Is Fast Matrix Multiplication of Practical Use?

By Nicholas J. Higham

A fast matrix multiplication method forms the product of two $n \times n$ matrices in $O(n^{\omega})$ arithmetic operations where $\omega < 3$. Such a method is more efficient asymptotically than direct use of the definition

$$(AB)_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \qquad (1)$$

which requires $O(n^3)$ operations. Several fast matrix multiplication methods are known, but in view of the possibility of large constant multipliers in the operation counts, it is necessary to question the practical use of such methods. Many researchers have assumed that they have none, but recent evidence has shown this to be untrue, at least for one method.

The History of Matrix Multiplication

The English mathematician Joseph Sylvester introduced the name "matrix" for a rectangular array of numbers in 1850. Matrix algebra was developed by Sylvester and a friend, the mathematician Arthur Cayley. They regarded a matrix as representing a linear transformation, and consideration of products of linear transformations led to the definition (1) of matrix multiplication.

For more than a century this definition provided the only known method for multiplying matrices. In 1967, however, Shmuel Winograd found, to the surprise of many, a way to exchange half the multiplications for additions in the basic formula [17]. His method rests on the identification of certain inner products that can be computed and reused. Winograd's paper generated immediate practical interest because floating-point multiplication was typically two or three times slower than floating-point addition on the computers of the 1960s. (On today's machines these two operations are usually similar in cost.)

Shortly after Winograd's discovery, Volker Strassen astounded the computer science community by finding a method for matrix multiplication that requires $O(n^{\log_2 7})$ operations ($\log_2 7 \approx 2.807$). A variant of this technique can be used to compute A^{-1}, and thereby to solve $Ax = b$, both in $O(n^{\log_2 7})$ operations. (Hence the title of Strassen's 1969 paper [16], which refers to the question of whether Gaussian elimination is asymptotically optimal for solving linear systems.)

$$p_1 = (a_{11} + a_{22})(b_{11} + b_{22}),$$
$$p_2 = (a_{21} + a_{22})b_{11},$$
$$p_3 = a_{11}(b_{12} - b_{22}),$$
$$p_4 = a_{22}(b_{21} - b_{11}),$$
$$p_5 = (a_{11} + a_{12})b_{22},$$
$$p_6 = (a_{21} - a_{11})(b_{11} + b_{12}),$$
$$p_7 = (a_{12} - a_{22})(b_{21} + b_{22}),$$
$$C = \begin{bmatrix} p_1 + p_4 - p_5 + p_7 & p_3 + p_5 \\ p_2 + p_4 & p_1 + p_3 - p_2 + p_6 \end{bmatrix}$$

Figure 1. Strassen's formulas for $C = AB$.

Strassen's method is based on a circuitous way to form the product of a pair of 2×2 matrices in seven multiplications and 18 additions, instead of the usual eight multiplications and four additions; see Figure 1 (a fairly natural derivation is given in [18]). As a means of multiplying 2×2 matrices, these formulas have nothing to recommend them. Strassen observed, however, that the formulas remain valid when a_{ij} and b_{ij} are matrices. Thus, general $n \times n$ matrices A and B can be multiplied as follows: Partition A and B into four equally sized blocks (if n is even) and apply the formula to the 2×2 block product; this involves seven half-sized matrix multiplications and 18 half-sized matrix additions. For large n, these matrix additions are of negligible cost compared with the matrix multiplications, and a computational saving of about 12% is achieved.

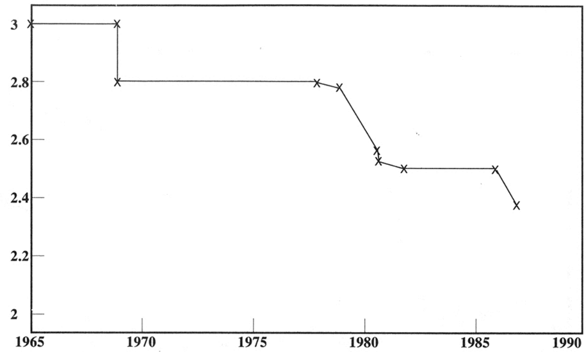

Figure 2. Exponent versus year of publication.

