Data Structures and Algorithms

Spring 2014

Cesar Agustin Garcia Vazquez

April 3, 2014

1 Programming Assignment 02

1.1 Introduction

Strassen's divide and conquer matrix multiplication algorithm for n by n matrices is asymptotically faster than the conventional $O(n^3)$ algorithm. This means that for sufficiently large values of n, Strassen's algorithm will run faster than the conventional algorithm. For small values of n, however, the conventional algorithm is faster. Indeed, the textbook Algorithms in C (1990 edition) suggests that n would have to be in the thousands before offering an improvement to standard multiplication, and "Thus the algorithm is a theoretical, not practical, contribution". Here we test this analysis.

Since Strassen's algorithm is a recursive algorithm, at some point in the recursion, once the matrices are small enough, we may want to switch from recursively calling Strassen's algorithm and just do a "base case" of a 1 by 1 matrix, but to switch earlier and use conventional matrix multiplication. Let us define the *cross-over point* between the two algorithms to be the value of n for which we want to stop using Strassen's algorithm and switch to conventional matrix multiplication. The goal of this assignment is to implement the conventional algorithm and Strassen's algorithm and to determine their cross-over point, both analytically and experimentally. One important factor our simple analysis will not take into account is memory management, which may significantly affect the speed of your implementation.

Tasks:

1. Assume that the cost of any single arithmetic operation (adding, subtracting, multiplying, or dividing two real numbers) is 1, and that all other operations are free. Consider the following variant of Strassen's algorithm: to multiply two n by n matrices, start using Strassen's algorithm, but stop the recursion at some size n_0, and use the conventional algorithm below that point. You have to find a suitable value for n_0 - the cross-over point. Analytically determine the value of n_0 that optimizes the

running time of this algorithm in this mode. (That is, solve the appropriate equations, somehow, numerically.) This gives a crude estimate for the cross-over point between Strassen's algorithm and the standard matrix multiplication algorithm.

2. Implement your variant of Strassen's algorithm and the standard matrix multiplication algorithm to find the cross-over point experimentally. Experimentally optimize for n_0 and compare the experimental results with your estimate from above. Make both implementations as efficient as possible.

   The actual cross-over point, which you would like to make as small as possible, will depend on how efficiently you implement Strassen's algorithm. Your implementation should work for any matrices, not just those whose dimensions are a power of 2.

   To test your algorithm , you might try matrices where each entry is randomly selected to be 0 or 1; similarly, you might try matrices where each entry is randomly selected to be 0, 1, or 2, or instead o, 1, or -1. You might also try matrices where each entry is a randomly selected real number in the range $[0, 1]$. You need not try all of these, but do test your algorithm adequately.

Code setup:

So that we may test your code ourselves as necessary, please make sure your code accepts the following command line form:

$$\$./\text{strassen } 0 \text{ dimension inputfile}$$

The flag 0 is meant to provide you some flexibility; you may use other values for your own testing, debugging, or extensions. The dimension, which we refer to henceforth as d, is the dimension of the matrix you are multiplying, so that 32 means you are multiplying two 32 by 32 matrices together. The parameter input file is an ASCII file with $2d^2$ integer numbers, one per line, representing two matrices A and B; you are to find the product $AB = C$. The first integer number is matrix entry $a_{0,0}$, followed by $a_{0,1}, a_{0,2}, \ldots, a_{0,d-1}$; next comes $a_{1,0}, a1, 1$, and so on, for the first d^2 numbers. The next d^2 numbers are similar for matrix B.

Your program should put on standard output (printf, cout, System.out, etc) a list of the values of the *diagonal entries* $c_{0,0}, c_{1,1}, \ldots, c_{d-1,d-1}$, one per line, including a trailing new line. The output may be checked by a script - add no clutter. (You should not output the whole matrix, although of course all entries should be computed.)

The inputs we present will be small integer, but you should make sure your matrix multiplication can deal with results that are up to 32 bits.

If you expect there will be any problems with running your work on the Harvard nice system (or a standard UNIX system), please contact the TFs early on in the process; also, be sure to include a makefile, or give explicit instructions on how to compile if necessary. Ideally, your program will be compiled, run, and tested by a script, so we would

prefer the following: you would be able to sit in the directory you submit, on nice, type "make", then "./strassen 0 ¡dimension¿ ¡inputfile¿" for some dimension and input file, and have it work.

What to hand in:

Hand in a project report (on paper) describing your analytical and experimental work (for example, carefully describe optimizations you made in your implementations). Be sure to discuss the results you obtain, and try to give explanations for what you observe. How low was your crossover point? What difficulties arose? what matrices did you multiply, and does this choice matter?

Your grade will be based primarily on the correctness of your program, the crossover point you find, your interpretation of the data, and your discussion of the experiment.

Hints:

It is hard to make the conventional algorithm inefficient; however, you may get better caching performance by looping through the variables in the right order (really, try it!). For Strassen's algorithm:

- Avoid excessive memory allocation and deallocation. This requires some thinking.

- Avoid copying large blocks of data unnecessarily. This requires some thinking.

- Your implementation of Strassen's algorithm should work even when n is odd! This requires some additional work, and thinking. (One option is to pad with 0's; how can this be done most effectively?) However, you may want to first get it to work when n is a power of 2 - this will get you most of the credit - and then refined it to work for some more general values of n.

2 First steps

The listing 1 presents the basic implementation of the multiplication algorithm which is $\Theta(n^3)$, which is going to be used to test Strassen's implementation is computing the multiplication of two matrices correctly.

Listing 1: Basic implementation of straight algorithm

```
public double[][] multiply(double[][] a, double[][] b){
            int numberOfRowsOfA = a.length;
            int numberOfRowsOfB = b.length;
            int numberOfColumnsOfB = b[0].length;

            double[][] c = new double[numberOfRowsOfA][numberOfColumnsOfB];

```

Table 1: Straightforward multiplication algorithm run with different test cases and sizes

Test\n	2	4	8	16	32	64	128	256	512	1024
{0, 1}	0	0	0	0	0	1	7	76	1,175	15,102
{0, 1, 2 }	0	0	0	0	0	1	7	75	1,117	14,380
{0, 1, -1}	0	0	0	0	0	1	7	76	1,119	14,382
$[0, 1] \in \mathbb{R}$	0	0	0	0	0	1	7	77	1,120	14,420

```
8          for(int i = 0; i < numberOfRowsOfA; i++){
9              for(int j = 0; j < numberOfColumnsOfB; j++){
10                 for(int m = 0; m < numberOfRowsOfB; m++){
11                     c[i][j] += a[i][m] * b[m][j];
12                 }
13             }
14         }
15
16         return c;
17     }
```