Algorithms for Fast Matrix Operations

Gregory V. Bard

December 7, 2005

Abstract

Matrix multiplication, inversion, LU-factorization, QR-decomposition, and the finding of determinants and characteristic polynomials are important steps in solving many problems. They are, however, very expensive operations. For this reason, much research on these operations and their complexity was undertaken in the period 1969–1987, with four major results. First, these operations are all of equal asymptotic complexity. For this reason, most future research in this area was done on fast matrix multiplication alone. Second, algorithms faster than the naive are possible, principally by finding a shortcut for a small matrix, and applying it recursively for larger ones. Third, partial matrix algorithms can be used for total matrices. Fourth, proofs can be written which give upper bounds for the complexity of matrix operations while not giving an algorithm, but rather an algorithm for finding the algorithm.

1 Introduction

This paper deals with the methods discovered in the period 1969–1987 for operating upon dense, unstructured, total[1] matrices. When matrices were first conceived in the mid 19-th Century, the algorithms that we know as Gauss-Jordan Elimination and matrix multiplication were assumed to be optimal.

Volker Strassen surprised the community by publishing an algorithm that multiplies a 2×2 matrix in 7 multiplications, rather than 8 [10]. At the time, multiplications were very expensive operations compared to subtractions and additions, and so the number of multiplications was a good choice for quantifying the complexity of these operations [1]. This algorithm can be extended recursively (as will be shown below) to handle matrices of any size. The complexity of the operation is $O(n^{2.81})$ for $n \times n$ matrices, which is not only unusual for beating cubic time, but also for having an irrational exponent (which will be derived below).

That paper also showed how the algorithm could be used to invert matrices, or find their determinants. These methods did not immediately apply to matrices in general, even over the real numbers, but it suggested that perhaps inverting a matrix, multiplying a matrix, and finding its determinant might have similar complexity. As it turns out, it was proved over the course of several papers that LU-factorization, QR-decomposition, finding the characteristic polynomial, as well as multiplication, inversion and determinants can all be done in time $O(M(n))$ [6],

[1] Matrix algorithms where certain fixed elements of a matrix are assumed to be zero are called partial algorithms. Total matrix algorithms are those which are not partial. This is not to be confused with sparse matrix operations, where most entries are zero but the positions of the zeroes is flexible.

where $M(n)$ is the time to multiply two $n \times n$ matrices. These results were originally proved over $\mathbb{R}$ and $\mathbb{C}$ [5] but Aho, Hopcroft and Ullman [1] proved them over fields in general in their landmark textbook. There is not space enough in this paper (or its margins) to prove these results, but one example is below, taken from [5].

Suppose we have a fast operation for inverting $n \times n$ matrices that runs in time $I(n)$. If we want to calculate the matrix product AB, we can construct

$$\begin{bmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{bmatrix}$$

This demonstrates that we can multiply two $n \times n$ matrices in time $I(3n)$, or $M(n) \leq I(3n)$. Inverting matrices with Gaussian Elimination is $O(n^3)$ (though faster algorithms exist) and so $I(3n) \leq 27I(n)$ and $M(n) \leq 27I(n)$, finally $M(n) = O(I(n))$. A more complex proof shows $I(n) = O(M(n))$, resulting in $M(n) = \Theta(I(n))$ and $I(n) = \Theta(M(n))$.

Ten years went by without a new algorithm, but then a flurry of papers 1979–1987 resulted in a series of new results which eventually showed the complexity of multiplying matrices is $O(n^{2.376})$. In particular, Aho, Hopcroft and Ullman, moved the discussion from $\mathbb{R}$ and $\mathbb{C}$ to operations over a general ring [1]. Later, Bini, Capovani, Lotti and Romani found a method for achieving $O(n^{2.7799})$ [2] but more importantly discovered the technique of approximate tensors, or border rank, for matrix multiplication. Schonhage generalized this idea, and also showed that partial algorithms can be used for total matrices [8], which will be the focus of this paper. He achieves $O(n^{2.695})$. Coppersmith and Winograd in 1982 proved a very interesting theorem which shows that for any matrix multiplication algorithm, there is one that is asymptotically faster. In plainer words, that means that the exponent of matrix multiplication is a limit point [3]. Victor Pan, both in earlier papers and his monograph of 1984, created the notation of trilinear maps which makes this field much easier to understand, and achieves a bound of $O(n^{2.52})$, though the algorithm given is one for generating a matrix multiplication algorithm. The two capstone papers of this field, are Coppersmith and Winograd's 1987 paper, which has the best exponent so far of $O(n^{2.367})$, and Volker Strassen's theoretical work which generalizes the notions of border rank and tensor rank to tackle problems related and unrelated to matrix multiplication [9], and explains some previously known results in terms of topology and algebraic geometry.

