Fast Matrix Multiplication

Alex Fogelson and Naren Karur

Abstract

In this report, we tour both the theoretical and practical implementations of matrix multiplication, especially for square matrices. In particular, we discuss methods of bounding the time complexity, primarily with tensor analysis and tensor approximations, although with other methods as well. The structure and explanation draws heavily from Markus Bläser's paper *Fast Matrix Multiplication*. Finally, we discuss the practicality of various methods, which we experiment with by emulating optimizations for different algorithms. We conclude that with the current collection of fast multiplication algorithms, the most reliably efficient implementations indeed have comparatively high theoretical time complexity.

Overview

1. General Matrix Multiplication (GEMM) (3)
    - Trivial Algorithm
    - Upper Bound
    - Lower Bound
    - Problem Framing
2. Motivation: Karatsuba (3)
    - Polynomials
    - Karatsuba's Algorithm
3. Strassen's First Approach (4)
    - Intro & Assumptions
    - Strassen's First Algorithm
    - Cost Analysis
4. Introduction to Tensors (6)
    - Tensor Basics
    - Matrix Multiplication as a Tensor
5. The Exponent of Matrix Multiplication (7)
    - Omega (ω)
    - Generalizing Smaller Bounds
6. Tensor Approximations and Border Ranks (7)
    - Approximations
    - Border Rank
    - Trajectory
7. Schönhage's τ-theorem (8)
8. Strassen's Second Method (The Laser Method) (9)
    - Strassen's Tensor
    - Approximation of Strassen's Tensor
    - Final Haul
9. Further Improvements (10)
    - Copersmith and Winograd
    - Alman and Williams
    - Ambainis and Filmus
10. Experimental Results (11)
    - Background
    - Optimization Emulation
    - Approximation Algorithm
11. Citations (14)

1 General Matrix Multiplication (GEMM)

Matrix multiplication is typically defined element-wise, by fixing an i, jth component in the output matrix, and performing a calculation for this element in isolation. The definition leads to a natural upper bound on the computation when floating point operations are assumed to have cost 1.

Trivial Algorithm

Let F be a field. If we have a matrix $A \in F^{k\times m}$ and $B \in F^{m\times n}$, we can compute $AB \in F^{k\times n}$ as follows:

$$AB[i,j] = \sum_{w=1}^{m} A[i,w]B[w,j]$$

That is, the i, jth element of AB is simply the dot product between the ith row of A and the jth column of B.

It is not obvious that there even exists a more efficient way of computing this other than the sum given above. A motivating example will be the topic of the next section.

Upper Bound $\mathcal{O}(n^3)$

The dot product given runs in $m + (m - 1)$ steps (m multiplications, $m - 1$ additions). Thus as we have kn of such computations to perform, the cost is simply $(2m - 1)kn \in \mathcal{O}(knm)$.
Without loss of generality, if n is the max, then $\mathcal{O}(knm) \subseteq \mathcal{O}(n^3)$. So we have a clear upper bound on the optimal complexity.

Lower Bound $\mathcal{O}(n^2)$

On the other hand, we need at least one operation for each of the kn elements in the output (access or computation), thus we have a lower bound of $kn \in \mathcal{O}(kn)$ operations. Again, without loss of generality, if n is the max, this is in $\mathcal{O}(n^2)$.

Problem Framing

With these respective bounds, we can confirm that the optimal algorithm (for which a formal definition will come in the section title *The Exponent of Matrix Multiplication*) needs to be in $\mathcal{O}(n^3)$ but not in $\mathcal{O}(n^{2-\varepsilon})$, for $\varepsilon > 0$. This will motivate our exploration: the problem of matrix multiplication continues to push the best algorithm towards the lower bound complexity and further from the trivial algorithm.

2 Motivation: Karatsuba

In the last section, we mention how non-obvious it is that the trivial algorithm for matrix multiplication can be improved. In this section, we provide a motivating topic with similarly unintuitive results, which related deeply to the problem of matrix multiplication.

