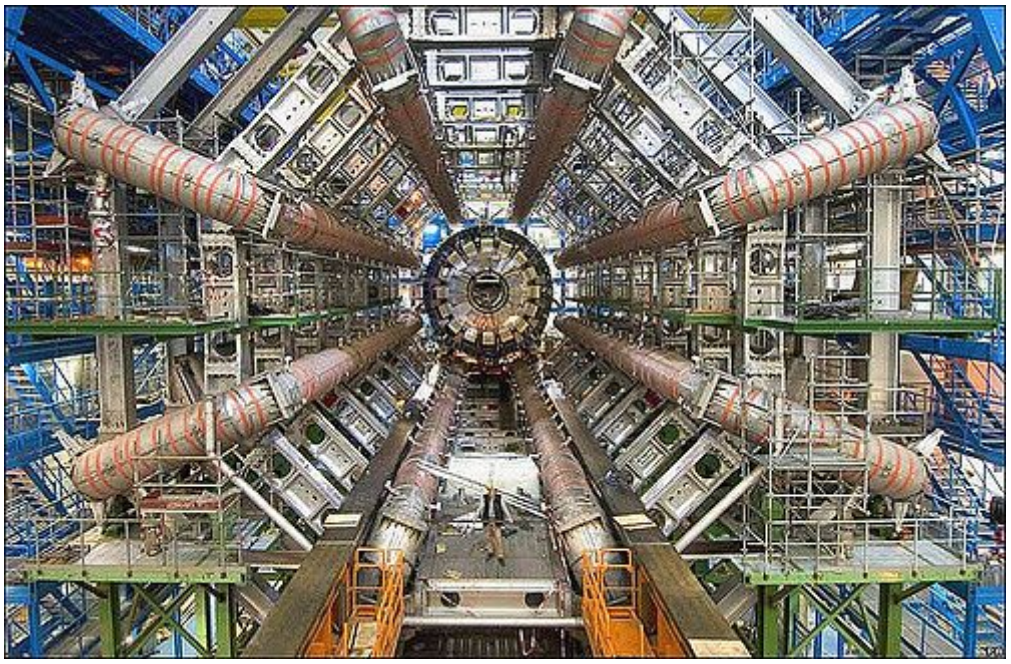
Faster matrix algebra for ATLAS

Project proposal



Student: José Francisco Lorenzo Hernández

March 2018

Google Summer of Code 2018

INDEX

I. INTRODUCTION

On this project proposal, we will define what we have done until this moment, what we are planning to do and my personal motivation and skills, in order to show how I am and how I can contribute to this project.

I.1. Project introduction

This is the project introduction, that could be found on [1].

“In an environment like the Large Hadron Collider, we have millions of collisions per second, each producing thousands of particles passing through our detectors. Particle physics experiments like ATLAS rely on thousands of computers using clever algorithms to identify different types of particles and find interesting collisions, such as the production of Higgs bosons or even new particles. The optimisation of these algorithms can dramatically improve the physics we are sensitive to. ATLAS pattern recognition makes heavy use of matrix algebra, implemented with the Eigen library”.

I.2. Project mentors

The mentors of this project are:

- Stewart Martin-Haugh. Email: stewart.martin-haugh@stfc.ac.uk
- Dmitry Emeliyanov. Email: dmitry.emeliyanov@stfc.ac.uk

II. Main Contents

II.1. Student Information

II.1.1. Contact Information

Name: José Francisco Lorenzo-Hernández
Address: Joaquín Belón Street, Nº 18, 4ºC, Las Palmas de Gran Canaria, Las Palmas, Spain. Zip code: 35013.
Birth Date: 20/04/1995 (22 years old)
Phone number: +34 678142739
Email: josee.loren[at]gmail.com
Linkedin: www.linkedin.com/in/joselorenzohernandez
Github: http://github.com/joseeloren

II.1.2. Studies Information

University: University of Las Palmas de Gran Canaria
Degree: Double Degree in Computer Science and Business Administration and Management
Year: Fifth - Finishing in 2018.

