CS 466: Divide and Conquer Algorithms and Reduction

Steph Durocher

September 18, 2008

Today we begin a new section in course material: Divide and Conquer Algorithms and Reduction
CLR Reference:

- matrix multiplication 25.1
- Strassen's algorithm 28.2
- divide and conquer algorithms 2.3
- Master Theorem 4.3

today:

- matrix multiplication using Strassen's algorithm
- review of Master Theorem

next lecture:

- transitive closure

Divide and Conquer.

1. Divide the problem into subproblems.
2. Solve (Conquer) the subproblems recursively (or directly if the subproblem is small enough: base case).
3. Combine solutions to the subproblems into a solution to the original problem.

Today we analyze various divide-and-conquer algorithms for matrix multiplication.

Matrix Multiplication.

Given an $s \times t$ matrix A and a $t \times u$ matrix B, return the $s \times u$ matrix $C = A \cdot B$.

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1u} \\ c_{21} & c_{22} & \cdots & c_{2u} \\ \vdots & \vdots & \ddots & \vdots \\ c_{s1} & c_{s2} & \cdots & c_{su} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1t} \\ a_{21} & a_{22} & \cdots & a_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ a_{s1} & a_{s2} & \cdots & a_{st} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1u} \\ b_{21} & b_{22} & \cdots & b_{2u} \\ \vdots & \vdots & \ddots & \vdots \\ b_{t1} & b_{t2} & \cdots & b_{tu} \end{pmatrix}$$

Recall, that for every $i \in \{1, \dots, s\}$ and every $j \in \{1, \dots, u\}$,

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

We can easily implement a sequential solution:

```
for i = 1 to s
    for j = 1 to u
        C[i,j] = 0
        for k = 1 to t
            C[i,j] += A[i,k] * B[k,j]
```

Running time: $\Theta(stu) = \Theta(n^3)$, where $n = \max(s, t, u)$.
Note, this is not a divide-and-conquer approach.
Let's consider divide-and-conquer algorithms. For simplicity, assume $s = t = u = n = 2^i$ for some non-negative integer i; all results we describe generalize to arbitrary s, t, and u.
Divide each $n \times n$ array into four arrays of size $n/2 \times n/2$.

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

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

Base case: multiply two 1×1 matrices (i.e., two scalars).
Computing each matrix C_{ij} requires matrix addition and subtraction and two matrix multiplications.
Therefore, computing each C_{ij} requires $2T(n/2) + \Theta(n^2)$ time.
Therefore, the total running time is

$$\begin{aligned} T(n) &= 8T(n/2) + \Theta(n^2) \\ T(1) &= 1 \end{aligned}$$

What is the closed-form solution solution for $T(n)$?
Long method: solve using substitution and induction.
Fast method: use the Master Theorem.

Review: Master Theorem.

The Master Theorem applies to recurrence relations of the form

$$T(n) = aT(n/b) + f(n),$$

where $a \geq 1$ and $b > 1$.
There are three cases:

1. If $f(n) \in O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.
2. If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \log_2 n)$.
3. If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and $af(n/b) \leq cf(n)$ for some $c < 1$ and all sufficiently large n, then $T(n) \in \Theta(f(n))$.

Let's use the Master Theorem, to find the running time of our recursive matrix multiplication algorithm.

$$\begin{aligned} T(n) &= 8T(n/2) + \Theta(n^2) \\ T(1) &= 1 \end{aligned}$$

$a = 8$, $b = 2$, $f(n) \in \Theta(n^2)$.
$f(n) \in O(n^{\log_b a-\epsilon}) = O(n^{3-\epsilon})$.
Therefore, Case 1 of the Master Theorem applies and $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^3)$.
The running time is still cubic. Can we do better with a straightforward divide and conquer algorithm?

Strassen's Algorithm.

Define seven matrices of size $n/2 \times n/2$, $M_1, \ldots, M_7$:

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{aligned}
$$

The four $n/2 \times n/2$ submatrices of C can be defined in terms of $M_1, \ldots, M_7$:

$$
\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned}
$$

Let's verify one of these (you can check the remaining three):

$$
\begin{aligned}
C_{12} &= M_3 + M_5 \\
&= [A_{11} \cdot (B_{12} - B_{22})] + [(A_{11} + A_{12}) \cdot B_{22}] \\
&= A_{11}B_{12} - A_{11}B_{22} + A_{11}B_{22} + A_{12}B_{22} \\
&= A_{11}B_{12} + A_{12}B_{22}
\end{aligned}
$$

What is the running time?
Computing each matrix M_i requires matrix addition and subtraction but only one matrix multiplication.
Therefore, computing each M_i requires $T(n/2) + \Theta(n^2)$ time.
Therefore, the total running time is

$$
\begin{aligned}
T(n) &= 7T(n/2) + \Theta(n^2) \\
T(1) &= 1
\end{aligned}
$$

Again, we can use the Master Theorem to solve the recurrence.
$a = 7$, $b = 2$, $f(n) \in \Theta(n^2)$
$f(n) \in O(n^{\log_b a-\epsilon}) \approx O(n^{2.81-\epsilon})$.
Therefore, Case 1 of the Master Theorem applies and $T(n) \in \Theta(n^{\log_b a}) \approx \Theta(n^{2.81})$.
Faster matrix multiplication algorithms exist, but these are more complicated. The current best algorithm has running time approximately $O(n^{2.376})$. The best lower bound is $\Omega(n^2)$ (i.e., the number of elements in an $n \times n$ matrix).