Polynomials

Suppose we are given a field K and two polynomials of the form $a+bX$ and $c+dX$, where $a, b, c, d \in K$, and $X \in K[x]$. What is the smallest number of multiplications needed to compute the coefficients of $(a + bX)(c + dX)$?

$$(a + bX)(c + dX) = ac + (ad + bc)X + bdX^2$$

$$\text{(coefficients are } ac,\ ad + bc,\ bd)$$

The incorrect but obvious answer is 4, because by simply foiling, we would get all of the desired products.

Karatsuba's Algorithm:

The following algorithm uses only 3 multiplications and 4 additions.

$$\begin{aligned} w_1 &= ac \\ w_2 &= bd \\ w_3 &= (a+b)(c+d) - w_1 - w_2 \\ &= ac + bc + bd + da - ac - bd \\ &= ad + bc \end{aligned}$$

Here lies the insight in Karatsuba's algorithm: by picking clever multiplications, one can *implicitly* compute all four products with only three multiplications and some number of additions.
Importantly, we don't particularly care about the individual products *ad* and *bc*. Therefor, we can construct a more clever multiplication scheme to get us their sum.

3 Strassen's First Algorithm

Intro & Assumptions

Strassen's first algorithm effectively builds on the insights of Karasuba's polynomial multiplication algorithm. By splitting the matrices of A, B appropriately, we can yield a recursive algorithm which limits our use of matrix multiplication. Note that Strassen has additional insights later, hence the enumeration of his algorithms.

We first make the simplifying assumption that we are multiplying $N \times N$ matrices to yield another such matrix. Moreover, we will assume that $N = 2^n$, for some $n \in \mathbb{Z}^+$. The latter assumption will be justified at the end.

Strassen's First Algorithm

Let $X, Y \in F^{N\times N}$, where F is a field. We first define the submatrices of X and Y as follows:

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$$Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

Where $A, ..., H \in F^{N/2 \times N/2}$, which is where we make use of the assumption that N is a power of 2. Now begin by computing the following, using 10 additions and 7 multiplications.

$$\begin{aligned}
M1 &= (A + C)(E + F)\\
M2 &= (B + D)(G + H)\\
M3 &= (A - D)(E + H)\\
M4 &= A(F - H)\\
M5 &= (C + D)(E)\\
M6 &= (A + B)(H)\\
M7 &= D(G - E)
\end{aligned}$$

Next, we compute XY as follows:

$$XY = \begin{pmatrix} AE + BG & CE + DG \\ AF + BH & CF + DH \end{pmatrix} = \begin{pmatrix} M2 + M3 - M6 - M7 & M4 + M6 \\ M5 + M7 & M1 - M3 - M4 - M5 \end{pmatrix}$$

This requires an additional 8 matrix additions, yielding 18 additions and 7 multiplications overall.

As in with Karatsuba's algorithm, although computing the submatrices in XY using the typical algorithm would use 8 matrix multiplications and 4 additions, we don't care about each individual product, and thus can minimize the number of multiplications performed.

Cost Analysis

The recurrence relation can be written for $N > 1$ as

$$W(N) = 7W(\frac{N}{2}) + 18N^2$$

It's not difficult to show that $W(N) \in \mathcal{O}(N^{\log_2 7})$, where $\log_2 7 \approx 2.807$. This is the first improvement we've seen on the cubic bound.

Now we justify the assumption that N is a power of 2. Let $M \in \mathbb{N}$, and let M' be the smallest power of 2 greater than or equal to M. We know $M \leq M' \leq 2M$, and therefor since $n^{\log_2 7}$ is monotonically increasing on positive values, we get the following bound:

$$M^{\log_2 7} \leq (M')^{\log_2 7} \leq (2M)^{\log_2 7} = 2^{\log_2 7} M^{\log_2 7} = 7M^{\log_2 7}$$

Therefor, we can first embed an $M \times M$ matrix into a $M' \times M'$ matrix by padding with zeros (in $O(M^2)$ time), and the cost of this multiplication using Strassen's first algorithm will be in $\mathcal{O}(7M^{\log_2 7}) = \mathcal{O}(M^{\log_2 7})$. Overall, this is still in $\mathcal{O}(M^{\log_2 7})$.