II.1.3. Work experience

I have worked on a company called Duocom Europe developing some software about VoIP (Voice over IP), using C++ and wxWidgets library, mainly. Some of the projects done on this company are:

- "Working clock" to clock in when you are at working place, using GPS API from Mozilla.
- libCurl library compilation for Windows and OpenSSL.
- A system to automate fixed lines portability.
- A diary monitor of back transactions (virtual POS).
- Network activity notification system.
- Virtual Fax app for desktop environments.
- Mozilla Firefox extension to upload documents from browser to company's private cloud.
- Web application to setup VoIP phones automatically.

Some others interesting works on other companies (Secret Source, e.g.), are:

- Audit system to check vulnerabilities on Web Servers.
- Linear regression to predict how much a webpage costs.
- Web platform for the new Historical Spanish Dictionary for RAE (Real Academia Española) using Tomcat, Java EE and web stack.

II.1.4. Courses, projects and awards.

On this section, we list all the interesting courses, projects and awards that could be interesting for this GSoC project:

- Arduino Introductory Course (10 hours)
- Linux Introductory Course (Linux Foundation, 50 hours)
- OPC UA Client project for an industrial robot using C++, Delta Robot (Project Mi5, ITQ, 2016) – Further information: https://projectmi5.com/live/
- Nacional Award "I+D+i Investigate" in Artificial Intelligence (Fundación San Patricio, 2011)) – Further information in Linkedin
- Demola Project – Computerize an old-style company to improve their business using smartphone apps and a web app. Further information (2017): https://canaryislands.demola.net/capross-reinventando-servicios-gran-canaria

As a previous GSoC student, I have done:

- Google Summer of Code project with wxWidgets organisation. Development of JavaScript WebView Integration in wxWidgets library using C++ (2017) – Further information: https://summerofcode.withgoogle.com/projects/#6085667476996096
- Speaker at a talk (2018) about “Google Summer of Code” at School of Computer Science, University of Las Palmas de Gran Canaria

My experience about open source is just related with GSoC and a little bit off GSoC with the same organization (wxWidgets), because I am always busy with studies and work, and it is impossible to contribute outside summer with a good contribution.

Also, I am currently developing my final project for university, that it is mainly about predict not socially responsible practices on Spanish hotels using Artificial Intelligence and R as programming language.

I have done other small projects (for companies and just for study a technology) using Java, Python and other programming languages, like JavaScript, e.g.

II.1.5. Availability during GSoC period.

I do not have any schedule problems with exams or projects from the university during the GSoC period to complete the project successfully and investing the time that it requires.

II.2. My personal motivations

II.2.1. Why I chose CERN-HSF?

On summer 2011, I was on a CERN laboratory in Switzerland, learning about atoms and molecules. At that time, I was studying at the high school but I was really interested in research. That was a part of a serie of visits to some research institutions around Switzerland and France because I won a R&D project for high school students [2].

I could not see the Large Hadron Collider, but I felt in love of all that scientific environment.

II.2.2. Why I chose ATLAS?

This decision was because we like the project, but I found that “ATLAS is one of two general-purpose detectors at the Large Hadron Collider (LHC)” [3]. For me, it is an important detector that helps to research fields in Europe and around the globe.

II.2.3. Why I chose this project?

We checked all the organizations that we know, that we recognize the name, and we checked the project. We love mathematics and that was one of the reason we study computer science, because it involves programming and mathematics too. So, the project we prefer of all the project I saw was this, because we like algebra and programming, and for me it challenging be involved on this kind of project. Also, we feel secure and motivate

with this project, and it is not an effort to develop this, because we love the idea and the project, as we said.

Even more, the project makes us be more self-taught that we are and learn to do our best to finish it successfully and with the best results. Also, I want to improve the world, specially the research, that helps to improve technology and the science, that help people to have a better life quality.

