Sparse Matrix-Vector Multiplication using CSR
High Performance Programming
Uppsala University

Victoria Sedig
victoria.sedig.0605@student.uu.se

March 2021

1 Introduction

1.1 Sparse Matrices

Sparse matrix-vector multiplication is important to a lot of fields where you need to solve a linear system $Ax = b$. For example in electromagnetism, astrophysics and computational fluid dynamics the matrix A is often sparse and very large. Therefore it is interesting to improve the multiplication with sparse matrices to save computational cost.

A sparse matrix is a matrix where a lot of the elements are zero values. There is no clear definition of how many elements should be zero values for a matrix to be sparse. However if the matrix is sparse it should be cheaper to use CSR or some other kind of sparse matrix-vector multiplication method than to use traditional matrix-vector multiplication.[1]

1.2 CSR

CSR which stands for "Compressed sparse row" is the most frequently used way to store sparse matrices to do sparse matrix-vector multiplication.[2] Instead of storing all elements in a matrix, all non-zero elements are stored in an array in row-wise order from the matrix. This means that storing a sparse matrix in this format saves a lot of memory space since the zero valued elements do not need a memory allocation. There will also be one array stating which column each value belong to. There is also a row offset array stating what index in the value array has the first element for each row (it is assumed that each row will have at least one non-zero element). In short there are three arrays storing all necessary information needed about the elements locations.

The multiplication using CSR format will work in the same fashion as a traditional matrix-vector multiplication since all necessary information to perform it is available, except it will skip all unnecessary computations with the zero elements. This means that the "more sparse" a matrix is the faster CSR will be.[3]

2 Problem Description

The goal of this project is to do sparse matrix-vector multiplication as fast as possible.

To make the multiplication faster the CSR format is implemented to store matrices. The cost of doing CSR format matrix-vector multiplication will be evaluated and compared to traditional matrix-vector multiplication. Parallelization and optimization will be applied to speed up the performance.

The experiments will investigate three aspects: sparsity, matrix size and parallelization.

3 Solution method

3.1 Sparse Matrix allocation and CSR

Both the CRS format and the matrix format are done in the code *sparsematrix.c*. First a sparse matrix with a certain percentage of non zero values is made and then the CSR arrays are made based on the sparse matrix. Then the two kinds of multiplications are performed and the results are compared to make sure that the output vector is correct.

A problem with this is that on the computer used, it was only possible to do up to 10000×10000 sized matrices, no matter what percentage of sparsity. One reason to this problem is that to do 100000×100000 matrices the malloc call to do memory allocation will take very long time and a lot of memory space (8×10^{10} bytes). An other problem is that the program was based on (int) matrices and arrays which has a maximal value around 2 million and $100000 \times 100000 = 10000000000 >> 2000000000$, which means it would not be possible to call for an allocation of that size, at least not for a simple (int). Possibilities with using (long int) or other formats were not investigated in this project. Even though it was not possible to make a larger matrix than 10000×10000 the code is useful since the efficiency of CSR format becomes clear.

3.2 Only CSR

In the code *onlycsr.c*, the sparse matrix is never allocated in the traditional format. Only the CSR format is created which means that only the values of the sparse matrix needs to be allocated. There is still a problem with making to long malloc allocations but this problem can be handled with making the matrix more sparse. In practice this means that the bigger matrix, the more sparse it has to be to be.

3.3 Parallelization using OMP

The sparse matrix-vector multiplication is done over a for loop which is computationally workload even over the whole loop. Therefor the Parallelization method applied was the "pragma omp for" directive.

It is possible to use Pthreads or normal "pragma omp" directive but since the only thing to parallelise is a loop and all the threads should do the same thing it is more practical and efficient to use the "pragma omp for" directive.

3.4 Optimization

Making CSR format instead of keeping the matrix format is in itself an optimization since it requires less memory allocations and computations.