Strassen recognized that the half-sized matrix multiplications can be carried out by applying the very same method recursively, with a 12% saving on each level of the recursion. It is this recursive, divide-and-conquer application that results in the lowering of the exponent. In the exponent $\log_2 7$, the '2' and the '7' arise from the use of a method for multiplying 2×2 matrices in seven multiplications. In addition to a lower exponent, the method has a reasonable constant: Strassen showed that his method requires at most $4.7n^{\log_2 7}$ arithmetic operations.

Strassen's paper raised the question "what is the minimum exponent ω such that matrix multiplication can be done in $O(n^{\omega})$ operations?" Clearly, $\omega \geqslant 2$, since each element of each matrix must take part in at least one operation. Despite intensive research, the minimum ω (or a limiting value) is still unknown. The current "world record" is 2.376 [4], and some hope is expressed in [4] of reaching "the elusive $\omega = 2$." Finding an asymptotically optimal algorithm for matrix multiplication has been described as "one of the most famous outstanding problems of computer science" [15, p. 533].

One approach to lowering the exponent is to search for p and q such that two $p \times p$ matrices can be multiplied in q scalar multiplications and $\log_p q < \log_2 7$. Victor Pan discovered a suitable pair—$p = 70$, $q = 143{,}640$—in 1978 [13]. This and subsequent lowerings of the exponent involve sophisticated applications of tensors and bilinear and trilinear forms; see [14] for a survey of this work. A graph of exponent versus time of publication is given in Figure 2 (not all publications are represented in the graph); the period from 1850 to 1964 has been omitted to save space!

Implementation of Strassen's Method

In addition to stimulating research in the complexity of matrix multiplication, Strassen's paper led several authors to look at the practical implementation of his method. Before a useful implementation can be attained, a number of issues have to be addressed: how to program the recursion, how best to handle arbitrary values of n (since the basic method is defined only for n a power of 2), and how to deal with the extra storage required by the method, among others.

Richard Brent [3] implemented Strassen's method in Algol-W on an IBM 360/67 computer and obtained a speed increase over conventional multiplication for n as small as 110, but these timings were overshadowed by the even better performance of Winograd's method for the values of $n < 300$ considered. In [6, 9, 10] various implementation details of Strassen's method are investigated, but computer timings are not presented.

Not until the recent work of David Bailey at NASA Ames [1] was the practical utility of Strassen's method convincingly demonstrated. Bailey implemented the method in Fortran 77 on a Cray 2 computer and obtained speedups over conventional multiplication ranging from 1.45 for $n = 128$ to 2.01 for $n = 2{,}048$ (although 35% of these speedups are due to Cray-specific techniques). To achieve these speedups, Bailey prematurely terminated the recursions in Strassen's method so as to minimize the bookkeeping costs and to exploit the architecture of the Cray 2—once the dimension reached 127 or less, multiplications were performed by the conventional method. In joint work with King Lee and Horst Simon, Bailey has since developed a more sophisticated implementation of Strassen's method for the Cray 2 and the Cray Y-MP [2]. This version handles arbitrary dimensions and has a reduced storage requirement.

In its bid for acceptance as a practical tool, Strassen's method needs to overcome a further obstacle—the myth that it is unstable. An error analysis of the method in [3] has often been overlooked. That study, together with further analysis in [11], dispels the myth. Strassen's method is not as stable as conventional multiplication (not surprisingly, in view of the formulas underlying the method), but it is stable enough to be a contender for practical use.

Large-Scale Computations

With the steady increases in computer processing power and storage, scientists are attempting to solve larger and larger problems.

Continued on page 14

Netlib News: Greetings

By Eric Grosse

This is the first of what will be a regular series of columns, appearing four times a year, about **netlib.** Never heard of it? Then read "Distribution of Mathematical Software via Electronic Mail," by Jack J. Dongarra and Eric Grosse (*Communications of the ACM*, Vol. 30, 1987, pages 403-407).

If that's too much trouble, just send an e-mail message containing the line "help" to the Internet address **netlib@research.att.com** or the uucp address **uunet!research!netlib.** A few minutes later, if you have speedy mail connections, you will receive information on how to use netlib along with an overview of the many mathematical software libraries and databases in the collection.

