Matrix Multiplication

Adapted from the CLRS book slides

MULTIPLYING SQUARE MATRICES

Input: Three $n \times n$ (square) matrices, $A = (a_{ij})$, $B = (b_{ij})$, and $C = (c_{ij})$.

Result: The matrix product $A \cdot B$ is added into C, so that

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

for $i, j = 1, 2, \ldots, n$.

If only the product $A \cdot B$ is needed, then zero out all entries of C beforehand.

Straightforward method

MATRIX-MULTIPLY(A, B, C, n)

for $i = 1$ **to** n // compute entries in each of n rows
    for $j = 1$ **to** n // compute n entries in row i
        for $k = 1$ **to** n
            $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ // add in one more term of equation (4.1)

Time: $\Theta(n^3)$ because of triply nested loops.

SIMPLE DIVIDE-AND-CONQUER ALGORITHM

Simple divide-and-conquer algorithm

For simplicity, assume that C is initialized to 0, so computing $C = A \cdot B$.

If $n > 1$, partition each of A, B, C into four $n/2 \times n/2$ matrices:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

Rewrite $C = A \cdot B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

giving the four equations

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

Each of these equations multiplies two $n/2 \times n/2$ matrices and then adds their $n/2 \times n/2$ products. Assume that n is an exact power of 2, so that submatrix dimensions are always integer.

SIMPLE DIVIDE-AND-CONQUER ALGORITHM (continued)

Use these equations to get a divide-and-conquer algorithm:

MATRIX-MULTIPLY-RECURSIVE(A, B, C, n)

if $n == 1$
// Base case.
    $c_{11} = c_{11} + a_{11} \cdot b_{11}$
    return
// Divide.
partition A, B, and C into $n/2 \times n/2$ submatrices
    $A_{11}, A_{12}, A_{21}, A_{22}$; $B_{11}, B_{12}, B_{21}, B_{22}$;
    and $C_{11}, C_{12}, C_{21}, C_{22}$; respectively
// Conquer.
MATRIX-MULTIPLY-RECURSIVE$(A_{11}, B_{11}, C_{11}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{11}, B_{12}, C_{12}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{21}, B_{11}, C_{21}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{21}, B_{12}, C_{22}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{12}, B_{21}, C_{11}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{12}, B_{22}, C_{12}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{22}, B_{21}, C_{21}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{22}, B_{22}, C_{22}, n/2)$

The book briefly discusses the question of how to avoid copying entries when partitioning matrices. Can partition matrices without copying entries by instead using index calculations.

ANALYSIS

Let $T(n)$ be the time to multiply two $n \times n$ matrices.

Base case: $n = 1$. Perform one scalar multiplication: $\Theta(1)$.

Recursive case: $n > 1$.

- Dividing takes $\Theta(1)$ time, using index calculations. [Otherwise, $\Theta(n^2)$ time.]
- Conquering makes 8 recursive calls, each multiplying $n/2 \times n/2$ matrices $\Longrightarrow 8T(n/2)$.
- No combine step, because C is updated in place.

Recurrence (omitting the base case) is $T(n) = 8T(n/2) + \Theta(1)$. Can use master method to show that it has solution $T(n) = \Theta(n^3)$.

ANALYSIS (continued)

Bushiness of recursion trees:

Compare this recurrence with the merge-sort recurrence $T(n) = 2T(n/2) + \Theta(n)$. If we draw out the recursion trees, the factor of 2 in the merge-sort recurrenece says that each non-leaf node has 2 children. But the factor of 8 in the recurrence for MATRIX-MULTIPLY-RECURSIVE says that each non-leaf node has 8 children. Get a bushier tree with many more leaves, even though internal nodes have a smaller cost.

STRASSEN'S ALGORITHM (1969)

Idea: Make the recursion tree less bushy. Perform only 7 recursive multiplications of $n/2 \times n/2$ matrices, rather than 8. Will cost several additions/subtractions of $n/2 \times n/2$ matrices.

Since a subtraction is a "negative addition," just refer to all additions and subtractions as additions.

