

Fast matrix algorithms and multiplication formulae

Warren D. Smith WDSmith@fastmail.fm, wds@math.temple.edu

Project summary

Broad impact. Computation with dense matrices is a highly developed, extremely successful field; software packages such as LAPACK and its relatives are now among the most important pieces of scientific software in the world and underlie everything from MATLAB^(TM), to statistics, to radar scattering calculations, to quantitative economics, to signal processing, to computational chemistry, to weather prediction.

V.Strassen discovered in 1969 how to multiply $N \times N$ matrices in $O(N^{2.81})$, not N^3 , steps. While this led to a 25-year-long revolution as far as algorithm theorists are concerned, it has had almost no effect on LAPACK and all its applications. This research should begin to change that.

Merit. Part of the merit of this research is to see that such subcubic times are possible, not just for matrix multiplication, but indeed for most of the matrix computations of interest, and the consequent savings are achievable not just in theory, for unrealistically enormous matrices, but in practice, for matrices of realistic sizes.

This proposal concerns:

I. Finding formulae like Strassen's famous 7-multiplication method for multiplying 2×2 matrices. These formulae underlie "fast matrix multiplication" algorithms.

II. Find ways to make most of the standard matrix algorithms in Golub & Van Loan's book *Matrix computations* (e.g.: Gaussian elimination, orthogonalization, least squares problems, eigenproblems, and many more) become asymptotically fast (subcubic runtime) without sacrificing their practicality and while retaining, in at least some form, whatever guaranteed numerical properties they have.

Smith has done enough research already to have confidence he'll make substantial progress in both areas. Namely:

(I) Smith's new search techniques have (so far) produced 25 new-record Strassen-like formulae (comparable to the total number of constructions discovered by all other investigators combined). They've revealed an important new empirical phenomenon, the "quantum effect." Smith has numerous new concepts and theoretical tools related to Strassen-like formulae, e.g., new notions of, and theorems about, their "canonical forms," new notions of "symmetric" matrix multiplication formulae, and new notions of "inexact" formulae and their theoretical uses. These have been key to the computer search-programs which found most of these new records. The project description lists reasons why Smith's search program is probably at least 40 billion times faster than previous attempts to create such programs.

(II) Smith has a menagerie of new algorithm-design paradigms. These have reached the point where they powerfully synergize with each other; one can now usually find a way to mix and match these tools to produce a fast algorithm for most matrix tasks. (The project description clearly explains why almost all previous work in this area has been impractical, and why one can do better.) Smith can use these design techniques to speed up most of the algorithms in Golub & Van Loan's classic text *Matrix Computations* to subcubic (and in most cases to provably optimal, up to a constant factor) runtimes. Some of these are explained in the text of the proposal. But much remains to be done – Smith's work so far can be regarded as a proof of principle and as assurance this research will get somewhere, but it is not yet tuned, optimized, well-explained algorithms, complete with experimental studies. Moving in that direction is the object of the proposal.

Project description – Fast matrix algorithms & multiplication formulae – W.D.Smith

1. Computational linear algebra

Computations with (mostly dense) matrices is a highly developed area. Some of its greatest achievements are enshrined in the book *Matrix computations* by Golub and Van Loan [16], and in the software package LAPACK. LAPACK and its relatives are probably the most-used nontrivial scientific software in the world.

There has been more and more interest in, and need for, matrix computations with very *large* dense matrices. Today's hardware is now capable of handling the job. (Also, non-iterative computations with large *sparse* matrices usually require, underneath, good routines for handling dense matrices.)

Examples:

1998: A full-complex Hermitian Eigensystem (problem arising in the computation of chemical quantum energy levels; 16-byte-wide matrix entries) was found [31] for a 39936×39936 Hermitian matrix (9200-processor Intel ASCI red, 200 MHz Ppros);

2001: Gaussian elimination with partial pivoting was used to solve a dense 525000×525000 linear system (3000-processor Compaq AlphaServer SC/ES45 1GHz; 7.3 hour run; note just *storing* such a matrix takes over 2000 gigabytes);

2002: R.A.Wilson et al. multiplied two elements of the “monster” sporadic finite simple group of order

8080, 17424, 79451, 28758, 86459, 90496, 17107, 57005, 75436, 80000, 00000,

each represented by a 196882×196882 matrix over a small finite field. (“Most of the computing resources of Lehrstuhl D für mathematik, RWTH Aachen, for about 45 hours”). According to Wilson, such computations have now “effectively tamed the monster.”

Therefore, *asymptotically* fast matrix methods are becoming more and more relevant.

2. Strassen and fast matrix multiplication

V.Strassen [33] in 1969 found a formula for multiplying 2×2 matrices using only 7, instead of the usual 8, multiplications. This formula still works even if the entries of the matrices are *noncommuting*. Strassen then noted that by recursive use of his formula one could multiply $N \times N$ matrices in $O(N^{2.81})$ arithmetic operations, instead of the usual $\approx 2N^3$. Over the next 21 years, the exponent ω fell from 2.81 to its present record low of 2.38 (due to D.Coppersmith and S.Winograd [9] in 1990, by combining many previous techniques with some new ideas). However, progress has stalled; no further exponent reductions have occurred since 1990, nor do any seem likely soon (although nobody believes 2.38 is best possible).

It was also pointed out that various linear algebra tasks (including solving linear systems – this pointed out by Strassen in his original paper) could also be performed in $O(N^\omega)$ arithmetic operations, where ω is any upper bound on the exponent of matrix multiplication.

A good review of some of the simpler exponent-reducing techniques is Pan's [28], and the book of Bini and Pan [5] and of Burgisser et. al [7] give theoretical (but often not practical) discussions of matrix and polynomial algorithms.

3. Gap between theoretical-asymptotic results and practical

While the methods proposed in 1970-1990 for reducing ω below 2.8 were astonishingly ingenious, none of them are practical, in the sense that none of them will beat Strassen's $N^{2.81}$ method if $N < 10^{20}$. Thus, from the practical point of view, no progress whatever has been made since 1969.

Not only has no real improvement been made on Strassen, but furthermore, even today, *still* no “good” *Strassen* code has even been implemented! This is regarding a “good” code-package as including all of the following:

1. Ability to multiply (rectangular) matrices of arbitrary dimensions.
2. Recursive use of Strassen's $\langle 2, 2, 2 \rangle = 7$ basic formula, with handling of "leftover" rows (arising if matrix dimensions are odd) by use of "brute force" rank-1 updates. (The $\langle a, b, c \rangle$ notation will be explained in §5.)
3. Use of a Strassen-variant having minimized additions and temporary storage.
4. Use of other basic formulae besides Strassen's, such as $\langle 3, 3, 3 \rangle \leq 23$ (Laderman/Sykora [24][34]), $\langle p, 2, q \rangle \leq \lceil \frac{3pq + \max\{p, q\}}{2} \rceil$ for $p, q \geq 1$ (Hopcroft et al. [18][19]).
5. Use of the best possible combination of these smaller formulae (and the naive method).
6. Ability to handle common specially simple matrix multiplication tasks such as $A^T A$, A^2 , lower and upper triangular multiplicands, symmetric multiplicands, banded multiplicands.
7. Handling of matrices with complex entries by means of the 3-multiplication formula $(A + iB)(C + iD) = AC - BD + i[(A + B)(C + D) - AC - BD]$, saving a factor of 25%.
8. "Cache-oblivious algorithms" techniques [15].
9. Possible combination with the methods of Kaporin [22], and Laderman, Pan, Sha [25].