Each column in this series will start with a background discussion of how netlib is run, and then address applications in other fields, security horror stories, and other topics. The second half of the column will briefly describe recent additions to the collection and important updates of old codes. If there are specific topics you would like to see addressed in future issues, let me know.

Strictly speaking, this column applies only to the netlib running at the AT&T Bell Laboratories in New Jersey. If you're accessing the copy at Oak Ridge, or Oslo, or Wollongong, or perhaps elsewhere, then the files either should be there already or will show up shortly, when our semi-automatic procedures have resynchronized the collections.

This first column provides a nice opportunity for a public thanks to our sponsors. The U.S. National Science Foundation provided an early grant to help get us started and implicitly helps by funding the national network. AT&T has donated machine resources, communication facilities, and my time. Sequent generously loaned a machine, operated by Oak Ridge National Laboratory, to support netlib. The Norwegian government, through a grant to Petter Bjørstad, purchased a machine to provide service to Europe. The Association for Computing Machinery agreed to the redistribution of its Collected Algorithms, and algorithms editor R.J. Renka arranged for prompt updates. SIAM contributed its membership database. To all these groups and the many others who have contributed, the community owes its thanks.

Naturally, this thanks should not be expressed in the form of a lawsuit. If you're unhappy with some piece of software, keep in mind that none of the contributing organizations had anything to do with the content; even the editors make no claims about the suitability of the software for any purpose. That's the meaning of the disclaimer "Anything free comes with no guarantee."

On the other hand, don't be completely frightened off by this warning. The mathematical algorithms in netlib include some of the most sophisticated and robust methods to be found anywhere. Just remember that a healthy skepticism is appropriate when you get software from any source.

Recent Additions

PLTMG, version 6.0, written by Randy Bank of the University of California at San Diego, is a package for solving elliptic partial differential equations in general regions of the plane. It features adaptive local mesh refinement, multigrid iteration, and a pseudo-arclength continuation option for parameter dependencies. The package includes an initial mesh generator and several graphics packages. Full documentation can be obtained in the users' guide, a recent volume in the SIAM Frontiers in Applied Mathematics series (*PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 6.0*, by Randolph E. Bank).

This library consists of a number of Fortran files and a few C files, most of which (aside from the graphics) are machine independent. Since the package is rather large, it has been made available via **ftp research.att.com,** for those who have Internet access. Log in as **anonymous** and **cd dist/pltmg.** You must uncompress the .Z files once you have a copy of them. Someday we plan to make all of netlib available by ftp.

Version 2.1 of the **dhrystone benchmarks** in Ada, C, and Pascal is a new release from

Continued on page 16

Fast Matrix Multiplication,

continued from page 12

For example, John Brown of Multiflow Computer and John Lewis of Boeing Computer Services are both interested in solving dense systems of linear equations of order 10,000 or more with complex coefficient matrices; these systems arise in the solution of integral equations by boundary element techniques. A promising way to make such massive computations efficient, or even just feasible, is to employ an "asymptotically fast" algorithm. Here, the level 3 Basic Linear Algebra Subprograms (BLAS3) [8] play an important role.

The BLAS3 are specifications of Fortran programs for four main tasks: general matrix multiplications, rank-r and rank-$2r$ updates of a symmetric matrix, multiplication by a triangular matrix, and solution of triangular systems with multiple right-hand sides. They provide building blocks for a wide variety of numerical algorithms; for example, they are used by many of the routines in LAPACK [7]. Strassen's method, or any other fast matrix multiplication technique, can be used to produce asymptotic speedups in all the BLAS3, as shown in [11], and hence in any algorithm that can be expressed in terms of the BLAS3. Thus, Strassen's method has the potential for producing useful speedups in many numerical algorithms.

