Project Proposal

Google Summer of Code 2018

Faster matrix algebra for ATLAS

CERN-HSF

High energy physics software and computing

Mentored By:

- Stewart Martin-Haugh
- Dmitry Emeliyanov

Abhishek Kumar

akumar9_be15@thapar.edu

abhisingh10p14@gmail.com

Ph: +91 8195901203

Index

Contact Information

Name : Abhishek Kumar
University : Thapar Institute Of Engineering & Technology
Email-id : akumar9_be15@thapar.edu
abhisingh10p14@gmail.com
Github username : abhising10p14
Bitbucket : https://bitbucket.org/abhisingh10p14/
Blogs : http://abhi5631.blogspot.in/
http://machineanddata.blogspot.in/
Phone : +91 8195901203, +91 7367028486
Address : EF-406 Hostel J
Thapar Institute Of Engineering & Technology
Patiala, Punjab, India.
Time-zone : UTC +5:30

Introduction

The interactions due to collision of particle in the ATLAS detectors create an enormous flow of data. Complex data-acquisition and computing systems are then used to analyze the collision events recorded.
When the LHC is operating, 40 million packets of protons collide every second at the center of the ATLAS detector. Every time there is a collision, the ATLAS Trigger selects interesting collisions and writes them to disk for further analysis. The Event Data Model(EDM) which is intended to contain the detailed output of tracking the trajectories analyzes around **1PB** of Raw Data per year.
To track the trajectory of charged particles and their collisions a large number of alignment parameters are required which can range from **10^4** to **10^5**. Several alignment algorithms as well as heavy CPU consumption is used to optimize these parameters. One such method requires the solution of a system of linear equations. To store these equations, a *symmetric n-by-n matrix* is used.Symmetric matrices make up a large fraction of the matrices used in track reconstruction. ATLAS Tracking Software makes heavy use of matrix algebra, implemented with the **Eigen** library. The CLHEP maths library was previously used throughout the ATLAS software, but after investigating potential alternatives, the decision was made to move to Eigen. Eigen was chosen since it offered the largest performance improvements for ATLAS.

Literature Survey

1. **The Eigen Algebra Library**- Eigen is a C++ template library for linear algebra. It is a pure C++ library and does not have any dependencies beyond libstdc++. A separate compilation of Eigen is not necessary because it is composed of header files only and is then compiled with the application. Eigen provides different optimizations appropriate to the size of the matrices it is using. It supports SIMD(single instruction multiple data) vectorized instructions for basic operations such as 4 X 4 matrix multiplication. Unlike the ROOT Framework, Eigen can be used for dynamically sized matrices. It is used by e.g. the Google Tensorflow, Ceres Large Survey Synoptic Telescope projects as well as ATLAS. Several performance profiling tests using PAPI, PIN Tool(for CPU time) e.t.c were performed on these frameworks and libraries(ROOT,CLHEP,EIGEN). In case of the CPU time for multiplication of two four-dimensional matrices, Eigen performed best. These results combined with the results of the CPU performance

comparisons in a small test framework showed Eigen performs the **fastest** for linear algebra operations.

Issue

One of the shortcomings of Eigen is that currently, it does not provide any separate template class for Symmetric Matrices which are heavily used in ATLAS software. So to store a symmetric matrix, one has to use a general **Eigen:: Matrix**. Due to this, lot of space (think in terms of PB) is wasted for nothing since the upper and lower triangular parts of a symmetric matrix are same. Also, a lot of CPU computing power is used. Eigen uses symmetric matrices for several purposes like inversion(using the **PLU** decomposition) of a matrix where it needs the self adjoint matrix of the given matrix which is formed by **SelfAdjointView.** In these cases,using a general matrix where a symmetric matrix can be used, costs space as well as time.

Solution

The current need is a seperate Symmetric Matrix class just similar to the **Eigen:: Matrix** with working implementation of all the operations related to symmetric matrices as well as with the generic matrices, ready to be submitted as a patch for Eigen. This will help ATLAS find particles while using less computing power and less storage space.