2 Strassen's Algorithm

In 1969 Strassen [10] found the following method for multiplying two matrices of size 2×2:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} r & s \\ t & u \end{bmatrix}$$

$$\begin{aligned} P_1 &= a(f-h) & P_2 &= (a+b)h \\ P_3 &= (c+d)e & P_4 &= d(g-e) \\ P_5 &= (a+d)(e+h) & P_6 &= (b-d)(g+h) \end{aligned}$$

$$P_7 = (a-c)(e+f)$$
$$r = P_5 + P_4 - P_2 + P_6 \qquad s = P_1 + P_2$$
$$t = P_3 + P_4 \qquad u = P_5 + P_1 - P_3 - P_7$$

As can be seen, the calculations of $P_1, \ldots, P_7$ require exactly one multiplication each. All other steps are additions, which are quadratic and therefore insignificant, since we assume $M(n) = \omega(n^2)$.

Also, the commutative property of multiplication over a field is not used in this calculation, nor is division used. Therefore it works over a ring such as the ring of $n \times n$ matrices over a field. Therefore, one can write a $2n \times 2n$ matrix as a 2×2 matrix whose 4 entries are $n \times n$ matrices and use the above algorithm.

Therefore we've shown that $M(n) = 7M(n/2) + O(n^2)$, which can be repeated to get $M(n) = 49M(n/4) + O(n^2)$ and $M(n) = 343M(n/8) + O(n^2)$ and so on. How many times can we cut n in half before reaching 1? The answer is $\log_2 n$ of course!

Thus we get $M(n) = 7^{log_2 n} M(1)$. But, a field element multiplication can be calculated in one step. Thus $M(n) = \Theta(7^{log_2 n}) = \Theta(n^{log_2 7}) = \Theta(n^{2.81})$.

Practicality When this algorithm was first published, while it showed that this method of matrix multiplication was asymptotically more efficient than the naive method, it was not widely adopted. The first reason for this is that the cross-over point, or smallest problem for which this algorithm is faster than the naive one, was assumed to be high. Second, the closeness of $log_2 7 \approx 2.81$ to three means that Strassen's algorithm's performance improves relative to the naive at a slow rate, and so the benefit is only felt at large size. Third, it was assumed that the numerical errors must be large, since no simple way of modeling that error was available. The first of these notions turned out to be false, as was proved via numerical experiment [6]. Such experiments also showed that while the instability makes it inappropriate for some applications, it is acceptable for most. The CRAY UNICOS operating system adopted Strassen's algorithm for matrices larger than 130×130 [6]. However, the second notion is true, and Strassen's Algorithm is twice as fast as the naive approach only at about 5000×5000. Nonetheless, over finite fields where there is no rounding error, and where cryptanalysis sometimes calls for million by million matrices, Strassen's Algorithm would be about 5.5 times faster.

3 Tensor Notation

Just as every matrix is a linear transformation, from U to V for example, and thus an element of the tensor product $U^* \otimes V$, then matrix multiplication can be expressed as an element of $U^* \otimes V^* \otimes W$ [7]. Generally, the linear functionals from U^* and V^* "select" elements from their respective matrices, and then the vector in W is the "address" for the destination of this calculation. This is easier explained by example. Also, it should be noted that it is customary to drop the $\otimes$ symbol when writing these tensors. The rank of a tensor is the minimum number of simple tensors required to write it.

Every tensor is a sum of simple tensors, each of which is one calculation. The symbol a_{11} or b_{22} represents the linear functional which returns the value of the cell a_{11} or b_{22}. The symbol c_{33} signifies that the calculation implied by that simple tensor should be stored in c_{33}. The

format in use here is called trilinear form, because if the a_{ij}'s, b_{ij}'s and c_{ij}'s are all treated as indeterminants, then the tensor becomes a trilinear map $f(\vec{a}, \vec{b}, \vec{c})$. This approach is due to [2, 7].

3.1 Naive Algorithm in Trilinear Form

The following is the naive matrix multiplication algorithm in trilinear form.

$$\begin{aligned}
\begin{bmatrix} a_{11} & a_{12} \\ b_{11} & b_{12} \end{bmatrix} \begin{bmatrix} a_{21} & a_{22} \\ b_{21} & b_{22} \end{bmatrix} &= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix} \\
f(\vec{a}, \vec{b}, \vec{c}) &= a_{11}b_{11}c_{11} + a_{12}b_{21}c_{11} \\
&\quad + a_{11}b_{12}c_{12} + a_{12}b_{22}c_{12} \\
&\quad + a_{21}b_{11}c_{21} + a_{22}b_{21}c_{21} \\
&\quad + a_{21}b_{12}c_{22} + a_{22}b_{22}c_{22}
\end{aligned}$$

Here we see that a triad of a linear combination of a_i's (here just one a_i) times a linear combination of b_j's (here just one b_j) times a single c_k represents a computer command similar to

$\mathtt{c_k} += \mathtt{a_i b_j}$

There is no claim that 8 is the shortest number of terms needed to write this form. In fact, it is not, because Strassen's Algorithm computes *the same trilinear form* with seven terms, see below.