(Some, but not all, of these have been implemented, in Strassen codes written and experimented with by several independent groups, including the PI. But without all of these combined, one's code is obviously improvable by $\gtrsim 25\%$.)

3.1 Gaussian elimination - an embarrassing case study

Furthermore, many of the methods that have been proposed (and that are discussed in the book of Bini and Pan [5]) for "fast" matrix computations besides matrix multiplication (e.g. solving $N \times N$ linear systems), are still *less* practical – even to the extent that they all are of no practical interest whatever. *This* point is unfortunately not discussed at all well in the book of Bini and Pan, or anywhere I've seen, so let me dwell on it now.

Consider the task of solving an $N \times N$ linear system $A\vec{x} = \vec{b}$. One could use block-based recursive Gaussian-elimination-like techniques to factor $A = LU$, where L is lower and U is upper triangular, in $O(n^\omega)$ flops, and then find $\vec{x} = U^{-1}L^{-1}\vec{b}$ by backsubstitution. But due to the absence of pivoting, this could involve division by zero, and even without that could involve arbitrarily large amplification of tiny numerical roundoff errors. (E.g. in Gaussian elimination, a pivot 100 times smaller than maximum possible could be used, which would result in amplification of tiny errors by 100. If this happened during 100 pivots, the result would be complete garbage.)

So Strassen (in advice later repeated in the book by Bini and Pan) suggested instead solving $A^T A \vec{x} = A^T \vec{b}$ for \vec{x} . This requires an extra matrix multiplication to create $A^T A$, but then we can LU -factorize *this*. Since it is automatically a positive definite symmetric matrix, $A^T A$, by results of Wilkinson [36], may be LU -factorized stably *without* pivoting. But, unfortunately, Strassen neglected to point out that this approach squares the condition number. To illustrate the effects of that, consider

$$A = \begin{pmatrix} \epsilon & 0 \\ 1 & 1 \end{pmatrix}, \quad A^T A = \begin{pmatrix} 1 + \epsilon^2 & 1 \\ 1 & 1 \end{pmatrix}. \quad (1)$$

Here the original system could be solved immediately by backsubstitution in a backwards stable (with respect to relative errors in the coefficients) manner. Meanwhile the new system, if ϵ^2 is smaller than unit roundoff, would be a singular matrix which would yield a division by zero and/or complete garbage. To avoid that fate and to get backwards stability, it is necessary to compute $A^T A$ and its later LU factorization with *double* the usual precision arithmetic, i.e. with at least 4 times the cost.

The cost of performing (conventional) Gaussian elimination with partial pivoting on an $N \times N$ matrix is $2N^3/3$ flops. Meanwhile, the "improved" approach (if done via conventional methods) requires N^3

additional flops for forming $A^T A$ (here advantage is taken of the symmetry of $A^T A$ to save a factor of 2 versus the naive flopcount $2N^3$), for a total of $8N^3/3$ flops. (Actually, by using our new fast-Cholesky factorization described later, $A^T A$'s LL^T factorization could be found while saving a factor of 2 in effort, i.e., the conventional flop count for Cholesky factorization is of order $N^3/3$ not $2N^3/3$. That would bring the total conventional flop count down to only $7N^3/3$.)

This is 4 times larger (or with our Cholesky approach, $7/2$ times larger). That may not seem so bad, but since we must count each of these flops as *four* flops since they are double precision, the total number of flops is not 4 (or $7/2$) times larger, but really 16 (or 14) times larger. This could be cut slightly if it were realized that $A^T A$ is computed using double precision multiplications of two single precision floats, which is not quite as expensive. So let me be generous and assume the “improved” method only suffers a factor-10 slowdown. *Our point is:* this factor-10 penalty, to be made up for by the improvement arising from using $O(N^{2.81})$ -step “fast” matrix methods instead of conventional $O(N^3)$ -step methods, requires N to be made larger by a factor of $10^{1/.19} \approx 183298$. In other words, the breakeven N will not be ≈ 100 (as it is for Strassen’s matrix multiplication versus conventional matrix multiplication on modern machines) but instead will be 18329800. Since this far exceeds the record largest Gaussian elimination performed so far ($N = 525000$), we conclude that this “fast” method of Gaussian elimination is of no practical interest.

Consequently, essentially every other “fast” matrix algorithm (besides plain multiplication) described in the Bini-Pan book has similar shortcomings. Things like this have led to a general distrust of the fast-matrix-algorithm / computer science community by the numerical practitioner community.

But it does not have to be that way. If Gaussian elimination *with* pivoting were done “fast” in *single* precision, then asymptotically-fast methods *would* be practical. I have, in fact, invented for the first time, fast GEPP (discussed later), and it indeed seems likely to be practical.

3.2 Fast eigenvalues - an ultra-embarrassing case study

The method (apparently due to W.Keller-Gehrig [23]) recommended in the books by Burgisser et al. and by Bini and Pan for finding eigenvalues involves a technique especially deserving of disdain: fast “Krylov matrices.” Given an $n \times n$ matrix M and a (unit norm) n -vector \vec{x} , the $n \times n$ “Krylov matrix” has columns

$$\text{Kry}(M, \vec{x}) \stackrel{\text{def}}{=} [\vec{x} | M\vec{x} | M^2\vec{x} | \dots | M^{n-1}\vec{x}]. \quad (2)$$

Call this matrix K . It may be computed naively in $2n^3 + O(n^2)$ flops by multiplying each K -column by M to get the next one (and doing it $n - 1$ times). But it may be computed in $O(n^\omega \log_2 n)$ flops by repeatedly squaring M to get M^2, M^4, M^8 etc. each time using it to double the number of known columns of K . One may then use K for various purposes. For example, since by the Cayley-Hamilton theorem $M^n \vec{x} + \sum_{\ell=0}^{n-1} c_\ell K_\ell = 0$ where K_ℓ is the ℓ th column of K , $\ell = 0, 1, \dots, n - 1$, and c_ℓ is the coefficient of λ^ℓ in the expansion of the monic characteristic polynomial $P(\lambda) = \det(M - \lambda I)$ of M , we can (if K is nonsingular) find M 's characteristic polynomial by solving $K\vec{c} = M^n \vec{x}$ for \vec{c} . This yields a “fast” algorithm ($O(n^\omega \log_2 n)$ flops) for finding the characteristic polynomial and thence (if we now run a polynomial rootfinder) for the eigenvalues of M .

