



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages 
Български
Čeština
Deutsch
Español
Esperanto
فارسی
Français
한국어
Bahasa Indonesia
Italiano
Magyar
日本語
Polski
Português
Русский
Slovenčina
Српски / srpski
Українська
中文

 Edit links


Create account Log in

Article Talk

Read

Edit

More ▾

Search 

Strassen algorithm

From Wikipedia, the free encyclopedia

Not to be confused with the [Schönhage–Strassen algorithm](#) for multiplication of polynomials.

In the [mathematical](#) discipline of [linear algebra](#), the **Strassen algorithm**, named after [Volker Strassen](#), is an [algorithm](#) used for [matrix multiplication](#). It is faster than the standard matrix multiplication algorithm and is useful in practice for large matrices, but would be slower than [the fastest known algorithms](#) for extremely large matrices.

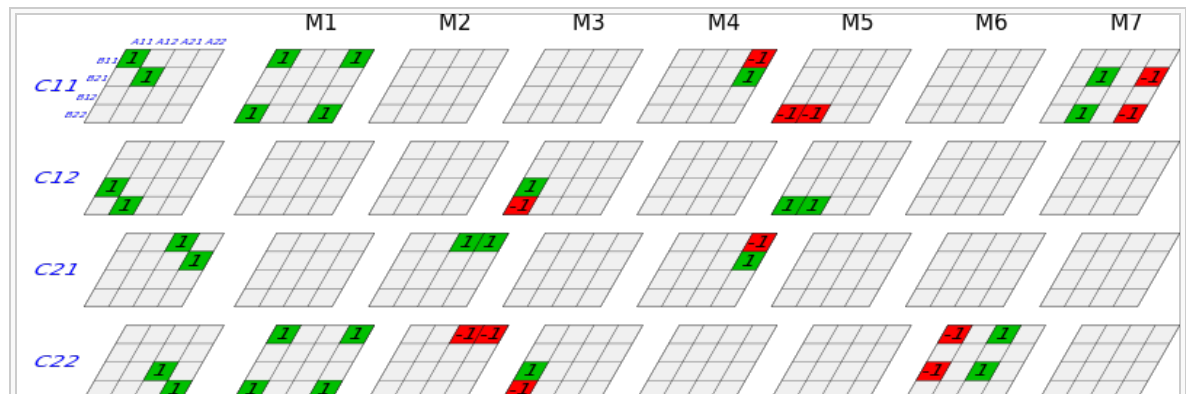
Contents [hide]

- History
- Algorithm
- Asymptotic complexity
 - Rank or bilinear complexity
- Implementation considerations
- See also
- References
- External links

History [edit]

[Volker Strassen](#) first published this algorithm in 1969 and proved that the n^3 general matrix multiplication algorithm wasn't optimal. The **Strassen algorithm** is only slightly better, but its publication resulted in much more research about matrix multiplication that led to faster approaches, such as the [Coppersmith–Winograd algorithm](#).

Algorithm [edit]



The left column represents 2×2 [matrix multiplication](#). Naïve matrix multiplication requires one multiplication for each "1" of the left column. Each of the other columns represents a single one of the 7 multiplications in the algorithm, and the sum of the columns gives the full matrix multiplication on the left.

Let A , B be two [square matrices](#) over a [ring](#) R . We want to calculate the matrix product C as

$$C = AB \quad A, B, C \in R^{2^n \times 2^n}$$

If the matrices A , B are not of type $2^n \times 2^n$ we fill the missing rows and columns with zeros.

We partition A , B and C into equally sized [block matrices](#)

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

with

$$A_{i,j}, B_{i,j}, C_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

then

$$\begin{aligned} C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

With this construction we have not reduced the number of multiplications. We still need 8 multiplications to calculate the $C_{i,j}$ matrices, the same number of multiplications we need when using standard matrix multiplication.

Now comes the important part. We define new matrices

$$\begin{aligned} \mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\ \mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\ \mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\ \mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\ \mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\ \mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\ \mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2}) \end{aligned}$$

only using 7 multiplications (one for each M_k) instead of 8. We may now express the C_{ij} in terms of M_k , like this:

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\ \mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\ \mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\ \mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6 \end{aligned}$$

We iterate this division process n times (recursively) until the [submatrices](#) degenerate into numbers (elements of the ring R). The resulting product will be padded with zeroes just like A and B , and should be stripped of the corresponding rows and columns.