3.2 Strassen's Algorithm in Trilinear Form

One can write Strassen's Algorithm in trilinear form as follows [10]. Here we see there are 7 terms instead of the expected 8, and this represents the increased efficiency.

$$\begin{aligned}
f(\vec{a}, \vec{b}, \vec{c}) &= \underbrace{(a_{11})(b_{12} - b_{22})}_{P_1}(c_{12} + c_{22}) + \underbrace{(a_{11} + a_{12})(b_{22})}_{P_2}(-c_{11} + c_{12}) + \\
&\quad \underbrace{(a_{21} + a_{22})(b_{11})}_{P_3}(c_{21} - c_{22}) + \underbrace{(a_{22})(b_{21} + b_{11})}_{P_4}(c_{11} + c_{21}) + \\
&\quad \underbrace{(a_{11} + a_{22})(b_{11} + b_{22})}_{P_5}(c_{11} + c_{22}) + \underbrace{(a_{12} - a_{22})(b_{21} + b_{22})}_{P_6}(c_{11}) + \\
&\quad \underbrace{(a_{11} - a_{21})(b_{11} + b_{12})}_{P_7}(-c_{22})
\end{aligned}$$

Take for example, the first term $(a_{11})(b_{12} - b_{22})(c_{12} + c_{22})$. Here we see that a triad of a linear combination of a_i's times a linear combination of b_j's times a linear combination of c_k's represents a computer command similar to

$\mathtt{c_{12}} += \mathtt{a_{11}}(\mathtt{b_{12}} - \mathtt{b_{22}})$

$\mathtt{c_{22}} += \mathtt{a_{11}}(\mathtt{b_{12}} - \mathtt{b_{22}})$

... other than the fact that one would compute $a_{11}(b_{12}-b_{22})$ only once and store it in a temporary variable to avoid the wastefulness of computing it twice.

4 Border Rank and Epsilons

Consider the following partial matrix multiplication, as a special case of the 2×2 by 2×2 total matrix multiplication. (This follows [8].)

$$\begin{bmatrix} a & b \\ c & 0 \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce & cf \end{bmatrix}$$

An algorithm might be

$$\begin{array}{cc} c_{11} \leftarrow a_{11}b_{11}+a_{12}b_{21} & c_{12} \leftarrow a_{11}b_{12}+a_{12}b_{22} \\ c_{21} \leftarrow a_{21}b_{11} & c_{22} \leftarrow a_{21}b_{12} \end{array}$$

In trilinear form this would be

$$\begin{aligned} t &= (a_{11}b_{11}+a_{12}b_{21})c_{11}+(a_{11}b_{12}+a_{12}b_{22})c_{12}+a_{21}b_{11}c_{21}+a_{21}b_{12}c_{22} \\ &= a_{11}b_{11}c_{11}+a_{12}b_{21}c_{11}+a_{11}b_{12}c_{12}+a_{12}b_{22}c_{12}+a_{21}b_{11}c_{21}+a_{21}b_{12}c_{22} \end{aligned}$$

Since there are six trilinear terms in the final sum, one knows the rank is at most six. Note also that this can be thought of as the trilinear form for naive matrix multiplication, with some of the terms zeroed out (and therefore dropped.) It is therefore termed a sub-tensor form of the original. Clearly if g is a sub-tensor of f, then the rank of g cannot be higher than the rank of f.

4.1 Tensor Product of two Forms

At times the tensor product of two forms will be calculated, which represents repeated evaluation recursively (this will be demonstrated by example, below). The tensor product is calculated as usual, but one must merely distinguish between the a_{ij}'s of one form and those of the other with some sort of mark, like a prime, then calculate the tensor product as normal.

The isomorphic commutativity of the tensor product, yields $(U_1^*\otimes V_1^*\otimes W_1)\otimes(U_2^*\otimes V_2^*\otimes W_2)\approx(U_1^*\otimes U_2^*)\otimes(V_1^*\otimes V_2^*)\otimes(W_1\otimes W_2)$. Thus in the tensor product, each simple tensor can be written as $(a_{ij}a'_{i'j'})(b_{mn}b'_{m'n'})(c_{rs}c'_{r's'})$.