Unfortunately, K is usually very near to being singular: Distinguishing K from a singular matrix requires recognizing that corrections of size $(\lambda_{\min})^n$ have been added to the terms of size $\|M\|_2^n$ which dominate the entries of K . Here λ_{\min} is the smallest absolute value among the eigenvalues of M . Even if we are so lucky that $\|M\|_2/\lambda_{\min} \approx 2$, this will require carrying at least n bits of precision in all the n^2 numbers in K . But the situation will be far worse than that if M were originally poorly conditioned – the condition number of M , essentially, is raised to the n th power when we get K – and matters will be even worse still if the initial choice of \vec{x} happened to be unlucky. This need for precision $\geq n$ arithmetic will slow down Krylov-based algorithms tremendously, *dwarfing* any savings obtained from the use of

fast matrix multiplication.

In fact, the need for precision- $O(n)$ arithmetic means that these asymptotically “fast” algorithms in fact are far slower, *especially* asymptotically, than conventional methods (not to mention ridiculously space-hungry). So they are completely and irredeemably useless for practical floating point computations. (In the comparatively rare event that one were interested in characteristic polynomials of matrices over *finite fields*, though, these methods could be effective. But new fast eigenmethods that are part of this proposal both are far better behaved numerically, and which do not suffer any log n runtime factor, so even then these Krylov based methods seem obsolete.)

3.3 What they were thinking

Of course, these theorists may not have been unaware of numerical issues, but simply wanted to investigate theoretical limits, untrammled by practical considerations.

4. What I can do

I propose to push beyond these investigations into the realm of the practical, while also getting new theoretical results. The attack is 2-pronged (**A** and **B** below, discussed in more detail in §5 and §6 respectively).

A. I have invented new methods for seeking new basic formulas such as Strassen’s 7-mul formula. I have successfully used these methods to find 25 new-record basic formulae¹, whose parameters are given in the table below. In this work I discovered, empirically, a new phenomenon I call the “quantum effect.” If this effect can be confirmed rigorously, it should have a dramatic impact on fast matrix theory.

<i>abc</i>	<i>M</i>	<i>Q</i>	ω	<i>abc</i>	<i>M</i>	<i>Q</i>	ω		
222	7	R	0	2.807354921	333	21	R	0.0012969046	2.771243750
223	10	R	9.995×10^{-7}	2.779885224	334	27	R	0.038130622	2.759162368
224	13	N	0.0018442103	2.775329788	335	33	N	0.0011552417	2.755569403
225	16	N	0.035930044	2.776538557	336	39	N	0.027019967	2.755259186
226	19	N	0.05897878	2.779473668	344	36	N	0.004451691	2.777059828
233	14	N	0.01079759	2.739153525	345	44	N	0.055593688	2.772743898
234	18	N	0.14680153	2.728435621	346	54	R	0.0003079792	2.798196494
235	23	N	0.0082581753	2.765638565	355	55	N	0.049808575	2.784489323
236	27	N	0.0016879719	2.759162368	356	69	N	0.016032822	2.822857064
244	24	N	0.063169541	2.750977500	366	84	R	0.0012646144	2.838974337
245	30	N	0.04111826	2.766041090	444	48	N	0.00048567336	2.792481251
246	37	R	0.00033098825	2.798292754	445	60	N	0.0044387035	2.803048614
255	39	N	0.0033880461	2.809463269	455	75	N	0.035392092	2.812591895
256	45	N	0.0097947787	2.789210165	555	90	N	0.060438642	2.795888947
266	54	N	0.0075372439	2.798196494	666	140	R	(223 & 332)	2.757983149
999	527	N	0	2.852325890	14, 14, 14	1743	N	0	2.828041271
234	19	N	4.371×10^{-5}	2.779473668	244	25	N	8.928×10^{-5}	2.786313714

Figure 1. New (and old) matrix multiplication formulae found by my computer searches, and the upper bounds ω on the exponent of matrix multiplication they yield. (The upper bound 2.728435621 is better than any other achieved so far by a small formula without using Schönhage’s tau-theorem.) Formulae all are bilinear and multiply $a \times b$ by $b \times c$ matrices (getting $a \times c$ matrix) using M noncommutative multiplications. Cases marked ‘R’ are rediscoveries of old records (or rediscoveries of records that can

¹These were discovered with the aid of computing resources no longer available to me. Quite possibly with more-modern computers and improved searchers, more records will be found.

be got in “non-basic” ways, i.e. by combining smaller record formulae; in some cases, e.g. 666 (see §5.6), these are actually new because some needed smaller formulae are new), although in many cases the computer’s discovery is inequivalent to the old construction method. The remaining cases are *new* “basic” records. (There are also a large number of consequent derived “non-basic” records, not listed.) Formulae with non-zero Q values (see §5 for definitions) are presumably ϵ -approximate inexact (although even this has not been proven and is believed based on empirical evidence); those with $Q = 0$ are exact. Due to limitations of floating point arithmetic, it can be difficult to tell which. (The two exact formulae below the line were found by me – the human – without computer assistance.) The entries for $234 \rightarrow 18$ and $244 \rightarrow 24$ are the least convincing new records (since their Q values are the largest); so in these cases I also list (at the bottom of the table) matmul formulae with the same parameters but one additional multiplication, having much smaller Q values. These still are new records. [It is impractical for me to give any of these new record formulae in full detail here, since they are described by 100s or 1000s of real numbers.]

B. I have invented a family of new techniques for making a large variety of matrix computations simultaneously theoretically *and* practically faster, i.e. subcubic asymptotic runtime, and runtime which is both never slower than the best conventional algorithm, and which becomes faster at practically reachable N . (I also have new lower-bounding techniques, which I won’t discuss.) Roughly speaking, I have gone through Golub & Van Loan’s *Matrix Computations* book, trying to redo everything in it to make it fast, while not sacrificing pleasant numerical properties and not sacrificing practicality. In most cases I have succeeded.

I will describe enough of (the simplest of) my new techniques in enough detail to convince you I really have them. A full exposition of **A** is largely completed and is about 50 (large) pages long. Its evolving current version is www.math.temple.edu/~wds/prospector.{ps,pdf}. (at this date *not* in final form; will be reposted occasionally). The status of **B** is, however, much less completed, and it will be considerably longer. (Consider the size of the Golub & Van Loan book to get an idea of how much has to be written...)

The purpose of this proposal is to complete the task.

