Matrix Multiplication

Rob H. Bisseling

Mathematical Institute, Utrecht University

Course Introduction Scientific Computing
February 8, 2021

Universiteit Utrecht

Traditional matrix multiplication

Strassen matrix multiplication

Universiteit Utrecht

Definition of matrix multiplication

- Let A be an $m \times n$ matrix and B an $n \times r$ matrix. Then $C = AB$ is the $m \times r$ matrix defined by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \quad \text{for } 1 \leq i \leq m,\ 1 \leq j \leq r.$$

- For ease of explanation, we will restrict ourselves here, without loss of generality, to square matrices, where A, B, C are matrices of size $n \times n$.
- Furthermore, we will assume that n is a power of 2.

Block matrix multiplication

- The matrix multiplication for square $n \times n$ matrices is given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \quad \text{for } 1 \leq i, j \leq n.$$

- Assume that A is divided into blocks A_{ij} of size $p \times p$,

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qq} \end{bmatrix},$$

  where $q = n/p$.
- Assume that B and C are divided in the same way as A. Then, the matrix multiplication in block form is

$$C_{ij} = \sum_{k=1}^{q} A_{ik} B_{kj}, \quad \text{for } 1 \leq i, j \leq q.$$

Traditional MM
Strassen MM

The choice $q = 2$

- For $q = 2$, the block matrix multiplication becomes

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

Recursive algorithm

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

function $\mathrm{MM}(A, B)$
  $n := \mathit{length}(A)$;
  if $n = 1$ **then**
    $C(1,1) := A(1,1) * B(1,1)$;
  else
    $m := n/2$;
    $A_{11} := A(1 : m, 1 : m)$; $B_{11} := B(1 : m, 1 : m)$;
    $A_{12} := A(1 : m, m+1 : n)$; $B_{12} := B(1 : m, m+1 : n)$;
    $A_{21} := A(m+1 : n, 1 : m)$; $B_{21} := B(m+1 : n, 1 : m)$;
    $A_{22} := A(m+1 : n, m+1 : n)$; $B_{22} := B(m+1 : n, m+1 : n)$;

    $C(1 : m, 1 : m) := \mathrm{MM}(A_{11}, B_{11}) + \mathrm{MM}(A_{12}, B_{21})$;
    $C(1 : m, m+1 : n) := \mathrm{MM}(A_{11}, B_{12}) + \mathrm{MM}(A_{12}, B_{22})$;
    $C(m+1 : n, 1 : m) := \mathrm{MM}(A_{21}, B_{11}) + \mathrm{MM}(A_{22}, B_{21})$;
    $C(m+1 : n, m+1 : n) := \mathrm{MM}(A_{21}, B_{12}) + \mathrm{MM}(A_{22}, B_{22})$;

  return C

Computation time

- A measure of computation time for an algorithm is the total number of floating-point operations, i.e. flops.
- If T_n is the number of flops for a recursive matrix multiplication of size n, we have

$$T_n = 8T_{n/2} + 4\left(\frac{n}{2}\right)^2 = 8T_{n/2} + n^2.$$

because of 8 matrix multiplications of size $n/2$ and 4 matrix additions. Furthermore, $T_1 = 1$. Therefore,

$$\begin{aligned} T_n &= 8T_{n/2} + n^2 = 8\left(8T_{n/4} + \left(\frac{n}{2}\right)^2\right) + n^2 \\ &= 8^2 T_{n/4} + (2+1)n^2 = \dots \\ &= 8^{\log_2 n} T_1 + (2^{\log_2 n - 1} + \dots + 2 + 1)n^2 \\ &= n^3 + (n-1)n^2 = 2n^3 - n^2. \end{aligned}$$

Advantages of the recursive computation

- The same number of flops as for the nonrecursive computation: n^2 elements of C each need n multiplications and $n-1$ additions, giving $n^2(2n-1) = 2n^3 - n^2$ flops.
- Advantage: once the submatrices involved become small enough, they fit in the computer cache and the flops are done much faster.
- We even do not have to know the cache size: the algorithm is cache-oblivious.
- This also works for cache hierarchies, such as for a node of the Dutch national supercomputer Cartesius:
  - L1 cache of 32 kB,
  - L2 cache of 256 kB,
  - L3 cache of 40 MB,
  - RAM memory of 64 GB.

Disadvantages of the recursive computation

- Disadvantage: the function calls have overhead, so don't get carried away; it is better to stop at $n = 8$ and perform 8×8 matrix multiplication in the straightforward way.
- Disadvantage: at the higher levels, many memory copies are performed; this may be improved by storing the matrix in a recursive data structure.
- In Matlab and Fortran: matrix storage is by columns. In C and C++, by rows.