All that is needed is an explanation of the meaning of the symbol $c_{ij}c'_{i'j'}$. If c_{ij} comes from a 3×5 matrix and $c'_{i'j'}$ from a 4×7 matrix, then the matrix represented by all $c_{ij}c'_{i'j'}$ is a 12×35 matrix, written as a 3×5 matrix whose elements are 4×7 matrices. Thus $c_{ij}c'_{i'j'}$ implies the submatrix represented by c_{ij}, and the entry represented by $c'_{i'j'}$. In particular, $c_{23}c'_{45}$ represents the submatrix is the second row, third column of submatrices, and the element in the fourth row and fifth column of that submatrix, which in normal notation would be the seventh row and fifteenth column. For this reason, $c_{23}c'_{45}$ could be replaced with $c''_{7,15}$, likewise for all the $a_{ij}a'_{i'j'}$ and $b_{ij}b'_{i'j'}$ [9].

Some Extra Properties of Rank The following properties are not surprising, but will be useful below. The above multiplication algorithm results in rr' terms for the new trilinear form, with r and r' being the ranks of the original trilinear forms. There may be a way to write it with fewer terms, however.

$$\begin{aligned} rk(t \otimes t') &\leq rk(t)rk(t') \\ rk(t^{\otimes s}) &\leq rk(t)^s \end{aligned}$$

Above

$$t^{\otimes s} = \underbrace{t \otimes t \otimes t \otimes \cdots \otimes t}_{s \text{ times}}$$

4.2 The Dual Numbers

In the days of Newton and Liebnitz, there were no limits in the epsilon/delta sense as we know them today. Instead, calculus was performed via infinitesimals. Algebraically, this can be done by adjoining an ϵ to a field, declaring it a transcendental element. The resulting field's elements can be written as rational functions, that is ratios of polynomials with coefficients from the original field. (These are called quolynomials in some German papers [9, 8].)

In fields of characteristic zero, one can then perform operations, and at the end take the limit as epsilon goes to zero. In finite fields, however, $F[\epsilon]/(\epsilon^k)$ will suffice. That is to say, taking the ring of polynomials in ϵ and with coeffecients in F, but modulo ϵ^k, forces all multiples of ϵ^k to be equal to zero. Naturally this is no longer a field since $\epsilon^{k-1}\epsilon = 0$. If $k = 2$, the term "dual numbers" is used to describe this ring.

4.3 Schonhage's First Algorithm

The following is a creative trick, from [8]:

$$\begin{gathered} t' = (a_{12} + \epsilon a_{11})(b_{12} + \epsilon b_{22})c_{21} + (a_{21} + \epsilon a_{11})b_{11}(c_{11} + \epsilon c_{12}) - \\ a_{12}b_{12}(c_{11} + c_{21} + \epsilon c_{22}) - a_{21}(b_{11} + b_{12} + \epsilon b_{21})c_{11} + \\ (a_{12} + a_{21})(b_{12} + \epsilon b_{21})(c_{11} + \epsilon c_{22}) = \end{gathered}$$

$$\begin{gathered} \cancel{a_{12}b_{12}c_{21}} + \epsilon a_{11}b_{12}c_{21} + \epsilon^2 a_{11}b_{22}c_{21} + \epsilon a_{12}b_{22}c_{21} + \cancel{a_{21}b_{11}c_{11}} + \epsilon a_{11}b_{11}c_{11} + \\ \epsilon a_{21}b_{11}c_{12} + \epsilon^2 a_{11}b_{11}c_{12} + -\cancel{a_{12}b_{12}c_{11}} - \cancel{a_{12}b_{12}c_{21}} - \cancel{\epsilon a_{12}b_{12}c_{22}} - \cancel{a_{21}b_{11}c_{11}} \\ -\cancel{a_{21}b_{12}c_{11}} - \cancel{\epsilon a_{21}b_{21}c_{11}} + \cancel{a_{12}b_{12}c_{11}} + \cancel{a_{21}b_{12}c_{11}} + \epsilon a_{12}b_{21}c_{11} + \cancel{\epsilon a_{21}b_{21}c_{11}} \\ \epsilon a_{21}b_{12}c_{22} + \cancel{\epsilon a_{12}b_{12}c_{22}} + \epsilon^2 a_{12}b_{21}c_{22} + \epsilon^2 a_{21}b_{21}c_{22} \end{gathered}$$

$$\begin{gathered} = \epsilon[a_{11}b_{12}c_{21} + a_{12}b_{22}c_{21} + a_{11}b_{11}c_{11} + \\ a_{21}b_{11}c_{12} + a_{21}b_{12}c_{22} + a_{12}b_{21}c_{11}] + \epsilon^2(\cdots) \end{gathered}$$

