Stassen's Matrix Multiplication

Dr. Bibhudatta Sahoo
Communication & Computing Group
Department of CSE, NIT Rourkela
Email: bdsahu@nitrkl.ac.in, 9937324437, 2462358

Multiplication of Large Integers

Consider the problem of multiplying two (large) *n*-digit integers represented by arrays of their digits such as:

A = 12345678901357986429 B = 87654321284820912836

The grade-school algorithm:

$$\begin{array}{r} a_1\ a_2 \ldots\ a_n \\ b_1\ b_2 \ldots\ b_n \\ \hline (d_{10})\, d_{11} d_{12} \ldots d_{1n} \\ (d_{20})\, d_{21} d_{22} \ldots d_{2n} \\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots \\ (d_{n0})\, d_{n1} d_{n2} \ldots d_{nn} \end{array}$$

Efficiency: $\Theta(n^2)$ single-digit multiplications

First Divide-and-Conquer Algorithm

A small example: $A * B$ where $A = 2135$ and $B = 4014$

$A = (21 \cdot 10^2 + 35)$, $B = (40 \cdot 10^2 + 14)$

So, $A * B = (21 \cdot 10^2 + 35) * (40 \cdot 10^2 + 14)$

$= 21 * 40 \cdot 10^4 + (21 * 14 + 35 * 40) \cdot 10^2 + 35 * 14$

In general, if $A = A_1A_2$ and $B = B_1B_2$ (where A and B are n-digit, A_1, A_2, B_1, B_2 are $n/2$-digit numbers),

$A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$

Recurrence for the number of one-digit multiplications $M(n)$:

$M(n) = 4M(n/2) + c\,n$,

$M(1) = 1$

Solution: $M(n) = n^2$

Second Divide-and-Conquer Algorithm

$$A * B = \mathbf{A_1 * B_1} \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$$

The idea is to decrease the number of multiplications from 4 to 3:

$$(A_1 + A_2) * (B_1 + B_2) = A_1 * B_1 + (A_1 * B_2 + A_2 * B_1) + A_2 * B_2,$$

i.e., $(A_1 * B_2 + A_2 * B_1) = (A_1 + A_2) * (B_1 + B_2) - A_1 * B_1 - A_2 * B_2,$ which requires only 3 multiplications at the expense of (4-1) extra add/sub.

Recurrence for the number of multiplications M(n):

$$M(n) = 3M(n/2), \quad M(1) = 1$$

Solution: $M(n) = 3^{\log_2 n} = n^{\log_2 3} \approx n^{1.585}$

Example of Large-Integer Multiplication

2135 * 4014

= (21*10^2 + 35) * (40*10^2 + 14)

= (21*40)*10^4 + c1*10^2 + 35*14

where c1 = (21+35)*(40+14) - 21*40 - 35*14, and

21*40 = (2*10 + 1) * (4*10 + 0)

= (2*4)*10^2 + c2*10 + 1*0

where c2 = (2+1)*(4+0) - 2*4 - 1*0, etc.

This process requires 9 digit multiplications as opposed to 16.

Sample problems

- **Complex multiplication:** If we multiply two complex numbers *a+bi* and *c+di* in the standard way, it requires four multiplications and two additions of real numbers. Since multiplications are more expensive than additions (and subtractions), it pays to minimize the number of multiplications if one would allow large numbers. Find an algorithm that uses only three multiplications (but more additions) to multiply two complex numbers.

Matrix Multiplication

Let A and B to be two n*n matrices. We wish to compute their product C=AB, $n = 2^k$; for some nonnegative integer k

The Problem:

$$\text{Given } A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \text{ and } B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{bmatrix}$$

$$\text{Compute } C = A \times B = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mk} \end{bmatrix} \text{ where } c_{ij} = \sum_{s=1}^{n} a_{is} \times b_{sj}$$

Complexity (on uniprocessor)

- Best known lower bound: $\Omega(n^2)$
  (assume $m = \Theta(n)$ and $k = \Theta(n)$)
- Straightforward algorithm: $O(n^3)$.
- Strassen's algorithm: $O(n^{\log 7}) = O(n^{2.81})$. [1969]
- Best known sequential algorithm: $O(n^{2.376})$ [1990]
- The best algorithm for this problem is still open.