See the section labeled *Experimental Result* for implementation and cost analysis for small value of N

4 Introduction to Tensors

Tensor Basics

A tensor is essentially a multidimensional matrix. For these algorithms, we will consider only tensors which are "three dimensional", that is, can be written as the sum of output products of triads, such that:

$$\sum_{i=1}^{q} v_i \otimes u_i \otimes w_i$$

Each $v_i \oplus u_i \oplus w_i$ is called a *Rank* 1 *Tensor.*

Definition (Rank of a Tensor): The rank of a tensor is defined by the minimum number of rank-1 tensors that sum to it. We denote this value with $R(t)$, if t is a tensor.

Matrix Multiplication as a Tensor

For a field K, we can in fact encode matrix multiplication from $K^{k\times m} \times K^{m\times n} \to K^{k\times n}$ as a binary tensor of size $K^{(km)\times mn\times kn}$. We call this tensor $\langle k, m, n\rangle$, and it is defined as follows, where $A \in K^{k\times m}, B \in K^{m\times n}$ are matrices with indeterminant enumerated elements $a_{1,1}, a_{1,2}, ..., a_{k,m}$ and $b_{1,1}, a_{1,2}, ..., b_{m,n}$, respectively:

$$\langle k, m, n\rangle(i, j, l) = \begin{cases} 1 & \text{if } a_i b_j \in \text{Terms}[(ab)_l] \\ 0 & \text{otherwise} \end{cases}$$

where Terms denotes the polynomial terms in $(ab)_l$.

Example: An example is enlightening. Consider matrix multiplication of 2×2 square matrices. Namely:

$$\begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix} \begin{pmatrix} y_{1,1} & y_{1,2} \\ y_{2,1} & y_{2,2} \end{pmatrix} = \begin{pmatrix} x_{1,1}y_{1,1} + x_{1,1}y_{2,1} & x_{1,1}y_{1,2} + x_{1,1}y_{2,2} \\ x_{2,1}y_{1,1} + x_{2,1}y_{2,1} & x_{2,1}y_{1,2} + x_{2,1}y_{2,2} \end{pmatrix}$$

Then we consider the tensor of size $K^{4\times 4\times 4}$ of the following form:

$$\left(\begin{array}{ccccc} & x_{1,1} & x_{1,2} & x_{2,1} & x_{2,2} \\ y_{1,1} & \text{x} & \text{x} & \text{x} & \text{x} \\ y_{1,2} & \text{x} & \text{x} & \text{x} & \text{x} \\ y_{2,1} & \text{x} & \text{x} & \text{x} & \text{x} \\ y_{2,2} & \text{x} & \text{x} & \text{x} & \text{x} \end{array}\right)$$

where the 4 matrices are

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We have some really wonderful characterization of matrix multiplication cost bounds based on this tensor analysis, so this kind of encoding is incredibly helpful later on.

5 The Exponent of Matrix Multiplication

We finally arrive at a formal definition of the cost bound we desire.

Omega (ω)

Definition (The Ostrowski Model): In the Ostrowski Model, denoted $C^{*/}$, the cost of floating point multiplication and division is 1, while the cost of floating point addition and subtractions are 0. Similarly, C^{*} only counts multiplications.

