GPU Project

TRÉMAS Loumi

loumi-05@hotmail.fr

Master of Science in Informatics in Grenoble

27 février 2022

Table des matières

1 Summary

This project is about optimizing a previous project I had to do during my first year of master internship for the Turing Center for Living Systems as an intern of the team MMG under the supervisions of Anthony Baptista and Anaïs Baudot .

2 Background

My first year of the master's internship was about prioritizing nodes in Multi-layer networks using the similarity of Katz as the measurement. This project was built under the ambition of predicting potential new interactions between some Multi-layer networks and more precisely for the medical network.

This method is working pretty well , the only issue we had was the time of computation as the similarity measure :

$$S = \sum_{k=1}^{N} \alpha^k (C)_k$$

with $\alpha \in [0, 1]$ and $(C)_k$ a matrix .

We see that we rely a lot on matrix multiplication and so we instantly figured that it could be optimized by using GPU computing as it is known for calculating matrix multiplication more efficiently.

This entire work was programmed using the language Python, so naturally, we tried a GPU computing implementation using the Cuda library called cupyx.
At the end of my internship, our measurements were working on a small network representing the interaction between the different airports and with a small depth in the path.

Unfortunately, it was not working as fast and as good as expected for the huge genetic network representing the interaction between 3 multi-layers networks composed of (genes, drugs, diseases).

So my aim for this project will be to optimize the computation with GPU computation. More specifically using sparse matrix multiplication.

3 Theoretical organisation

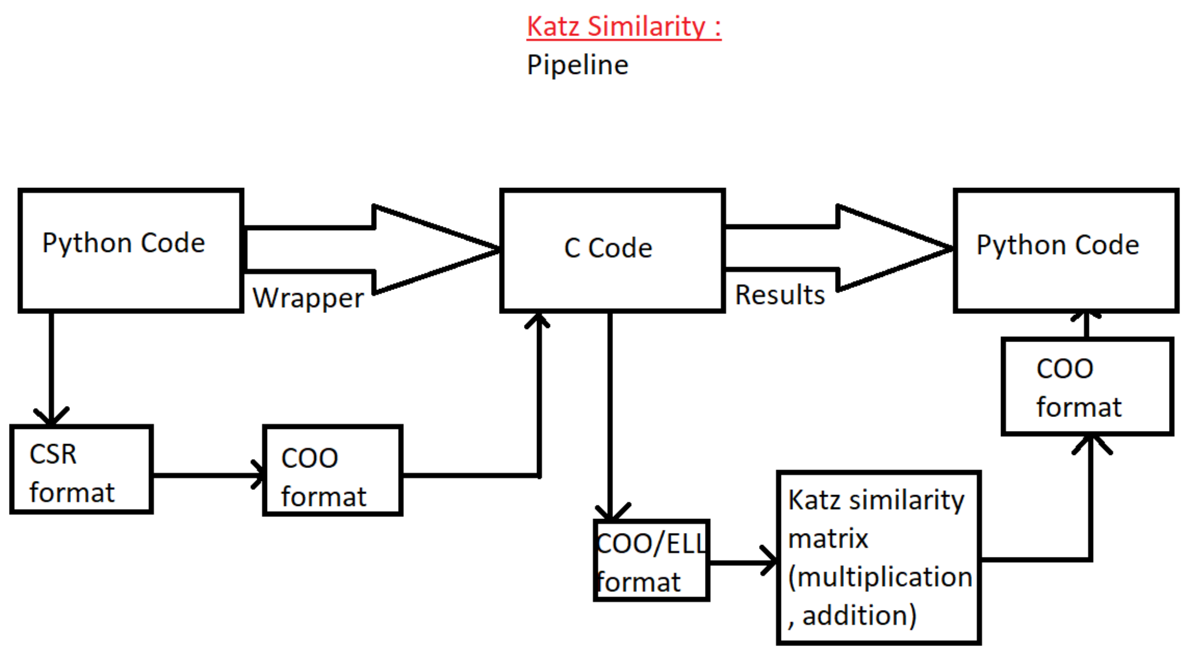
3.1 Global overview

3.1.1 First Pipeline

As the global project was built using python and by time necessity I will not build everything from scratch. My plan is to use the python programming already done, to read and build the data with the library NetworkX giving me a supra-adjacency matrix which is composed of multiplexes and bipartite matrices in the format of CSR.

