Optimizing Matrix Multiplication

Carl Gao & Michelle Deng
CS 124 – Programming Assignment 2
April 1, 2013

1 Introduction

Suppose we want to multiply two n-by-n matrices $A = \{a_{ij}\}$ and $B = \{b_{ij}\}$, where the subscript ij denotes the element at the ith row and jth column of a matrix. By definition, the product of A and B is the n-by-n matrix $C = \{c_{ij}\}$, where

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

The conventional matrix multiplication algorithm simply uses the above formula to compute each c_{ij} individually. Each c_{ij} requires n scalar multiplications and $n-1$ additions, and there are n^2 elements in C, giving a total of $n^2(n+n-1) = 2n^3 - n^2$ arithmetic operations. Assuming that such primitive arithmetic operations take constant time, which is realistic when a_{ij} and b_{ij} are not extremely large, and that all other operations (e.g. data-copying, memory access, etc.) are free, the overall computation time $T_c(n)$ using the conventional algorithm on an n-by-n matrix is

$$T_c(n) = 2n^3 - n^2 = O(n^3). \tag{2}$$

Famously, Strassen provides an $O(n^{\log_2 7}) \approx O(n^{2.81})$ recursive algorithm for matrix multiplication, which is asymptotically more efficient for large n. The algorithm breaks the n-by-n multiplication into seven matrix multiplications and 10 matrix additions using various $\frac{n}{2}$-by-$\frac{n}{2}$ quadrants of A and B, generating seven $\frac{n}{2}$-by-$\frac{n}{2}$ auxiliary matrices $P_1, ..., P_7$. These P matrices are then recombined in a sequence of 8 $\frac{n}{2}$-by-$\frac{n}{2}$ matrix additions to generate the four quadrants of C.[1] Since an n-by-n matrix addition requires n^2 constant-time additions, the total computation time $T_s(n)$ using Strassen's algorithm can be described by the following recurrence:

$$T_s(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2$$

which is $O(n^{\log_2 7})$ by the master theorem.

Note that we have not specified a base case for the recursion. Strassen's algorithm is only *asymptotically* more efficient than the conventional algorithm; for small n, the conventional algorithm is faster. Thus, to optimize matrix multiplication, one may want to use Strassen's algorithm whenever $n \geq n_0$, and the conventional algorithm for $n < n_0$. That is, n_0 is the cross-over point between the two methods, and the full recurrence for Strassen's is

$$T_s(n) = \begin{cases} 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 & \text{for } n \geq n_0 \\ T_c(n) & \text{for } n < n_0 \end{cases} \tag{3}$$

We will first estimate n_0 analytically and then experimentally.

[1] Lecture Notes 8 contains a complete description of the algorithm and definitions of these variables/matrices.

2 Analytical estimate of n_0

To estimate n_0, we will use the simplifications described above, where primitive arithmetic operations are constant time, and all other operations are free. However, in a real-world implementation, Strassen's recursive algorithm requires many more memory-access and data-copying steps than the conventional algorithm, so this bound is likely an underestimate of any empirical cross-over point. For simplicity, we will assume that n is even.

The cross-over point n_0 is defined as the smallest integer such that $T_c(n_0) > T_s(n_0)$. In particular,

$$\begin{aligned} T_c(n_0) &> T_s(n_0) \\ 2n_0^3 - n_0^2 &> 7T\left(\frac{n_0}{2}\right) + 18\left(\frac{n_0}{2}\right)^2 \\ &= 7\left(2\left(\frac{n_0}{2}\right)^3 - \left(\frac{n_0}{2}\right)^2\right) + 18\left(\frac{n_0}{2}\right)^2 \\ 8n_0^3 - 4n_0^2 &> 7n_0^3 + 11n_0^2 \\ 0 &> n_0^3 - 15n_0^2 = n_0^2(n_0 - 15) \end{aligned}$$

Hence $n_0 > 15$, and our analytic estimate is $n_0 = 16$.

3 Implementation

3.1 Dealing with odd n

[describe static padding yay]

3.2 Cache efficiency

4 Conclusion