CIS 575. Introduction to Algorithm Analysis
Material for February 26, 2024

Multiplying Large Matrices by Divide & Conquer

©2020 Torben Amtoft

The topic of this note is covered in *Cormen*'s Section 4.2.

1 Matrix Multiplication

We shall look at how to multiply two $n \times n$ matrices, and develop various algorithms. As is customary for this problem, we shall measure running time as a function of n (though strictly speaking it should be as a function of $n \cdot n$ since this is the input size).

First let us settle on notation: if A is a matrix, then a_{ij} is the element in the i'th row (from top) and the j'th column (from left). If A and B both are $n \times n$ matrices, then $A\ B$ is the $n \times n$ matrix C defined by

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

For example, with

$$A = \begin{pmatrix} 1 & 3 & 5 & 8 \\ 4 & 2 & 7 & 3 \\ 3 & 0 & 2 & 1 \\ 4 & 1 & 3 & 6 \end{pmatrix} \text{ and } B = \begin{pmatrix} 2 & 2 & 4 & 3 \\ 1 & 0 & 7 & 2 \\ 4 & 3 & 1 & 5 \\ 2 & 1 & 0 & 1 \end{pmatrix}$$

we have

$$A\ B = \begin{pmatrix} 1\cdot 2+3\cdot 1+5\cdot 4+8\cdot 2 & 1\cdot 2+3\cdot 0+5\cdot 3+8\cdot 1 & 1\cdot 4+3\cdot 7+5\cdot 1+8\cdot 0 & 1\cdot 3+3\cdot 2+5\cdot 5+8\cdot 1 \\ 4\cdot 2+2\cdot 1+7\cdot 4+3\cdot 2 & 4\cdot 2+2\cdot 0+7\cdot 3+3\cdot 1 & 4\cdot 4+2\cdot 7+7\cdot 1+3\cdot 0 & 4\cdot 3+2\cdot 2+7\cdot 5+3\cdot 1 \\ 3\cdot 2+0\cdot 1+2\cdot 4+1\cdot 2 & 3\cdot 2+0\cdot 0+2\cdot 3+1\cdot 1 & 3\cdot 4+0\cdot 7+2\cdot 1+1\cdot 0 & 3\cdot 3+0\cdot 2+2\cdot 5+1\cdot 1 \\ 4\cdot 2+1\cdot 1+3\cdot 4+6\cdot 2 & 4\cdot 2+1\cdot 0+3\cdot 3+6\cdot 1 & 4\cdot 4+1\cdot 7+3\cdot 1+6\cdot 0 & 4\cdot 3+1\cdot 2+3\cdot 5+6\cdot 1 \end{pmatrix}$$

which evaluates to

$$C = \begin{pmatrix} 41 & 25 & 30 & 42 \\ 44 & 32 & 37 & 54 \\ 16 & 13 & 14 & 20 \\ 33 & 23 & 26 & 35 \end{pmatrix}$$

The definition suggests a simple implementation:

```
for i ← 1 to n
    for j ← 1 to n
        c_ij ← 0
        for k ← 1 to n
            c_ij ← c_ij + a_ik · b_kj
```

which obviously (with our usual assumption that arithmetic operations can be executed in constant time) runs in time $\mathbf{\Theta(n^3)}$.

1.1 Naive Divide & Conquer

Given A and B that we want to multiply, we may view A as a 2×2 matrix where each a_{ij} may be a number, or a *smaller* matrix. To find $C = A\,B$ we recursively compute

$$\begin{aligned} c_{11} &= a_{11} \cdot b_{11} + a_{12} \cdot b_{21} \\ c_{12} &= a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ c_{21} &= a_{21} \cdot b_{11} + a_{22} \cdot b_{21} \\ c_{22} &= a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{aligned}$$

For our above example, we have

$$a_{11} = \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix} \quad a_{12} = \begin{pmatrix} 5 & 8 \\ 7 & 3 \end{pmatrix} \quad b_{11} = \begin{pmatrix} 2 & 2 \\ 1 & 0 \end{pmatrix} \quad b_{12} = \begin{pmatrix} 4 & 3 \\ 7 & 2 \end{pmatrix}$$
$$a_{21} = \begin{pmatrix} 3 & 0 \\ 4 & 1 \end{pmatrix} \quad a_{22} = \begin{pmatrix} 2 & 1 \\ 3 & 6 \end{pmatrix} \quad b_{21} = \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix} \quad b_{22} = \begin{pmatrix} 1 & 5 \\ 0 & 1 \end{pmatrix}$$

We can now compute the product, with a result that agrees with our previous calculations:

$$\begin{aligned} c_{11} &= \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix} \begin{pmatrix} 2 & 2 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 5 & 8 \\ 7 & 3 \end{pmatrix} \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 2 \\ 10 & 8 \end{pmatrix} + \begin{pmatrix} 36 & 23 \\ 34 & 24 \end{pmatrix} = \begin{pmatrix} 41 & 25 \\ 44 & 32 \end{pmatrix} \\ c_{12} &= \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix} \begin{pmatrix} 4 & 3 \\ 7 & 2 \end{pmatrix} + \begin{pmatrix} 5 & 8 \\ 7 & 3 \end{pmatrix} \begin{pmatrix} 1 & 5 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 25 & 9 \\ 30 & 16 \end{pmatrix} + \begin{pmatrix} 5 & 33 \\ 7 & 38 \end{pmatrix} = \begin{pmatrix} 30 & 42 \\ 37 & 54 \end{pmatrix} \\ c_{21} &= \begin{pmatrix} 3 & 0 \\ 4 & 1 \end{pmatrix} \begin{pmatrix} 2 & 2 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 2 & 1 \\ 3 & 6 \end{pmatrix} \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 6 & 6 \\ 9 & 8 \end{pmatrix} + \begin{pmatrix} 10 & 7 \\ 24 & 15 \end{pmatrix} = \begin{pmatrix} 16 & 13 \\ 33 & 23 \end{pmatrix} \\ c_{22} &= \begin{pmatrix} 3 & 0 \\ 4 & 1 \end{pmatrix} \begin{pmatrix} 4 & 3 \\ 7 & 2 \end{pmatrix} + \begin{pmatrix} 2 & 1 \\ 3 & 6 \end{pmatrix} \begin{pmatrix} 1 & 5 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 12 & 9 \\ 23 & 14 \end{pmatrix} + \begin{pmatrix} 2 & 11 \\ 3 & 21 \end{pmatrix} = \begin{pmatrix} 14 & 20 \\ 26 & 35 \end{pmatrix} \end{aligned}$$

