Making faster multiplications
Saad Mneimneh

1 A simple divide-and-conquer inspired by Gauss

Consider the multiplication of two complex numbers:

\[(a + bi)(c + di) = ac + (bc + ad)i - bd\]

which involves four multiplications: \(ac\), \(bc\), \(ad\), and \(bd\). Gauss observed that those quantities can be obtained by performing three multiplications only: \(ac\), \(bd\), and \((a + b)(c + d)\), then the term \((bc + ad)\) can be obtained as \((a + b)(c + d) - ac - bd\). Since multiplication of \(n\) bit numbers required \(\Theta(n^2)\) bit operations, compared to \(\Theta(n)\) for addition and subtraction, it may be worth reducing the number of multiplications. Consider two \(n\) bit numbers \(u\) and \(v\), and let us write

\[u = a \cdot 2^{n/2} + b\]
\[v = c \cdot 2^{n/2} + d\]

where \(a\), \(b\), \(c\), and \(d\) are \(n/2\) bit numbers. For simplicity, we may assume that \(n\) is a power of 2, but we can use floors and ceilings to adjust for an \(n\) that is not a power of 2.

\[uv = (a \cdot 2^{n/2} + b)(c \cdot 2^{n/2} + d) = ac \cdot 2^n + (bc + ad)2^{n/2} + bd\]

Using Gauss’ idea, we can perform three multiplications to obtain all terms. Note that multiplication by a power of 2 is really just a shift operation; therefore, if we apply this idea recursively, we obtain a recurrence for the time:

\[T(n) = 3T(n/2) + \Theta(n)\]

2 The Master theorem

Consider the following recurrence:

\[T(n) = \begin{cases} 
\Theta(1) & 1 \leq n \leq n_0 \\
adT(n/b) + \Theta(g(n)) & n > n_0
\end{cases}\]
where:
• $a \geq 1$
• $b > 1$
• $g$ is asymptotically positive

Then,
• $g(n)/n^{\log_b a} = O(n^{-\epsilon})$ for some $\epsilon > 0 \Rightarrow T(n) = \Theta(n^{\log_b a})$
• $g(n)/n^{\log_b a} = \Theta(\log^k n)$ for some $k \geq 0 \Rightarrow T(n) = \Theta(n^{\log_b a \log^{k+1} n})$
• $g(n)/n^{\log_b a} = \Omega(n^\epsilon)$ for some $\epsilon > 0$ and $ag(n/b) \leq cg(n)$ for some $c < 1$ and $n > n_0 \Rightarrow T(n) = \Theta(g(n))$

We usually interpret $n/b$ as either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. The proof of the Master theorem can be found in the book Introduction to Algorithms by CLRS.

3 A better Master theorem, the Bazzi method

Consider the following recurrence:

$$T(n) = \begin{cases} 
\Theta(1) & 1 \leq n \leq n_0 \\
\sum_{i=1}^{k} a_i T(n/b_i) + \Theta(g(n)) & n > n_0
\end{cases}$$

where:
• $n_0 > b_i$ and $n_0 \geq b_i/(b_i - 1)$ for $1 \leq i \leq k$
• $a_i > 0$ for $1 \leq i \leq k$
• $b_i > 1$ for $1 \leq i \leq k$
• $k \geq 1$
• $g(n)$ is non-negative and satisfies:

$$u \in [n/b_i, n] \Rightarrow c_1 g(n) \leq g(u) \leq c_2 g(n)$$

for $1 \leq i \leq k$ where $c_1$ and $c_2$ are positive constants

Then,

$$T(n) = \Theta\left(x^p \left(1 + \int_{1}^{n} \frac{g(u)}{u^{p+1}} du\right)\right)$$

where $p$ is the unique solution of $\sum_{i=1}^{k} a_i b_i^{-p} = 1$.

Again, we usually interpret $n/b_i$ as either $\lfloor n/b_i \rfloor$ or $\lceil n/b_i \rceil$. The proof of this theorem can be found at http://courses.csail.mit.edu/6.046/spring04/handouts/akrabazzi.pdf.

\footnote{Any function $g(n)$ of the form $n^\alpha \log^\beta n$ satisfies that condition.}
Examples:

- If $T(n) = 2T(n/4) + 3T(n/6) + \Theta(n \log n)$, then $p = 1$ and $T(n) = \Theta(n \log^2 n)$.
- If $T(n) = 2T(n/2) + \frac{8}{5}T(3n/4) + \Theta(n^2 / \log n)$, then $p = 2$ and $T(n) = \Theta(n^2 / \log \log n)$.
- If $T(n) = T(n/2) + \Theta(\log n)$, then $p = 0$ and $T(n) = \Theta(\log^2 n)$.
- If $T(n) = \frac{1}{2}T(n/2) + \Theta(1/n)$, then $p = -1$ and $T(n) = \Theta((\log n)/n)$.
- If $T(n) = 4T(n/2) + \Theta(n)$, then $p = 2$ and $T(n) = \Theta(n^2)$.