Practical implementations of Strassen's algorithm switch to standard methods of matrix multiplication for small enough submatrices, for which those algorithms are more efficient. The particular crossover point for which Strassen's algorithm is more efficient depends on the specific implementation and hardware. Earlier authors had estimated that Strassen's algorithm is faster for matrices with widths from 32 to 128 for optimized implementations.^[1] However, it has been observed that this crossover point has been increasing in recent years, and a 2010 study found that even a single step of Strassen's algorithm is often not beneficial on current architectures, compared to a highly optimized traditional multiplication, until matrix sizes exceed 1000 or more, and even for matrix sizes of several thousand the benefit is typically marginal at best (around 10% or less).^[2]

Asymptotic complexity [\[edit\]](#)

The standard matrix multiplication takes approximately $2N^3$ (where $N = 2^n$) arithmetic operations (additions and multiplications); the asymptotic complexity is $\Theta(N^3)$.

The number of additions and multiplications required in the Strassen algorithm can be calculated as follows: let $f(n)$ be the number of operations for a $2^n \times 2^n$ matrix. Then by recursive application of the Strassen algorithm, we see that $f(n) = 7f(n-1) + l4^n$, for some constant l that depends on the number of additions performed at each application of the algorithm. Hence $f(n) = (7 + o(1))^n$, i.e., the asymptotic complexity for multiplying matrices of size $N = 2^n$ using the Strassen algorithm is

$$O([7 + o(1)]^n) = O(N^{\log_2 7 + o(1)}) \approx O(N^{2.8074}).$$

The reduction in the number of arithmetic operations however comes at the price of a somewhat reduced [numerical stability](#),^[3] and the algorithm also requires significantly more memory compared to the naive algorithm. Both initial matrices must have their dimensions expanded to the next power of 2, which results in storing up to four times as many elements, and the seven auxiliary matrices each contain a quarter of the elements in the expanded ones.

Rank or bilinear complexity [\[edit\]](#)

The bilinear complexity or **rank** of a [bilinear map](#) is an important concept in the asymptotic complexity of matrix multiplication. The rank of a bilinear map $\phi : \mathbf{A} \times \mathbf{B} \rightarrow \mathbf{C}$ over a field \mathbf{F} is defined as (somewhat of an [abuse of notation](#))

$$R(\phi/\mathbf{F}) = \min \left\{ r \mid \exists f_i \in \mathbf{A}^*, g_i \in \mathbf{B}^*, w_i \in \mathbf{C}, \forall \mathbf{a} \in \mathbf{A}, \mathbf{b} \in \mathbf{B}, \phi(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^r f_i(\mathbf{a})g_i(\mathbf{b})w_i \right\}$$

In other words, the rank of a bilinear map is the length of its shortest bilinear computation.^[4] The existence of Strassen's algorithm shows that the rank of 2×2 matrix multiplication is no more than seven. To see this, let us express this algorithm (alongside the standard algorithm) as such a bilinear computation. In the case of matrices, the [dual spaces](#) \mathbf{A}^* and \mathbf{B}^* consist of maps into the field \mathbf{F} induced by a scalar [double-dot product](#), (i.e. in this case the sum of all the entries of a [Hadamard product](#).)

	Standard algorithm			Strassen algorithm		
i	$f_i(\mathbf{a})$	$g_i(\mathbf{b})$	w_i	$f_i(\mathbf{a})$	$g_i(\mathbf{b})$	w_i
1	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
2	$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 0 & 0 \\ 1 & -1 \end{bmatrix}$
3	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$

4	$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$
5	$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}$
6	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$
7	$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$
8	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} : \mathbf{a}$	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} : \mathbf{b}$	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$			
$\mathbf{ab} = \sum_{i=1}^8 f_i(\mathbf{a})g_i(\mathbf{b})w_i$				$\mathbf{ab} = \sum_{i=1}^7 f_i(\mathbf{a})g_i(\mathbf{b})w_i$		

It can be shown that the total number of elementary multiplications L required for matrix multiplication is tightly asymptotically bound to the rank R , i.e. $L = \Theta(R)$, or more specifically, since the constants are known, $\frac{1}{2}R \leq L \leq R$. One useful property of the rank is that it is submultiplicative for [tensor products](#), and this enables one to show that $2^n \times 2^n \times 2^n$ matrix multiplication can be accomplished with no more than 7^n elementary multiplications for any n . (This n -fold tensor product of the 2×2 matrix multiplication map with itself—an n th tensor power—is realized by the recursive step in the algorithm shown.)

Implementation considerations [\[edit\]](#)



This section **needs additional citations for verification**. Please help [improve this article](#) by [adding citations to reliable sources](#). Unsourced material may be challenged and removed. *(January 2015)*

The description above states that the matrices are square, and the size is a power of two, and that padding should be used if needed. This restriction allows the matrices to be split in half, recursively, until limit of scalar multiplication is reached. The restriction simplifies the explanation, and analysis of complexity, but is not actually necessary;^[5] and in fact, padding the matrix as described will increase the computation time and can easily eliminate the fairly narrow time savings obtained by using the method in the first place.