This trilinear form t' has rank five over $F(\epsilon)$. In order to perform a partial matrix multiplication, we had a rank six trilinear form (without epsilons) called t. Now we have a trilinear form with the property that

$$t' = \epsilon t + \epsilon^2(\cdots).$$

The resemblance to calculus is interesting.

$$(x+\epsilon)^2 - x^2 = 2x\epsilon + \epsilon^2(1)$$

Just as $\frac{(x+\epsilon^2-x^2)}{\epsilon}$ is an approximation to the derivative $2x$ then one can say that $\frac{t'}{\epsilon}$ is an approximation to t. In fields like $\mathbb{Q}, \mathbb{R}, \mathbb{C}$, the following ideas would work:

For many small values of ϵ converging in a sequence toward 0, the value of t' approximates that of $t\epsilon$. The forms which are given by substituting these small values for ϵ, are a sequence of rank five forms which approximate $t\epsilon$, a rank six form. One could say that though $t\epsilon$ is a rank six form, it is surrounded by rank five forms in a tiny neighborhood. In practice, over $\mathbb{R}$ or $\mathbb{C}$, choose a small value for ϵ, evaluate $\frac{t'}{\epsilon}$ for that ϵ.

Thus the rank of t is six, while t' demonstrates that an ϵ-first-degree approximation exists of t that is rank five. The minimal rank of all ϵ-nth-degree approximations, for all n, is called the border rank. Thus the border rank of t', denoted $\underline{rk}(t')$ is at most five.

4.4 What about the epsilon squared terms?

We have two choices for isolating the ϵ term.

1. Choose ϵ to be small, and then ϵ^2 will be rather tiny. (Works because $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ are fields on which an ordering or metric can be used.)

2. Evaluate the tensor product symbolically, and only keep the terms with the correct degree of ϵ.

3. Work in $F[\epsilon]/\epsilon^2$, where all multiples of ϵ^2 are congruent to zero.

The problem with the first idea is that there will still be error, and we'd like our algorithms to be exact. The problem with the third idea is that ϵ is a zero divisor in this case, $\epsilon^{k-1}\epsilon = 0$, and zero divisors never have inverses. If $\delta\epsilon = 1$ then $\delta\epsilon\epsilon^{k-1} = \epsilon^{k-1}$ and $\delta 0 = \epsilon^{k-1}$ and $0 = \epsilon^{k-1}$. However, that statement is false, so δ does not exist. Since ϵ has no inverse, we cannot divide by it.

It seems that for any sufficiently small ϵ (in $\mathbb{Q}, \mathbb{R}, \mathbb{C}$) that $\frac{t'}{\epsilon}$ is a good formula for t, but that an explosion takes place if we use $\epsilon = 0$. Strassen uses an argument with the Zariski Topology (where closed sets are only those which can be written as the zeroes of a set of polynomials) to show that the formula will still work if $\epsilon = 0$.

The problem with the second idea is more subtle. While a symbolic calculation need be executed only once, the algorithms have cross-over points that require scaling up to higher tensorial powers. Multiplying two ten-degree polynomials requires 20 field multiplications, and as the degree gets higher, more work is required. This adds to the complexity of the system, and it is not clear that such an approach would be feasible, let alone more efficient than the naive algorithm.

Instead, select k values in the field (or its algebraic closure, which is always infinite), and evaluate $\frac{t'}{\epsilon}$ with ϵ set equal to these values. Since the tensor elements are $k-1$ degree polynomials, then a Lagrange Interpolator will approximate the polynomial exactly [8]. Create a Lagrange polynomial in a dummy variable x, then evaluate this polynomial at $x = 0$ (representing $\epsilon = 0$). This will give the exact value of the tensor $\frac{t'}{\epsilon}$ at $\epsilon = 0$. The only downside is the resulting expression will have k times as many terms as the original.

4.5 Efficiency

In our example, we started with a rank six trilinear form t, and found an approximation t' of rank five, provided we could cancel terms which were multiples of ϵ^2 (keeping terms which were multiples of ϵ.) Then this means we need $k = 2$ times as many terms for our Lagrange Interpolator, or 10 terms!

Now suppose that we consider $t^{\otimes s}$ instead. This will have rank less than or equal to 6^s. Likewise $(t')^{\otimes s}$ will have rank less than or equal to 5^s, with many ϵ's. Since t' is a good approximation for $t\epsilon$, then $(t')^{\otimes s}$ will be a good approximation for $t^{\otimes s}\epsilon^s$. Thus we should cancel terms which are multiples of ϵ^{s+1}.