4 Back to Section 1

Our recurrence is:

$$T(n) = 3T(n/2) + \Theta(n)$$

Applying the Bazzi method (just for a change from the classical Master method), we get $3 \cdot 2^{-p} = 1 \Rightarrow p = \log_2 3$.

$$\int_1^n \frac{u}{u^{\log_2 3} + 1} du = \int_1^n \frac{u^{1 - \log_2 3} - 1}{1 - \log_2 3} du = \Theta(n^{1 - \log_2 3})$$

$$T(n) = \Theta(n^{\log_2 3} (1 + n^{1 - \log_2 3})) = \Theta(n^{\log_2 3} + n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.59})$$

5 Strassen’s divide-and-conquer algorithm

Consider the multiplication of two $n \times n$ matrices. If $c = ab$, then

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

Therefore, the running the basic algorithm for matrix multiplication is $\Theta(n^3)$ (each of the $n^2$ entries in $c$ requires $n$ multiplications and $n - 1$ additions). Strassen observed that if we divide the matrices into four blocks, we have the following (here $a$, $b$, $c$, $d$, $e$, $f$, $g$, and $h$ are all $\frac{n}{2} \times \frac{n}{2}$ matrices):

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & df + dh \end{bmatrix}$$

Therefore, in the most straightforward way, we require eight multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices. Once we have those results, we need $\Theta(n^2)$ time to combine them by adding $\frac{n}{2} \times \frac{n}{2}$ matrices. If we apply this idea recursively we get:

$$T(n) = 8T(n/2) + \Theta(n^2)$$
which leads to \( T(n) = \Theta(n^3) \). Strassen’s idea is to perform seven multiplications only, and combine them in \( \Theta(n^2) \) time as follows:

\[
\begin{bmatrix}
  a & b \\
  c & d
\end{bmatrix}
\begin{bmatrix}
  e & f \\
  g & h
\end{bmatrix}
= \begin{bmatrix}
  p_5 + p_4 - p_2 + p_6 \\
  p_3 + p_4 \\
  p_1 + p_2
\end{bmatrix}
\begin{bmatrix}
  p_1 + p_2 \\
  p_1 + p_5 - p_3 - p_7
\end{bmatrix}
\]

where
- \( p_1 = a(f - h) \)
- \( p_2 = (a + b)h \)
- \( p_3 = (c + d)e \)
- \( p_4 = d(g - e) \)
- \( p_5 = (a + b)(e + h) \)
- \( p_6 = (b - d)(g + h) \)
- \( p_7 = (a - c)(e + f) \)

Strassen’s algorithm leads to the following recurrence:

\[ T(n) = 7T(n/2) + \Theta(n^2) \]

which has \( T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.81}) \) as a solution (using results of Section 2 and/or Section 3).

6 Fast Fourier transform

Consider the problem of multiplying two polynomials \( a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_r x^r \) and \( b(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_s x^s \) (assume \( a_r \neq 0 \) and \( b_s \neq 0 \)). If \( c(x) = a(x)b(x) \), then \( c(x) \) has degree \( r + s \). We can expand the two polynomial to have \( n \) terms by adding zero coefficients. Therefore, let \( n - 1 \geq r + s \) and write:

\[
\begin{align*}
  a(x) &= a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} \\
  b(x) &= b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1} \\
  c(x) &= c_0 + c_1 x + c_2 x^2 + \ldots + c_{n-1} x^{n-1}
\end{align*}
\]

where \( c_j = \sum_{k=0}^{j} a_k b_{j-k} \) for \( 0 \leq j \leq n - 1 \).

Therefore, the most straightforward way for computing all \( c_j \)'s requires \( \Theta(n^2) \) time. We will explore a way to compute all \( c_j \)'s in \( \Theta(n \log n) \) using the Discrete Fourier transform (DFT), more specifically, an implementation of the it known as Fast Fourier Transform (FFT).

Given a polynomial \( a(x) \) of degree \( n - 1 \), let \( a(x_0), \ldots, a(x_{n-1}) \) be the values of \( a(x) \) on \( n \) distinct points \( x_0, \ldots, x_{n-1} \). One can show that \( a(x_0), \ldots, a(x_{n-1}) \) uniquely determine the polynomial \( a(x) \).
When \( x_0, \ldots, x_{n-1} \) are distinct, the matrix on the left is known as the Vandermonde matrix and is always invertible (the determinant is different than 0). Therefore, \( a_0, \ldots, a_{n-1} \) are uniquely determined.