As recommended by my teacher I will use a hybrid format using two-part composed of an ELL part and a COO part (we will call the ELL/COO format a hybrid format) ; to compute all calculations on the sparse matrices.
For doing so I will first need to convert all the data generated in CSR format to COO format and then create a wrapper in C which will take the COO matrices as input ; will convert them into the Hybrid format and then the whole computation from matrix multiplication to matrix addition getting a final matrix as results of the computation. This final matrix would be converted back to the COO format and then transferred back to python to be used or printed using the python toolbox.
The pipeline can be resumed as the following .

Figure 1 – Katz Similarity Pipeline



3.1.2 Second Pipeline

The previous pipeline was the first sketch of my project, after some questioning and interrogation I decide to go with the same tools to pass data from python to C, but this time I would just pass the address and size of the numpy structure through a pipe to the C code avoiding a costly transfer of data.

Due that the biggest sparse matrix is composed of 840 058 elements , so as a COO format we have to transfer 3 times these size .

This is without saying too big to be transferred, moreover, each element is a float coded as float32, which is coded on 4 bytes, with a quick calculation we find that we have to transfer 0.0093 GB just for one sparse matrix.

For the whole supra-adjacency matrix, we would have to transfer 0,025 GB just from python to C, so it's not a good option as we're trying to save time and energy.

After 2 weeks of trying I have found how to get the NumPy array information and how to transfer it to the C code, after some more weeks of trying I was unable to use the given address, blocking my reading of those needed data.

3.1.3 Third Pipeline

For my last try on using the data I needed for the calculation, I decided to turn myself to another hybrid Cython, this time it's a combination of the programming languages Python and C.

Due to lack of time, I decided to stop focusing on this part of the project which has taken me around 65% of my time on this project and as it's not on how to transfer data but how to improve time calculation with GPU computing.

4 Approach

4.1 Research

4.1.1 Hybrid format

To encode a COO format into the Hybrid format I had first to do some research and answer some questions :

- Such as how does it look ?
- How can I pass from COO to Hybrid ?
- How would it be better ?

For the first question, I think the better way to grab the idea is with a good illustration. The best illustration (shown below) I have found is from the research paper :
Characterizing Dataset Dependence for Sparse Matrix-Vector Multiplication on GPUs

