Notes on Fast Matrix Multiplication

Timothy Chan

July 30, 2020

These notes describe a few subcubic matrix multiplication algorithms that go beyond Strassen's original $O(n^{2.81})$ algorithm, and are not too difficult to understand, and should hopefully be accessible to students. There are already several extensive surveys on the topic (e.g., Pan'84, Pan'14, and Bläser'13), but our intention is to be direct, concise, and concrete, forgoing abstract notation and general frameworks (like trilinear forms, tensor products, rank and border rank, the τ theorem, ...). We will not explain the notation in the headings (but the readers can guess...).

Algorithm 0: $R(\langle 2,2,2\rangle) \le 7 \;\Rightarrow\; \omega < 2.808$ (Strassen'69)

We begin with a quick review of Strassen's original algorithm. Given 2×2 matrices $A = (a_{ij})_{i,j\in\{1,2\}}$ and $B = (b_{ij})_{i,j\in\{1,2\}}$, the product $C = AB = (c_{ij})_{i,j\in\{1,2\}}$ can be computed by the following formulas, which use 7 multiplications:

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

To multiply two $n\times n$ matrices, we divide each matrix into 4 $(n/2)\times(n/2)$ submatrices and apply the above formulas, where the elements are now $(n/2)\times(n/2)$ submatrices. The 7 multiplications of elements can be computed by 7 recursive calls. The running time satisfies the recurrence $T(n) = 7\,T(n/2) + O(n^2)$, which solves to $O(n^{\log_2 7}) = O(n^{2.808})$.

Remark. Although verification of the formulas is a straightforward exercise, they appear to work "by magic", and aren't exactly easy to remember...

Algorithm 1: $R(2 \odot \langle 13,13,13\rangle) \le 2704 \;\Rightarrow\; \omega < 2.811$ (Winograd/Pan)

Given two pairs of 13×13 matrices $A = (a_{ij})_{i,j\in\{1,\ldots,13\}}$, $B = (b_{ij})_{i,j\in\{1,\ldots,13\}}$, $A' = (a'_{ij})_{i,j\in\{1,\ldots,13\}}$, and $B' = (b'_{ij})_{i,j\in\{1,\ldots,13\}}$, here is one way to compute the two products $C = AB = (c_{ij})_{i,j\in\{1,\ldots,13\}}$ and $C' = A'B' = (c'_{ij})_{i,j\in\{1,\ldots,13\}}$ simultaneously, using $13^3 + 3\cdot 13^2 = 2704$ multiplications: for each $i,j,k \in \{1,\ldots,13\}$,

$$\begin{array}{rcl} p_{ikj} &=& (a_{ik} + a'_{kj})(b_{kj} + b'_{ji}) \\ q_{kj} &=& a'_{kj} b_{kj} \\ r_{ij} &=& \left(\sum_{k=1}^{13} (a_{ik} + a'_{kj})\right) b'_{ji} \\ s_{ki} &=& a_{ik} \left(\sum_{j=1}^{13} (b_{kj} + b'_{ji})\right) \\ \\ c_{ij} &=& \sum_{k=1}^{13} (p_{ikj} - q_{kj}) - r_{ij} \\ c'_{ki} &=& \sum_{j=1}^{13} (p_{ikj} - q_{kj}) - s_{ki}. \end{array}$$

To multiply two pairs of $n \times n$ matrices, we divide each matrix into 13^2 $(n/13) \times (n/13)$ submatrices and apply the above formulas where the elements are now submatrices. The number of recursive calls is $2704/2$, leading to the recurrence $T(n) = (2704/2)T(n/13) + O(n^2)$, which solves to $O(n^{\log_{13}(2704/2)}) = O(n^{2.811})$.

Remarks. This is slightly slower than Strassen's, but the formulas are easier to verify and more intuitive, with a clearer pattern that extends to larger numbers of parts (13 is best here). The above is one variant among a series of algorithms by Pan and others (with the basic idea tracing back to Winograd). Pan'78 (see also Laderman, Pan, and Sha'92) extended the idea to three products to obtain the first improvement over Strassen, but the formulas are much messier to write down.

