+7)\cdot 18n^2 \\
M(2^i n_{\min}) &= 7^i M(n_{\min})(4^{i-1} + 4^{i-2}\cdot 7 + \cdots + 4\cdot 7^{i-2} + 7^{i-1})18n_{\min}^2 \\
&= 7^i M(n_{\min}) + 7^{i-1}\left(\sum_{k=1}^{i}\left(\frac{4}{7}\right)^{k-1}\right)18n_{\min}^2 \\
&= 7^i M(n) + 7^{i-1}\left(\frac{1-\left(\frac{4}{7}\right)^i}{1-\frac{4}{7}}\right)18n_{\min}^2
\end{aligned}$$

where $M(n)$ is the number of arithmetic operations needed by Strassen's algorithm to multiply an $n \times n$ matrix [Bar06]. For large $n = 2^i n_{\min}$ we can further generalize $M(n)$ as

$$\begin{aligned}
M(2^i n_{\min}) &\approx 7^i M(n_{\min}) + 7^i 6n_{\min}^2 \\
M(n) &\approx \frac{7^{\log_2 n}}{7^{\log_2 n_{\min}}} M(n_{\min}) + \frac{7^{\log_2 n}}{7^{\log_2 n_{\min}}} 6n_{\min}^2 \\
M(n) &\approx n^{\log_2 7}\frac{M(n_{\min}) + 6n_{\min}^2}{n_{\min}^{\log_2 7}}.
\end{aligned}$$

If using the naive method for matrix multiplication, then Strassen's will require

$$M(n) \approx n^{\log_2 7}\frac{n_{\min}^3 - n_{\min}^2 + 6n_{\min}^2}{n_{\min}^{\log_2 7}}.$$

Here it can be seen that the value of $n_{\min}$ is important when implementing Strassen's. If we compare

$$n^{\log_2 7}\frac{n_{\min}^3 - n_{\min}^2 + 6n_{\min}^2}{n_{\min}^{\log_2 7}} = 2n^3 - n^2,$$

we can find that Strassen's only becomes faster at $n_{\min} = 40$ [Bar06], although here we only worry about square matrices whose sizes are powers of 2.

Regardless, when we make n very large, asymptomatically the algorithm will run in $O(n^{\log_2 7}) \approx O(n^{2.807})$.

While this algorithm may be asymtomatically faster, even with an optimized $n_{\min}$ issues with memory hierarchies in modern computers result in the algorithm only being effective for matrices of larger sizes. This is primarily due to the fact that in memory it is difficult to store matrices in block form, opposed to column-major or row-major orders. Storing in block form makes the matrices more difficult to keep in contiguous memory, meaning read and write times can take longer. Experimentally, it has been found that $n_{\min}$ on most modern systems is typically somewhere around 1200, a size which is rarely practical [DN]. Certain implementations however do take advantage of the occasional specialized hardware to achieve speedups and beat these caching issues.

3 Triangular Matrices

Upper triangular matrix multiplication–without loss of generality–under the naive method may be modifed to compute the product in $\frac{n^3}{3}$ time, by ignoring the computations underneath the diagonals [GL13]. Instead, we may write upper triangular matrices of size $2n$ in block form as

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix},$$

where A_{11} and A_{22} are restricted to being upper triangular and each block is of size n. The naive way to multiply triangular matrices of size 2^n with eachother would be to use the naive method of block matrix multiplication to obtain

$$\begin{aligned}\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}\begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix} &= \begin{bmatrix} A_{11}B_{11} + A_{12}0 & A_{11}B_{12} + A_{12}B_{22} \\ 0B_{11} + A_{22}0 & 0B_{12} + A_{22}B_{22} \end{bmatrix} \\ &= \begin{bmatrix} A_{11}B_{11} & A_{11}B_{12} + A_{12}B_{22} \\ 0 & A_{22}B_{22} \end{bmatrix}.\end{aligned}$$

Since A_{11} and B_{11} are upper-triangular, their product $A_{11}B_{11}$ is upper-triangular and similarly $A_{22}B_{22}$ is upper triangular. We may then recurse with block triangular multiplication on the top left and bottom right blocks, and use the normal form of naive matrix multiplication to recurse on the top-right. For the triangular recursion, when the blocks degrade to scalars it is enough to perform only the 4 multiplications and 1 addition since there is a 0 in the matrix.