Example of reducing multiplications: Given x and y, compute $x^2 - y^2$. Obvious way uses 2 multiplications and one subtraction. But observe:

$$\begin{aligned} x^2 - y^2 &= x^2 - xy + xy - y^2 \\ &= x(x-y) + y(x-y) \\ &= (x+y)(x-y), \end{aligned}$$

so at the expense of one extra addition, can get by with only 1 multiplication. Not a big deal if x, y are scalars, but can make a difference if they are matrices.

STRASSEN'S ALGORITHM (continued)

The algorithm:

1. Same base case as before, when $n = 1$.
2. When $n > 1$, then as in the recursive method, partition each of the matrices into four $n/2 \times n/2$ submatrices. Time: $\Theta(1)$, using index calculations.
3. Create 10 matrices $S_1, S_2, \ldots, S_{10}$. Each is $n/2 \times n/2$ and is the sum or difference of two matrices created in previous step. Time: $\Theta(n^2)$ to create all 10 matrices.
4. Create and zero the entries of 7 matrices $P_1, P_2, \ldots, P_7$, each $n/2 \times n/2$. Time: $\Theta(n^2)$.
5. Using the submatrices of A and B and the matrices $S_1, S_2, \ldots, S_{10}$, recursively compute $P_1, P_2, \ldots, P_7$. Time: $7T(n/2)$.
6. Update the four $n/2 \times n/2$ submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of C by adding and subtracting various combinations of the P_i. Time: $\Theta(n^2)$.

Analysis

Recurrence will be $T(n) = 7T(n/2) + \Theta(n^2)$. By the master method, solution is $T(n) = \Theta(n^{\lg 7})$. Since $\lg 7 < 2.81$, the running time is $O(n^{2.81})$, beating the $\Theta(n^3)$-time methods.

STRASSEN'S ALGORITHM (continued)

Details

Step 2: Create the 10 matrices

$$
\begin{aligned}
S_1 &= B_{12} - B_{22} \, , \\
S_2 &= A_{11} + A_{12} \, , \\
S_3 &= A_{21} + A_{22} \, , \\
S_4 &= B_{21} - B_{11} \, , \\
S_5 &= A_{11} + A_{22} \, , \\
S_6 &= B_{11} + B_{22} \, , \\
S_7 &= A_{12} - A_{22} \, , \\
S_8 &= B_{21} + B_{22} \, , \\
S_9 &= A_{11} - A_{21} \, , \\
S_{10} &= B_{11} + B_{12} \, .
\end{aligned}
$$

Add or subtract $n/2 \times n/2$ matrices 10 times $\Rightarrow$ time is $\Theta(n^2)$.

STRASSEN'S ALGORITHM (continued)

Step 5: Compute the 7 matrices

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \,, \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \,, \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \,, \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \,, \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \,, \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} \,, \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12} \,.
\end{aligned}
$$

The only multiplications needed are in the middle column; right-hand column just shows the products in terms of the original submatrices of A and B.

STRASSEN'S ALGORITHM (continued)

Step 6: Add and subtract the P_i to construct submatrices of *C*:

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

EXAMPLE

Example of how C_{11} is reconstructed using additions and the previously defined P and S matrices.

$$
\begin{array}{lllllllll}
A_{11} \cdot B_{11} & + A_{11} \cdot B_{22} & + A_{22} \cdot B_{11} & + A_{22} \cdot B_{22} & & & \\
& & - A_{22} \cdot B_{11} & & + A_{22} \cdot B_{21} & & \\
& - A_{11} \cdot B_{22} & & & & - A_{12} \cdot B_{22} & \\
& & & - A_{22} \cdot B_{22} & - A_{22} \cdot B_{21} & + A_{12} \cdot B_{22} & + A_{12} \cdot B_{21} \\
\hline
A_{11} \cdot B_{11} & & & & & & + A_{12} \cdot B_{21}
\end{array}
$$

All four examples are fully worked out in the text.

THEORETICAL AND PRACTICAL NOTES

Strassen's algorithm was the first to beat $\Theta(n^3)$ time, but it's not the asymptotically fastest known. A method by Coppersmith and Winograd runs in $O(n^{2.376})$ time. Current best asymptotic bound (not practical) is $O(n^{2.37286})$.

Practical issues against Strassen's algorithm:

- Higher constant factor than the obvious $\Theta(n^3)$-time method.
- Not good for sparse matrices.
- Not numerically stable: larger errors accumulate than in the obvious method.
- Submatrices consume space, especially if copying.