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

About me

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/

http://abbhi5631.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

Personal Information

I am a third-year 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 do participate in competitive coding contests. I have participated in Google Code Jam in 2017 and had qualified for the second round.

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, CLPHED.

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

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 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 **BLASS,** GPU approaches of **CUDA**, **LAPACK** 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 overloading.
   - 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

   - Add different types of Symmetric matrices like skew, Symmetrizable, sparse, Complex , providing more space efficiency in case of sparse symmetric matrices
   - 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 **BLASS** 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
  - 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 an edit it. For example: the use of **auto** 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.
- Update the documentation and 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.
- Comparison of storage size and execution time of different approach for the functions and choose the best ones.
- 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

Note that several parts of Tasks have already been implemented during the evaluation phase of the project. The code and compilation details are available at this GitHub link http://github.com/abhising10p14/Symmetric-Matrix *.*
Check out the current work regarding this project and the current issues on my blog.
Any kind of suggestion and feedback are most welcome.(Link)
Also, continuous contributions are being made for this project.

Timeline

Duration	Task
March 27	**Deadline for submission of proposal**
March 27-April 22	- Learn more about optimized methods for storage of symmetric matrices - Learn more about the Eigen library, GEMM, BLASS - work on the matching the timing constraints for operations similar to Eigen.
April 23-May 14	**Official community bonding period** - Get to know the community better - Chalk 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.

May 14-June 5	**Official Coding period starts** ● Finish the implementation of all of the members and basic operations. ● Collect the information regarding possible efficient methods and algorithms for complex operations ● Test for storage size, running time and the possible bugs (**Task 3)** ● Update the documentation and blog
June 5-June 10	● The time period for any unexpected delays
June 11-June 15	**Submitting Phase 1 Evaluations** ● Submit code in the git repository with documentation
June 16-July 4	● Implement the optimized method and algorithm of all the major operations and functions. ● Test for storage size, running time and the possible bugs of the new methods.(**Task 3**). ● Update the documentation and blog
July 5-July 8	● The time period for any unexpected delays **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** ● 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.
● Documentation and blog for the Class and it's members.

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 BLASS implementation[1][2][3]
12. Strassen Algorithm for Multiplication
13. BLIS -high-performance BLAS-like dense linear algebra library