A good implementation will observe the following:

- It is not necessary or desirable to use the Strassen algorithm down to the limit of scalars. Compared to conventional matrix multiplication, the algorithm adds a considerable $O(n^2)$ workload in addition/subtractions; so below a certain size, it will be better to use conventional multiplication. Thus, for instance, if you start with matrices that are 1600x1600, there is no need to pad to 2048x2048, since you could subdivide down to 25x25 and then use conventional multiplication at that level.
- The method can indeed be applied to square matrices of any dimension.^[2] If the dimension is even, they are split in half as described. If the dimension is odd, zero padding by one row and one column is applied first. Such padding can be applied on-the-fly and lazily, and the extra rows and columns discarded as the result is formed. For instance, suppose the matrices are 199x199. They can be split so that the upper-left portion is 100x100 and the lower-right is 99x99. Wherever the operations require it, dimensions of 99 are zero padded to 100 first. Note, for instance, that the product M_2 is only used in the lower row of the output, so is only required to be 99 rows high; and thus the left factor $(A_{2,1} + A_{2,2})$ used to generate it need only be 99 rows high; accordingly, there is no need to pad that sum to 100 rows; it is only necessary to pad $A_{2,2}$ to 100 columns to match $A_{2,1}$.

Furthermore there is no need for the matrices to be square. Non-square matrices can be split in half using the same methods, yielding smaller non-square matrices. If the matrices are sufficiently non-square it will be worthwhile reducing the initial operation to more square products, using simple methods which are essentially $O(n^2)$, for instance:

- A product of size $[2N \times M] * [N \times 10M]$ can be done as 20 separate $[N \times N] * [N \times N]$ operations, arranged to form the result;
- A product of size $[N \times 10M] * [10N \times M]$ can be done as 100 separate $[N \times M] * [N \times M]$ operations, summed to form the result.

These techniques will make the implementation more complicated, compared to simply padding to a power-of-two square; however, it is a reasonable assumption that anyone undertaking an implementation of Strassen, rather than conventional, multiplication, will place a higher priority on computational efficiency than on simplicity of the implementation.

See also [\[edit\]](#)

- [Computational complexity of mathematical operations](#)
- [Gauss–Jordan elimination](#)
- [Coppersmith–Winograd algorithm](#)
- [Z-order matrix representation](#)
- [Karatsuba algorithm](#), for multiplying n -digit integers in $O(n^{\log_2 3})$ instead of in $O(n^2)$ time

- [Gauss's complex multiplication algorithm](#) multiplies two complex numbers using 3 real multiplications instead of 4

References [\[edit\]](#)

- ↑ Skiena, Steven S. (1998), "§8.2.3 Matrix multiplication", *The Algorithm Design Manual*, Berlin, New York: Springer-Verlag, ISBN 978-0-387-94860-7.
 - ↑ *a* *b* D'Alberto, Paolo; Nicolau, Alexandru (2005). *Using Recursion to Boost ATLAS's Performance* (PDF). Sixth Int'l Symp. on High Performance Computing.
 - ↑ Webb, Miller (1975). "Computational complexity and numerical stability". *SIAM J. Comput.* 97–107.
 - ↑ Burgisser, Clausen, and Shokrollahi. *Algebraic Complexity Theory*. Springer-Verlag 1997.
 - ↑ Higham, Nicholas J. (1990). "Exploiting fast matrix multiplication within the level 3 BLAS" (PDF). *ACM Transactions on Mathematical Software (TOMS)* **16** (4): 352–368.
- Strassen, Volker, *Gaussian Elimination is not Optimal*, Numer. Math. 13, p. 354-356, 1969
 - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Chapter 28: Section 28.2: Strassen's algorithm for matrix multiplication, pp. 735–741.

External links [\[edit\]](#)

- Weisstein, Eric W., "Strassen's Formulas", *MathWorld*. (also includes formulas for fast [matrix inversion](#))
- Tyler J. Earnest, *Strassen's Algorithm on the Cell Broadband Engine*
- *Simple Strassen's algorithms implementation in C* (easy for education purposes)

v · t · e		Numerical linear algebra	[hide]
Key concepts	Floating point · Numerical stability		
Problems	Matrix multiplication (algorithms) · Matrix decompositions · Linear equations · Sparse problems		
Hardware	CPU cache · TLB · Cache-oblivious algorithm · SIMD · Multiprocessing		
Software	BLAS · Specialized libraries · General purpose software		
Categories: Matrix multiplication algorithms			