Strassen's Matrix Multiplication

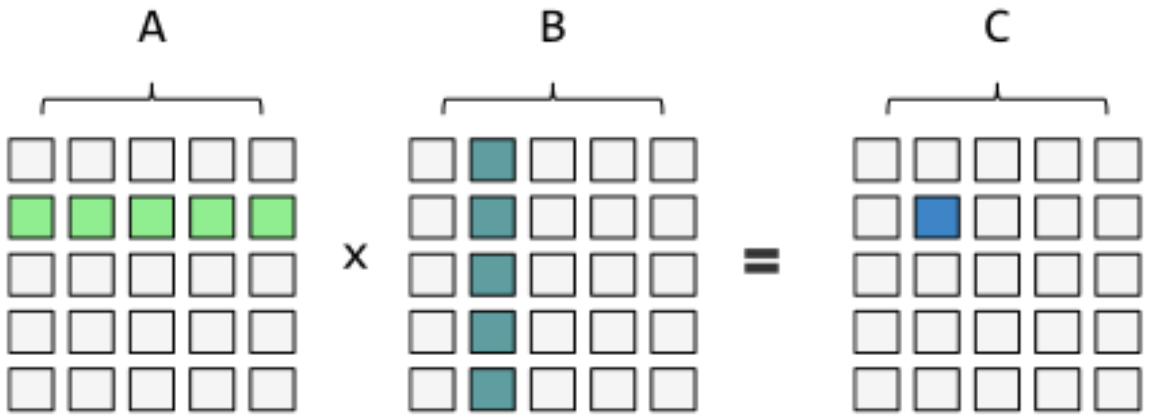
- **History**

  Volker Strassen is a German mathematician born in 1936. He is well known for his works on probability, but in the computer science and algorithms he's mostly recognized because of his algorithm for matrix multiplication that's still one of the main methods that outperforms the general matrix multiplication algorithm.

  Strassen firstly published this algorithm in 1969 and proved that the n^3 algorithm isn't the optimal one. Actually the given solution by Strassen is slightly better, but his contribution is enormous because this resulted in many more researches about matrix multiplication that led to some faster approaches, i.e. the Coppersmith-Winograd algorithm with $O(n^{2,3737})$.

Matrix Multiplication

MATRIX MULTIPLICATION




```
C[i][j] = sum(A[i][k] * B[k][j]) for k = 0 ... n
```

```
In our case:
C[1][1] =>
A[1][0]*B[0][1] + A[1][1]*B[1][1] + A[1][2]*B[2][1] + A[1][3]*B[3][1] + A[1][4]*B[4][1]
```

The Straightforward Method

```
procedure MATRIX MULTIPLICATION(A, B, C)
    for i = 1 to m do
        for j = 1 to k do
            (1) c_ij ← 0
            (2) for s = 1 to n do
                    c_ij ← c_ij + a_is×b_sj
                end for
        end for
    end for.
```

- It takes $O(m \cdot n \cdot k) = O(n^3)$ time.

Conventional Matrix Multiplication

8 multiplications

4 additions

- Brute-force algorithm

$$\begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} * \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix}$$

$$= \begin{pmatrix} a_{00} * b_{00} + a_{01} * b_{10} & a_{00} * b_{01} + a_{01} * b_{11} \\ a_{10} * b_{00} + a_{11} * b_{10} & a_{10} * b_{01} + a_{11} * b_{11} \end{pmatrix}$$

$$T(n) = \begin{cases} b & n \leq 2 \\ 8T\left(\frac{n}{2}\right) + cn^2 & n > 2 \end{cases}$$

Efficiency class in general: Θ (n^3)

Divide and conquer

A Divide & Conquer approach

Partition A, B and C into $\frac{n}{2} \times \frac{n}{2}$ matrices as such:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad B = \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

$$C = \begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} ae + bf & ag + bh \\ ce + df & cg + dh \end{pmatrix}$$

This reduces the $n \times n$ problem to 8 $\frac{n}{2} \times \frac{n}{2}$ problems, with an overhead of $4\left(\frac{n}{2}\right)^2$ additions.