5. Search for Strassen-like basic formulae

Define the “dot” notation: $X \cdot Y \stackrel{\text{def}}{=} \sum_{i=1}^d \sum_{j=1}^a X_{ij} Y_{ij}$, where X and Y are $d \times a$ matrices. (In other words, we regard the two matrices as “big vectors” and take their inner product.) Then a general “non-commutative m -multiplication **bilinear algorithm** [20]” for performing the matrix multiplication $C = AB$ is specified by $3m$ matrices of *coefficients* $X^{(r)}$, $Y^{(r)}$, $Z^{(r)}$, $r = 1, 2, \dots, m$, of the same dimensions as A , B , C respectively, such that

$$C = \sum_{r=1}^m \left(A \cdot X^{(r)} \right) \left(B \cdot Y^{(r)} \right) Z^{(r)}. \quad (3)$$

For conciseness, denote the existence statement for such a bilinear algorithm by the **angle bracket notation**

$$\langle n_1, n_2, n_3 \rangle \leq m. \quad (4)$$

If any such algorithm, for any n_1, n_2, n_3 , exists, then it may be recursed to show there is a method for multiplying $n \times n$ matrices in $n^{\omega+o(1)}$ arithmetic operations, $n \rightarrow \infty$, where

$$\omega \leq \frac{3 \log m}{\log(n_1 n_2 n_3)}. \quad (5)$$

EQ 3 is a valid identity for indeterminate A, B, C if and only if it always holds for matrices A and B containing all 0s, except for a single 1, hence if and only if

$$\sum_{r=1}^m X_{ij}^{(r)} Y_{j'k}^{(r)} Z_{i'k'}^{(r)} = \delta_{ii'} \delta_{jj'} \delta_{kk'}, \quad 1 \leq i, i' \leq n_1, 1 \leq j, j' \leq n_2, 1 \leq k, k' \leq n_3. \quad (6)$$

EQ 6 constitutes, in total, $(n_1 n_2 n_3)^2$ scalar validity conditions, each of which must be satisfied by the $X^{(r)}, Y^{(r)}, Z^{(r)}$.

For real $p \geq 1$, define the non-negative-valued “cost function”

$$Q \left(X^{(1, \dots, m)}, Y^{(1, \dots, m)}, Z^{(1, \dots, m)} \right) \stackrel{\text{def}}{=} \sum_{\substack{1 \leq i, i' \leq n_1, \\ 1 \leq j, j' \leq n_2, \\ 1 \leq k, k' \leq n_3}} \left| \delta_{ii'} \delta_{jj'} \delta_{kk'} - \sum_{r=1}^m X_{ij}^{(r)} Y_{j'k}^{(r)} Z_{i'k'}^{(r)} \right|^p \quad (7)$$

in $D = (n_1 n_2 + n_2 n_3 + n_1 n_3)m$ -dimensional parameter space. For any particular $p \geq 1$ (we will *prefer* $p = 2$, but $p = 1$ and $p \rightarrow \infty$ also have advantages), cost_p is zero if and only if $X^{(1, \dots, m)}, Y^{(1, \dots, m)}, Z^{(1, \dots, m)}$ represents a valid $\langle n_1, n_2, n_3 \rangle$ matrix multiplication algorithm.

This reduces the problem of finding Strassen-like formulae, to the problem of minimizing a certain sextic polynomial Q over $(n_1 n_2 + n_2 n_3 + n_1 n_3)m$ -dimensional space. Obviously $Q = 0$ if and only if we have a valid formula for matrix multiplication. Less obviously (but provably) the infimal value of Q is 0 if and only if an “arbitrary precision approximation” (APA, also called “ ϵ ” [4]) formula exists.

So, by simply running a conjugate gradient minimizer in high dimensional space, one can try to find Strassen-like matrix multiplication formulae. Indeed my program *does* find Strassen’s own formula (by performing minimizations in 84-dimensional space) in about 1 second. On the other hand, by running rigorous global-min-finding branch and bound codes [17], one can try to *disprove* the existence of a Q -minimum below some nonzero threshold, thereby disproving the existence of a matrix multiplication formula.

5.1 Quantum effect – a mysterious empirical discovery

An amazing thing happened when I ran this program (and this has been confirmed by hundreds of different kinds of runs): The final minimized values of Q that it finds are almost always *integers*, or slightly above integers. For example, 0.001, 1.001, 2.003. Furthermore, whenever convergence stalls (i.e. the minimizer is having trouble reducing Q further), the value of Q is (almost always) slightly above an integer. If, however, the minimizer “gets lucky” and manages to get *below* that integer, then dramatic rapid cost reductions re-ensue – until we get slightly above some smaller integer, at which point the reductions again slow to a crawl. Another thing: it empirically appears that “slightly above an integer” means “by optimizing for long enough and weakening any bounds on the sizes of the undetermined coefficients enough, Q can be reduced arbitrarily close to that integer.” Quantum effects also seem to apply, at least sometimes, if one allows *complex* coefficients, or if one is searching for other bilinear formulae besides matrix multiplication, e.g. formulae for multiplying octonions. (The naive method to multiply two octonions – also called “Cayley numbers” – would involve 64 multiplications, but my computer and I have reduced that to 24 [apparently exact], and 22 [if ϵ -algorithms are permitted].)

Where are these integers coming from??!

I call this integers-from-nowhere effect, the “quantum effect.” I do not understand it. I’ve asked other mathematicians about this and whatever other integers-from-nowhere effects they’d heard of – and this is the clear winner for mystery. (The fact that, e.g., $e^{\pi\sqrt{163}} = 262537412640768743.999999999999925\dots$ is very close to an integer, takes the silver medal. *That* has been wholly understood for many decades.)

Essentially, there is a very subtle conspiracy going on among 100s or 1000s of coefficients, all of which are random-looking real numbers (not integers at all!) such that certain 6th degree polynomials Q of them, magically come out integer, or slightly above, when one tries to minimize Q . The closest I have come to an explanation, is, that *after* removing certain “equivalence transformation” symmetries (discussed later??), many of the coefficients, often, come out mysteriously close to certain rational numbers (although there are usually some other coefficients which do not look close to any recognizable values). I have (with the computer’s aid) explored many possible putative explanations of the quantum effect – with the result that all the putative easy explanations are wrong.

Obviously, if the quantum effect *can* be understood, it would have a dramatic impact on the fast matrix area, which has been stalled, with zero theoretical progress in reducing the exponent, since 1990. E.g., to prove a matrix multiplication ϵ -formula existed, all one would need to do would be to prove that Q could be reduced below 0.99; the quantum effect would do the rest. The quantum effect and observations surrounding it are also, by the way, the most important reason I believe that my computer’s numerical results indeed represent genuine matrix multiplication formulae, although strictly speaking this is only known for certain in the $Q = 0$ cases in which exact arithmetic has been employed.