Algorithm 2: $\underline{R}(\langle 3,3,3\rangle) \leq 21 \;\Rightarrow\; \omega < 2.772$ (Schönhage'81)

Given 3×3 matrices $A = (a_{ij})_{i,j\in\{1,2,3\}}$ and $B = (b_{ij})_{i,j\in\{1,2,3\}}$, here is one way to compute an *approximate* product $C = (c_{ij})_{i,j\in\{1,2,3\}}$ using $6+6+3+3+3 = 21$ multiplications: for each $i,j,k \in \{1,2,3\}$,

$$\begin{array}{rcll} p_{ij} & = & (\varepsilon^2 a_{i1} + a_{j3})(b_{1j} + \varepsilon b_{3i}) & \text{if } i \neq j \\ q_{ij} & = & (\varepsilon^2 a_{i2} + a_{j3})(b_{2j} - \varepsilon b_{3i}) & \text{if } i \neq j \\ r_j & = & a_{j3}(b_{1j} + b_{2j}) & \\ p_{ii} & = & (\varepsilon^2 a_{i1} + a_{i3}) b_{1i} & \\ q_{ii} & = & (\varepsilon^2 a_{i2} + a_{i3})(b_{2i} + \varepsilon^2 b_{3i}) & \\ & & & \\ c_{ij} & = & \frac{1}{\varepsilon^2}(p_{ij} + q_{ij} - r_j) + \frac{1}{\varepsilon}(p_{ji} - p_{ii}). & \end{array}$$

Here, we think of ε as a very small number (an "infinitesimal"). It can be checked that $c_{ij} \equiv \sum_{k=1}^{3} a_{ik} b_{kj} \pmod{\varepsilon}$, where "$\pmod{\varepsilon}$" means that we ignore terms having at least one factor of ε. (This is because for $i \neq j$, we have $p_{ij} + q_{ij} - r_j \equiv \varepsilon^2(a_{i1}b_{1j} + a_{i2}b_{2j}) \pmod{\varepsilon^3}$, and $p_{ji} - p_{ii} \equiv \varepsilon a_{i3} b_{3j} \pmod{\varepsilon^2}$; on the other hand, for $i = j$, we have $p_{ii} + q_{ii} - r_i \equiv \varepsilon^2(a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j}) \pmod{\varepsilon^3}$.)

To compute an approximate product of two $n \times n$ matrices A and B, we divide each matrix into 9 $(n/3) \times (n/3)$ submatrices and apply the above formulas, where the elements are now submatrices. Each of the 21 multiplications of elements is replaced by a recursively computed approximate product of submatrices (using the same ε at all levels of recursion). The number of operations satisfies the recurrence $T(n) = 21\,T(n/3) + O(n^2)$, which solves to $O(n^{\log_3 21}) = O(n^{2.772})$.

To justify that the recursion yields good approximation at the end, let $A * B$ denote the approximate product computed by the above algorithm. First it can be checked (by induction) that $*$ is a *bilinear* operator (i.e., it satisfies the properties $(A + A') * B = A * B + A' * B$, and $A * (B + B') = A * B + A * B'$, and $(cA) * B = A * (cB) = c(A * B)$, for all A, B, A', B' and all constants c; equivalently, $A * B$ can be expressed as a sum where each term is a constant times an element of A times an element of B). Because of the bilinearity of $*$, when we replace every individual multiplication of elements with the $*$ operator, at the end we get $c_{ij} \equiv \sum_{k=1}^{3} a_{ik} * b_{kj} \pmod{\varepsilon}$. We can now[1] apply the induction hypothesis that $a_{ik} * b_{kj} \equiv a_{ik} b_{kj} \pmod{\varepsilon}$, to deduce that $C = A * B \equiv AB \pmod{\varepsilon}$.

Dealing with ε. Instead of setting ε to be a very small number (which would require high precision arithmetic, with the number of bits increased by an $O(\log n)$ factor), one way to turn the approximation algorithm into an exact algorithm is to treat ε as a symbolic variable. Then each matrix element is a polynomial in ε, with degree bounded by $O(\log n)$ (the number of levels of recursion), and can be represented by an array of $O(\log n)$ coefficients. At the end, since $A * B \equiv AB \pmod{\varepsilon}$, we can remove all but the nonconstant terms in the final polynomials. Each addition or multiplication of $O(\log n)$-degree polynomials takes polylogarithmic time, so the running time increases by a polylogarithmic factor.