Given \( a(x_0), \ldots, a(x_{n-1}) \), and similarly, \( b(x_0), \ldots, b(x_{n-1}) \), we can determine \( c(x_0), \ldots, c(x_{n-1}) \) in \( \Theta(n) \) time by simply multiplying the corresponding terms, i.e. \( c(x_j) = a(x_j)b(x_j) \).

Multiply \( a(x) \) and \( b(x) \)
1. obtain \( a(x_0), \ldots, a(x_{n-1}) \) from \( a_0, \ldots, a_{n-1} \)
2. obtain \( b(x_0), \ldots, b(x_{n-1}) \) from \( b_0, \ldots, b_{n-1} \)
3. compute \( c(x_j) = a(x_j)b(x_j) \) for \( 0 \leq j \leq n-1 \) in \( \Theta(n) \) time
4. obtain \( c_0, \ldots, c_{n-1} \) from \( c(x_0), \ldots, c(x_{n-1}) \)

We will show that each of steps (1), (2), and (4) can be done in \( \Theta(n \log n) \) time. The idea is to consider a special set of \( n \) values for \( x_0, \ldots, x_{n-1} \); they will consist of the \( n \) complex \( n^{th} \) roots of 1 (so they will be complex numbers).

Let \( n \) be a power of 2. Consider the complex number \( w = e^{i2\pi/n} = \cos 2\pi/n + i \sin 2\pi/n \). The powers of \( w \) are:

\[
1, w, w^2, \ldots, w^{n-1}
\]

where \( w^k = e^{i2\pi k/n} = \cos 2\pi k/n + i \sin 2\pi k/n \). Note that \( (w^k)^n = 1 \) and that’s why we call them the \( n \) complex \( n^{th} \) roots of 1, with \( w \) being the principal \( n^{th} \) root of 1.

\[
\begin{bmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & w & w^2 & \ldots & w^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & w^{n-1} & w^{2(n-1)} & \ldots & w^{(n-1)(n-1)}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
\vdots \\
a_{n-1}
\end{bmatrix}
= 
\begin{bmatrix}
a(1) \\
a(w) \\
\vdots \\
a(w^{n-1})
\end{bmatrix}
\]

We call \((a(1), \ldots, a(w^{n-1}))\) the Discrete Fourier Transform of \((a_0, \ldots, a_{n-1})\) where:

\[
a(w^j) = \sum_{i=0}^{n-1} w^{ij} a_i
\]
If we let \( a(x) = a_0(x) + xa_1(x) \) where
\[
a_0(x) = a_0 + a_2x + a_4x^2 + \ldots a_{n-1}x^{n/2-1}
\]
\[
a_1(x) = a_1 + a_3x + a_5x^2 + \ldots a_{n-2}x^{n/2-1}
\]
then \( a(w^k) = a_0(w^{2k}) + w^k a_1(w^{2k}) \). This means to evaluate \( a(x) \) at 1, \( w \), \( w^2 \), \ldots, \( w^{n-1} \), we need to evaluate \( a_0(x) \) and \( a_1(x) \) at \( 1^2 \), \( w^2 \), \ldots, \( (w^{n-1})^2 \). But the squares of the \( n^{th} \) roots of 1 are exactly the \( n/2^{nd} \) roots of 1. In fact
\[
(w^{k+n/2})^2 = (w^k)^2 \cdot w^n = (w^k)^2 \cdot 1 = (w^k)^2 = e^{\frac{ij\pi}{n/2}}
\]
Therefore, to evaluate \( a(x) \) on \( n \) points, we need to evaluate \( a_0(x) \) and \( a_1(x) \) on \( n/2 \) points. If we apply this recursively, it leads to the following recurrence for time:
\[
T(n) = 2T(n/2) + \Theta(n)
\]
This means that obtaining \( a(1), a(w), \ldots, a(w^{n-1}) \) requires \( \Theta(n \log n) \) time. That’s the Fast Fourier Transform (FFT). Once we obtain \( c(1), c(w), \ldots, c(w^{n-1}) \) we need to compute \( c_0, \ldots, c_{n-1} \).

\[
\begin{bmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & w & w^2 & \ldots & w^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & w^{n-1} & w^{2(n-1)} & \ldots & w^{(n-1)(n-1)}
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1 \\
\vdots \\
c_{n-1}
\end{bmatrix}
= 
\begin{bmatrix}
c(1) \\
c(w) \\
\vdots \\
c(w^{n-1})
\end{bmatrix}
\]

If we denote the matrix on the left by \( V \), where \( V_{ij} = w^{ij} \) (assuming indexing starts at 0), it is not hard to see that \( V^{-1} \) is such that \( V_{ij}^{-1} = \frac{1}{n} w^{-ij} \).