FIGURE 2 – Hybrid format

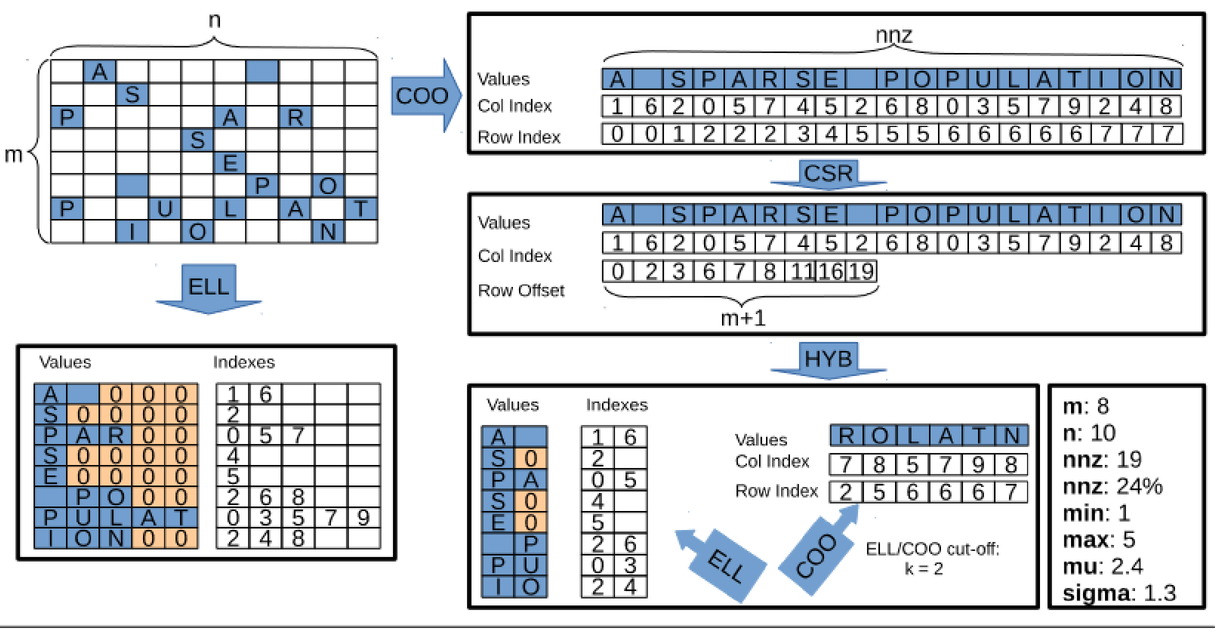


Figure 1. Sparse storage representations for matrix A in COO, CSR, ELL and HYB.

4.2 Tools used

4.2.1 Python IDE & Libraries

IDE :

- Pycharm
- Spyder

Librairies :

- Numpy
- NetworkX
- Scipy
- Pandas

4.2.2 C & Cuda IDE

IDE :

- Visual Studio code
- Gedit
- Vim

4.2.3 Version control

- GitHub

4.2.4 Scheduling

- Trello

4.3 Hardware used

4.3.1 Personal Hardware

- CPU : Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz
- RAM : 16,0 Go
- OS : Windows 10
- GPU : NVIDIA GeForce RTX 2060 with Max Q Design
- GPU-Memory : 8,0Go

4.3.2 University Hardware

- CPU : 4 Intel Core Processor (Skylake)
- RAM : 12,0 Go
- OS : Ubuntu 20.04.1
- GPU : Quadro RTX 6000/8000

5 Implementation

5.1 Creation of an Hybrid type

As in C, we can only return one element for each function. I had to choose how to transmit and retrieve multiple arrays for each calculation.
For the implementation I first chose to transmit all addresses to the function, which would work on their addresses, meaning that I would not have to return those data.

For practical testing reason , I decided to implement a new type of structure which would represent this Hybrid format .

This type is composed of :

— The 2 array and size of the ELL part
— The 3 array and size of the COO part
— The cut off size , allowing to read the ELL part and to convert back to COO