Strassen matrix multiplication algorithm

function STRASSEN(A, B)
  $n := length(A)$;
  if $n = 1$ **then**
    $C(1,1) := A(1,1) * B(1,1)$;
  else
    $m := n/2$;
    $A_{11} := A(1 : m, 1 : m)$; ... also $A_{12}, A_{21}, A_{22}, B_{11}, B_{12}, B_{12}, B_{22}$

    $P_1 = \text{Strassen}(A_{11} + A_{22}, B_{11} + B_{22})$;
    $P_2 = \text{Strassen}(A_{21} + A_{22}, B_{11})$;
    $P_3 = \text{Strassen}(A_{11}, B_{12} - B_{22})$;
    $P_4 = \text{Strassen}(A_{22}, B_{21} - B_{11})$;
    $P_5 = \text{Strassen}(A_{11} + A_{12}, B_{22})$;
    $P_6 = \text{Strassen}(A_{21} - A_{11}, B_{11} + B_{12})$;
    $P_7 = \text{Strassen}(A_{12} - A_{22}, B_{21} + B_{22})$;

    $C(1 : m, 1 : m) := P_1 + P_4 - P_5 + P_7$;
    $C(1 : m, m + 1 : n) := P_3 + P_5$;
    $C(m + 1 : n, 1 : m) := P_2 + P_4$;
    $C(m + 1 : n, m + 1 : n) := P_1 + P_3 - P_2 + P_6$;

Correctness

- This algorithm does the job.
- Here, we will verify that C_{12} is computed correctly. For C_{11}, C_{21}, C_{22}, the proof is similar (but sometimes a bit tedious).
- We may assume by induction on the size of the matrix that Strassen works correctly for sizes up to $n/2$.
- Then

$$\begin{aligned} C_{12} &= P_3 + P_5 = A_{11}(B_{12} - B_{22}) + (A_{11} + A_{12})B_{22} \\ &= A_{11}B_{12} + A_{12}B_{22}, \end{aligned}$$

  which is the same as for the traditional computation of C_{12}.
- Note that matrix multiplication is not commutative ($AB \neq BA$). The algorithm always keeps submatrices of A on the left, and those of B on the right.

Fast matrix multiplication

- This magic algorithm has been proposed by Volker Strassen in 1969.
- It has only 7 multiplications of half-size matrices, instead of 8.
- This comes at the price of 18 additions/subtractions of half-size matrices, instead of 4.
- For $n = 2$, this is not worthwhile: we perform 25 flops instead of 12.
- But for $n > 2$, we benefit: matrix multiplications are much more expensive than additions or subtractions.

Computation time

- If T_n is the number of flops for a Strassen matrix multiplication of size n, we have

$$T_n = 7T_{n/2} + 18\left(\frac{n}{2}\right)^2 = 7T_{n/2} + \frac{9}{2}n^2.$$

because of 7 matrix multiplications of size $n/2$ and 18 matrix additions. Furthermore, $T_1 = 1$.

Computation time (cont'd)

The computation time of Strassen matrix multiplication equals

$$
\begin{aligned}
T_n &= 7T_{n/2} + \frac{9}{2}n^2 = 7\left(7T_{n/4} + \frac{9}{2}\left(\frac{n}{2}\right)^2\right) + \frac{9}{2}n^2 \\
&= 7^2 T_{n/4} + \frac{9}{2}\left(\frac{7}{4} + 1\right) n^2 = \ldots \\
&= 7^{\log_2 n} T_1 + \frac{9}{2}\left(\left(\frac{7}{4}\right)^{\log_2 n - 1} + \cdots + \frac{7}{4} + 1\right) n^2 \\
&= n^{\log_2 7} + 6\left(\left(\frac{7}{4}\right)^{\log_2 n} - 1\right) n^2 \\
&= n^{\log_2 7} + 6\left(n^{\log_2 \frac{7}{4}}\right) n^2 - n^2 \\
&= 7n^{\log_2 7} - n^2 \\
&\approx 7n^{2.81}.
\end{aligned}
$$

Summary

- Traditional multiplication of two $n \times n$ matrices takes $2n^3$ floating-point operations (flops).
- Strassen matrix multiplication takes only $7n^{2.81}$ flops.
- Recursive algorithms can be formulated concisely and may have cache benefits.
- Recursions should be finished early to avoid excessive function-call overheads.
- Now let's try it out: Live demo!