Definition (The Exponent of Matrix Multiplication): Our aim from here on out will be to bound the complexity of tensor ranks corresponding to matrix multiplication. In particular, we define the following quantity ω:

$$\omega = \inf\{\beta : R(\langle n, n, n\rangle) \in \mathcal{O}(n^{\beta})\}$$

Note that this quantity only takes into account matrix multiplications rather than all operations (C^{*}). We can define a similar quantity:

$$\overline{\omega} = \inf\{\beta : C^{*}(\langle n, n, n\rangle) \in \mathcal{O}(n^{\beta})\}$$

In the former definition, we are bounding the rank of the tensor which corresponds to matrix multiplication, while the latter definition is the cost in the Ostrowski model of matrix multiplication. If our field K is infinite, it turns out $\omega = \overline{\omega}$ (Bläser, Theorem 5.2), therefor it's clear that this ω is the complexity exponent we are searching for.

We call omega *the exponent of matrix multiplication.*

Generalizing Smaller Bounds

Theorem (Bläser, 5.9)[1] If $R(\langle k, m, r\rangle) \leq r$, then $\omega \leq 3\log_{kmn} r$.

For example, we can show that $R(\langle 2, 2, 2\rangle) \leq 7$, therefore $\omega \leq 3\log_8(7) = \log_2(7)$, which is exactly Strassen's original bound. Of course, since $R(\langle 2, 2, 2\rangle)$ must be the minimal number of multiplications, and Strassen's approach showed 7 was sufficient, there is a direct correspondence here.

6 Tensor Approximations and Border Rank

Approximations

Often helpful for our analysis of tensors is to find lower rank approximations. Namely, $t(\varepsilon)$ is a lower rank approximation of t if as $\varepsilon \to 0$, $t(\varepsilon) \to t$, yet $\forall \epsilon > 0$, $R(t(\varepsilon)) < R(t)$. As an example, consider the following slices of a tensor $t \in K^{2\times 2\times 2}$

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

It's not too tricky to show that $R(t) = 3$. However, consider the following sum:

$$t(\varepsilon) = (1, \varepsilon) \times (1, \varepsilon) \times (0, 1/\varepsilon) + (1, 0) \times (1, 0) \times (1, -1/\varepsilon)$$

The first slice of this is

$$0 \begin{pmatrix} 1 & \varepsilon \\ \varepsilon & \varepsilon^2 \end{pmatrix} + 1 \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

So the first slice matches. Now we compute the second slice (where $\varepsilon > 0$):

$$(1/\varepsilon) \begin{pmatrix} 1 & \varepsilon \\ \varepsilon & \varepsilon^2 \end{pmatrix} + (-1/\varepsilon) \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & \varepsilon \end{pmatrix}$$

Therefor as $\varepsilon \to 0$, we get $t(\varepsilon) \to t$, even though $R(t(\varepsilon)) = 2$.

This leads to a natural definition of *border rank*.

Border Rank

Definition (Border Rank): Bläser calls $\underline{R}(t)$ the border rank of t, defined as $\underline{R}(t) = \min_h R_h(t)$, where $R_h(t)$ is defined as follows:

$$R_h(t) = \min\{r | \exists u_\rho \in F[\varepsilon]^k, v_\rho \in F[\varepsilon]^m, w_\rho \in F[\varepsilon]^n : \sum_{\rho=1}^{n} u_\rho \otimes v_\rho \otimes w_\rho = \varepsilon^h t + \mathcal{O}(\varepsilon^{h+1})\}$$

Although this definition seems strange at first, terms u_ρ, v_ρ, w_ρ simply define epsilon approximations over our field F, and this says the error needs to be within $\mathcal{O}(\varepsilon)$ with respect to the order of ε in the final tensor. So it's simply the smallest rank of any low rank approximation of t, which at least approaches t linearly.

Trajectory

We have introduced a lot of concepts so far, including tensors, rank of tensors, approximations, and border rank. We now begin to bring these together to bound ω (or at least give an overview without regurgitating the paper).

7 Schönhage's τ-Theorem

Here we present a famous theorem that is used in proving the bound for Strassen's. Namely, it says that if we have a bunch of tensors representing matrix multiplication, and we can bound the rank of their tensor sums, then we have a bound on ω. It says the following:

τ-theorem: If $\underline{R}(\bigoplus_{i=1}^{\rho} \langle k_i, m_i, r_k \rangle) \leq r$ where $r > \rho$, then we can bound $\omega \leq 3\tau$, where we define τ as follows:

$$\sum_{i=1}^{\rho} (k_i m_i n_i)^\tau = r$$

One can see that such a τ must exists by the mean value theorem.