5.2 Conversion COO to Hybrid

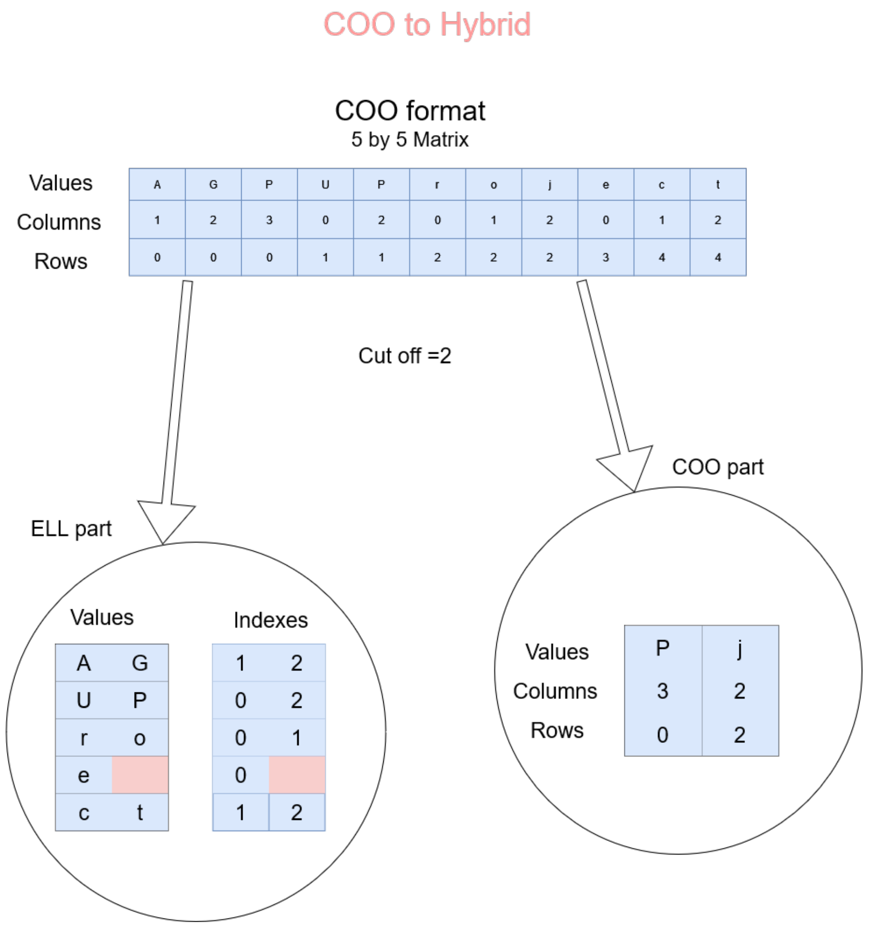
In order to use the practicability of the Hybrid format we first need to transform our sparse matrix under a COO format to a Hybrid format composed of the ELL and COO format .

To perform this transformation we need as input the size of the array containing the values, the three arrays containing values, columns and rows of one element, how much cut_off we want and the number of rows of the original matrix.

From all those elements I can calculate the new size of the future COO array and future ELL array. Then to do the transition I just need to check whether the number of elements coming from the same row in my ELL part is not exceeding my cut off if it is then I put it in the COO part.

The rest is just handling some specific situation of completion. This will give as output two arrays representing the ELL part and three arrays for the COO part.

Figure 3 – COO2Hyrbid



5.3 Size of the new COO

To get the size of the new COO array .
We need to have the rows of the former COO format,the size and the cut_off as input .
For the calculation, we just need to count the number which exceeds the cut off

0	0	0	1	1	2	2	2	3	4	4

Here if we take a cut off of size 2 then we would have a new COO of size 2 , because we only got one 0 and one 2 that exceed the cut off size.

5.4 Hybrid multiplication

For this function we will as inputs :

— All components of the first matrix with Hybrid format
— The size of the ELL and COO array
— The ELL and COO values of the second matrix
— The cut off size

And this will give us as output two arrays representing respectively the ELL and COO values array .

This function is kernel function as those calculations can be performed in parallel.

5.5 Hybrid scalar multiplication

For this function we will as inputs :

— All components of the first matrix with Hybrid format
— The size of the ELL and COO array
— A scalar value to multiply our matrix with
— The cut off size

And this will give us as output two arrays representing respectively the ELL and COO values array .

This function would also be a kernel function as those calculations can be performed in parallel.