II.4. Task basics

On project proposal web page [1], there is an introduction to the task:

“Eigen is a fast, robust, open source linear algebra library, used by e.g. the Google Tensorflow and Large Survey Synoptic Telescope projects as well as ATLAS. Most algorithms in ATLAS software use symmetric matrices, which are unfortunately missing from Eigen. This would make our calculations more efficient: since the upper and lower triangular parts of the matrix are the same, only the upper/lower half of the matrix needs to be stored or used in operations.

Your task will be to implement an efficient symmetric matrix class storing only the needed parts. This will help ATLAS find particles while using less computing power and less storage space”.

From this task, we can conclude that the requirements basics are:

- Implement algorithms for symmetric matrices on Eigen library.
- Store only the upper/lower triangular part of the matrix.
- Use less computing power.
- Use less storage space.

II.5. Evaluation test

II.5.1. Evaluation test statements

For this project, there was a small evaluation test [4] to filter the candidates for this project. Mainly, the mentors wanted to know if the students have skills on C++, Git/Github, linear algebra basis and documentation, basically.

As a summary, these are the evaluation tests requirements:

- Write a standalone C++ class for a symmetric matrix (SymMat class).
- Store only the upper triangular part of a matrix.
- A constructor to make a SymMat object from a Eigen::Matrix
- S(i,j) should return S(j,i) and vice versa.
- SymMat +/- SymMat and SymMat +/- Eigen::Matrix
- SymMat*SymMat and SymMat*Eigen::Matrix
- Throw exceptions if there are incongruent data (matrices dimensions e.g.)

- Write test cases and comment
- Upload the code to GitHub
- Write it using C++11 standard and compile it with clang or gcc.
- Some documentation like compilation instructions.

II.5.2. Evaluation test solution

The solution to this evaluation test could be found on [5]. However, on this section we will comment some remarkable features that we think that are important for the final project.

II.5.2.1. Algorithm notes.

This notes are on the project wiki [5], but we are going to rewrite the ideas in order to comment some important keys. Also, here we can markdown better the equations.

Get the number of rows or columns of the matrix when we don't know the size of the matrix that it comes from.

This section is about how to get the size of the matrix. If we store the matrix as a vector, the most basic way to store the upper part of a matrix (because it lose the matrix shape), we lose it measurement.

If we this on a triangular matrix, the first row has n elements, the second n-1 until the end, that has 1 element. If we do the arithmetic progression sum, we get:

$$e = (n+1) * \frac{n}{2} \; [1]$$

Where e is the number of elements.

What we need is n, so if we isolate n we get:

$$n^2 + n - 2e = 0 \; [2]$$

Finally, we get:

$$n = \frac{-1 + \sqrt{1+8e}}{2} \; [3]$$

As I said, this a summary. For more information, check the project wiki.

About this, square roots consume a lot of computacional time, so we need to store this value in order to save this time. If we compare computer time vs. memory, in this case, it worth.

Get the number of rows or columns of the matrix when we don't know the size of the matrix that it comes from.

If we have stored the data as a vector, we need to calculate the corresponding m(i,j) from the corresponding position of the element in the vector.

If we think a 4x4 matrix:

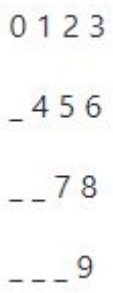


Figure 1. SymMat 4x4 matrix positions

Having this example keys and how to calculate them:

- First-row first position: 0. We don't have to do nothing until second-row.
- Second-row first position: 4. This is equal to matrix size: n=4.
- Third-row first position: 7. This is equal to n+(n-1)=7
- Fourth-row first position: 9. This is equal to n+(n-1)+(n-2)=9.

So, to get this in general, we have an arithmetic progression:

$$n + (n - (i - 1)) * (i/2) = (2n - (i - 1)) * (i/2) \; [5]$$

Where i is the row that we want the element from.

If we add the column, we have:

$$(2n - (i - 1)) * (i/2) + (j - i) \; [6]$$

We have to check if $i \leq j$ or $i > j$, to have the lower value or upper value, that are the same.