The total number of arithmetic operations then becomes

$$\begin{aligned} T(2n) &= 2T(n) + 2M(n) + n^2 \\ T(4n) &= 2\left(2T(n) + 2M(n) + n^2\right) + 2(M(2n)) + (2n)^2 \\ &= 2^2T(n) + (4 + 2\cdot 8)M(n) + (2\cdot 4 + 2^2)n^2 \\ T(8n) &= 2\left(2^2T(n) + (4 + 2\cdot 8)M(n) + (2\cdot 4 + 2^2)n^2\right) + 2(8^2M(n) + (8+4)4n^2) + (4n)^2 \\ &= 2^3T(n) + (2\cdot 4 + 2\cdot 16 + 2\cdot 64)M(n) + (2^7 + 2^3)n^2 \end{aligned}$$

where $T(n)$ is the number of arithmetic operations needed by a triangular multiplication and $M(n)$ is the the number of arithmetic operations needed by conventional naive matrix multiplication. We can substitute $2n^3 + n^2$ for $M(n)$ and ignore the n^2 to obtain

$$\begin{aligned} T(2^i n_{\min}) &\approx 2^i T(n_{\min}) \left(\sum_{k=1}^{i} 2^{i+2(k-1)}\right)(2n_{\min}^3 + n_{\min}^2). \\ &\approx 2^i T(n_{\min}) + \left(\frac{2^{3i} - 2^i}{3}\right)(2n_{\min}^3) \\ T(n) &\approx \frac{2^{\log_2 n}}{2^{\log_2 2}}T(2) + \frac{16\left(2^{3\log_2 n - 3\log_2 2} - 2^{\log_2 n - \log_2 2}\right)}{3}. \end{aligned}$$

Triangular Strassen's

Using Strassen's on triangular matrix multiplication does not give a speedup. We substitute the 0 matrix for A_{21} and B_{21} to obtain

$$\begin{aligned}
M_1 &= (A_{11}+A_{22})(B_{11}+B_{22}) & M_2 &= A_{22}B_{11} \\
M_3 &= A_{11}(B_{12}-B_{22}) & M_4 &= A_{22}(-B_{11}) \\
M_5 &= (A_{11}+A_{12})B_{22} & M_6 &= (-A_{11})(B_{11}+B_{12}) \\
M_7 &= (A_{12}-A_{22})B_{22}.
\end{aligned}$$

Observe that M_2 and M_4 are additive inverses, meaning we only need M_2 and can flip the sign everywhere M_4 is needed. Continuing with Strassen's gives

$$\begin{aligned}
M_8 &= M_1 - M_2 \\
C_{11} &= M_8 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= 0 \\
C_{22} &= M_8 + M_3 + M_6.
\end{aligned}$$

Strassen's here is clearly at a disadvantage. For every recursion 3 triangular additions are needed, 2 triangular multiplications are needed, however 4 normal multiplications and 9 normal additions are needed. In the triangular case, it will be quicker just to use the naive method of matrix multiplication.

4 Inversion Using Strassen's

In addition to providing a way to compute matrix multiplication with fewer multiplications Strassen showed that the inverse of a matrix could be computed recursively for a cost of $\leq 5.64n^{\log_2 7}$ arithmetic operations[BH74]. Most importantly, the cost of Strassen's inversion is the same as the cost for multiplication.

The inversion relies on a block LDU decomposition to compute the inverse. The LDU decomposition is

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & \Delta \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}$$

where $\Delta = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is the Schur complement of A_{11}, given that A_{11} and Δ are nonsingular. Then,

$$\begin{aligned}
A^{-1} &= \begin{bmatrix} I & -A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & \Delta^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \\
&= \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}\Delta^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}\Delta^{-1} \\ -\Delta^{-1}A_{21}A_{11}^{-1} & \Delta^{-1} \end{bmatrix}
\end{aligned}$$