5.6 Hybrid addition

5.7 Combination

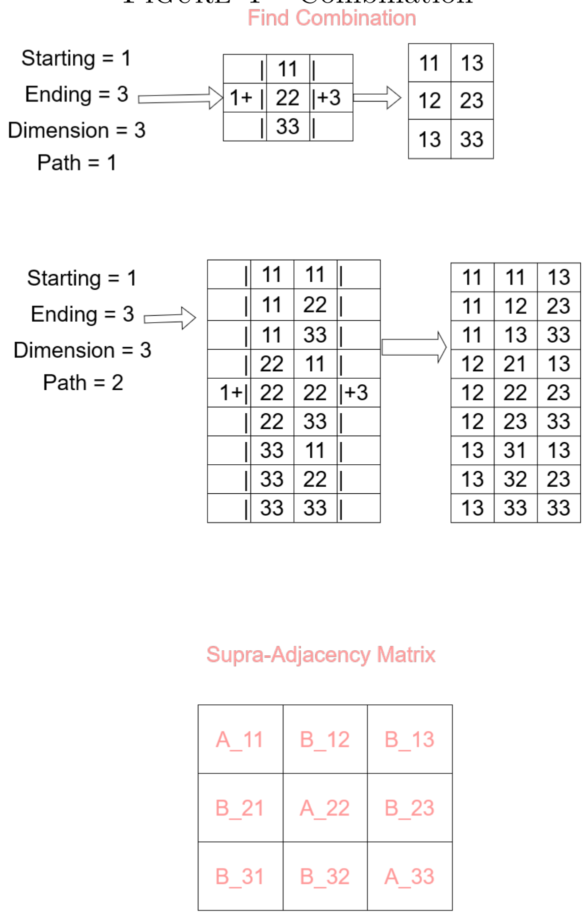
The purpose of this function is to find in which order we should multiply the matrices between them , as to perform the Katz similarity .

In order for this function to perform we need the following input : :

— The path length
— The matrix dimension
— The starting and ending Adjacency matrix

And this function returns the ordered list of matrices indexes to be multiplied. This function does not need parallelism, so it will be executed as a normal function.

Figure 4 – Combination



5.8 Matrix from Combination

This function will be a crucial part of our pipeline as it is the function that will launch all the matrix multiplication.

For this doing the function will need as inputs :

— The length path and the dimension
— An array composed of all the Hybrid matrices
— The combination array

From the length path and the dimension we can determine the size of the combination array ; which is $dimension^{(length\_path-1)}$. As for the Hybrid array size we just need to know the dimension of the supra_adjacency matrix , as saw in 4 this matrix will always be a squared matrix , so the size of the supra_adjacency matrix is calculated with $(dimension)^2$.

This function will first initialize a new hybrid matrix, then through the use of loop it will multiply for all length path elements, which will give one hybrid matrix that will be stored onto the new hybrid matrix, for the first loop iteration we just need to perform sparse matrix multiplication and storage ; but for the rest of the loop, we will perform an addition between the previously stored matrix and the new one that has been calculated. To clarify a little we will sum matrices that have been obtained using sparse matrix calculation, providing one final matrix .

This function will give as output the final calculated matrix , and as it will ask for different operation on sparse matrices such as addition and multiplication , this function will launches some kernel function but will not be a kernel function by itself .

5.9 Katz Similarity

This function is the core of all this project as it will be the function giving all the order to other functions and kernel functions.

In order to used this function need to receive as inputs :

— The supra adjacency matrix dimension
— The path length we want to calculate
— The start and ending matrix
— A scalar $\Theta \in [0; 1]$
— An array composed of all the Hybrid matrices

This function will first initialize a new hybrid matrix and will calculate the possible combination using a previous function, then the function will iterate by the size of the path that we have given .
For all those iterations, this function will calculate and store the product of the scalar matrix multiplication using a previous function .
Then from this resulting matrix this function will perform the sum for all values of the length path.

$$S = \sum_{k=1}^{N} \alpha^k (C)_k$$

with $\alpha \in [0, 1]$ and $(C)_k$ a matrix.

Here the sum would take place between $k = 1$ to N ; in practice and in other papers the norm is to never exceed a $k \geq 4$ as after this path length we don't gain much information in the network.
This fact was also verified during my internship as we had done some comparison of path length and the gains were indeed appearing to be low.