As we said before, this is a small and quick definition of what we did on the evaluation test algorithms. Please check project wiki for more information.

II.5.2.2. Doubts about the Evaluation Test document.

Personally, I found the evaluation test not clear enough. Because of this, I did it what was strictly required. On the final project, we will include this, because we have time to ask the doubts and discuss them with the community.

Some of them we think that they should be on the final project:

- Commutative multiplication operator.
- Template for other data types from Symmetric Matrix.
- Non square matrix multiplication, but pseudo-symmetric. This could be useful, but it is not the most important doubt that we should contrast with the community.

II.5.2.3. Future work.

Some future work that we are going to do on the final project, if we use some of this code on the new Eigen class is:

- Add more tests and refactor tests and test data.
- Test bigger and lower numbers.

- Check run time in order to improve calculations, if it is possible.
- Refactor some common code in methods.
- Use C++ template to allow other types of data. On the evaluation test is not provided if there are more types of data, but we assume that it is necessary on the final class, even if it is not used by ATLAS, because it will be used by other projects.
- Create doxygen documentation. It is important because final library users should know how to use the new class without looking at the implementation.

II.6. Expected results by mentors

As a results, on project proposal web page [1], is said:
“A working implementation of symmetric matrices in Eigen, ready to be submitted as a patch for Eigen”.

With this information, we need to work with Eigen library in order to develop a new class for symmetric matrices, that could be sent as a patch to Eigen library and could be added to the library easily.

II.7. Eigen library analysis

On the evaluation test we checked how the library work in order to implement what we need to do it successfully. However, to plan how to develop the new class for Symmetric Matrices is necessary check what methods are implemented on Eigen and if they are easy enough to develop all of them thought the summer.

It could be interesting during community bounding and even on the first month, check with mentors and the community related to ATLAS, what they need. For example, do they need only basic arithmetic operations? Also, it is important to check with Eigen community what they think about this project and what is the minimum features of the new class, to send it as a patch. We are writing about basic and minimum because we should be realistic to do a robust and error-free class during the summer, not plan more that we could do.

Moreover, we should discuss about the interface we could use to operate with Symmetric Matrices.

II.7.1. Eigen library modules

To fast-introduce Eigen library, we are going to just describe a little bit their modules. The Eigen library has the following modules:

- Core: Matrix and Array classes, basic linear algebra (including triangular and selfadjoint products), array manipulation.
- Geometry: Transform, Translation, Scaling, Rotation2D and 3D rotations (Quaternion, AngleAxis).
- LU: Inverse, determinant, LU decompositions with solver (FullPivLU, PartialPivLU)
- Cholesky: LLT and LDLT Cholesky factorization with solver

- Householder: Householder transformations; this module is used by several linear algebra modules.
- SVD: SVD decompositions with least-squares solver (JacobiSVD, BDCSVD)
- QR: QR decomposition with solver (HouseholderQR, ColPivHouseholderQR, FullPivHouseholderQR)
- Eigenvalues: Eigenvalue, eigenvector decompositions (EigenSolver, SelfAdjointEigenSolver, ComplexEigenSolver)
- Sparse: Sparse matrix storage and related basic linear algebra (SparseMatrix, SparseVector)

We wrote an email on March, 2nd to one of the mentors and we do not get response about more details for this project. So, for this proposal, we assume that we need just to implement class for Core module. However, we strongly believe we need to talk with mentors and Eigen community to know what it is needed.

II.7.2. The Matrix class

Because we do not know what are the exact requirements, that we will discuss on the community bonding, we assume that we are going to re-implement Matrix class to allow Symmetric Matrix support.

Our proposal, that we have to check with mentors, is add a default template parameter to create a Symmetric Matrix or not.

This includes some of the following methods:

- Matrix constructors: empty, from another matrix, from scalars, etc.
- Set the matrix to zeros, ones, constant…
- Resize the matrix
- Get the norms
- Transpose the matrix
- Arithmetic operations +,- and *.
- Determinant
- Inverse
- And more...