[1] The argument may appear simple, but note that had we apply the induction hypothesis too early, the formula for c_{ij} would naively seem to yield an error term of $\frac{1}{\varepsilon^2} O(\varepsilon) = O(\frac{1}{\varepsilon})$, which would be too large! The bilinearity of $*$ is crucial here.

Remarks. Although the formulas above may still look magical at first, they are a bit easier to verify than Strassen's (and easier to recreate), since the use of the infinitesimal ε offers more flexibility and allows us to discard higher-degree terms.

This idea of using ε (leading to so-called "any precision approximation (APA) algorithms" and the "border rank") was pioneered by Bini et al.'79, and was the starting point behind all the theoretically faster algorithms. The extra polylogarithmic-factor overhead makes this type of algorithms less practical, however.

Algorithm 3: $\underline{R}(2 \odot \langle 8,8,8 \rangle) \leq 640 \Rightarrow \omega < 2.774$

Given two pairs of 8×8 matrices $A = (a_{ij})_{i,j \in \{1,\ldots,8\}}$, $B = (b_{ij})_{i,j \in \{1,\ldots,8\}}$, $A' = (a'_{ij})_{i,j \in \{1,\ldots,8\}}$, and $B' = (b'_{ij})_{i,j \in \{1,\ldots,8\}}$, here is a way to compute an approximate product $C = (c_{ij})_{i,j \in \{1,\ldots,8\}}$ of A and B and an approximate product $C' = (c'_{ij})_{i,j \in \{1,\ldots,8\}}$ of A' and B' simultaneously, using $8^3 + 2 \cdot 8^2 = 640$ multiplications: for each $i, j, k \in \{1, \ldots, 8\}$,

$$\begin{array}{rcl} p_{ikj} & = & (a_{ik} + a'_{kj})(b_{kj} + \varepsilon b'_{ji}) \\ q_{kj} & = & a'_{kj} b_{kj} \\ s_{ki} & = & a_{ik} \sum_{j=1}^{8} (b_{kj} + \varepsilon b'_{ji}) \\ & & \\ c_{ij} & = & \sum_{k=1}^{8} (p_{ikj} - q_{kj}) \\ c'_{ki} & = & \frac{1}{\varepsilon} \left(\sum_{j=1}^{8} (p_{ikj} - q_{kj}) - s_{ki} \right). \end{array}$$

It is easy to check that $C \equiv AB \pmod{\varepsilon}$ and $C' \equiv A'B' \pmod{\varepsilon}$.

Consequently, given two pairs of $8^2 \times 8^2$ matrices (A, B) and (A', B'), we can compute approximate products of these pairs by applying the above formulas, with elements replaced by an 8×8 submatrices. Each such product of 8×8 submatrices is in turn computed by the above method, mod ε^2 (i.e., with ε replaced by ε^2). This is sufficient to guarantee that at the end, $C \equiv AB \pmod{\varepsilon}$ and $C' \equiv A'B' \pmod{\varepsilon}$. The total number of multiplications of elements is $(640/2) \cdot 640$.

Iterating ℓ times, we can thus multiply two pairs of $8^\ell \times 8^\ell$ matrices, computed mod ε, using $O((640/2)^\ell)$ multiplications of elements. (When iterating ℓ times, the degree in ε increases to 2^ℓ, but theoretically this is acceptable as ℓ will be a constant.)[2]

To compute an approximate product of two $n \times n$ matrices A and B, we divide each matrix into $(n/8^\ell) \times (n/8^\ell)$ submatrices and apply the above method (appended with a dummy matrix pair), where each of the $O((640/2)^\ell)$ multiplication of elements is replaced by a recursively computed approximate product of submatrices (here, we can use the same ε at all levels of recursion, as justified by the same bilinearity argument as before). The number of operations satisfies the recurrence $T(n) = O((640/2)^\ell)\, T(n/8^\ell) + O(2^{O(\ell)} n^2)$, which solves to $O(n^{\log_8(640/2)+\delta}) = O(n^{2.774})$ for an arbitrarily small constant $\delta > 0$, by making ℓ an arbitrarily large constant. As before, we can obtain an exact algorithm at the expense of an extra polylogarithmic factor.