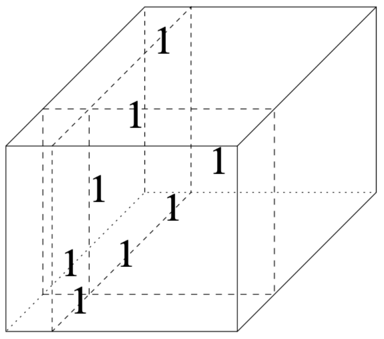
8 Strassen's Second Method (The Laser Method)

Strassen's Tensor

We first define Strassen's tensor to be the following:

$$\sum_{i=1}^{q} e_i \otimes e_0 \otimes e_i + e_0 \otimes e_i \otimes e_i$$

Note that the first term in the summand is simply $\langle q, 1, 1\rangle$ and the second $\langle 1, 1, q\rangle$, so Strassen's tensor is sum over q of such tensors. We can guess that this will eventually lead to the τ-theorem.



Strassen's Tensor, as depicted by Bläser, p. 35

Approximation of Strassen's Tensor

Consider the following approximation for Strassen's tensor:

$$\begin{aligned}
&\sum_{i=1}^{q}(e_0 + \varepsilon e_i) \otimes (e_0 + \varepsilon e_i) \otimes e_i - (e_0 \otimes e_0 \otimes \sum_{i=1}^{q} e_i) \\
&= -(e_0 \otimes e_0 \otimes \sum_{i=1}^{q} e_i) + \sum_{i=1}^{q} e_0 \otimes e_0 \otimes e_i + \varepsilon \sum_{i=1}^{q}(e_i \otimes e_0 \otimes e_i + e_0 \otimes e_i \otimes e_i) + \mathcal{O}(\varepsilon^2) \\
&= \varepsilon \sum_{i=1}^{q}(e_i \otimes e_0 \otimes e_i + e_0 \otimes e_i \otimes e_i) + \mathcal{O}(\varepsilon^2) && \text{(Distributive laws)}
\end{aligned}$$

Alas, the first term is just a sum of $q + 1$ rank-1 tensors, therefor we have that the rank of this approximation is bounded by $q + 1$, and therefor $\underline{R}(\text{Str}) \leq q + 1$.

Final Haul

We now give an overview of how Bläser proves Strassen's improved bound.

1. By applying decompositions and permutations to Strassen's tensor, we can make a new tensor "Str-Sym" by multiplying together 3 permutation on Strassen's tensor. The resulting tensor has an outer structer equivalent to $\langle 2, 2, 2\rangle$ (corresponding to 2×2 matrix mulitplication), and each inner structure is of the form $\langle k, m, n\rangle$ where $kmn = q^3$. So the whole tensor looks like matrix multiplication both on the inside and the "outside" (by viewing chunks as elements).

2. We call the "outside" view of this tensor Sym-Str$_{\mathcal{D}}$, and note that since $\underline{R}(Str) \le q+1$, we have $\underline{R}(\text{Sym-Str}_{\mathcal{D}}) \le (q+1)^3$. We take it to the sth tensorial power, and get that $\underline{R}(\text{Sym-Str}_{\mathcal{D}}^{\oplus s}) \le (q+1)^{3s}$

3. We show that the identity tensor of size $\frac{3}{4}2^{2s}$ is actually a good approximation (or in the language of the paper, a degeneration) of Sym-Str$_{\mathcal{D}}^{\oplus s}$, and moveover the all the 1's in this identity tensor correspond directly with the subtensors of the form $\langle k, m, n\rangle, kmn = q^3$ described earlier.