Project Goals

Objectives

- Create a standalone Symmetric class similar to **Eigen:: Matrix** which can store a symmetric matrix for a given **Eigen:: Matrix**.
- Add all the functionalities of **Eigen:: Matrix** to the class Symmetric Matrix
- Deploy all the member functions, constructors, Exception handling cases
- Implement all the higher computational operations of linear algebra related to the Symmetric matrix using optimized subroutines similar to that of **Blas,** GPU approaches of **CUDA**, **LAPACK, Blaze** e.t.c .
- Using the **functional programming** approach for computational purpose.
- Keep on adding all my ongoing work and implementations on my blog(link) as well as in the documentation of the project.

- Submit the class as a patch for Eigen.

Tasks

1. **Define a Symmetric class in a separate header file “Symmetric.h”**
    - Define all the member variables like rows, columns e.t.c of the class which are available in **Eigen:: Matrix.**
    - Define all the member functions.
    - Define all the constructors.
    - Define all the operator overloadings.
    - Add different types of typedefs similar to the typedefs of **Eigen:: Matrix.**
    - Add some other functions like :
        - joining two Symmetric Matrices together into a new one
        - e.g: symmatA<< symmatA, symmatA/10, symmatA/10, symmatA;
        - **.row(i), .col(j)** accessors
        - .dot()
        - .vector()
        - .mean()
        - .trace()
        - .minCoeff()
        - .maxCoeff()
        - .transpose()
        - Triangular views
        - .noalias()
        - Static methods such as **Zero()** and **Constant()**
        - **SymMat:: Random() :** like the Gaussian, Laplacian
        - **transpose()** -> same as the Symmetric matrix
        - **transposeInplace()** -> same as transpose()
        - **trace()**
        - **adjoint()**
        - **adjointInplace()**
        - **JacobiRotation()**
        - Other rotations like **applyonLeft, applyonRight** etc.
        - Decompositions and solving linear equations
            - **PartialPivLU**
            - **FullPivLU**

        - **HouseholderQR**
        - **ColPivHouseholderQR**
        - **FullPivHouseholderQR**
        - **LLT**
        - **LDLT**
        - **JacobiSVD**
    - Functions to compute eigenvalues and eigenvectors, similar to:
        - **EigenSolver**
        - **ComplexEigenSolver**
    - Function for solving the least squares
    - Add different types of Symmetric matrices like skew, Symmetrizable, sparse, Complex , providing more space efficiency in case of sparse symmetric matrices.
    - Sparse Symmetric matrix can be stored in more space efficient ways.
    - The header file must be similar to that of an Eigen header file.
    - Update the Documentation.

2.**Create a Source file "Symmetric.cpp" for the above-defined header file**

- Implement the member functions defined in the header files.
- Implement the constructors for all the defined typedefs.
- Implement all the operator overloadings, like Coefficient assessors, '*', '+', '-' e.t.c.
- Try to use the **functional programming** approach if possible and useful.
- Explore the possibility of using efficient methods and algorithms for different functions like :
    - Optimize the matrix multiplication by using the **GEMM(General Matrix Multiplication)** routine of **Blas** as the basic matrix multiplication takes **(O(n^3))**
    - *Strassen algorithm* **O((n^(2.8074))***, Karatsuba algorithm* for multiplication of two numbers
    - Using multithreading() -> This reduced the time of multiplication by 27% when 4 threads were used(for a 100 by 100 matrix)
    - Use libraries like **Boost** for threading
    - GPU approaches with **CUDA** in **C++**
    - Optimize the matrix inversion for symmetric matrices using the methods like **PLU(**Permutation, lower and upper triangle **)** decomposition. **Eigen** uses **PLU** for matrices of size greater than 4x4.

        - Fast I/O methods like using **std::ios::sync_with_stdio(false)** for faster I/O operations.

