Serial Matrix-Matrix Multiplication

Holger Dell, Martin Aumüller, Riko Jacob

Finish by: September 28, 2020

Format of Hand-in This exercise can be solved in groups of up to three people. Please limit your report to 1 page of text (figures and graphs don't count).

We expect you to do two implementations, and perform an experiment comparing at least the two implementations. The optional tasks are a little more open-ended. For example, installing a library might be a demanding and annoying task. If you—also with the help of the TAs—just don't get it to work on your computer, you can just skip the task.

The "really optional" task of implementing Strassen's algorithm fits very nicely into the lecture now, but it is not easy. Please surprise us by making it work (and claim that it was easy)!

This is not a mandatory activity. We will give you feedback on whatever you have if you hand it in before the deadline.

1 Implement naive MxM

Given two $n \times n$ input matrices $A = (a_{ij})_{i=0,\dots,n-1;j=0,\dots,n-1}$ and $B = (b_{ij})_{i=0,\dots,n-1;j=0,\dots,n-1}$, compute the product $C = (c_{ij})_{i=0,\dots,n-1;j=0,\dots,n-1}$, given by $c_{ij} = \sum_{k=0,\dots,n-1} a_{ik}b_{kj}$.

File format On codejudge, the input format is one line containing n, then n lines with n space separated (floating point or integer) numbers specifying A; the second matrix, B, follows after a blank line in the same format as A. The output consists of n lines with n space separated numbers specifying C.

2 Implement the recursive MxM algorithm

You can test your implementation if you make sure that you always recurse on even sized matrices (at least if n is a power of 2), and replace the 2x2 base case by the following:

$$\begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix} = \begin{pmatrix} a_{00}b_{00} - a_{01}b_{10} & a_{00}b_{01} - a_{01}b_{11} \\ a_{10}b_{00} - a_{11}b_{10} & a_{10}b_{01} - a_{11}b_{11} \end{pmatrix}$$

3 Implement Freivalds' Algorithm (optional)

The algorithm chooses a random n-dimensional vector x, and then compares the result of $y = A(Bx)$ with $y' = Cx$. If $y == y'$ it answers "correct", otherwise it answers "faulty". (If indeed $C = AB$, this algorithm always answers "correct", while for $C \neq AB$, it has probability of at least $1/2$ to answer "faulty".)

On codejudge, the input consists of the integer matrices A and B as above, followed by a blank line and the third matrix C, another blank line, and then 3 "random" vectors x, i.e., 3 lines with n space separated integers.

4 Implement using a library (optional)

Choose a library and use it to implement matrix-matrix-multiplication. The list of libraries at `https://en.wikipedia.org/wiki/List_of_numerical_libraries` might be useful. Test the correctness of this implementation by comparing it with the other implementations and/or by applying Freivalds' algorithm (perhaps also implemented by the library)

Please share your experience with (installing) the libraries on piazza. Feel free to include code snippets in this context. Please ignore posts with code snippets, or the whole piazza folder, if you want to do this on your own without spoilers.

5 Implement Strassen's MxM algorithm (really optional)

Follow the description of the algorithm at `https://en.wikipedia.org/wiki/Strassen_algorithm`. You can test it on the codejudge instance for the naive MxM algorithm.

6 Experimental comparison

Compare the running times of the different implementations (including further variants, like in python storing the matrix in a list of lists versus storing it in a numpy multiarray). Make sure to describe your experiment using the concepts presented in Chapter 2 of McGeoch.

Some additional directions for an experimental study can be: Is there a threshold for the matrix size such that recursive / Strassen / whatever is better for bigger matrices than the nave approach? Is there a dependency on the input? Do many 0 entries help? Is there a difference between double (64bit) and float (32 bit)? Is the running time different if all numbers are smaller than 1/100 and many of the floating point numbers are small (say around 2^{-1020} for doubles)?