To estimate the general running time, observe that we do 8 (different) multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices. Since addition can be done in time $\Theta(n^2)$, we get the recurrence

$$T(n) \in 8\,T(\frac{n}{2}) + \Theta(n^2)$$

and thus $T(n) \in \mathbf{\Theta(n^3)}$. We see that so far, applying the Divide & Conquer paradigm hasn't gained us anything; we have the same asymptotic running time as the straightforward implementation (which probably even runs faster). This illustrates that Divide & Conquer is not a panacea; to benefit from it, one usually needs to add a clever idea to the mix.

1.2 Clever Divide & Conquer

In 1969, Volker Strassen got such a clever idea: he showed that rather than doing eight multiplications, *seven* would suffice. One way of doing that (sources disagree about what was Strassen's original scheme) is to do the seven multiplications

$$\begin{aligned}
m_1 &= (a_{21} + a_{22} - a_{11})(b_{22} - b_{12} + b_{11}) \\
m_2 &= a_{11}b_{11} \\
m_3 &= a_{12}b_{21} \\
m_4 &= (a_{11} - a_{21})(b_{22} - b_{12}) \\
m_5 &= (a_{21} + a_{22})(b_{12} - b_{11}) \\
m_6 &= (a_{12} - a_{21} + a_{11} - a_{22})b_{22} \\
m_7 &= a_{22}(b_{11} + b_{22} - b_{12} - b_{21})
\end{aligned}$$

and then define

$$\begin{aligned}
c_{11} &= m_2 + m_3 \\
c_{12} &= m_1 + m_2 + m_5 + m_6 \\
c_{21} &= m_1 + m_2 + m_4 - m_7 \\
c_{22} &= m_1 + m_2 + m_4 + m_5
\end{aligned}$$

In our example, we would compute

$$\begin{aligned}
m_1 &= \begin{pmatrix} 4 & -2 \\ 3 & 5 \end{pmatrix} \begin{pmatrix} -1 & 4 \\ -6 & -1 \end{pmatrix} = \begin{pmatrix} 8 & 18 \\ -33 & 7 \end{pmatrix} \\
m_2 &= \begin{pmatrix} 5 & 2 \\ 10 & 8 \end{pmatrix} \\
m_3 &= \begin{pmatrix} 36 & 23 \\ 34 & 24 \end{pmatrix} \\
m_4 &= \begin{pmatrix} -2 & 3 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -3 & 2 \\ -7 & -1 \end{pmatrix} = \begin{pmatrix} -15 & -7 \\ -7 & -1 \end{pmatrix} \\
m_5 &= \begin{pmatrix} 5 & 1 \\ 7 & 7 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 6 & 2 \end{pmatrix} = \begin{pmatrix} 16 & 7 \\ 56 & 21 \end{pmatrix} \\
m_6 &= \begin{pmatrix} 1 & 10 \\ 4 & -2 \end{pmatrix} \begin{pmatrix} 1 & 5 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 15 \\ 4 & 18 \end{pmatrix} \\
m_7 &= \begin{pmatrix} 2 & 1 \\ 3 & 6 \end{pmatrix} \begin{pmatrix} -5 & 1 \\ -8 & -2 \end{pmatrix} = \begin{pmatrix} -18 & 0 \\ -63 & -9 \end{pmatrix} \\
c_{11} &= \begin{pmatrix} 41 & 25 \\ 44 & 32 \end{pmatrix} \\
c_{12} &= \begin{pmatrix} 30 & 42 \\ 37 & 54 \end{pmatrix} \\
c_{21} &= \begin{pmatrix} 16 & 13 \\ 33 & 23 \end{pmatrix} \\
c_{22} &= \begin{pmatrix} 14 & 20 \\ 26 & 35 \end{pmatrix}
\end{aligned}$$

which does indeed agree with our previous calculations!

For this algorithm, we get the recurrence

$$T(n) = \mathbf{7}\, T(\frac{n}{2}) + \Theta(n^2)$$

and thus $T(n) \in \boldsymbol{\Theta}(\mathbf{n}^{\lg(\mathbf{7})})$ where $n^{\lg(7)} \in O(n^{2.81})$.

Since Strassen's discovery, more than half a century ago, a lot of researchers have tried to improve the asymptotic running time of matrix multiplication. In January 1980, a small step for mankind was taken as someone was able to improve the previously know best running time of $O(n^{2.521813})$ to $O(n^{2.521801})$. From what I can tell, the asymptotically best algorithm currently known runs in time $O(n^{2.373})$. The quest for faster algorithms continues!

1.3 Concluding Remarks

It must be emphasized that

- unless one needs to multiply some really big matrices, the simple algorithm (three nested `for`-loops) is probably the fastest.
- if the matrices to be multiplied are known to be "sparse", in the sense that all but relatively few elements are zero, then it is possible to employ even faster algorithms.