Remarks. This is of course a variant of Algorithm 1 but using ε, which simplifies the formulas by eliminating the r_{ij} products. Although the time bound is slightly worse than in Algorithm 2, the formulas are intuitive.

[2] There are ways to avoid the exponential blow-up in the degree in ε. One may be tempted to reuse the previous bilinearity argument, but this has to be done carefully (for example, the recursively computed approximate product for q_{kj} will be a function of other variables besides a'_{kj} and b_{kj}, depending on which other product it is paired with...).

Algorithm 4: $\underline{R}(\langle 7,1,7\rangle \oplus \langle 1,7,7\rangle) \leq 63 \;\Rightarrow\; \omega < 2.66$ (Pan/Schönhage'81)

Part I. Given a 7×1 matrix $A = \begin{pmatrix} a_1 \\ \vdots \\ a_7 \end{pmatrix}$, a 1×7 matrix $B = (b_1 \cdots b_7)$, a 1×7 matrix $A' = (a'_1 \cdots a'_7)$, and a 7×7 matrix $B' = \begin{pmatrix} b'_{11} & & b'_{17} \\ & \ddots & \\ b'_{71} & & b'_{77} \end{pmatrix}$, here is one way to compute an approximate product $C = \begin{pmatrix} c_{11} & & c_{17} \\ & \ddots & \\ c_{71} & & c_{77} \end{pmatrix}$ of A and B, and an approximate product $C' = (c'_1 \cdots c'_7)$ of A' and B', using $7^2 + 2 \cdot 7 = 63$ multiplications: for each $i, j \in \{1, \ldots, 7\}$,

$$\boxed{\begin{array}{rcl} p_{ij} & = & (a_i + a'_j)(b_j + \varepsilon b'_{ji}) \\ q_j & = & a'_j b_j \\ s_i & = & a_i \sum_{j=1}^{7} (b_j + \varepsilon b'_{ji}) \\ & & \\ c_{ij} & = & p_{ij} - q_j \\ c'_i & = & \frac{1}{\varepsilon}\left(\sum_{j=1}^{7}(p_{ij} - q_j) - s_i\right). \end{array}}$$

It is easy to check that $C \equiv AB \pmod{\varepsilon}$ and $C' \equiv A'B' \pmod{\varepsilon}$.

Consequently, given one pair $(A^{(1)}, B^{(1)})$ of matrices of dimensions $7^2 \times 1$ and 1×7^2, two pairs $(A^{(2)}, B^{(2)})$ and $(A^{(3)}, B^{(3)})$ of matrices of dimensions 7×7 and 7×7^2, and one pair $(A^{(4)}, B^{(4)})$ of matrices of dimensions 1×7^2 and $7^2 \times 7^2$, we can compute approximate products of these pairs by applying the above formulas for the pairs $(A^{(1)}, B^{(1)})$ and $(A^{(2)}, B^{(2)})$, with elements of the A's replaced by 7×1 submatrices and elements of the B's replaced by 1×7 submatrices; and applying the above formulas for the pairs $(A^{(3)}, B^{(3)})$ and $(A^{(4)}, B^{(4)})$, with elements of the A's replaced by 1×7 submatrices and elements of the B's replaced by 7×7 submatrices. Each product of submatrices is in turn computed by the above method, mod ε^2. This guarantees that the products of the given matrix pairs are correct mod ε. The total number of multiplications of elements is 63^2.

Iterating ℓ times, we can thus do the following: given $\binom{\ell}{s}$ pairs of matrices of dimensions $7^s \times 7^{\ell-s}$ and $7^{\ell-s} \times 7^\ell$ for every $s \in \{0, \ldots, \ell\}$, we can compute the products of all pairs, mod ε, using 63^ℓ multiplications of elements. (The degree in ε increases to 2^ℓ, but ℓ will be a constant.)

In particular, setting $s = \ell/2$, we can compute $\binom{\ell}{\ell/2} = 2^{\ell - o(1)}$ products for pairs of matrices of dimension $7^{\ell/2} \times 7^{\ell/2}$ and $7^{\ell/2} \times 7^\ell$, mod ε, using 63^ℓ multiplications of elements.