This means our tensor in trilinear form will be an s degree polynomial, and we will need $s+1$ data points. Furthermore the Lagrange Interpolator will have $s+1$ times as many terms as $(t')^{\otimes s}$ in trilinear form. Therefore it will have $(s+1)5^s$ terms.

- If $s = 15$ then $t^{\otimes 15}$ has rank $6^{15} = 4.70 \times 10^{11}$ but the Lagrange Interpolator of $(t')^{\otimes 15}$ would have rank $16(5^{15}) = 4.88 \times 10^{11}$.
- If $s = 16$ then $t^{\otimes 16}$ has rank $6^{16} = 2.82 \times 10^{12}$ but the Lagrange Interpolator of $(t')^{\otimes 15}$ would have rank $17(5^{16}) = 2.59 \times 10^{12}$.
- If $s = 30$ then $t^{\otimes 30}$ has rank $6^{30} = 2.21 \times 10^{23}$ but the Lagrange Interpolator of $(t')^{\otimes 15}$ would have rank $31(5^{30}) = 2.89 \times 10^{22}$. The savings for $s = 30$, which signifies multiplying pairs of dense square matrices with 2^{30} rows (slight over one million), is a factor of 7.66, which is very significant.

However, the reason that such high cross-over is required is that rank 6 and rank 5 are not very different. For multiplying 3×3 matrices, another trick converts a rank 27 tensor into a rank 21 tensor, with both ϵ and ϵ^2 terms. The cross-over is found by $(1+2s)(21^s) \leq 27^s$, or $s > 14$.

5 Using Partial Operations for Total Operations

In this section, the third tensorial power will be used, instead of the 16th (as [8]), because it would be hard to fit a matrix of 64,000 rows and columns on a single page. However, the

ideas are the same, other than the fact that the algorithm here will be slower than the naive algorithm. Since we started with a $k \times m$ by $m \times n$ into $k \times n$ algorithm, with $k = m = n = 2$, the third tensorial power [2] is an algorithm for computing $k^s \times m^s$ by $m^s \times n^s$ into $k^s \times n^s$ of the form

$$\begin{bmatrix} X & X & X & X & X & X & X & X \\ X & 0 & X & 0 & X & 0 & X & 0 \\ X & X & 0 & 0 & X & X & 0 & 0 \\ X & 0 & 0 & 0 & X & 0 & 0 & 0 \\ X & X & X & X & 0 & 0 & 0 & 0 \\ X & 0 & X & 0 & 0 & 0 & 0 & 0 \\ X & X & 0 & 0 & 0 & 0 & 0 & 0 \\ X & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} X & X & X & X & X & X & X & X \\ X & X & X & X & X & X & X & X \\ X & X & X & X & X & X & X & X \\ X & X & X & X & X & X & X & X \\ X & X & X & X & X & X & X & X \\ X & X & X & X & X & X & X & X \\ X & X & X & X & X & X & X & X \\ X & X & X & X & X & X & X & X \end{bmatrix} = 8 \times 8$$

It would be a great coincidence if our target matrices for multiplication were of this form. Therefore one must hope to produce an algorithm for total matrix multiplication. Denote the cells that are not forced to be zero "variable cells."

The trilinear form to compute

$$\begin{bmatrix} 0 & X & X & 0 & X & 0 & 0 & 0 \\ 0 & 0 & X & 0 & X & 0 & 0 & 0 \\ 0 & X & 0 & 0 & X & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & X & 0 & 0 & 0 \\ 0 & X & X & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & X & 0 & 0 & 0 & 0 & 0 \\ 0 & X & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ X & X & X & X & X & X & X & X \\ X & X & X & X & X & X & X & X \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ X & X & X & X & X & X & X & X \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = 8 \times 8$$

is clearly a special case of our third tensorial power, and therefore its trilinear form is a sub-tensor form of $t'^{\otimes 3}$. These additional zeroes were not selected at random. First, partition $s = 3$ into $m = 2$ pieces, via $3 = 2 + 1$ in this case, or $s = s_1 + s_2 + \cdots + s_m$ in general.