5.2 Symmetries and canonization

There are a large class of “equivalence transformations” (both discrete and continuous; an almost-complete list is given in [20]) of bilinear matrix multiplication formulae. These convert any such formula into another, equally valid, formula.

The most interesting class of them is this: For any nonsingular matrices P, Q, R of respective sizes $n_1 \times n_1, n_2 \times n_2, n_3 \times n_3$, perform

$$(X^{(r)}, Y^{(r)}, Z^{(r)}) \rightarrow (PX^{(r)}Q^{-1}, QY^{(r)}R^{-1}, P^{-T}Z^{(r)}R^T), \quad r = 1, \dots, m. \quad (8)$$

I have invented several “canonical form” theorems, showing, essentially, how, mechanically, to reduce any bilinear matrix multiplication formula (under either these general real equivalence transformations, or the unimodular subgroup of integer-preserving ones) to simpler-than-usual forms “canonical” under these transformations. These allow automated “beautification” of formulae found by computer, and my associated invariant theory allows automating proofs that two matrix multiplication formulae are inequivalent.

Example (a particularly easy observation): the set of m spectra of $X^{(r)}Y^{(r)}(Z^{(r)})^T, r = 1, 2, \dots, m$, is an equivalence transformation invariant.

I have also invented the notion of a “symmetric” bilinear matrix multiplication formula – one which, under some equivalence transformation, is mapped to *itself*. It turns out (although this had not been previously recognized) that most of the previously known famous bilinear matrix multiplication formulae (Strassen’s among them) both are symmetric, and are already in canonical form (in many cases, in several of my canonical forms simultaneously). This suggests that these ideas lead in the right direction: naturally occurring good matrix multiplication formulae often are symmetric, and my notions of “canonical form” often correspond to human notions of a “beautiful presentation.”

Let us make a geometric analogy. Suppose one were seeking the “best” way to place 12 points on a sphere, to maximize the volume of their convex hull. One could solve this by a 24-dimensional maximization. However, one could canonize under the $SO(3)$ rotation group of equivalence transformations by demanding that one of the points lie at the North pole, and a second lie on the Greenwich meridian. That would reduce the problem to a 21-dimensional maximization. If, further, one hypothesized that, since the problem is $SO(3)$ -symmetric, it is plausible that its optimal solution should be symmetric under some finite subgroup of $SO(3)$, and in particular if one hypothesized the icosahedral group of 60 rotations, then there would be a *unique* 12-point set invariant under this group (namely, the vertices of

the regular icosahedron), so the problem would be reduced to a *zero*-dimensional search problem and would be solved instantly with no search at all. (Since, in fact, it is known [12] that the icosahedron is the unique optimal solution of the original problem. If one did not know that, one would have the optimum solution but would not be certain of its optimality.) However, if one foolishly guessed, instead, the rotation group of a regular 12-gon, then the only 12-point sets invariant under *this* group would have *zero* convex hull volume – pessimal. This illustrates the immense power of symmetry, but also the fact that, to harness that power, one must guess the right symmetry group!

To return to the matrix multiplication area: I have incorporated numerous putative symmetry groups (finite subgroups of the known full group of equivalence transformations) into my optimizer, to allow it to search for symmetric matrix multiplication formulae using a much lower-dimensional search. When one guesses the right symmetry group, the result is huge increases in searching power. Many of the new record formulae the searcher found, are symmetric. (See §5.4 for an example.)

Strassen’s 7-multiplication formula has been conjectured to be optimal among ϵ -formulae, i.e. it is conjectured that there is no way to multiply 2×2 matrices arbitrarily accurately with only 6 multiplications. Embarrassingly, this conjecture remains open, even after over 30 years. Our canonizations can be used to reduce the search problem from 72 dimensions to 30 dimensions, so that this conjecture could now be settled by finding a nonzero lower bound on the global minimum of a certain function in 30 dimensional space. This task may be within the reach of the best rigorous min-finding programs currently under development, but so far the (non-best) such programs available to me have been unable to settle this conjecture by machine.

5.3 Net increase in power

Optimization-based approaches had been used before, albeit with much less success, to search for Strassen-like matrix multiplication formulae. There are several reasons, some interesting and some boring, why my searcher has been so much more successful than others. I have reason to believe my search program is, on hard problems, effectively 4×10^{10} times faster than previous ones, due to the following approximately-measured speedup factors:

- Use of tuned conjugate gradient optimizer and efficient analytic differentiation of the cost function (as opposed to: ad hoc minimizers): 100
- Performance-statistics-guided “Futility cutoffs:” 100
- “Hall of fame” mechanism: 8
- Faster hardware: 1000
- symmetry groups (when applicable): ≥ 500 , but probably often much more.

There are at least 4 more speedup ideas which, so far, have *not* been incorporated into the searcher, and which might yield an additional factor-8000 speedup:

- “Many searches in one:” 4
- Library of records used to generate initial guesses by truncation and combining tricks, “genetic algorithms [27],” discrete searchers, etc.: 100?
- Take advantage of quantum effect to make more-informed cutoff decisions: 10?
- Use hand-optimized machine code in inner loops: 2?

5.4 A simple example symmetry

My new record ϵ -procedure for multiplying 5×5 matrices in only 90 multiplications (the previous low [26] was 100) is symmetric under the following 5-group: cyclically permute all the rows and columns of all 3 of the matrices, simultaneously. By postulating this group, the dimension of the search space was reduced from 6750 to 1350. Without the imposed symmetry, the search program probably would have taken centuries to find this.

5.5 Inexact formulae

It has not been pointed out previously, but actually Strassen-like basic matrix multiplication formulae which are *not* exact, and which also are *not* arbitrary precision approximate, *still* can lead, if they have small enough (but nonzero!) error bound, to asymptotically faster-than-naive matrix algorithms. This is with work now reckoned (more realistically) in terms of bit-operations rather than (less realistically) arithmetic operations. (The tradeoffs involved are, unfortunately, now considerably messier.) The key ideas are (1) recursion to increase the matrix sizes and (2) a different kind of recursion to increase the number of decimal places of precision. (Once one removes one’s blinders about arithmetic-count versus bit-op-count, one sees that really, both kinds of recursion are needed even with “exact” methods. Our point is that, thanks to this theorem, it is *not* necessary, even for theoretical purposes, to find, or prove one has found, ϵ -formulae; “inexact” formulae, which are in comparison trivial to find, are good enough.)

5.6 Rectangular matrices

There is a widespread delusion that it is best to find formulae, like Strassen’s, for multiplying two *square* matrices. In fact, the best known ϵ -method (140 muls) for multiplying 6×6 matrices is to use my new 14-mul formula for multiplying 3×3 by 3×2 matrices, in combination with the previously known [4] 10-mul way (or my computer’s discovery of a new inequivalent method with the same matrix sizes and multiplication counts, but a smaller error term) to multiply 2×2 by 2×3 matrices. In general, rectangular formulae are *more* useful, since they are more flexible, than square formulae.