- Implement the comma-initialization method.
- Add the Exception Handling block for each and every possible functions, constructors, operators.
- Add the **Assertions** for detecting the bugs.
- Add all the resizing methods like resize(), Assignment and resize methods similar to **Eigen:: Matrix.**
- Add the functionality of dynamic matrix initialization.
- Providing the maximum possible storage space(like **unsigned long long int**)to the given Class member variables.
- Test all the functions, variables, constructors, operators, typedefs with all the corner cases and fix bugs.
- Explore for extra functionalities which can be added either from the past used ROOT, CLHEP or new ones like distributed work using multithreading for faster results in case of largely sized matrices.
- Follow the **C++ 98** standards:
    - As the code is self explanatory and anyone can use and edit it. For example: the use of **auto** in **c++11** renders the code unreadable
    - It can build on older machines also.
    - Debugging would be easier especially in case of conversions
- If possible, use the **C++11** features like thread class, rvalue references, copying the pointer to buffer. Add the ifdef so that the code compiles fine under C++ 98.
- Update the documentation and the blog.

3**. Integrate all the class members with Eigen:: Matrix**

- Test the operation of all the functions of Symmetric Matrix with Eigen::Matrix using every possible test cases.
- Provide test cases for all of the operations.
- Comparison of storage size and execution time of different approach for the functions and choose the best ones.
- Build the libraries using **Cmake** and the cross check it’s working on other different system.
- Update the documentation and blog regarding the tests.

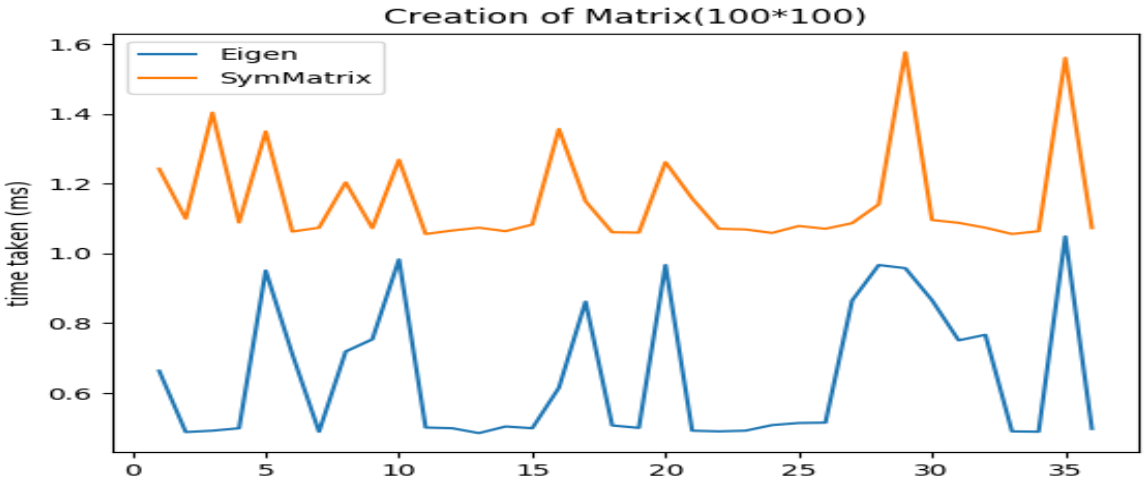
4. **Submit the project as a patch for Eigen**

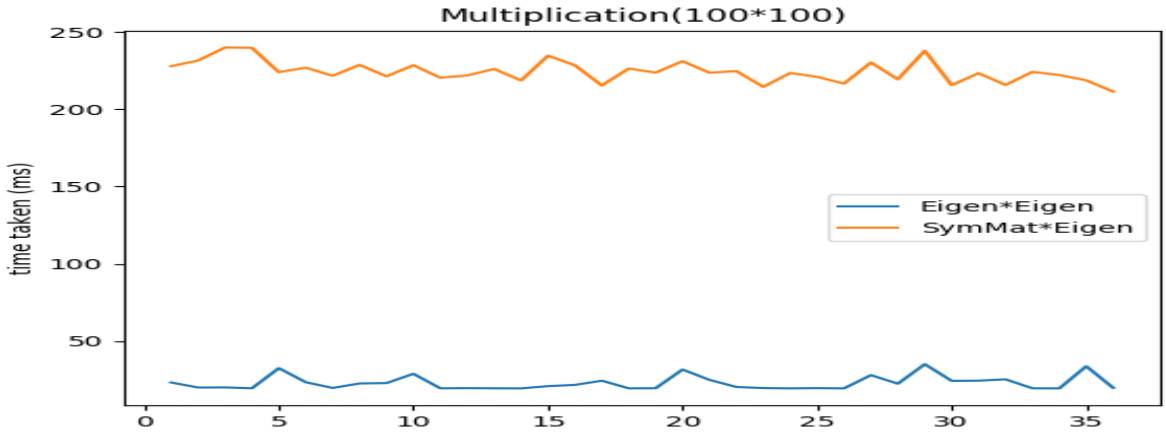
- Test the class on mathematical works of ATLAS