Now, consider the set $\mathcal{M}$, to be the set of all s-dimensional vectors, with elements in $1, 2, \ldots, m$, such that exactly s_i of the elements have i as a value. In our case, the set of all 3-dimensional vectors, with elements 1 or 2, such that exactly 2 elements are 1, and 1 element is 2. Clearly $\mathcal{M} = \{(2,1,1); (1,2,1); (1,1,2)\}$, and in general

$$|\mathcal{M}| = M = \frac{s!}{s_1! s_2! \cdots s_m!}$$

[2] The left matrix is given by the Kronecker Matrix product

$$\begin{bmatrix} X & X \\ X & 0 \end{bmatrix} \otimes \begin{bmatrix} X & X \\ X & 0 \end{bmatrix} \otimes \begin{bmatrix} X & X \\ X & 0 \end{bmatrix}$$

The vector $(2,1,1)$ signifies the second submatrix, first subsubmatrix, and first column of that in A, or row in B. This is column 5 or row 5 in normal notation. Its membership in $\mathcal{M}$ means it will not be zeroed out. All other columns of A or rows of B will be zeroed out. Dropping the all-zero rows and all-zero columns,

$$\begin{bmatrix} X & X & X \\ 0 & X & X \\ X & 0 & X \\ 0 & 0 & X \\ X & X & 0 \\ 0 & X & 0 \\ X & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} X & X & X & X & X & X & X & X \\ X & X & X & X & X & X & X & X \\ X & X & X & X & X & X & X & X \end{bmatrix} = 8 \times 8$$

gives us an 8×3 times a 3×8, with exactly 4 variable entries in the columns of the left matrix. In general this would be $k^s \times M$ by $M \times n^s$ into $k^s \times n^s$. The fact that all the columns of A and all the rows of B have the same number of entries ($K = 4$ and $N = 8$) is the reason $\mathcal{M}$ was constructed. We will take advantage of this algorithm to create an algorithm for total $U = 4 \times 3$ times $V = 3 \times 8$, yielding a $W = 4 \times 8$, or $K \times M$ by $M \times N$ into $K \times N$ in general. Note:

$$\begin{aligned} K &= k_1^{s_1} + k_2^{s_2} + \cdots k_m^{s_m} \\ N &= n_1^{s_1} + n_2^{s_2} + \cdots n_m^{s_m} \end{aligned}$$

To facilitate this, generate matrices G, H, Q, R such that:

$$\begin{aligned} U_{4\times3} &= G_{4\times8} A_{8\times3} \\ A_{8\times3} &= H_{8\times4} U_{4\times3} \\ V_{3\times8} &= B_{3\times8} Q_{8\times8} \\ B_{3\times8} &= V_{3\times8} R_{8\times8} \end{aligned}$$

$$U_{4\times3} V_{3\times8} = G_{4\times8} A_{8\times3} B_{3\times8} Q_{8\times8} = G_{4\times8} C_{8\times8} Q_{8\times8} = W_{4\times8}$$

Here the matrices G and Q will be used to "convert" the partial $AB = C$ algorithm into an algorithm to find $(GA)(BQ) = UV = W$. The matrices G and Q, H and R are calculated once, and are not large. (Actually in this case it turns out that R and Q are the 8×8 identity matrix. In other cases, this may not be possible.) So long as G and Q are full-rank, then they are injections, and so it is easy to find H and R.

A problem with this technique is that it has matrix multiplications inside it. However, our algorithm in trilinear form is written as a sum of terms, each with a constant, a_i, b_j and c_k. The

fact that $GCQ = W$ means that using G and Q, we can write the w_m's as linear combinations of the C_k's, and so make a trilinear form with a_i's, b_j's, and w_m's.

Likewise the fact that $GA = U$ and $BQ = V$ means that we can write the u_n's as linear combinations of the a_i's and the v_t's as linear combinations of the b_j's, thus making a trilinear form with u_n's, v_t's and w_m's. Call this new tensor t''.

The taking of linear combinations will not affect rank, and therefore the rank of this tensor, $\underline{rk}(t'') = \underline{rk}(t'^{\otimes 3}) \leq \underline{rk}(t')^3 \leq 5^3 = 125$, and it multiplies total 4×3 by 3×8 matrices to get 4×8 matrices. In general, the rank of the tensor would be r^s, where r was the rank of the original tensor for $k \times m$ times $m \times n$ into $k \times n$.

By permuting the variables a, b, and c, for a trilinear form that does matrix multiplication, one gets another trilinear form, that does matrix multiplication on the sizes likewise permuted. Thus by changing variables, we can get 3×4 by 4×8 is 3×8 or 8×4 by 4×3 is 8×3. Tensoring these three together, we get $(4 \cdot 3 \cdot 8 =) 96 \times 96$ times 96×96 yields 96×96, and has border rank at most $125^3 = 1,953,125$ instead of the trivial rank $96^3 = 884,736$. If s were larger than the cross-over point, however, this algorithm would beat the trivial one.