Using Parallelization is one way of making the code faster since the work is divided by different cores. To optimize the threads different schedules were tested. For normal matrix-vector multiplication schedule(guided) is used and for CSR multiplication schedule(static, 4) is used. The program will be run with different amount of threads in the experiments. The computer used has 4 cores, therefor 4 threads are expected to give the best result.

To optimize the code some optimisation flags were used. For example O3/Ofast and ffast-math are expected to speed up the loops and computation. Also autovectorization of the "pragma omp for" was tested.

4 Experiments

4.1 Number of Threads

In figure 1 the relation between cost and number of threads to do sparse matrix-vector multiplication for both CSR and traditional matrix format is visualised. In the right subfigure both the CSR method and matrix method is optimized. The size of the matrix and sparsity are unmodified.

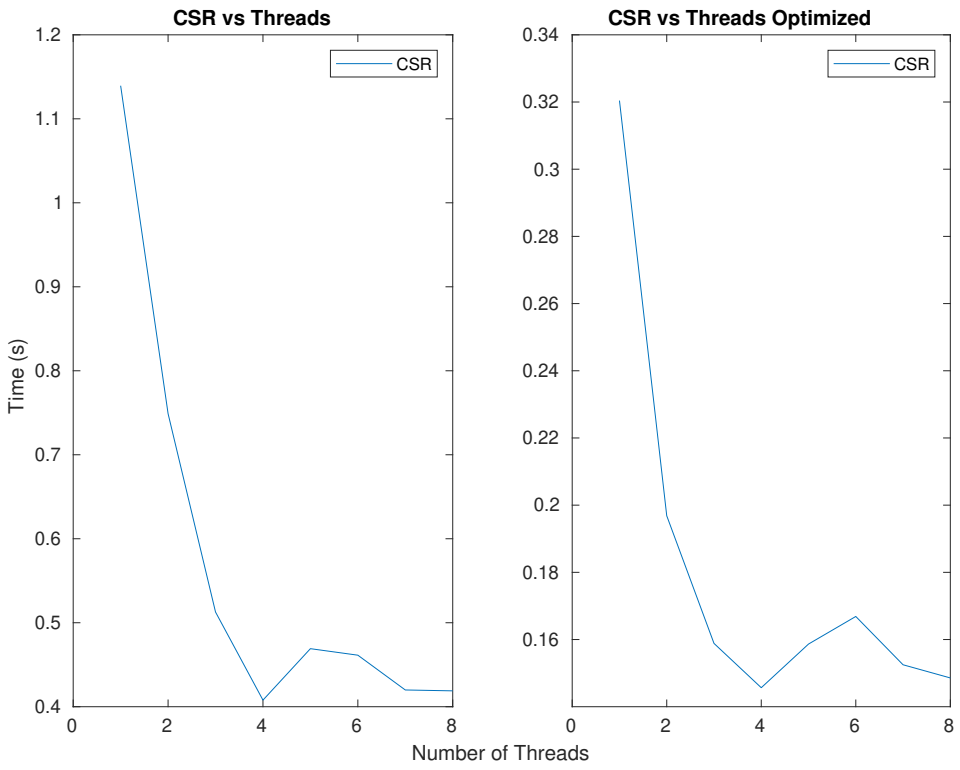

Figure 1: A graph over how the cost is affected for the CSR method respectively the traditional matrix-vector multiplication by changing the number of threads. The x-axis shows the number of threads from 1 to 8 threads and the y axis shows the time in seconds. The matrix is constantly of size $10000x10000$.

4.2 Sparsity

In figure 2 the relation between the cost and sparsity to do sparse matrix-vector multiplication for both CSR and traditional matrix format is visualised. In the right subfigure both the CSR method and matrix method is optimized. The size of the matrix and number of threads are unmodified for each subplot.