The number $T(n)$ of arithmetic operations used to compute C is thus:

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 8T(\frac{n}{2}) + n^2 & \text{if } n > 1. \end{cases}$$

We can show that $T(n) = \Theta(n^3)$.

Strassen's Matrix Multiplication

- Strassen's algorithm for two 2x2 matrices (1969):

$$\begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} * \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix}$$

$$= \begin{pmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{pmatrix}$$

- $m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$
- $m_2 = (a_{10} + a_{11}) * b_{00}$
- $m_3 = a_{00} * (b_{01} - b_{11})$
- $m_4 = a_{11} * (b_{10} - b_{00})$
- $m_5 = (a_{00} + a_{01}) * b_{11}$
- $m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$
- $m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11})$

This requires 7 multiplications and 10 additions

Strassen's algorithm

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

- $T(n) = 7T(n/2) + O(n^2)$

  $= O(n^{\log 7}) = O(n^{2.81})$

$$m_1 = (b - d)\cdot(f + h)$$
$$m_2 = (a + d)\cdot(e + h)$$
$$m_3 = (a - c)\cdot(e + g)$$
$$m_4 = (a + b)\cdot h$$
$$m_5 = a\cdot(g - h)$$
$$m_6 = d\cdot(f - e)$$
$$m_7 = (c + d)\cdot e$$

$$r = m_1 + m_2 - m_4 + m_6$$
$$= (b - d)\cdot(f + h) + (a + d)\cdot(e + h) - (a + b)\cdot h + d\cdot(f - e)$$
$$= bf + bh - df - dh + ae + ah + de + dh - ah - bh + df - de$$
$$= a\cdot e + b\cdot f$$

$$s = m_4 + m_5 = (a + b)\cdot h + a\cdot(g - h) = a\cdot g + b\cdot h$$

$$t = m_6 + m_7 = d\cdot(f - e) + (c + d)\cdot e = c\cdot e + d\cdot f$$

$$u = m_2 - m_3 - m_5 - m_7 = c\cdot g + d\cdot h$$

Time Analysis

$$T(1) = 1 \qquad (\text{assume } N = 2^k)$$

$$T(N) = 7T(N/2)$$

$$T(N) = 7^k T(N/2^k) = 7^k$$

$$T(N) = 7^{\log N} = N^{\log 7} = N^{2.81}$$

Time Analysis

$$T(n) = \begin{cases} b & n \leq 2 \\ 7T\left(\frac{n}{2}\right) + cn^2 & n > 2 \end{cases}$$

- With constant *a* and *b*
- Using master' s theorem
- as a > b^k

$$T(n) = O\left(n^{\log_2 7}\right)$$

$$T(n) = O\left(n^{\log_2 7}\right) \approx O(n^{2.81})$$

Recurrence for the number of additions required by Strassens's algorithm

$$T(n) = \begin{cases} b & n \leq 2 \\ 18T\left(\frac{n}{2}\right) + cn^3 & n > 2 \end{cases}$$