In the general case the rank becomes r^{3s}, and the algorithm is for $P \times P$ times $P \times P$ matrices, with

$$P = \frac{s!}{s_1! s_2! \cdots s_m!} (k_1 n_1)^{s_1} (k_2 n_2)^{s_2} \cdots (k_m n_m)^{s_m}$$

5.1 Exponent Calculation

Here the symbol $< A, B, C >$ represents matrix multiplication of type $A \times B$ by $B \times C$ into $A \times C$. The exponent of matrix multiplication induced by a recursive algorithm is given as

$$\omega \leq \frac{\ln rk(t)}{\ln P}$$

where a short-cut to execute $< P, P, P >$ matrix multiplication requires $rk(t)$ simple tensors. (That is, the non-border rank, thus without epsilons.) For example, Strassen had rank 7, and $P = 2$, thus $\omega \leq \frac{\ln 7}{\ln 2} = \log_2 7 \leq 2.81$

The above formula requires a P and this in turn depends on the selection of a partition. The optimal partition is not easily determined. However, a "good" bound along with an algorithm that achieves it, is given. Simply choose the rows/columns associated with the largest monomials in the multinomial expansion of $(k_1 n_1 + k_2 n_2 + \cdots + k_m n_m)^s$.

Recall that f is the rank of the trivial tensor for the $< k, m, n >$ starting partial operation, and so the rank of $< k^s, m^s, n^s >$ will be f^s. The above sum also equals f^s, and it has $\binom{s+m-1}{m-1}$ terms, so the average value of a term is $(f^s)/\binom{s+m-1}{m-1}$

And since that's the average value of a term in that formula, there exists one term of at least that value. (i.e., the worse case is if they are all equal, otherwise there is one above average, etc...) That average value, if taken as a value for P, will always be an available option.

$$\begin{aligned} P &\geq \frac{f^s}{\binom{s+m-1}{m-1}} \\ \ln P &\geq s \ln f - \ln \binom{s+m-1}{m-1} \\ \ln P &\geq s \ln f - (m-1)\ln(s+m-1) \end{aligned}$$

- The border rank of the $< k, m, n >$ was at most r, with degree h and so border rank of the $< k^s, m^s, n^s >$ was less than r^s, with degree hs. Then $< k^s, M, n^s >$ was a subtensor, and so has border rank at most r^s also, and degree hs. The total matrix multiplication $< K, M, N >$ was an extended linear combination of this, and so the approximate rank and degree do not change. The final square matrix multiplication was $< P = KMN, P, P >$ and so had border rank at most r^{3s} and degree $3hs$.
- Recalling $rk(t) \leq (1+2h)r_h(t)$ then

$$\begin{aligned} rk(< P, P, P >) &\leq (1 + \underbrace{6hs}_{\text{2x degree}})(r^{3s}) \\ \ln rk(< P, P, P >) &\leq \ln(1+6hs) + 3s \ln r \end{aligned}$$

- The ratio of these two logs is

$$\omega \leq \frac{\ln rk(< P, P, P >)}{\ln P} \leq \frac{\ln(1+6hs) + 3s \ln r}{s \ln f - (m-1)\ln(s+m-1)}$$

- Taking the limit as $s \to \infty$

$$\omega \leq \frac{3s \ln r}{s \ln f} = \frac{3 \ln r}{\ln f}$$

- In our case, $f = 6$, and $r = 5$, so $\omega \leq \frac{3 \ln 5}{\ln 6} \leq 2.695$, or $M(n) = O(n^{2.695})$.

References

[1] A. Aho, J. Hopcroft, and J. Ullman. "Chapter 6: Matrix Multiplication and Related Operations." *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] D. Bini, M. Capovani, G. Lotti, and F. Romani. "$O(n^{2.7799})$ Complexity for Approximate Matrix Multiplication", Information Processing Letters, No. 8, 1979.

[3] D. Coppersmith and S. Winograd. "On the Asymptotic Complexity of Matrix Multiplication." SIAM Journal of Computing, Vol 11, No. 3, August 1982.

[4] D. Coppersmith and S. Winograd. "Matrix Multiplication via Arithmetic Progressions." Proc. 19th Annual ACM Conf. on Theory of Computing, 1987.

[5] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. "Chapter 28: Matrix Operations." *Introduction to Algorithms, Second Edition*. MIT Press, McGraw-Hill Book Company, 2001.

[6] N. Higham. "Chapter 23: Fast Matrix Multiplication." *Accuracy and Stability of Numerical Algorithms, Second Edition.* SIAM, 2002.

[7] V. Pan. *How to Multiply Matrices Faster.* Springer-Verlag, 1984.

[8] A. Schönhage. "Partial and Total Matrix Multiplication." SIAM Journal of Computing, Vol 10, No. 3, August 1981.

[9] V. Strassen. "Relative Bilinear Complexity and Matrix Multiplication." J. Reine Angew. Math. Vols 375 & 376, 1987.

[10] V. Strassen. "Gaussian Elimination is not Optimal." Numerische Mathematik, Vol 13, No 3. 1969.