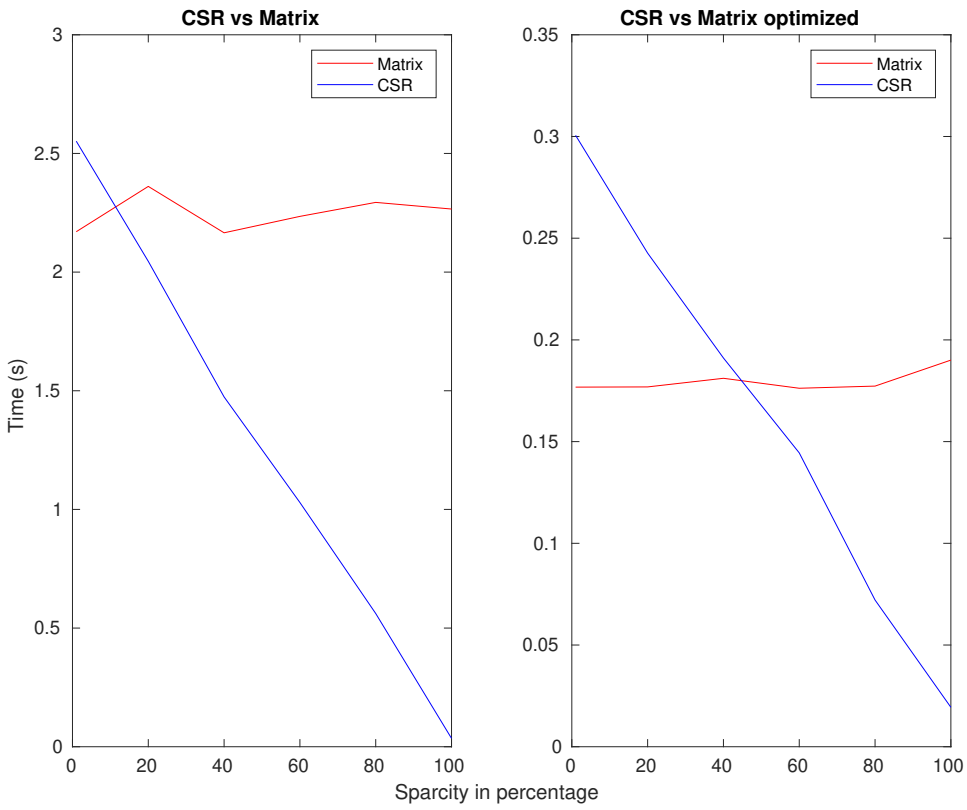

Figure 2: A graph over how the cost is affected for the CSR method respectively the traditional matrix-vector multiplication by changing the sparcity. The x-axis shows the sparcity in percentage and the y axis shows the time in seconds. The matrix is constantly of size $10000x10000$. The not optimized case was made with one thread and the optimized case was made with 4 threads.

4.3 Matrix Size

In figure 3 the relation between the cost and size of the matrix to do sparse matrix-vector multiplication for both CSR and traditional matrix format is visualised. In the right subfigure both the CSR method and matrix method is optimized. The number of non-zero elements and number of threads are unmodified for each subplot.