Iterating this whole process ℓ times and letting $L = \ell^2$, we can compute $2^{\ell - o(1)}$ products for pairs of matrices of dimension $7^{L/2} \times 7^{L/2}$ and $7^{L/2} \times 7^L$, mod ε, using $(63^\ell / 2^{\ell - o(1)})^\ell 2^\ell = (63/2)^{(1+o(1))L}$ multiplications of elements.

To compute an approximate product of a $\sqrt{n} \times \sqrt{n}$ matrix and a $\sqrt{n} \times n$ matrix, we divide the first matrix into $\sqrt{n/7^L} \times \sqrt{n/7^L}$ submatrices and the second into $\sqrt{n/7^L} \times n/7^L$ submatrices and apply the above method, where each of the multiplication of elements is replaced by a recursively computed approximate product of submatrices (here, we can use the same ε at all levels of recursion, as justified by the same bilinearity argument as before). The number of operations satisfies the recurrence $T(n) = (63/2)^{(1+o(1))L} T(n/7^L) + O(2^{O(L)} n^2)$, which solves to $O(n^{\log_7(63/2)+\delta})$ for an

arbitrarily small constant $\delta > 0$, by making $L = \ell^2$ an arbitrarily large constant. As before, we can obtain an exact algorithm at the expense of an extra polylogarithmic factor.

Part II. Given a 1×7 matrix $A = (a_1 \cdots a_7)$, a 7×7 matrix $B = \begin{pmatrix} b_{11} & & b_{17} \\ & \ddots & \\ b_{71} & & b_{77} \end{pmatrix}$, a 7×7 matrix $A' = \begin{pmatrix} a'_{11} & & a'_{17} \\ & \ddots & \\ a'_{71} & & a'_{77} \end{pmatrix}$, and a 7×1 matrix $B' = \begin{pmatrix} b'_1 \\ \vdots \\ b'_7 \end{pmatrix}$, here is one way to compute the approximate product $C = (c_1 \cdots c_7)$ of A and B, and the approximate product $C' = \begin{pmatrix} c'_1 \\ \vdots \\ c'_7 \end{pmatrix}$, using $7^2 + 2 \cdot 7 = 63$ multiplications: for each $i, j \in \{1, \ldots, 7\}$,

$$\boxed{\begin{array}{rcl} p_{kj} & = & (a_k + \varepsilon a'_{kj})(\varepsilon b_{kj} + b'_j) \\ q_j & = & \left(\sum_{k=1}^{7} (a_k + \varepsilon a'_{kj})\right) b'_j \\ s_k & = & a_k \sum_{j=1}^{7} (\varepsilon b_{kj} + b'_j) \\ \\ c_j & = & \frac{1}{\varepsilon}\left(\sum_{k=1}^{7} p_{kj} - q_i\right) \\ c'_k & = & \frac{1}{\varepsilon}\left(\sum_{j=1}^{7} p_{kj} - s_k\right). \end{array}}$$

It is easy to check that $C \equiv AB \pmod{\varepsilon}$ and $C' \equiv A'B' \pmod{\varepsilon}$.

By a similar argument, this leads to an algorithm that computes the product of a $\sqrt{n} \times n$ matrix and an $n \times \sqrt{n}$ matrix in $O(n^{\log_7(63/2)+\delta})$ time.

Combine. An algorithm symmetric to Part I computes the product of an $n \times \sqrt{n}$ matrix and a $\sqrt{n} \times \sqrt{n}$ matrix in $O(n^{\log_7(63/2)+\delta})$ time.

Observe that if we can multiply any $n_1 \times n_2$ and $n_2 \times n_3$ matrix in T operations and we can multiply any $n'_1 \times n'_2$ and $n'_2 \times n'_3$ matrix in T' operations, then we can multiply any $n_1 n'_1 \times n_2 n'_2$ and $n_2 n'_2 \times n_3 n'_3$ matrix in $O(TT')$ operations (by viewing each $n'_1 \times n'_2$ submatrix in the first matrix as an element and each $n'_2 \times n'_3$ submatrix in the second matrix as an element). Consequently, by combining all three algorithms above, we can multiply two $n^2 \times n^2$ matrices in $O(n^{3\log_7(63/2)+O(\delta)})$ time, i.e., two $n \times n$ matrices in $O(n^{(3/2)\log_7(63/2)+O(\delta)})$ time. The bound is $O(n^{2.660})$.