- Ask for feedback and suggestions from the users and implement them
- Ensure that the code is reusable
- Finalize the documentation and update my blog

My work so far:

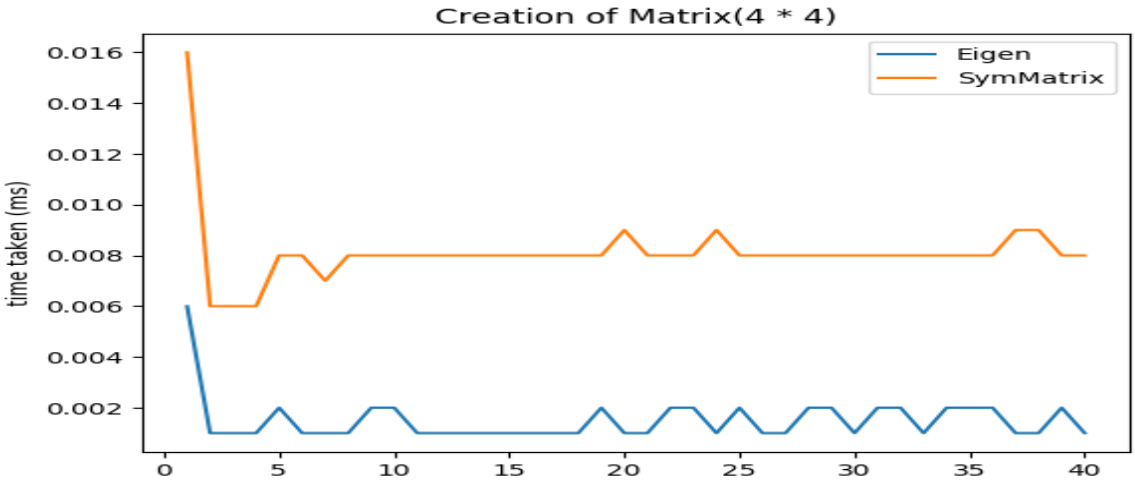
Appart from working on the evaluation test for this project, I have worked on several other parameters regarding the implementation of this Class. I have added other operators like <<,= . Also to compare and check the time for multiplication operation of class for large as well as small sized matrix, I applied multithreading approach on one of the multiplication operation between a Symmetric Matrix and an Eigen::Matrix. The result were more or less similar for Eigen and SymMat in case of small sized matrices where only single thread was used for multiplication. But the results were different in case of matrices of size 100 by 100. The time taken in multiplication and matrix creation of an Eigen:: Matrix(100 x 100) and a Symmetric Matrix (100 x 100) were noted and their matplotlib plot shows that:

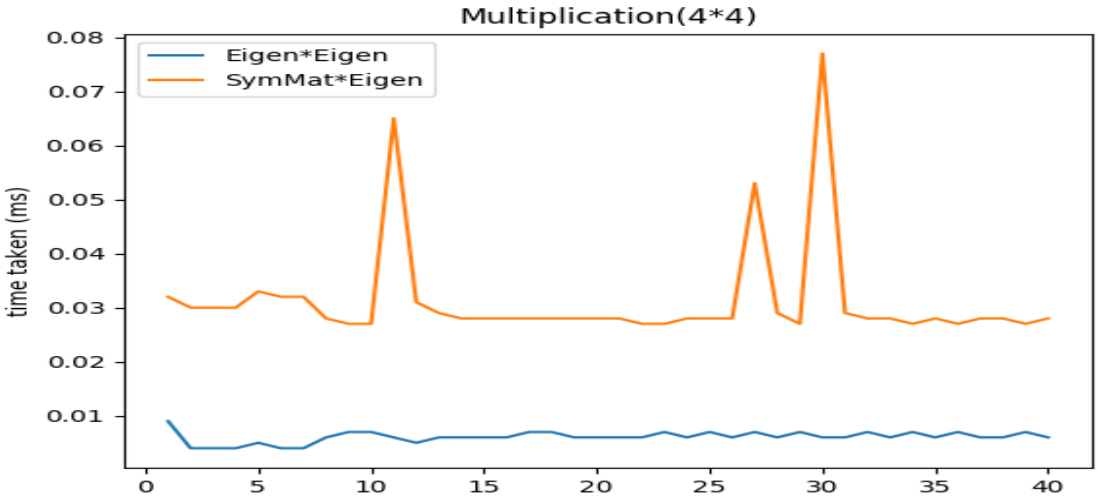




For the multiplication of SymMat and Eigen:: Matrix of size 100*100, multithreading with four threads running parallely were used.

For the size of matrix 4*4 (**without using multithreading**):





The current issue is that even after using multithreading, the speed has not yet matched with the Eigen. So the first priority is to enhance this speed for different operations at least to match with Eigen also, to reduce storage space as much possible.
Later on, my Mentor suggested me that single-threaded performance is better as ATLAS uses small to medium size matrices that easily fit on a single CPU. Also it can be easily compiled under **C++98**.
Two points to be noted here :

1. The multithreading should be used only if the size of the matrix is large.
2. The code should compile fine under C++98

So **#ifdef** macros have been used in those parts which compile only under c++ 11.
Also, the multithreading is used only if the size of the matrix is >=50(this can be changed later).
I have implemented the basic functions like transpose(), transposeInPlace(), trace() till now.

While working on this project, I came to know about several good as well as bad practices of coding especially in case of the making the Template class and building of the libraries which led to several commits in my repository. I adopted the good ones and am always trying to avoid the bad practices of coding.
This is my github id for this project.

Check out about the current work and issues regarding this project on my blog. Any kind of suggestions and feedbacks are most welcome.(Link)

Timeline

Duration	Task
March 27	**Deadline for submission of proposal**
March 27-April 22	• Learn more about optimized methods for storage of symmetric matrices, • Acquire all the knowledge about the best way to implement a template class, ways to store sparse Symmetric matrix • Learn more about the Eigen library, GEMM, Blas,Lapack, Blaze e.t.c • work on matching the timing constraints for Matrix operations better if possible or at least similar to that of Eigen.
April 23-May 14	**Official community bonding period** • Get to know the community better • Know about the current ideas and implementations being done regarding this project. • Discuss my ideas with the **Eigen** core development team as well as the mentors. • Discuss with mentors that how final class would be, about it's dependencies, about the other libraries to be used, about all the size,speed constraints in detail. • Explore the **Eigen::MatrixBase** class • Check out all details of implementation and decide on undecided areas of implementation

	possible such as the implementation of efficient algorithms for multiplication, inversion(**PLU**), search e.t.c. • **Begin Task 1 and Task2 simultaneously** • Define all the extra members, typedefs, accessors, operator overloadings in the header file • Declare the corresponding constructors, functions, accessors, initializers in the source file. • Push all my current work on a regular basis
May 14-June 5	**Official Coding period starts** • Finish the implementation of all the member functions and operations as listed in **Task 1** • Work on the possible efficient methods and algorithms for complex operations • Work on detecting the aliasing effect using **eval()** • Test for storage size, running time and the possible bugs (**Task 3)** • Test for the coding standards(c++98,c++11) being followed and implement any changes required • Update the documentation, testing and the blog
June 5-June 10	• The time period for any unexpected delays
June 11-June 15	**Submitting Phase 1 Evaluations** • Submit code for evaluations along with documentation