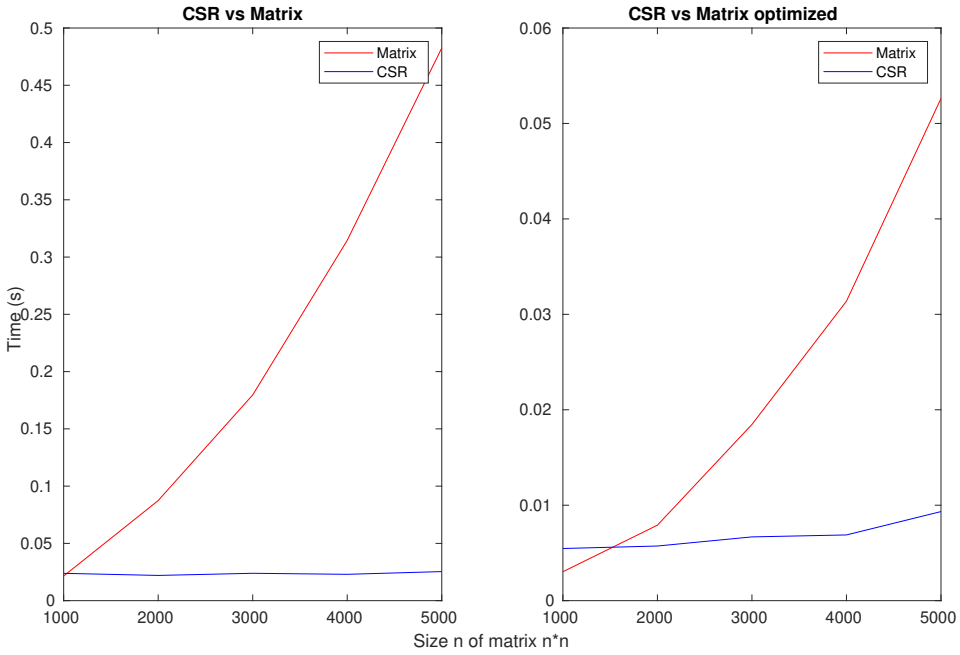

Figure 3: A graph over how the cost is affected for the CSR method respectively the traditional matrix-vector multiplication by changing the size of the matrix. The x-axis shows the size n of a nxn matrix and the y axis shows the time in seconds. The number of elements is constant and put to 10^6. The not optimized case was made with one thread and the optimized case was made with 4 threads.

5 Conclusions

In figure 1 it is visualised that the code is the fastest for 4 threads. Since the computer the experiment was made on has 4 cores that agrees to the expected result. 8 threads also give a rather good result since all the cores are used simultaneously but for smaller parts of the for loop.

In figure 2 it is illustrated that for the same size of matrix, traditional matrix-vector multiplication always takes the same time, independently of how sparse the matrix is. For the CSR format on the other hand it is clear that the cost and sparsity are proportional. The more sparse a matrix is the better CSR performs. This goes well along with the expected results since CSR only does computation with non-zero elements, while the matrix-vector always computes every element as normal no matter if they are zero or not. An interesting outcome that can be notified from the optimized and not optimized subplot is that the optimized matrix-vector multiplication is around the same time cost as a unoptimized CSR multiplication for most of the cases. According to the experiment the optimized matrix-vector multiplication performs better than the unoptimized CSR until around 90% sparsity, as the optimized normal multiplication only cost around 0.18 seconds.

In figure 3 the relation between the size of a matrix against time is shown. The CSR appears to behave constant in matrix size. This is expected since the number of non zero elements were kept constant in this experiment. The matrix-vector multiplication on the other hand is increasing with matrix size. This is one of the reasons why the code onlycsr.c is possible to run for very large matrices, as long as the matrix is sparse enough. On the computer used for the experiment it was possible to run up to 1^6 in matrix size while for the sparsematrix.c program it was only possible to run up to 10^4 no matter of how sparse the matrix is.

An other reason for why the program sparsematrix.c can not compute with matrices of size 10^5 is because the the allocated memory requires $10^5 x 10^5 = 10^{10}$ to store the matrix which takes a lot of time and the number is larger than the max number of an int (which was used in this code).

An idea for development of this project is to implement other formats than CSR. For example COO, DIY or HYB format. [3]

Possible ways to make the code faster is to do vectorisation on all loops. In this experiments only autovectorisation was applied. An other way to further speed up the code is to test more optimization flags. It is also possible to do more tricks to speed up the code, for example inline functions and declaring const for constant variables.

References

[1] Tim Davis, Siva Rajamanickam, and Wissam Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numerica*, 25:383–566, 05 2016.

[2] Guixia He and Jiaquan Gao. A novel csr-based sparse matrix-vector multiplication on gpus. 2016.

[3] Dimitar Lukarski. Sparse matrix-vector multiplication and matrix formats. `https://www.it.uu.se/education/phd_studies/phd_courses/pasc/lecture-1`. Accessed: 2021-03-05.