Remarks. The formulas in Part I are of course from the same family used in Algorithm 3, and are slightly simpler because one dimension is 1. The formulas in Part II are "equivalent" to those in Part I when converted to *trilinear form* (which implies symmetry of rectangular matrix multiplication exponents).

While the algebraic side gets simpler, the algorithmic side gets more complicated, as we deal with multiple rectangular products of different dimensions (Schönhage's τ theorem provides a general analysis of the exponent in such recursion).

Algorithm 5: $\underline{R}(\langle 4,1,4\rangle \oplus \langle 1,9,1\rangle) \leq 17 \;\Rightarrow\; \omega < 2.548$ (Schönhage'81)

Part I. Given a 4×1 matrix $A = \begin{pmatrix} a_0 \\ \vdots \\ a_3 \end{pmatrix}$, a 1×4 matrix $B = (b_0 \cdots b_3)$, a 1×9 matrix $A' = (a'_{11} \cdots a'_{33})$, and a 9×1 matrix $B' = \begin{pmatrix} b'_{11} \\ \vdots \\ b'_{33} \end{pmatrix}$, we want to compute an approximate product $C = \begin{pmatrix} c_{00} & & c_{03} \\ & \ddots & \\ c_{30} & & c_{33} \end{pmatrix}$ of A and B, and an approximate product $C' = (c')$ of A' and B'. (Note the unconventional indexing for the vectors A' and B'.)

First set $a'_{0i} = b'_{j0} = 0$ and $a'_{j0} = -\sum_{i=1}^{3} a'_{ji}$ and $b'_{0i} = -\sum_{j=1}^{3} b'_{ji}$ for $i, j \in \{0, \ldots, 3\}$. (This ensures that $\sum_{i=0}^{3} a'_{ji} = \sum_{j=0}^{3} b'_{ji} = 0$.) We use the following formulas, which requires $4^2 + 1 = 17$ products: for each $i, j \in \{0, \ldots, 3\}$,

$$\boxed{\begin{array}{rcl} p_{ij} & = & (a_i + \varepsilon a'_{ji})(b_j + \varepsilon b'_{ji}) \\ q & = & \left(\sum_{i=0}^{3} a_i\right)\left(\sum_{j=0}^{3} b_j\right) \\ \\ c_{ij} & = & p_{ij} \\ c' & = & \frac{1}{\varepsilon^2}\left(\sum_{i=0}^{3}\sum_{j=0}^{3} p_{ij} - q\right). \end{array}}$$

It can be checked that $C \equiv AB \pmod{\varepsilon}$ and $C' \equiv A'B' \pmod{\varepsilon}$.

Like before, iterating ℓ times, we can thus do the following: given $\binom{\ell}{s}$ pairs of matrices of dimensions $4^s \times 9^{\ell-s}$ and $9^{\ell-s} \times 4^s$ for every $s \in \{0, \ldots, \ell\}$, we can compute the products of all pairs, mod ε, using 17^ℓ multiplications of elements.

In particular, setting $s = \alpha\ell$ for some constant α to be chosen later, we can compute $\binom{\ell}{\alpha\ell} = 1/(\alpha^\alpha(1-\alpha)^{1-\alpha})^{(1-o(1))\ell}$ products for pairs of matrices of dimension $4^{\alpha\ell} \times 9^{(1-\alpha)\ell}$ and $9^{(1-\alpha)\ell} \times 4^{\alpha\ell}$, mod ε, using 17^ℓ multiplications of elements.

Iterating this whole process ℓ times and letting $L = \ell^2$, we can compute $1/(\alpha^\alpha(1-\alpha)^{1-\alpha})^{(1-o(1))\ell}$ products for pairs of matrices of dimension $4^{\alpha L} \times 9^{(1-\alpha)L}$ and $9^{(1-\alpha)L} \times 4^{\alpha L}$, mod ε, using $(17^\ell \cdot (\alpha^\alpha(1-\alpha)^{1-\alpha})^{(1-o(1))\ell})^{\ell+O(1)} = (17\alpha^\alpha(1-\alpha)^{1-\alpha})^{(1+o(1))L}$ multiplications of elements.