6. New fast-matrix methods

There are three main algorithm design paradigms (described in 6.1, 6.2, and 6.3 below) which I have found effective in producing numerous new fast matrix algorithms. Many of them look likely to be competitive in practice.

6.1. Block-based methods

Partition matrices into half-size blocks, and find a way to perform the computation using block multiplications, block additions, and other fast matrix computations of various kinds (perhaps including recursive calls to the same computation we are trying to do on the full matrix) performed on blocks. (If matrix dimensions are odd, so that we cannot split into two equal halves, then ways, such as zero padding, dealing with “leftover rows” by “brute force,” and/or unequal splits, must be found to deal with that.) For example, I have found ways, using this paradigm, to do “fast multiple backsubstitution,” “fast Cholesky factorization,” “fast Sylvester equation solving,” and “fast evaluation of analytic functions of matrices.” Many of these new algorithms are quite simple and it is surprising or embarrassing that they had not been discovered previously, since this algorithm-design paradigm was previously known. (In contrast, some of the algorithms we mention below are considerably more clever.)

Since some of these methods are simple enough to explain briefly here, let’s do so.

Fast Multiple back substitution: Given a lower triangular $n \times n$ matrix L and a full-dense $n \times n$ matrix A , compute $B = L^{-1}A$. **Algorithm:** If n is small enough use conventional backsubstitution methods. Otherwise, write

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}, \quad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad (9)$$

where the upper-left blocks are $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$. Compute $B_{11} = L_{11}^{-1}A_{11}$ and $B_{12} = L_{11}^{-1}A_{12}$ recursively. Now compute the right hand sides of (by fast matrix multiplication) then solve (recursively) $L_{22}B_{22} = A_{22} - L_{21}B_{12}$ and $L_{22}B_{21} = A_{21} - L_{21}B_{11}$. The runtime $T(n)$ (we assume now, for simplicity, that n is even) obeys the recurrence

$$T(n) = 4T\left(\frac{n}{2}\right) + 2\text{MatMult}\left(\frac{n}{2}\right) + O(n^2). \quad (10)$$

If $\text{MatMult}(n) = cn^\omega$ then the solution is

$$T(n) = \frac{2c}{2^\omega - 4}n^\omega + O(n^2). \quad (11)$$

Fast Cholesky: Given a positive definite symmetric $n \times n$ matrix M , factor it as $M = LL^T$ where L is lower triangular. (Alternate task: factor it as $M = LDL^T$ where L is *unit* lower triangular and D is diagonal with positive elements. Either task is easy to solve if we can solve the other.)

Algorithm: If n is smaller than some constant threshold, use any Cholesky factorization algorithm. Otherwise: Write

$$M = \begin{bmatrix} M_{11} & M_{21}^T \\ M_{21} & M_{22} \end{bmatrix}, \quad L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}, \quad L^T = \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix} \quad (12)$$

where the upper-left blocks are $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$. Cholesky-factor $M_{11} = L_{11}L_{11}^T$ recursively. Now find $L_{21} = M_{21}L_{11}^{-T}$ with the aid of the previous fast multiple backsubstitution algorithm. Note that because M is positive definite, all its principal minors, hence M_{11} are too, hence L_{11} must be of full rank, hence its inverse exists. Finally, Cholesky factor $M_{22} - L_{21}L_{21}^T = L_{22}L_{22}^T$ recursively. The runtime $T(n)$ (we assume now, for simplicity, that n is even) now obeys a similar recurrence, also with solution $T(n) = O(n^\omega)$.

6.2 Agglomeration of elementary transformations

View the matrix computation as a sequence of “elementary” transformations. (E.g. for Gaussian Elimination with Partial Pivoting [GEPP], these are “Gauss transformations.” In the case of QR factorization, these are “Householder transformations,” and for certain other matrix computations they are “Jacobi-Givens transformations.”) Develop fast methods for “agglomerating” such transformations, and find ways to “reschedule” the computation so that large chunks of the elementary transformations may be performed “in bulk,” i.e. using a single agglomerated multi-transformation. Often there is a mixture of bulk and individual transformations being applied to different parts of the matrix, and with different amounts of bulkiness. The goal, which in many cases can be achieved, is to cause the net savings from such “economies of scale” actually to dominate the flop count, resulting in an overall asymptotic speedup. For example, I have successfully sped up GEPP and Householder-QR in this way to $O(n^\omega)$ flops, and I believe these algorithms probably will be practically useful. My basis for this belief: A “rescheduling” paradigm sometimes allows one to set up a 1-to-1 correspondence between certain flops in the conventional $O(n^3)$ algorithm, and in the sped-up algorithm – were the sped-up algorithm run using conventional matrix multiplier instead of a fast one. If then this matrix-multiply subroutine is ripped out and replaced by a fast one, things can only become better. Other times, one can set up, e.g., a (≤ 2)-to-1 correspondence, proving that the constant factor work-increase is not very large (and by terminating the recursions at the breakeven ns , one need never suffer any slowdown whatever versus the “slow” algorithm).

The details of my fast GEPP algorithm are too long to describe here, so let it suffice to describe two key ideas: (a) The k th column of the matrix, which needs to be updated by k Gauss transformations (corresponding to its own and all the previous pivots), is updated by means of g_j agglomerated 2^j -in-one transformations, where $g_j \in \{0, 1\}$ is the j th binary digit of k . The runtime analysis depends heavily on the fact that geometric series tend to be dominated by their largest terms. (b) Products of Gauss transformations have known sparsity structure allowing them to be agglomerated by use of a “product tree” with the aid of “fast matrix block multiplications.”

6.3 Bandwidth reduction paradigm

In many cases it is possible to reorganize a diagonalizing computation to reduce the bandwidth of the matrix by, e.g., a constant factor per “stage.” The flop counts of the later stages are dominated by the flop count in the first stage (the runtime analysis consists largely of summing a geometric series; the contribution of the first term to such series is well known to be within a constant factor of the entire sum), and that stage can be made to consist mainly of operations on large blocks (i.e. asymptotically fast operations). This idea can often be used in combination with the earlier ideas. For example, I have used this idea to speed up the tridiagonalization of a symmetric matrix by orthogonal similarity transformations to $O(n^\omega)$ flops (leading to an asymptotically-fast and stable symmetric eigendecomposition algorithm) and similarly have asymptotically-fast SVD algorithms. I am not sure whether algorithms based on this attack will be practically competitive. It appears that algorithms based on the bandwidth reduction idea involve a certain amount of “wasted work,” i.e. it is not just a “rescheduling,” and hence it is less clear that they can be practically competitive. For example, variants of the Bunch-Kaufman [6] and Aasen [1] algorithms for stabilized LZL^T factorization of symmetric indefinite matrices (L lower triangular, Z of low bandwidth) can be done “fast” by use of this paradigm, but it is not clear that the savings can be made worth the extra overhead for reachably small matrix sizes (indeed: will it be faster than GEPP without taking advantage of symmetry?) and its numerical stability needs to be investigated empirically. Regardless of that, most of these methods certainly are better than previous (in many cases impractical and unstable) “fast” algorithms of purely-theoretical value.