\[
[VV^{-1}]_{ij} = \sum_{k=0}^{n-1} V_{ik}V_{kj}^{-1} = \frac{1}{n} \sum_{k=0}^{n-1} (w^{i-j})^k
\]
If \( i - j \) is a multiple of \( n \) (this happens only when \( i - j = 0 \), i.e. \( i = j \)), and hence \( w^{i-j} \) is 1, the above sum is 1. Otherwise, the sum is a geometric sum equal to
\[
\frac{(w^{i-j})^n - 1}{w^{i-j} - 1} = \frac{(w^n)^{i-j} - 1}{w^{i-j} - 1} = \frac{1 - 1}{w^{i-j} - 1} = 0
\]
Therefore,
\[
\begin{bmatrix}
c_0 \\
c_1 \\
\vdots \\
c_{n-1}
\end{bmatrix}
= 
\begin{bmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & w^{-1} & w^{-2} & \ldots & w^{-(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & w^{-(n-1)} & w^{-(2(n-1))} & \ldots & w^{-(n-1)(n-1)}
\end{bmatrix}
\begin{bmatrix}
c(1) \\
c(w) \\
\vdots \\
c(w^{n-1})
\end{bmatrix}
\]
The right side looks like a Discrete Fourier transform with \( w \) replaced by \( w^{-1} \).
Therefore, \( c_0, \ldots, c_{n-1} \) can be also obtained in \( \Theta(n \log n) \) time. The inverse DFT is given by:

\[
c_j = \frac{1}{n} \sum_{i=0}^{n-1} w^{-ij} c(w^i)
\]

7 Schönhage-Strassen algorithm

We revisit the problem of multiplying two \( n \) bit numbers \( u \) and \( v \). Divide \( u \) and \( v \) into \( K \) blocks of \( l \) bits each. We take \( K \) to be a power of 2 as follows:

\[
K = 2^k, \quad L = 2^l, \quad 2n \leq 2^k l < 4n
\]

Therefore, \( u \) and \( v \) can be viewed as \( K \) digit numbers in base \( L \)

\[
u = u_{K-1}L^{K-1} + \ldots + u_1L + u_0, \quad v = v_{K-1}L^{K-1} + \ldots + v_1L + v_0
\]

Note that since \( 2^{k-1}l \geq n \), \( u_j = v_j = 0 \) for \( j \geq K/2 \). We would like to compute \( w = uv \), and by applying FFT, we can find \( (w_{K-2}, \ldots, w_0) \).

\[
w = w_{K-2}L^{K-2} + \ldots + w_1L + w_0
\]

Assuming we are using \( m \) bits for carrying out the arithmetic operations for FFT and inverse FFT, the running of this procedure is \( O(K \log KM) = O(Mnk/l) \) where \( M \) is the time required for \( m \)-bit multiplications. Note that \( w_r < (r+1)L^2 < KL^2 \); therefore, each \( w_r \) has at most \( k + 2l \) bits and hence reconstructing the binary representation of \( w \) requires \( O(K(k+l)) = O(n+nk/l) \) time. The total running time of this algorithm is \( O(n) + O(Mnk/l) \).

Schönhage and Strassen showed that if \( k \geq 7 \), \( m \geq 4k + 2l \), and \( w^0, \ldots, w^{K-1} \) are computed in a specific way, then all \( m \)-bit multiplications of complex numbers will not propagate much error and will round to the correct integers \( w_r \). We omit the messy details. Therefore, we have

\[
2n \leq 2^k l < 4n
\]

\[
k \geq 7
\]

\[
m \geq 4k + 2l
\]

A practical example: if \( n = 2^{13} \), we can choose \( k = 11, l = 8 \), and \( m = 60 \). Therefore, with today’s double precision arithmetic, we can multiply 8192 bit numbers in practically \( O(n) \) time (thinking of \( M \) as a constant because we are using the hardware of the machine).

Theoretically, we can choose \( k = l \) and \( m = 6k \); this choice of \( k \) is always less than \( \log n \):

\[
2^k k < 4n
\]

\[
2^{k-2} k < n
\]
\[ k - 2 + \log k < \log n \]

Since \( k \geq 7 \), \( \log k > 2 \) and \( k < \log n \).

Therefore, if we apply the algorithm recursively for the \( m \)-bit multiplications, we get \( T(n) = O(nT(\log n)) \). Therefore,

\[ T(n) \leq cn(c \log n)(c \log \log n)(c \log \log \log n) \ldots \]

With a variant of this algorithm, and more careful analysis, Schönhage and Strassen achieved an \( O(n \log n \log \log n) \) time algorithm, which remained the best until 2007.