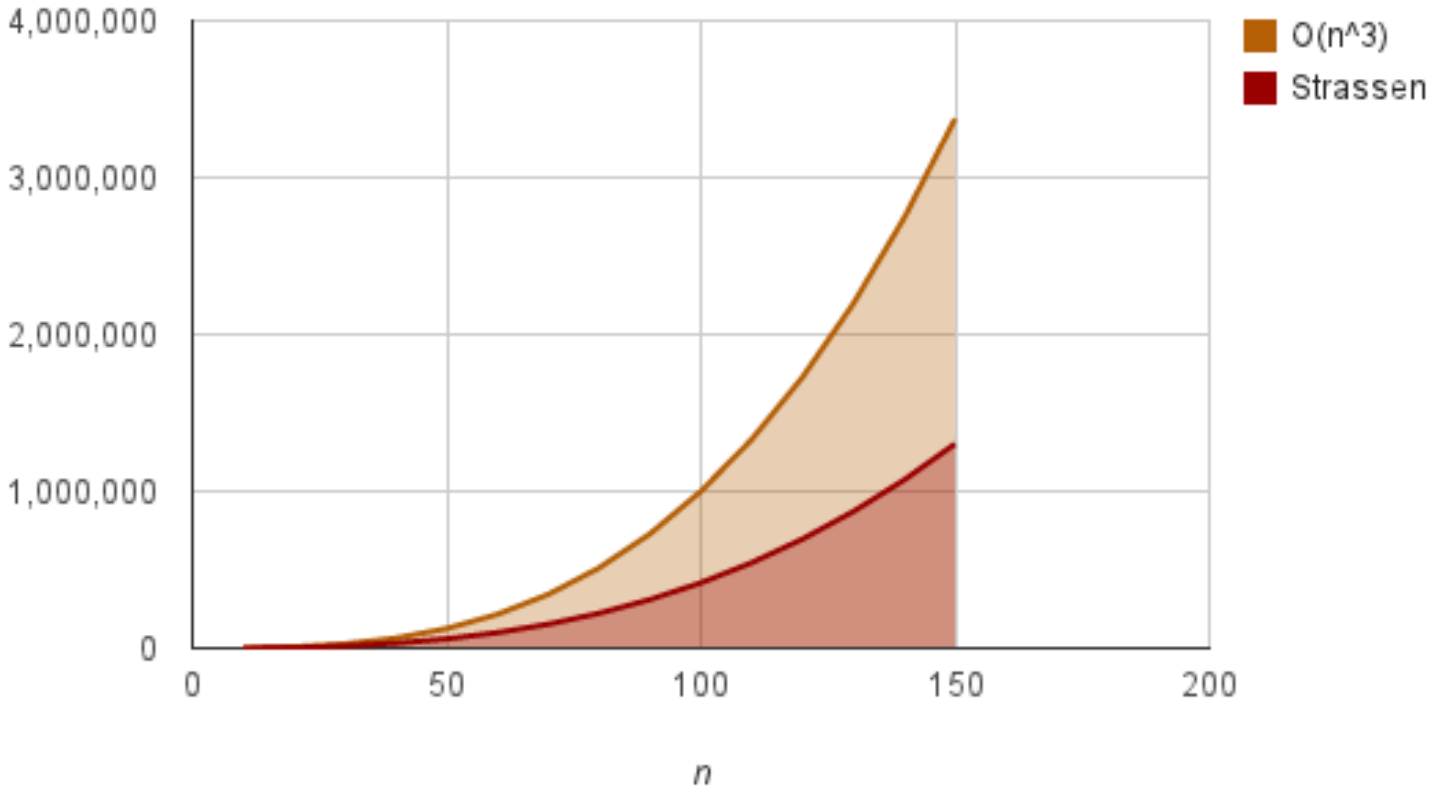
- Solve the recurrence relation ?

Strassen Algorithm

```
void matmul(int *A, int *B, int *R, int n) {
  if (n == 1) {
    (*R) += (*A) * (*B);
  } else {
    matmul(A, B, R, n/4);
    matmul(A, B+(n/4), R+(n/4), n/4);
    matmul(A+2*(n/4), B, R+2*(n/4), n/4);
    matmul(A+2*(n/4), B+(n/4), R+3*(n/4), n/4);
    matmul(A+(n/4), B+2*(n/4), R, n/4);
    matmul(A+(n/4), B+3*(n/4), R+(n/4), n/4);
    matmul(A+3*(n/4), B+2*(n/4), R+2*(n/4), n/4);
    matmul(A+3*(n/4), B+3*(n/4), R+3*(n/4), n/4);
  }
```

Divide matrices in sub-matrices and recursively multiply sub-matrices

Performance analysis



Algorithms for Multiplying Two *n*×*n* Matrices

A recursive algorithm based on multiplying two $m \times m$ matrices using k multiplications will yield an $O(n^{\log_m k})$ algorithm.

To beat Strassen's algorithm: $\log_m k < \log_2 7 \Rightarrow k < m^{\log_2 7}$.

So, for a 3×3 matrix, we must have: $k < 3^{\log_2 7} < 22$.