This function will not be a kernel function as it will be the chief function that will launch all the other kernel functions.
This function would return a hybrid matrix that would then be reconstructed into a COO

format and then sent back in the python code to perform other computations using all the tools available on python.

6 Difficulties

6.1 Pipeline

The implementation of a pipeline transmitting data between a python code that is used to build a network and represent it as a COO format of sparse matrix and then transmit those COO format to a C code was more time driving than the whole rest of the project. I had to search and test many path, such as the GitHub library A python to C pipeline , which was first used as a whole pipeline to transfer the whole three NumPy arrays representing a COO matrix. This method was quickly abandoned as it would be too costly and not efficient for the whole calculation, so instead, I tried to transmit the address of the NumPy array structure directly to the C code ; but unfortunately, I was unable to make it work properly, so I've directed myself to use Cython.

6.2 Time scheduling

The biggest issue I had with this project is that I have underestimated the loads of work I would have to provide all by myself.
The fact that the pipeline took me around 65% of my time, showed me that this project was too ambitious for the time I had to make it work.
This project could have been a subject for an internship or a 2-month team project .

6.3 C Programming

I have overestimated my ability in C programming, even if I was remembering a lot from my previous C programming experiences.
I must realize that my C programming was rusted, such as pointer and structure handling.

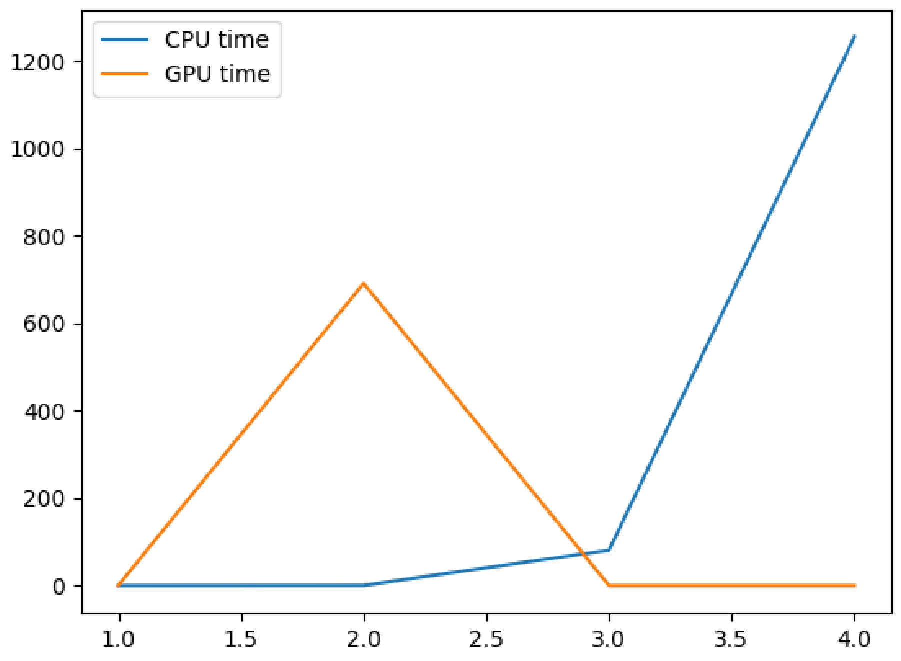
7 Results

Because I wasn't able to build a functional pipeline between the python and C code, I could not get any results using the code I have produced during this project as I couldn't transmit any data .
So to obtain some results I had to implement the same code but on python in some days, for those results I have to use the copy library which is the python version of Cuda onto the COO sparse matrices .

Here are the results for the genetic network :

It has a fixed theta of 0.5 the y axis represent the time in second and the x represent the size of the path .