Also, we need to know, as we said, that it is necessary to know which ones of this methods should be implemented and even on other modules, like LU and Cholesky.

II.8. Project schedule

Frankly speaking, I believe this is not possible to define a strict project schedule and it is going to be modify instantly on the community bonding period. However, we try to figure out what it is needed and roughly define what we are going to do.

II.8.1. Community bonding period

This period if for learning about the organization's community. We think we have two different communities on this project that is ATLAS related community (that included the mentors and people involved with ATLAS) and Eigen library community.

On this period, we are going to:

- Get in touch with the mentors.
- Check that are the necessities and the requirements for the new matrix class, for mentors and ATLAS community.
- Get in touch with Eigen community.
- Check what are the requirements to create a patch for a new matrix class.
- Compare ATLAS community necessities and requirements with Eigen community ones.
- Study Eigen matrix code
- Design an interface to check if it fits on what ATLAS community and Eigen community want.
- Define how to communicate with mentors and how often.

II.8.2. Summer period

We think this is too difficult to predict, because the schedule during the summer should be developed after community bonding, when the previous discussion is finished.

However, we draft the basics of any programming project:

1. Read and study the previous documentation available (code and documentation itseft).
2. Implement and document every part we do on the project.
3. Test what we do. Even more, do the tests before the implementation.
4. Go to the first task until the project finish.

II.8.3. Milestones during the summer

On the following table, we show a draft schedule proposal for the milestones of this project:

Table 1. General milestones for the summer

Community bonding	Getting in touch with ATLAS community, Eigen community and Eigen library, depthly. Learn Eigen library code style.
First month	Design what we have discuss on community bonding about how to develop the new matrix class. Develop easy methods of Core module about the matrix (addition/subtraction, transpose, etc.).

	Maybe multiplication can be developed here.
Second month	Develop other matrices methods, from other modules, like LU and Cholesky methods.
Third month	Finish some unfinished work that it is required for the project. Bug fixing. Do more testing, finish documentation and send the patch to Eigen repository. Do a final report.

Note: Development includes programming, documentation and tests.

Also, it is important during the project check runtime and memory consumption, using for example a profiler like Valgring. In addition, using multiple methods to analyze cpu usage and memory will be good to get the best algorithm performance. For measure memory and cpu there are unix/linux commands and functions, so we can check the two ways of performance measurements, e.g.

II.9. Project ideas

On this section, we write some ideas that are important for the project, and we propose for it. All the ideas are criticable and can be modified, to get the best from the time we use on the project. We are not going to suggest packages because Eigen will have some to generate documentation and continuous integration.

II.9.1. Project documentation and integration

The documentation we are going to generate, apart from a final report or even a blog with our advances, will be generated with doxygen. This will be code documentation.

We will use continuous integration, to check if the tests and the project compiles with our modification, apart from compile it on our machine. Also, we write unit test to check the functionality of every method.

II.9.2. The first critical part of the project: matrix multiplication

On the evaluation test we realize that multiplication is a critical part of the project, because it has a lot of computer complexity $O(n^3)$. This is the first critical issue we have on the project, and on the evaluation test we solved it using the easiest way of solving it, with classical matrix multiplication.

However, there are methods to reduce the complexity of the algorithm of multiplication. Coppersmith–Winograd algorithm [6] is used for extremely large matrices with $O(n^{2.807355})$ and Strassen algorithm [7] that is used for large matrices. There are not differences if the matrix is not large with Strassen algorithm and even on Coppersmith–Winograd algorithm.

On our problem, we think that the Coppersmith–Winograd algorithm will be in our solution, ~~because ATLAS uses extremely large matrices~~ (but, in general, most of the matrices are fairly small, so other algorithms could be used). However, there are other research papers about other algorithm based on Coppersmith–Winograd's one. For instance, we have Andrew Stothers in 2010 [8], Virginia Williams in 2011 [9] and François Le Gall in 2014[10]. The last one improved the algorithm to a complex of $O(2^{2.3728639})$.