6.4. Other remarks

In most cases in which the conventional algorithm can be and has been proven to be “backwards stable,” its sped-up version based on the above techniques can be shown to be (what I call) “normwise backwards stable” – a weaker condition which still should be adequate for many uses.

I also have several new ad hoc speedup techniques and ad hoc new error analysis techniques, as well as some new empirical observations about numerical stability. Among these may be mentioned: new results on Leslie Foster’s [14] provably-stable “rook pivoting” showing it can be done with subcubic overhead in the *worst case*. New versions of “Danilevsky’s method [11]” for finding the characteristic polynomial of a general square matrix – my empirical results suggest that this old method has been unfairly maligned and ignored in the modern literature.

7. Goals

The great future linear algebra package FLAPACK (“fast lapack”). would simultaneously achieve fast asymptotic speed, and at least as good speed in practice (i.e. non-asymptotically) as current LAPACK, with improvement at practically reachable N . FLAPACK would offer analogous (although in some cases weaker) numerical-quality guarantees to LAPACK.

But it is far too much to ask (with the resources available to the PI) that he write FLAPACK. But at least, as a start in that direction – a pilot project – I would like to produce a good matrix multiplier code, and perhaps a few other codes good at least for demonstration and experimental purposes.

This research already is underway, and enough so that success seems assured. But much remains to be done: investigating and comparing alternative possible algorithms, writing and optimizing their expositions, optimizing their “constant factors,” and doing more research (both theoretical and experimental) both on algorithms and on their error-analyses.

8. Broader impacts, education, minorities, dissemination plan

Matrices have been called “the language of science” and unquestionably play the central role in every treatment of multidimensional data, multidimensional spaces, algebra, statistics, and group theory. This centrality alone seems to justify understanding them and creating tools of the utmost efficiency for manipulating them. Not coincidentally, LAPACK and its relatives (EISPACK, LINPACK) appear to be the most important scientific software packages in the world.

Here is a list of areas which would benefit from fast linear algebra for large matrices (another, albeit dated, list was made by Edelman [10]):

- Commercial software such as MATLAB, MATHEMATICA, and GAUSS,
- design and control of large structures, systems theory [21],
- economic modeling,
- computational chemistry and numerical solution of partial differential equations such as Navier-Stokes equations of fluid dynamics, elasticity, and Maxwell's equations of electromagnetics [8], especially including integral and transform methods,
- weather prediction,
- design and simulation of electronic devices and circuits,
- signal processing,
- statistical analysis of large data sets,
- analysis of radar cross sections of, e.g., stealth aircraft, or quantum scattering cross sections of e.g., molecules,
- integer factoring and therefore cryptography [29],
- optimization of large engineering designs,
- linear (and related kinds of) programming [2][35] and therefore methods for scheduling, solution of traveling salesman problems, etc.
- graph algorithms[32].

Temple's student body is 30,000 people from very diverse ethnic and cultural backgrounds. Temple is one of the few large urban universities in the US with sizable proportions of students from each of these 3 minority groups: Blacks, Hispanics, Asians. More than half are women. A large percentage are urban. One of Temple's missions is to extend the opportunity of a college education to those with potential but a perhaps unproven record. Temple has a strong tradition of educating children from blue-collar families. A third of incoming freshman are the first in their families to attend college.

This proposal will involve at least one graduate student at Temple in programming, algorithms, and numerical methods at a world class level. Plus, much of the computer equipment purchased under the grant, after (and to an extent during) use by the grantees, will undoubtedly be recycled into the general Temple math department diaspora to help all there.

The plan of "dissemination to the wider community," as the NSF puts it, is to publish papers, and to make our programs, papers, and data freely available over the internet. The latter is in fact essential because the data describing my matrix multiplication formulae is too voluminous and messy to publish non-electronically (many of the formulae currently are described by thousands of numbers). Most or all of the results (algorithms and data) should be accessible to non-specialists.

References

- [1] J.O. Aasen: On the reduction of a symmetric matrix to tridiagonal form, BIT 11 (1971) 233-242.
- [2] D.S. Atkinson & P.M. Vaidya: A cutting plane algorithm for convex programming that uses analytic centers, Mathematical Programming 69 (1995) 1-43.
- [3] Dario Bini & Grazia Lotti: Stability of fast algorithms for matrix multiplication, Numerische Mathematik 36 (1980) 63-72.
- [4] D.Bini, M.Capovani, F.Romani, G.Lotti: $O(n^{2.7799})$ complexity for approximate matrix multiplication, Information Proc. Lett. 8,5 (1979) 234-235.
- [5] D.Bini & V.Pan: Polynomial and matrix computations, Birkhauser 1994.
- [6] J.R.Bunch & L.Kaufman: Some stable methods for calculating inertia and solving symmetric linear systems, Math. Computation 31 (Jan. 1977) 163-179.
- [7] P.Bürgisser, M.Clausen, M.A.Shokrollahi (with T.Lickteig): Algebraic complexity theory, Springer (GMW #315) 1997.
- [8] G. Cheng, G. C. Fox, K. A. Hawick: A Scalable Paradigm for Effectively-Dense Matrix Formulated Applications, Syracuse University, Syracuse, NY 1994, <http://www.npac.syr.edu/users/gcheng/CEM/hpcn94.ps>
- [9] D.Coppersmith & S.Winograd: Matrix multiplication via arithmetic progressions, J.Symbolic Comput. 9,3 (1990) 251-280. Abbreviated version in ACM Sympos. Theor. Computer Sci. 19 (1987) 1-6.
- [10] Alan Edelman: Large Dense Numerical Linear Algebra in 1993 (survey), Int'l Journal of Super-computer Applications 7 (1993) 113-128.
- [11] V.N.Fadeeva: Computational methods of linear algebra, Dover 1959 (authorized translation by Curtis D. Benster).
- [12] L. Fejes Toth: Über eine Volumenabschätzung für Polyeder, Monatshefte Math. 64 (1960) 374-377.
- [13] R. Fletcher: Practical Methods of Optimization, (2nd ed.) John Wiley & Sons, 1987.
- [14] Leslie V. Foster: The growth factor and efficiency of Gaussian elimination with rook pivoting, J Comput Appl Math 86,1 (1997) 177-194; erratum 98,1 (1997) 177.
- [15] Matteo Frigo, C. E. Leiserson, Harald Prokop, Sridhar Ramachandran: Cache-Oblivious Algorithms, SFOCS 40 (1999) 285-297.
- [16] G.H.Golub & C.L.Van Loan: Matrix Computations, Johns Hopkins Univ. Press 1996.
- [17] Pascal Van Hentenryck: A Gentle Introduction to Numerica, Artificial Intelligence 103, 1-2 (1998) 209-235.
- [18] J.E.Hopcroft & L.R.Kerr: On the number of multiplications necessary for matrix multiplication, SIAM J. Appl. Math. 20,1 (1971) 30-36.
- [19] J.E.Hopcroft & J.Musinski: Duality applied to the complexity of matrix multiplication and other bilinear forms, SIAM J. Comput. 2,3 (1973) 159-173.