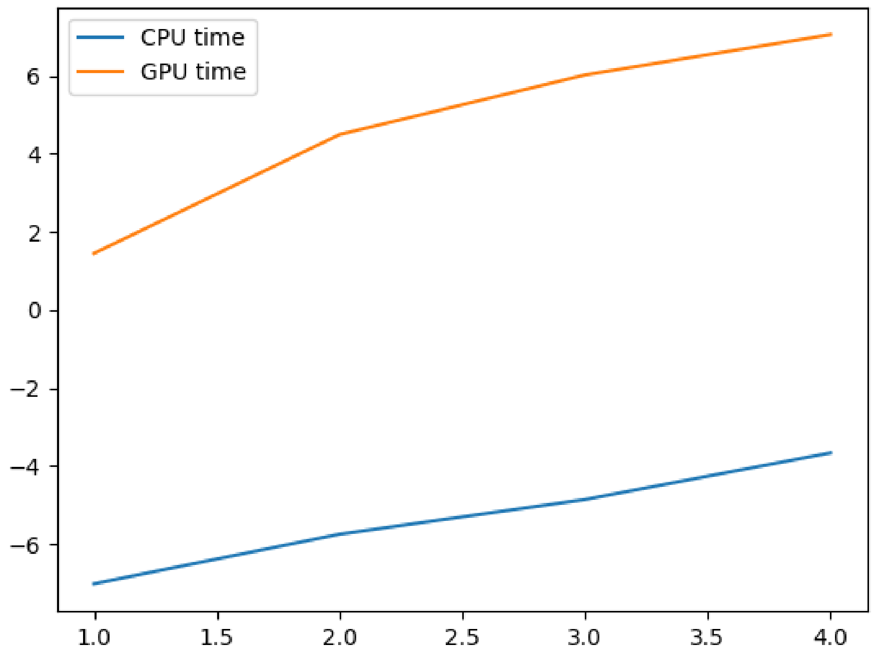
Figure 5 – Genetic network



The issue with the big network here is that the GPU card on my personal computer doesn't have enough memory to perform all calculations ; more likely it looks like the GPU memory is never released, the possible solution would be to perform the calculation using block partitioning matrix multiplication. We can see that the GPU calculation is taking way too much time compared to the expected time.

Same goes for the airports network results :

FIGURE 6 – Genetic network (with a logarithm scale)



Those results could be interpreted as the GPU is performing way worse than the CPU, but as I am using the python library of Cuda, I think those results a more due to my implementation and the fact that it is a high-level language and by so, couldn't be optimal for the usage I have done with it.
I expect that with a good implementation we should at least get a slightly lower time with the GPU when we have a bigger network .
As it would perform sparse matrix multiplication on bigger matrices.

8 Improvements

In a possible future it would be a great improvement to do the following :

For C part of the project :

— Transform the code using the type hybrid
— Implement the pipeline from Python to C and the other way around
— Implement some generated data test
— Trying multiple values of cut off to see which would be more effective

— Implement the same code but without GPU and compare the time between both implementation

And for the python part of the project :

— Implement a block matrix multiplication
— Explore the functionnality of the cupyx benchmark
— Implement manually the sparse multiplication matrix

9 Conclusion

This project would have made me learn that I should not be overconfident, when starting a project that I master and understand very well, even if it looks easy on paper and the planning, the time estimation for each task should be increased.
I've also learned that I should be less stubborn on the task I am doing .

As for GPU computing, this project would have brought a new way of thinking about programming, as I had to think about how this calculation time could be reduced and done in parallel.
This project would have also made me learn and exercised about compressed data structure and more especially about sparse matrices .
Even if this project was too big for me, it would have made me learn new skills such as a new way of conceptualizing a program. How to think at a parallelism way of computing, reducing the calculation time and by doing so the energy needed.
It would had made me learn how to better organize myself using tools such as Trello linked to my schedule but also by using version control tools such as GitHub linked to the project .

10 References

Characterizing Dataset Dependence for Sparse Matrix-Vector Multiplication on GPUs by Naser Sedaghati , Arash Ashari , Louis-Noël Pouchet , Srinivasan Parthasarathy, P. Sadayappan .