About Strassen algorithm, we are going to just comment a few things to not extend this document a lot. There are papers about this algorithms and improvement even on ATLAS (D' Alberto, P. et al., 2005) [11]. And at this date, we found an improvement on a paper from 2016 [12] about an improvement of Straassen's Algorithm. Most of this improvement are using more memory (cache). It is well-know that if you want less CPU you should use more memory, in most cases. So you have to analyse what is more important for your project.

On this project, we will search on research papers repositories (Google Scholar, WOS, Scopus, etc.) for new matrix multiplication algorithms and even come up with a new algorithm that performs better than the ones available, maybe that just work with symmetric matrices, if it is possible.

II.9.3. Checking students ideas.

I think if we get selected, we should check other students proposals, with their permission, because as we could review, there are really interesting proposals and ideas that could help to do the best project. Obviously, if they want to contribute with the ideas they had written, we will include them on acknowledgements at the end of the project. This is one of the basis of open source software, we believe.

II.10. Our expected results

We have the same expected results as the mentors, but we strongly believe that this should be discuss during the project and with the issues that could happen during it. To add more information, we think we could have a functional symmetric matrix class with all methods working, ones more optimized than others, depending on what is critical on this project. The other methods could be solved without the fastest solution or we could help outside GSoC to implement them, if we do not have enough time during the summer. We must be realistic and distribute the time to get the best results.

III. Summary and conclusions

As a summary, we believe that is necessary on the community bonding define with mentors and communities the schedule to follow, and even what is possible to develop on the period and what is necessary.

Also, we need to check the best way to implement the methods in order to get the best performance in CPU and memory, that is the main aim of this project, improve matrix operations. All other things like testing, documentation and clean code are assumed.

Finally, as appreciation, we think this project is really challenging to get the position. We believe that the organization should use another way to filter the candidates, because doing the evaluation test and the proposal request time and effort. However, if we do not get selected, I feel I have learnt about Eigen library, ATLAS project and more about linear algebra and matrices, that it will help me in the future on my professional life as computer scientist.

IV. Resources and bibliography

[1] Faster matrix algebra for ATLAS organization proposal. 2018. Available online.
[2] ”I want to be a Google programmer”. La Provincia Press. In Spanish. 2011. Available online.
[3] About ATLAS. CERN. Available online.
[4] EMELIYANOV, D. & MARTIN-HAUGH, S. HEPSF Google Summer of Code: Faster Matrix Algebra for ATLAS. STFC Rutherford Appleton Laboratory. February, 2018. Available online.
[5] ATLASEigen_EvalTest GitHub project. Jose Lorenzo. 2018. Available online.
[6] Coppersmith, D., & Winograd, S. (1990). Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, *9*(3), 251–280. https://doi.org/10.1016/S0747-7171(08)80013-2
[7] Strassen, V. (1969). Gaussian Elimination is Not Optimal. *Numer. Math.*, *13*(4), 354–356. https://doi.org/10.1007/BF02165411
[8] Stothers, A. (2010). On the Complexity of Matrix Multiplication. Available online.
[9] Williams V. (2014). Multiplying matrices in $O(n^{2.373})$ time. Stanford University. Available online.
[10] Gall, F. Le. (2014). Powers of Tensors and Fast Matrix Multiplication, 1–28. https://doi.org/10.1145/2608628.2608664
[11] D'Alberto, Paolo; Nicolau, Alexandru (2005). Using Recursion to Boost ATLAS's Performance . Sixth Int'l Symp. on High Performance Computing. Available online.
[12] Huang, J., Smith, T. M., Henry, G. M., & van de Geijn, R. A. (2016). Strassen’s Algorithm Reloaded. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (p. 59:1--59:12). Piscataway, NJ, USA: IEEE Press. Retrieved from http://dl.acm.org/citation.cfm?id=3014904.3014983