4. Then since Sym-Str$_{\mathcal{D}}^{\oplus s}$ is the sum of a bunch of tensors of the form $\langle k, m, n\rangle$, where $kmn = q^3$, by the τ-theorem, we can pick a τ such that $\omega \le 3\tau$ and the following holds:
$$\underline{R}(\text{Sym-Str}_{\mathcal{D}}^{\oplus s}) = \sum_{\text{inner tensors}} (q^{3s})^\tau = \frac{3}{4}2^{2s}(q^{3s})^\tau$$

5. Since we also have a bound on the border rank as $(q+1)^{3s}$, we get the following inequality:
$$\frac{3}{4}2^{2s}(q^{3s})^\tau \le (q+1)^3 s$$
Now we take the sth root of both sides, divide by 4, send $s \to \infty$ and take the log of both sides. We get
$$3\tau \le \log_q \frac{(q+1)^3}{4}$$
Which is minimal when $q = 5$ (which was the number of sums in our original approximation of Strassen's tensor). Then we can bound $\omega \le 2.48$ by the τ-theorem.

9 Further Improvements

Copersmith and Winograd[1]

Using a similar tensor to Strassen's, Copersmith and Winograd were able to improve this bound to $\omega \le 2.39$. The tensor is as follows:
$$\text{CW} = \sum_{i=1}^{q}(e_i \otimes e_0 \otimes e_i) + (e_0 \otimes e_i \otimes e_i) + (e_i \otimes e_i \otimes e_0)$$
The calculations are highly, highly trivial and left as nothing but a casual exercise to the reader, like a morning Sudoku (details in Bläser's paper)

Alman and Williams[2]

Improvements to the laser method are displayed, and the bound is slightly improved with $\omega \leq 2.37$.

Ambainis and Filmus[3]

A more generalized version of the laser method is presented, and it is shown that the method cannot prove the conjecture $\forall \varepsilon > 0, \omega < 2 + \varepsilon$.

10 Experimental Results

Purpose

So far, we've taken an in depth look at many theoretical bounds given on optimal matrix multiplication. However, with regard to the laser method, it's not clear how it should be implemented, if at all.

In particular, various optimizations, in practice, improve actual runtime for computation so much as to render them generally incomparable to theory-based approaches. For example, in the empirical studies from Kakaradov[4], the four main algorithms depicted are stated to be visually incomparable (on a graph of runtime) to the vectorized algorithm.

Most modern libraries, such as LAPACK or ATLAS (used in many python packages), actually use GEMM as it is optimized for cache and memory usage.[5] For this reason, in our experimental section, we don't focus on implementing the theoretically optimized bounds, but rather we compare theoretically varying algorithms with different cache-level optimization. The theoretical bounds given in the section above are galactic algorithm bounds; i.e. to yield improved results, we would need inordinately large input sizes.

Additionally, we implement an approximation algorithm for matrix multiplication, which if infinitely precise would yield $\omega \leq 2.78$, which is an improvement on Strassen first method. We see how the error increases with both size and iterations.

Optimization Emulation

Our emulation works as follows:

- A new object to emulate a cache is created, which must be called before a floating point operation is performed.
- Parameters to this object include approximate ram and cache access times, as well as time for one flop.
- Both the trivial algorithm and Strassen's algorithm are implemented from scratch, so that every addition, multiplication, and access can be monitored and counted.