- [20] Rodney W. Johnson & Aileen M. McLoughlin: Noncommutative bilinear algorithms for 3×3 matrix multiplication, *SIAM J. Comput.* 15,2 (1986) 595-603.
- [21] T. Kailath: *Linear Systems*, Prentice Hall, 1980.
- [22] Igor Kaporin: A practical algorithm for faster matrix multiplication, *Numerical Linear Algebra with Appl.* 6,8 (1999) 687-700.
- [23] W.Keller-Gehrig: Fast algorithms for the characteristic polynomial, *Theor. Comput. Sci.* 36 (1985) 309-317.
- [24] J.D.Laderman: A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications, *Bull. Amer. Math. Soc.* 82,1 (1976) 126-128.
- [25] J.Laderman, V.Pan, X-H.Sha: On practical algorithms for accelerating matrix multiplication, *Linear Algebra Applic.* 162-164 (1992) 557-588.
- [26] O.M.Makarov: A noncommutative algorithm for multiplying 5×5 matrices using 100 multiplications, *USSR Comput. Phys. Math. Phys.* 27,1 (1987) 205-207.
- [27] M.Mitchell: *An Introduction to Genetic Algorithms*, MIT Press 1996
- [28] V.Y. Pan: *How to Multiply Matrices Faster*, Springer-Verlag (Lecture Notes in Computer Science #179) 1984; How can we speed up matrix multiplication, *SIAM Review* 26,3 (1984) 393-415.
- [29] *Cryptology and Computational Number Theory*, (C. Pomerance, ed.), AMS Proc. Symp. Applied Math. #42, Providence 1990.
- [30] Arnold Schönhage: Partial and total matrix multiplication, *SIAM J. Comput.* 10,3 (1981) 434-455.
- [31] M.P. Sears, K. Stanley, G. Henry: Application of a High Performance Parallel Eigensolver to Electronic Structure Calculations, *Supercomputing '98* www.cs.berkeley.edu/~stanley/gbell/index.html
- [32] R. Seidel: On the all-pairs-shortest-path problem in unweighted undirected graphs, *J. Comput. Syst. Sci.* 51 (1995) 400-403.
- [33] Volker Strassen: Gaussian elimination is not optimal, *Numerische Mathematik* 13,4 (1969) 354-356.
- [34] Ondrej Sykora: A fast non-commutative algorithm for matrix multiplication, *Mathematical Foundations of Computer Sci.* 6 (1977) 504-512. (Springer Lecture Notes in Computer Science #53).
- [35] Pravin M. Vaidya: Speeding-Up Linear Programming Using Fast Matrix Multiplication (Extended Abstract), *Sympos. Foundations Computer Sci.* 30 (1989) 332-337.
- [36] J.H. Wilkinson: *Rounding Errors in Algebraic Processes*, Prentice-Hall 1963.
- [37] S.Winograd: On multiplication of 2×2 matrices, *Linear Algebra & Applics.* 4 (1971) 381-388.

Biographical sketch of Warren D. Smith

Education:

Massachusetts Institute of Technology Math BS 1984

Massachusetts Institute of Technology Physics BS 1984

Princeton University Applied Math PhD 1988 (PhD Advisors: J.H.Conway & R.E.Tarjan.)

Employment:

Temple University 2002-

NEC Research Institute 1990-2002

Bell Labs Math'l Sciences Research Center 1988-1990

Web page: <http://www.neci.nec.com/homepages/wds/>. (Or <http://www.math.temple.edu/~wds;> may move.) Contains publications lists and electronically available papers.

10 selected publications in rough descending order of relevance:

Serge Plotkin, Satish Rao, Warren D. Smith: Shallow excluded minors and improved graph decompositions, SODA: ACM-SIAM Symposium on Discrete Algorithms (1994) 462-470.

A.E.Brouwer, N.J.A.Sloane, J.B.Shearer, W.D.Smith: A New Table of Constant Weight Codes, IEEE Transactions on Information Theory, IT-36 (1990) 1334-1380.

E.B.Baum & W.D.Smith: A Bayesian Approach to Game Playing, Artificial Intelligence 97 (1997) 195-242.

W.D. Smith: How to find Steiner minimal trees in d-space, Algorithmica 7 (1992) 137-177.

Satish B. Rao and Warren D. Smith: Improved approximation schemes for geometrical graphs via 'spanners' and 'banyans' ACM Symposium on Theory of Computing 30 (Dallas Texas, 1998) 540-550.

Warren D. Smith and Nicholas C. Wormald: Geometric separator theorems and applications, IEEE Symposium on Foundations of Computing 39 (Palo Alto, California, Nov. 1998) 232-243.

W.D. Smith & J.M.Smith: On the Steiner ratio in 3-space, J. Combinatorial Theory A 69,2 (1995) 301-332.

Ding-Zhu Du & Warren D. Smith: Disproofs of Generalized Gilbert-Pollak Conjecture on the Steiner Ratio in Three or More Dimensions, Journal of Combinatorial Theory A 74 (1996) 115-130.

Craig D. Hodgson, Igor Rivin, Warren D. Smith: A characterization of convex hyperbolic polyhedra and of convex polyhedra inscribed in the sphere, Bull. Amer. Math. Soc. (N.S.) 27,2 (1992) 246-251.

W. D. Smith: A lower bound for the simplicity of the N-cube via hyperbolic volumes, European J. Combinatorics 21,1 (2000) 131-137.

Smith has worked in math, computer science, physics, political science, and even biochemistry. Smith is probably best known for having invented the fastest known algorithms for exact (with N.C.Wormald) and ϵ -inexact (with S.B.Rao) traveling salesman tour in n -dimensional space (the case $n = 2$ had been in his 1988 PhD thesis). Smith produced the first proofs of the algorithmicity of quantum mechanics and the *non*algorithmicity of Newtonian mechanics (both with point masses and $1/r$ interparticle potentials) and a partial proof (some facts need to be assumed, but if those assumptions are untrue, that is also extremely interesting and devastating) of the *non*algorithmicity of the Navier-Stokes equations of fluid mechanics. Much of Smith's research has been computer-intensive.