But the best known algorithm uses 23 multiplications!

Inventor	Year	Complexity
Classical	—	$\Theta(n^3)$
Volker Strassen	1968	$\Theta(n^{2.807})$
Victor Pan (multiply two 70×70 matrices using 143,640 multiplications)	1978	$\Theta(n^{2.795})$
Don Coppersmith & Shmuel Winograd (arithmetic progressions)	1990	$\Theta(n^{2.3737})$
Andrew Stothers	2010	$\Theta(n^{2.3736})$
Virginia Williams	2011	$\Theta(n^{2.3727})$

Limitations

- Generally Strassen's Method is not preferred for practical applications for following reasons.

1. The constants used in Strassen's method are high and for a typical application Naive method works better.
2. For Sparse matrices, there are better methods especially designed for them.
3. The sub-matrices in recursion take extra space.
4. Because of the limited precision of computer arithmetic on non-integer values, larger errors accumulate in Strassen's algorithm than in Naive Method .

Sample problems

1. Use Strassens's matrix multiplication algorithm to multiply the matrices

$$X = \begin{bmatrix} 3 & 2 \\ 4 & 8 \end{bmatrix} \qquad Y = \begin{bmatrix} 1 & 5 \\ 9 & 6 \end{bmatrix}$$

2. Program Strassens's matrix multiplication algorithm as well as classical matrix multiplication to (show that) determine when Strassens's matrix multiplication method outperforms the classical one.

3. Solve the recurrence for the number of additions required by Strassens's algorithm assuming that n is power of 2.

4. Practical implementation of Strassens's matrix multiplication algorithm usually switch to the brute force method after matrix sizes become smaller than some crossover point. Run an experiment to determine such crossover point on your computer system.

Sample problems

5. Why is it important that Strassen's algorithm does not use commutativity in the multiplication of 2 X2 matrices?
6. Show how to multiply two complex number X=a+ib and Y=c+id using only three multiplications.
7. [a] Show that how to multiply two numbers by solving five problems that are roughly one third of the original size.

   [b] Generalized this problem to obtain an $O(N^{1+\in})$ algorithm for any constant $\in > 0$.

   [c]Is the algorithm in part (b) better than O(N log N).

Sample problems

8. Using the "ordinary" method for multiplying 4-by-4 matrices, how many multiplications of numbers will be needed? Give a reason for your answer, more than just that your answer is given by a formula.
9. Use Strassen's method for multiplying matrices to see how many multiplications of numbers would be required to multiply 4-by-4 matrices in this case. [Hint: you must not only use Strassen's basic method, but also the divide-and-conquer strategy.] Give a reason for your answer. For full credit you must explain in some detail how Strassen's basic result allows one to multiply the 4-by-4 matrices using the number of multiplications that you indicate.

Sample problems

10. Matrix multiplication Strassen's algorithm multiplies two n×n-matrices in time $O(n^{2.808})$ by decomposition of 2×2-block matrices. It is faster than $O(n^3)$ because it computes seven multiplications instead of eight to form the product matrix. Another idea is to make the decomposition of 3×3-block matrices instead. A researcher at NADA tried a couple of years ago to find the minimum number of multiplications needed to multiply two 3× 3-matrices. He managed to get almost 22 multiplications. If he had succeeded, what would have been the time complexity to the multiplication of two n× n matrices?

Problems on Page no 194-195

7. It is possible to consider the product of matrices of size $n \times n$, where n is a power of 3. Using divide-and-conquer, the problem can be reduced to the multiplication of 3×3 matrices. The conventional method requires 27 multiplications. How many times must one be able to multiply 3×3 matrices so that the resultant computing time is smaller than $O(n^{2.81})$? How many times for 4×4 matrices?

8. For any even integer n, it is always possible to find integers m and k such that $n = m2^k$. To find the product of two $n \times n$ matrices, Strassen suggests partitioning them into $2^k \times 2^k$ submatrices each having $m \times m$ elements. Then start with Strassen's method to multiply the original matrices and uses the standard method for multiplying the required pairs of submatrices. Write a multiplication procedure for general n.

Thanks for Your Attention!