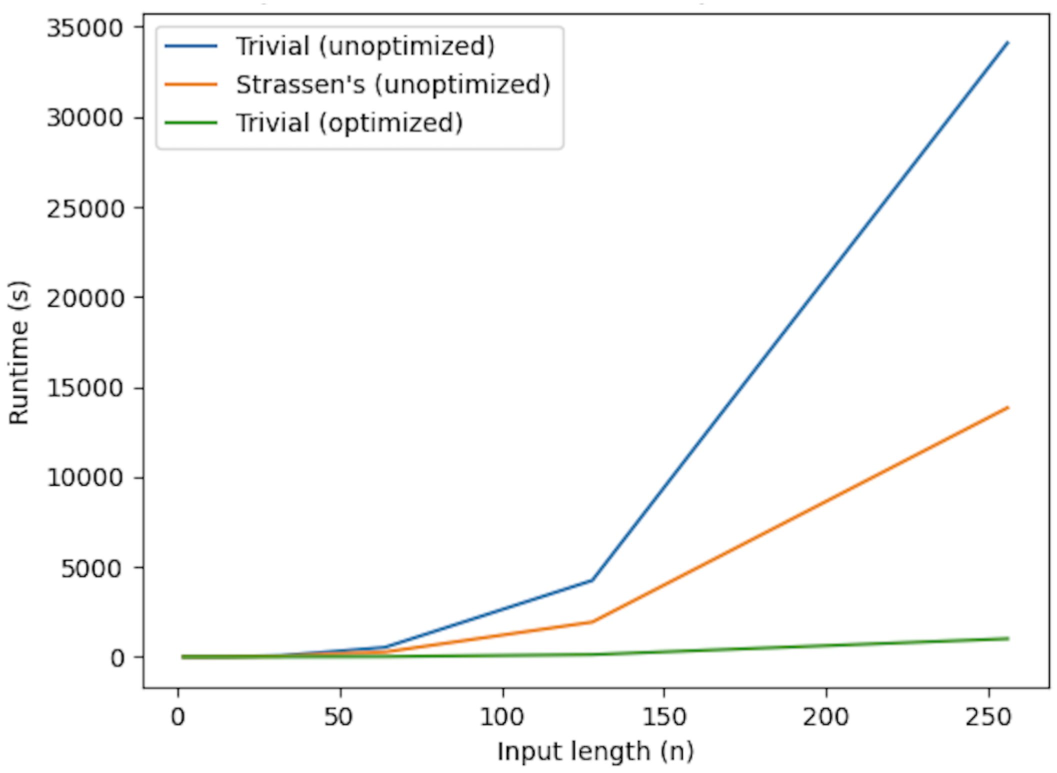
We compare three algorithms: *trivial unoptimized, Strassen's unoptimized* and *trivial optimized.* They were implemented as follows:

> *Trivial Unoptimized* – Here we use a cache access time of 1/100 the ram access time, and flop time equivalent to the cache time. The cache is a one row cache, which requires flushing to ram each time the row is updated.

Strassen's Unoptimized – The parameters are identical to the *trivial unoptimized* algorithm, except it is Strassen's algorithm instead.

Trivial Optimized – Here, we presume that all calculations are done with cache speed. Although this is clearly an optimistic assumption, it's a realistic approximation give how advanced many GPU's have become at doing just this.

Simulations are then run on random matrices, ranging in size from 2×2 through 256×256. The results are plotted below, with the caveat that the runtime should be taken relative to the other times, and not in absolute seconds, as the simulated values are scaled relative to each other.



Approximation Algorithms

Consider the following approximation (given by Bini, Capovani, Romani, Lotti) for matrix multiplication using 2×2 and 2×3 blocks:

$$\begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{13} \end{pmatrix} = \begin{pmatrix} z_{11} & z_{12} & z_{13} \\ z_{21} & z_{22} & z_{23} \end{pmatrix}$$

$$
\begin{aligned}
p_1 &= (x_{12} + \varepsilon x_{22})y_{21} \\
p_2 &= x_{11}(y_{11} + \varepsilon y_{12}) \\
p_3 &= x_{12}(y_{11} + y_{21} + \varepsilon y_{22}) \\
p_4 &= (x_{11} + x_{12} + \varepsilon x_{21})y_{11} \\
p_5 &= (x_{12} + \varepsilon x_{21})(y_{11} + y_{22})
\end{aligned}
$$

and then we get three of the six entries via:

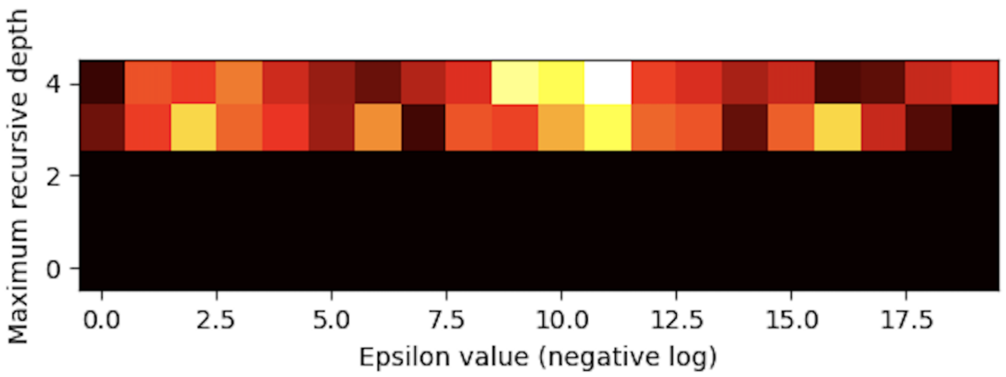
$$
\begin{aligned}
\varepsilon z_{11} &= \varepsilon p_1 + \varepsilon p_2 + \mathcal{O}(\varepsilon^2) \\
\varepsilon z_{12} &= p_2 - p_4 + p_5 + \mathcal{O}(\varepsilon^2) \\
\varepsilon z_{21} &= p_1 p_3 + p_5 + \mathcal{O}(\varepsilon^2).
\end{aligned}
$$

Now note that one can simply flip the rows of the right hand matrix, and switch the first and second column. Applying the same transformation on the right will yield the remaining three items by the same algorithm.

This means in total, we have 10 recursive multiplication instead of the traditional 12 which would be used in GEMM here. By solving the recurence, one can see that this algorithm runs with $\omega \leq 2.78$, which is an improvement on Strassen's recursive, divide and conquer algorithm.

But of course, there is an error term of $\mathcal{O}(\varepsilon)$ for each of these terms. How bad could that be? Well, it depends on how many recursive calls one has. And in fact, this very rapidly gives horrendous results.

We implement this algorithm, plotting a heat map on the the epsilon value on the x-axis and the maximum recursive depth on the y-axis (when the max depth is reached, we instead perform the trivial algorithm). It's clear that after two iterations, the algorithm becomes a terrible approximation (darker means lower error), and a small epsilon barely helps at all.



11 Citations

1. Bläser, M. (2013). Fast Matrix Multiplication. Theory of Computing, 1(1), 1–60. https://doi.org/10.4086/toc.gs.2013.005

2. Alman, Josh and Virginia Vassilevska Williams. "A Refined Laser Method and Faster Matrix Multiplication." SODA (2021).Andris Ambainis, Yuval Filmus, and François Le Gall. 2015. Fast Matrix Multiplication: Limitations of the Coppersmith-Winograd Method. In Proceedings of the forty-seventh annual ACM symposium on Theory of Computing (STOC '15). Association for Computing Machinery, New York, NY, USA, 585–593. DOI:https://doi.org/10.1145/2746539.2746554

3. Ambainis and Filmus, https://www.cs.toronto.edu/ yuvalf/AmbFilLeG14.pdf

4. Kakaradov, https://cs.stanford.edu/people/boyko/pubs/MatrixMult_SURJ_2004.pdf

5. https://www.quora.com/What-algorithm-does-BLAS-use-for-matrix-multiplication-Of-all-the-considerations-e-g-cache-popular-instruction-sets-Big-O-etc-which-one-turned-out-to-be-the-primary-bottleneck

6. Dario Bini, Milvio Capovani, Francesco Romani, and Grazia Lotti: $O(n^2.7799)$ complexity for $n \times n$ approximate matrix multiplication. Inform. Process. Lett., 8(5):234–235, 1979. [doi:10.1016/0020-0190(79)90113-3] 26