Scientific Software Report: Assignment 5

Daniel Loscos Barroso
`<daniel.loscosbarroso@student.kuleuven.be>`

December 20, 2018

`Time spent: 15 hours`

Design Choices

Compiling and linking

I decided to use a Makefile to easily compile and link the programs. Four possible programs can be compiled with it: `srda`, `time_xtx` and their debugging versions `srda_d` and `time_xtx_d`.

Debugging

I employed assertions throughout the code for debugging purposes. The debugging versions of the programs do not output any more extra information.

Header files

All the code for the classes is in-lined on the header files instead of declared and then implemented on a `.cpp` file with the same name. This felt more comfortable to do.

Testing

I overrode `time_vm` to adapt it to my testing needs for this problem. I wrote a script `datagen.sh` to repeatedly call `time_xtx` for the proper sizes of `N` and log the timing results for both methods. Al the code is included in the zip file.

Operators for vectors and matrices

I decided to include several extra functionalities in the matrix class: matrix-matrix multiplication, addition and subtraction and matrix-scalar multiplication. A second interface for the vector scalar multiplication was also added.

To make both classes more flexible in their capabilities I introduced `enable_if` handles and `decltype` to derive the return types of those operations.

Expression templates were not added to represent the operators. They would have been a better alternative but I was not familiar enough with them at the time of the implementation.

EXTRA: efficient SRDA functor

A third functor could be implemented as a three step operation instead of a two step one:

`t=X*x; v=transpose(X)*t; y=v+beta*x;`

In the two step implementation for the exercise `X` , `t`, and `x` all have to be brought to the cache to compute each element of `y` via `y=transpose(X)*t+beta*x;`.

The three step implementation does not have this problem avoiding cache collisions for big sizes of N.

Analysis

Theoretical complexity of the methods.

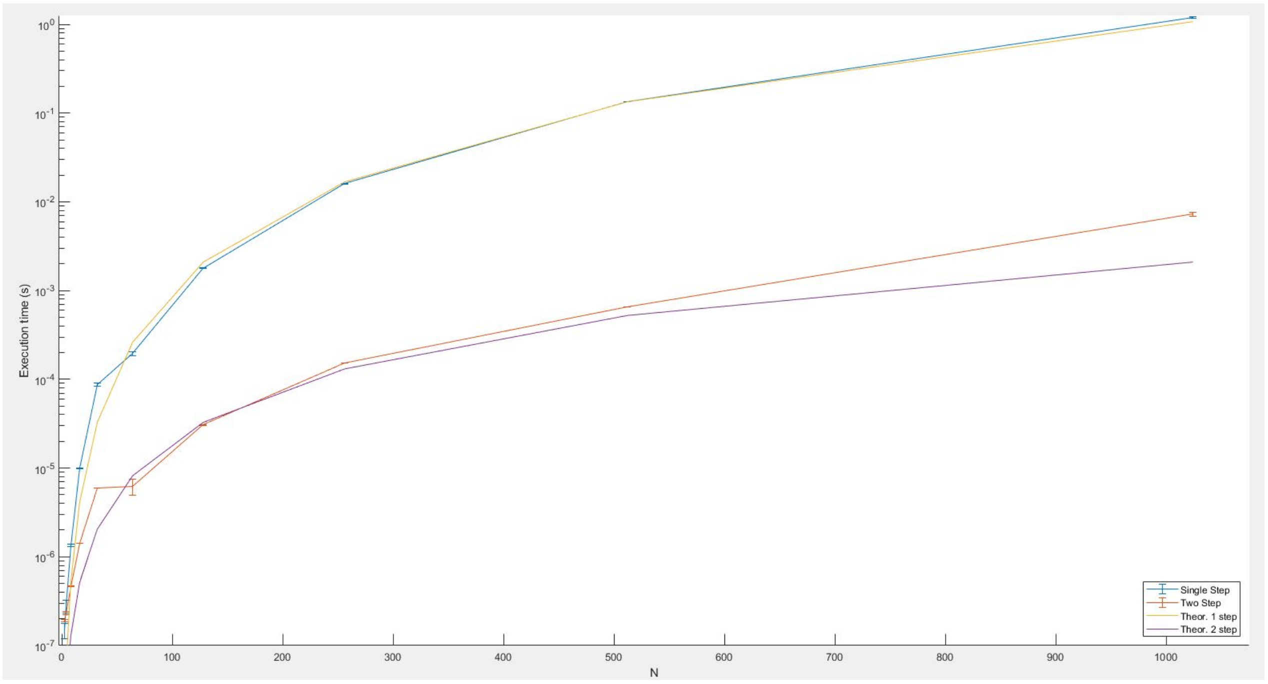
A matrix vector multiplication takes $2N-1$ flop to compute each element of the resulting vector. Computing the whole vector thus takes $2N^2-N$ flop. The way expression templates work is through lazy evaluation, this can lead to more efficiency in some cases, but not in the single step implementation, where the elements of `X*x` are computed N times (once per row of `transpose(X)`). Finally, the addition of `beta*x` takes $2N$ flop.