To compute an approximate product of an $n^\alpha \times n^{(1-\alpha)\log_4 9}$ matrix and an $n^{(1-\alpha)\log_4 9} \times n^\alpha$ matrix, we divide the first matrix into $(n/4^L)^\alpha \times (n/4^L)^{(1-\alpha)\log_4 9}$ submatrices and the second into $(n/4^L)^{(1-\alpha)\log_4 9} \times (n/4^L)^\alpha$ submatrices and apply the above method, where each of the multiplication of elements is replaced by a recursively computed approximate product of submatrices. The number of operations satisfies the recurrence $T(n) = (17\alpha^\alpha(1-\alpha)^{1-\alpha})^{(1+o(1))L} T(n/4^L) + O(2^{O(L)}n^2)$, which solves to $O(n^{\log_4(17\alpha^\alpha(1-\alpha)^{1-\alpha})+\delta})$ for an arbitrarily small constant $\delta > 0$ by making $L = \ell^2$ an arbitrarily large constant. As before, we can obtain an exact algorithm at the expense of an extra polylogarithmic factor.

Part II. Given a 1×4 matrix $A = (a_0 \cdots a_3)$, a 4×4 matrix $B = \begin{pmatrix} b_{00} & & b_{03} \\ & \ddots & \\ b_{30} & & b_{33} \end{pmatrix}$, a 9×1 matrix $A' = \begin{pmatrix} a'_{11} \\ \vdots \\ a'_{33} \end{pmatrix}$, and a 1×1 matrix $B' = (b')$, we want to compute an approximate product $C = (c_0 \cdots c_3)$ of A and B, and an approximate product $C' = \begin{pmatrix} c'_{11} \\ \vdots \\ c'_{33} \end{pmatrix}$ of A' and B'. (Again, note the unconventional indexing for the vectors A' and C'.)

First set $a'_{k0} = 0$ and $a'_{0j} = -\sum_{k=1}^{3} a'_{kj}$. (This ensures that $\sum_{k=0}^{3} a'_{ki} = 0$.) We use the following formulas, which require $4^2 + 1 = 17$ products: for each $j, k \in \{0, \ldots, 3\}$,

$$\boxed{\begin{array}{rcl} p_{kj} & = & (a_k + \varepsilon a'_{kj})(\varepsilon^2 b_{kj} + b') \\ q & = & \left(\sum_{k=0}^{3} a_k\right) b' \\ \\ c_j & = & \frac{1}{\varepsilon^2}\left(\sum_{k=0}^{3} p_{kj} - q\right) \\ c'_{kj} & = & \frac{1}{\varepsilon}(p_{kj} - p_{k0}). \end{array}}$$

It can be checked that $C \equiv AB \pmod{\varepsilon}$ and $C' \equiv A'B' \pmod{\varepsilon}$.

By a similar argument, this leads to an algorithm for computing the product of an $n^\alpha \times n^\alpha$ matrix and an $n^\alpha \times n^{(1-\alpha)\log_4 9}$ matrix in $O(n^{\log_4(17\alpha^\alpha(1-\alpha)^{1-\alpha})+\delta})$ time.

Combine. An algorithm symmetric to Part II computes the product of an $n^{(1-\alpha)\log_4 9} \times n^\alpha$ matrix and an $n^\alpha \times n^\alpha$ matrix in $O(n^{\log_4(17\alpha^\alpha(1-\alpha)^{1-\alpha})+\delta})$ time.

Consequently, by combining all three algorithms, we can multiply two $n^{2\alpha+(1-\alpha)\log_4 9} \times n^{2\alpha+(1-\alpha)\log_4 9}$ matrices in $O(n^{3\log_4(17\alpha^\alpha(1-\alpha)^{1-\alpha})+O(\delta)})$ time, i.e., two $n \times n$ matrices in $O(n^{3\log_4(17\alpha^\alpha(1-\alpha)^{1-\alpha})/(2\alpha+(1-\alpha)\log_4 9)+O(\delta)})$ time. By choosing $\alpha = 0.62$ to minimize the expression, the bound is $O(n^{2.548})$.

Remarks. Again, the formulas in Parts I and II are equivalent when converted to trilinear form.

Further small improvements can be obtained with more complicated refinements. But later Strassen'86 and Coppersmith and Winograd'90 came up with still more powerful techniques that led to more significant improvements, and eventually to the current record near $O(n^{2.373})$ by Stothers'10, Vassilevska Williams'12, and Le Gall'14...