June 16-July 4	• Implement the optimized methods and algorithms of all the major operations and functions. • Start the implementation of functions and the decompositions defined in **Task 1** • Implement the complex algorithms like the matrix inversion,Rotation base • Test for storage size, running time and the possible bugs of the new methods.(**Task 3**). • Update the documentation,testing and the blog
July 5-July 8	• The time period for any unexpected delays • Finish the implementation of functions and the decompositions **End Task 1 and Task 2 and Task 3**
July 9-July 13	**Submitting Phase 2 Evaluations** • Submit code
July 14-August 5	**Start Task 4** • Test the working of the code on different systems as well as different C++ compilers • Submit the code for test on ATLAS works • Implement any additional suggestions and fix any error or bug if any • Complete the code and final documentation
August 6-August 14	**End Task 4** **Final Submission** • Submit the final code and the documentation

Deliverables

- Working implementation for Symmetric Matrix Class and all it's corresponding functions and operations ready to be submitted as a patch for Eigen.
- A working library ready to be included in any code related to Symmetric Matrices.
- Documentation, unit tests and blog for the Class and it's members.

Benefits to The Community

Currently, if someone has to use symmetric matrix for calculations using Eigen, he would have to use **Eigen::Matrix**.This uses space and consumes time which can be reduced in case of a symmetric matrix.
This implementation of a separate class for symmetric matrix would let the users work on symmetric matrices in an optimal way which would save space as well as the computation time.

About Me

I am a third-year undergraduate student at Thapar Institute Of Engineering & Technology, India. I am pursuing a B.E degree in Computer Science with the additional courses of Machine Learning and Data Analysis.

I use Linux Mint 18 'Sarah' on my system. I've been coding in C and C++ for over 3 years and in Python for under a year and am proficient in all three of them now. I have used both Octave and Matlab for projects as well as my college courses.

I am very much interested in Mathematics, Machine Learning, and Image processing and have done many projects in these fields(link). I have also worked on **Qt** framework in standard C++ and have made a Text Editor using it(link). I have worked as a freelancer for one year on several major and side projects like word recognition, Image detection, Searching Algorithms. I love participating in competitive coding contests.

I have been using git and GitHub for quite some time and have enough knowledge to carry out this project successfully in the given time.I am also familiar with other version control systems like **Mercurial** and have also worked on **BitBucket** and am always willing to become more familiar and work with these.

My Mathematical, as well as Computer science-related background is enough for the project. I have taken the following courses relevant to this project :
Numerical analysis, Optimization Techniques, Linear Algebra, Discrete Mathematics, Physics, C, C++, Python, Data Structures and Algorithms and Machine Learning.

I am also familiar with the **Eigen** 3.0 and have used it for Matrix. I have gone through several research papers and articles regarding Eigen, Symmetric Matrix, Root, Blas.

I am updating all my work regarding this project on this blog of mine. Any sort of suggestion or feedback is most welcome.

References

1. Symmetric Matrix
2. V. Blobel, Software alignment for tracking detectors, NIM A 566, (2006) 5 – 13 , Track based Alignment Algorithms
3. ATLAS Tracking Software,LHC Run 2, Nicholas Styles et al 2015 J. Phys.: Conf. Ser. 608 012047[1]
4. Optimisation of the ATLAS track reconstruction software for Run-2 April 14, 2015 [1]
5. Event Data Model in ATLAS , Edward Moyse, (University of Massachusetts Amherst) CHEP 2006, Mumbai[1]
6. ROOT:: Smatrix
7. Symmetric matrix class of ROOT[1]
8. ATLAS offline software performance monitoring and optimization, N Chauhan et al 2014 J. Phys.: Conf. Ser. 513 052022 [1]
9. Optimize large matrices multiplication in Eigen[1]
10. Eigen:: MatrixBase , Eigen:: Matrix, Aliasing, GEMM
11. ATLAS BLAS implementation[1][2][3]
12. Strassen Algorithm for Multiplication

13. BLIS -high-performance BLAS-like dense linear algebra library
14. Comparison of linear algebra libraries.[1]