The **single step** method computes every element of `X*x` ($2N-1$) once for every element of every row of `transpose(X)` (N^2) to perform the multiplications. The additions needed amount to $N-1$ per row; thus needing $2N^3-N^2+N^2-N$ flop to compute `transpose(X)*X*x`. We can add now to this the cost of `beta*x` ($2N$ flop) and get the theoretical complexity: $2N^3-N^2+N^2-N+2N=2N^3+N$.

The **two step** method computes `t=X*x` ($2N^2-N$) only once. It then computes `transpose(X)*t` ($2N^2-N$) and finally performs the addition `beta*x` ($2N$). The theoretical complexity this yields is: $2N^2-N+2N^2-N+2N=4N^2$.

This estimations only take flop into account and to extrapolate as a time measure the assumption that floating point multiplications and additions take the same amount of cycles is made.

Testing the empirical results versus the theoretical complexities.



Both theoretical curves where scaled by the same factor (assumed to be the ratio between flop and second). The discrepancies between the experimental and the theoretical values for small values of N can be due to the cost of bringing all the data to cache memory to perform the computation, this cost is big compared to only a few flop. The theoretical and empirical values tend to converge for bigger values of N but we can notice that for $N = 1024$, in the two step method, there is a big difference and the experimental time is longer than expected.

Looking at the succession of experimental values for $N = 256, 512, 1024$ we can see that they scale almost linearly in the logarithmically scaled plot, while the curve for single step method does not behave in the same way. Since I am pretty confident in my theoretical cost derivation, I think that this exponential increase in the time of the computation for the two step method (that only appears for big values of N) is due to cache conflicts. While the single step method only needs to keep `X` and `x` near the processor, the two step method also accesses the values of `t`, which may remove pages storing `X` or `x` from the cache.

Results

The experimental results in seconds can be found in the following file. The structure of each row is as follows:

`N 1step_time 1step_standard_dev 2step_time 2step_standard_dev`

data.dat

```
 2 1.4881e-07 2.99406e-08 1.915e-07 6.39207e-09
4 2.7803e-07 4.51009e-08 2.3267e-07 1.00091e-08
8 1.34228e-06 3.60261e-08 4.6553e-07 2.06194e-09
16 9.93964e-06 4.34422e-08 1.42778e-06 3.64201e-09
32 8.75899e-05 4.02895e-06 5.94328e-06 9.82892e-09
64 0.000194196 9.23206e-06 6.17844e-06 1.24156e-06
128 0.00179429 2.3358e-05 3.05131e-05 2.29084e-07
256 0.016054 9.69137e-05 0.000152691 3.81049e-07
512 0.135265 0.000420998 0.000655263 6.1103e-06
1024 1.19917 0.0224603 0.00725133 0.000311392
```