To verify that our inverse is correct we perform the computation

$$\begin{aligned}
AA^{-1} &= \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & \Delta \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} I & -A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & \Delta^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \\
&= \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & \Delta \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & \Delta^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \\
&= \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \\
&= \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \\
&= I,
\end{aligned}$$

and see that A^{-1} is in fact the inverse of A. Strassen's modification may now be used to reduce the needed multiplications [PS09] to allow the inversion to be calculated quicker than $O(n^3)$. Similarly to Strassen's algorithm for multiplication the inversion requires the use of intermediate matrices. In order, the calculations required for one recursion are

1. $M_1 = A_{11}^{-1}$
2. $M_2 = A_{21}M_1$
3. $M_3 = M_1A_{12}$
4. $M_4 = A_{21}M_3$
5. $M_5 = M_4 - A_{22}$
6. $M_6 = M_5^{-1}$
7. $(A^{-1})_{12} = M_3M_6$
8. $(A^{-1})_{21} = M_6M_2$
9. $M_7 = M_3(A^{-1})_{21}$
10. $(A^{-1})_{11} = M_1 - M_7$
11. $(A^{-1})_{22} = -M_6$.

Subsequent recursions down to a block size of 1 are needed in steps 1. and 6., since those are the only steps that require an inversion. Otherwise, 6 multiplications and 3 additions are needed. In terms of complexity, we observe that the most complex operation is matrix multiplication, thus the complexity of the inversion will be approximately the complexity of the matrix multiplication. For example, using Strassen's multiplication algorithm for the multiplications would give a complexity of $O^{\log_2 7}$ for calculating the inverse of a large enough matrix.

5 Coppersmith-Winograd and Derivatives

The Coppersmith-Winograd algorithm is another algorithm targeted at reducing the complexity of matrix multiplication. The algorithm relies on checking the validity of a certain matrix multiplication using properties of tensor products, and achieves a complexity of $O^{2.496}$ [Wil14]. Multiple improvements have been made to the algorithm since, and have improved the theoretical complexity to as low as $\approx O^{2.37}$ [Wil14]. These algorithms are plagued however by worse caching issues and are even more difficult to implement on modern hardware, so they have no real use. Even the asymptomatic complexity shows that it is not practical to multiply matrices of under size 1000 in many cases.

6 Conclusion

The algorithms presented by Strassen and others clearly provide significant asymptomatic speedups when compared to typical naive matrix multiplication. We have also seen through the work of Strassen that matrix inversion can occur in $O(mul(n))$ time, providing significant speedup when solving linear systems of a large size. This is one of the crucial reasons to speed up matrix multiplication; it is a good idea to create implementations of matrix multiplication that run quickly when dealing with large dense linear systems. Unfortunately in triangular matrix multiplication Strassen's falls behind, and given the usefulness of a single tool to perform matrix multiplication it is clear why the naive method is the most commonly used metod of matrix multiplication.

References

[Bar06] Gregory V. Bard. "Achieving a log(n) Speed Up for Boolean Matrix Operations and Calculating The Complexity of the Dense Linear Algebra step of Algebraic Stream Ciper Attacks and of Integer Factorization Methods". In: *IACR E-PRINT* 2006 (2006), p. 163.

[BH74] James R. Bunch and John E. Hopcroft. "Triangular Factorization and Inversion by Fast Matrix Multiplication". In: *Mathematics of Computation* 28.125 (1974), pp. 231–236. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2005828.

[DN] Paolo D'alberto and Alexandru Nicolau. "Using Recursion to Boost ATLAS's Performance". In: *Lecture Notes in Computer Science* 4759 (). DOI: https://doi.org/10.1007/978-3-540-77704-5_12.

[GL13] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Baltimore, Maryland: The Johns Hopkins University Press, 2013. ISBN: 978-1421407944.

[Hua+16] Jianyu Huang et al. "Strassen's Algorithm Reloaded". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '16. Salt Lake City, Utah: IEEE Press, 2016. ISBN: 9781467388153.

[PS09] Marko D. Petković and Predrag S. Stanimirović. "Generalized matrix inversion is not harder than matrix multiplication". In: *Journal of Computational and Applied Mathematics* 230.1 (2009), pp. 270–282. ISSN: 0377-0427. DOI: https://doi.org/10.1016/j.cam.2008.11.012. URL: https://www.sciencedirect.com/science/article/pii/S0377042708006237.

[Wil14] Virginia Williams. "Breaking the Coppersmith-Winograd barrier". In: (Sept. 2014).