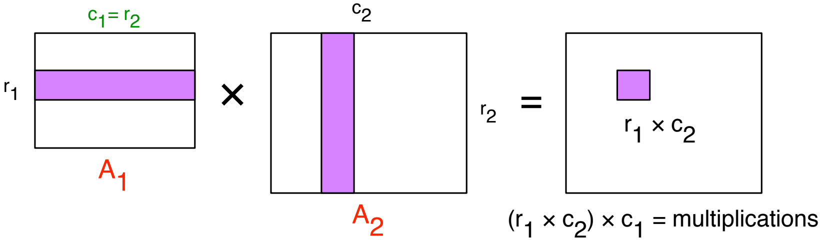
Strassen's Algorithm

Slides by Carl Kingsford

Feb. 21, 2014

Matrix Multiplication

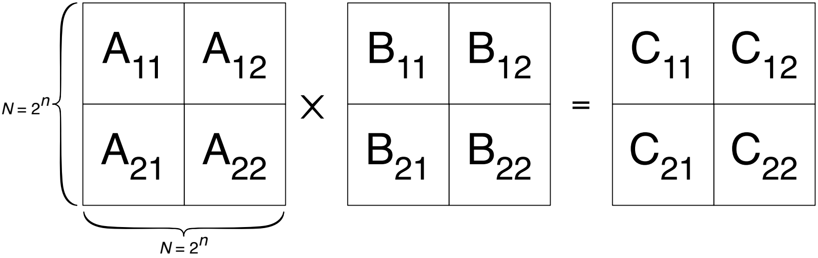


If $r_1 = c_1 = r_2 = c_2 = N$, this standard approach takes $\Theta(N^3)$:

- For every row $\vec{r}$ (N of them)
- For every column $\vec{c}$ (N of them)
- Take their inner product: $r \cdot c$ using N multiplications

Can we multiply faster than $\Theta(N^3)$?

For simplicity, assume $N = 2^n$ for some n. The multiplication is:



- $C_{11} = A_{11}B_{11} + A_{12}B_{21}$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21}$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22}$
- $C_{22} = A_{21}B_{12} + A_{22}B_{22}$

Uses 8 multiplications

Strassen's Algorithm

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\begin{aligned}
P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
P_2 &= (A_{21} + A_{22})B_{11} \\
P_3 &= A_{11}(B_{12} - B_{22}) \\
P_4 &= A_{22}(B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{12})B_{22} \\
P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
P_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}$$

$$\begin{aligned}
C_{11} &= P_1 + P_4 - P_5 + P_7 \\
C_{12} &= P_3 + P_5 \\
C_{21} &= P_2 + P_4 \\
C_{22} &= P_1 - P_2 + P_3 + P_6
\end{aligned}$$

Uses only 7 multiplications!

Since the submatrix multiplications are the expensive operations, we save a lot by eliminating one of them.

Apply the above idea recursively to perform the 7 matrix multiplications contained in $P_1, \ldots, P_7$.

Need to show how much savings this results in overall.

Recurrence

$$T(N) = T(2^n) = \underbrace{7T(2^n/2)}_{\text{recursive } \times} \quad + \quad \underbrace{c4^n}_{\textit{additions}}$$

Solving the recurrence:

$$\frac{T(2^n)}{7^n} = \frac{7T(2^{n-1})}{7^n} + \frac{c4^n}{7^n} = \frac{T(2^{n-1})}{7^{n-1}} + \frac{c4^n}{7^n}$$

The red term is same as the left-hand side but with $n-1$, so we can recursively expand:

$$\frac{T(2^n)}{7^n} = \gamma + \sum_{i=1}^{n} \frac{c4^i}{7^i} = \gamma + c\sum_{i=1}^{n}\left(\frac{4}{7}\right)^i \leq \alpha \quad \text{for some constants } \alpha, \gamma$$

So:

$$T(2^n) \leq 7^n\alpha = \alpha 2^{n\log_2(7)} = \alpha N^{2.807\ldots} = O(N^{2.807\ldots})$$

Summary

- Strassen first to show matrix multiplication can be done faster than $O(N^3)$ time.
- Strassen's algorithm gives a performance improvement for large-ish N, depending on the architecture, e.g. $N > 100$ or $N > 1000$.
- Strassen's algorithm isn't optimal though! Over the years it's been improved:

Authors	Year	Runtime
Strassen	1969	$O(N^{2.807})$
⋮		
Coppersmith & Winograd	1990	$O(N^{2.3754})$
Stothers	2010	$O(N^{2.3736})$
Williams	2011	$O(N^{2.3727})$

- Conjecture: an $O(N^2)$ algorithm exists.

Karatsuba's Algorithm for Integer Multiplication

Integer Multiplication

```
       10101110
     × 01011101
   ------------
       10101110
     10101110
    10101110
   10101110
 10101110
 --------------
 11111100110110
```

n numbers of *n* bits each
$O(n^2)$-time

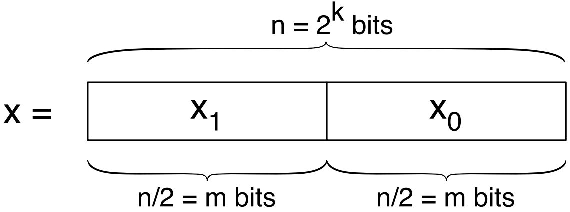
Start similar to Strassen's algorithm, breaking the items into blocks ($m = n/2$):

- $x = x_1 2^m + x_0$
- $y = y_1 2^m + y_0$

Then:

$$xy = (x_1 2^m + x_0)(y_1 2^m + y_0) = x_1 y_1 2^{2m} + (x_1 y_0 + x_0 y_1) 2^m + x_0 y_0$$

Breaking x and y into blocks



$x_1 2^m$ can be computed via "shift right by m"

So this multiplication only costs $O(n)$ operations.

4 Multiplications $\rightarrow$ 3 Multiplications

$$xy = x_1 y_1 2^{2m} + (x_1 y_0 + x_0 y_1) 2^m + x_0 y_0$$

We can write two multiplications as one, plus some subtractions:

$$x_1 y_0 + x_0 y_1 = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$$

But what we need to subtract is exactly what we need for the original multiplication!

- $p_0 = x_0 y_0$
- $p_1 = x_1 y_1$
- $p_2 = (x_1 + x_0)(y_1 + y_0) - p_1 - p_0$

$$xy = p_1 2^{2m} + p_2 2^m + p_0$$

Analysis

Assume $n = 2^k$ for some k (this is the common case when the integers are stored in computer words):

$$\begin{aligned} T(2^k) &= 3T(2^{k-1}) + c2^k \\ \frac{T(2^k)}{3^k} &= \frac{T(2^{k-1})}{3^{k-1}} + \frac{c2^k}{3^k} \\ &= \gamma + c\sum_{i=1}^{k} \frac{2^i}{3^i} \\ &\le \beta \quad \text{for some constants } \gamma, \beta \end{aligned}$$

(γ handles the constant work for the base case.) So:

$$T(2^k) \le \beta 3^k = \beta(2^k)^{\log_2(3)} = \beta n^{\log_2(3)} = O(n^{1.58\ldots})$$