An impressive example of the realization of this potential can be found in the work of Bailey, Lee, and Simon [2]. They substituted their Strassen's method code for the BLAS3 subroutine SGEMM and tested LAPACK's SGETRF. This routine performs LU factorization of dense matrices using a block algorithm. Using dimensions $n \leqslant 2{,}048$, Bailey, Lee, and Simon obtained a maximum rate of computation of 325 virtual megaflops on a Cray Y-MP (single processor), as compared with 291 Mflops when a conventional matrix multiply kernel was used. Here, "virtual" denotes that the computational cost was overestimated as $2n^3/3$ floating-point operations, which explains the otherwise puzzling fact that the peak performance of the machine (for one processor) was exceeded!

It seems likely that fast methods for numerical computation will be more widely used in the future. First, however, the numerical stability of the methods needs to be proven, and codes will have to be written carefully to exploit particular computer architectures. A step in this direction is the provision by two supercomputer manufacturers of Fortran 77 implementations of Strassen's method that are tuned for their particular machines. IBM's ESSL library [12] contains codes for the IBM 3090, and Cray Research Inc. provides codes in its UNICOS library [5] for Cray X-MP/Y-MP and Cray 2 systems.

References

[1] D.H. Bailey, *Extra high speed matrix multiplication on the Cray-2*, SIAM J. Sci. Stat. Comput., 9 (1988), 603-607.

[2] D.H. Bailey, K. Lee, and H.D. Simon, *Using Strassen's algorithm to accelerate the solution of linear systems*, Manuscript, 1990.

[3] R.P. Brent, *Algorithms for matrix multiplication*, Technical Report CS 157, Computer Science Department, Stanford University, 1970.

[4] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progression*, Proceedings of the Nineteenth Annual ACM Symposium of Theory of Computing (1987), 1-6.

[5] Cray Research Inc., *UNICOS Math and Scientific Library Reference Manual*, Number SR-2081, Version 5.0 (March 1989).

[6] J. Cohen and M. Roth, *On the implementation of Strassen's fast multiplication algorithm*, Acta Informatica, 6 (1976), 341-355.

[7] J.W. Demmel, J.J. Dongarra, J.J. Du Croz, A. Greenbaum, S.J. Hammarling, and D.C. Sorensen, *Prospectus for the development of a linear algebra library for high-performance computers*, Technical Memorandum No. 97, Mathematics and Computer Sciences Division, Argonne National Laboratory, Illinois, 1987.

[8] J.J. Dongarra, J.J. Du Croz, I.S. Duff, and S.J. Hammarling, *A set of Level 3 basic linear algebra subprograms*, Preprint No. 1, Mathematics and Computer Science Division, Argonne National Laboratory, 1988.

[9] P.C. Fischer, *Further schemes for combining matrix algorithms, in Automata, Languages and Programming*, J. Loeckx, ed., Lecture Notes in Computer Science 14, Springer Verlag, Berlin (1974), 428-436.

[10] P.C. Fischer and R.L. Probert, *Efficient procedures for using matrix algorithms, in Automata, Languages and Programming*, J. Loeckx, ed., Lecture Notes in Computer Science 14, Springer Verlag, Berlin, (1974), 413-427.

[11] N.J. Higham, *Exploiting fast matrix multiplication within the level 3 BLAS*, Technical Report 89-984, Department of Computer Science, Cornell University, 1989; to appear in ACM Trans. Math. Soft.

[12] IBM, *Engineering and Scientific Subroutine Library, Guide and Reference, Release 3*, Fourth Edition (Program Number 5668-863), 1988.

[13] V. Pan, *Strassen algorithm is not optimal. Trilinear technique of aggregating, uniting, and canceling for constructing fast algorithms for matrix multiplication*, Proc. 19th Annual Symposium on the Foundations of Computer Science, Ann Arbor, MI (1978), 166-176.

[14] V. Pan, *How can we speed up matrix multiplication?*, SIAM Review 26 (1984), 393-415.

[15] R. Sedgewick, *Algorithms*, Second Edition, Addison-Wesley, Reading, Massachusetts, 1988.

[16] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), 354-356.

[17] S. Winograd, *A new algorithm for inner product*, IEEE Trans. Comput., C-18 (1968), 693-694.

[18] G. Yuval, *A simple proof of Strassen's result*, Information Processing Letters, 7 (1978), 285-286.

Nicholas J. Higham is a member of the Department of Mathematics at the University of Manchester, Manchester, England.