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

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 course 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.

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 sonsumption 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. **ROOT Framework**

   ROOT is an object-oriented program and library developed by CERN. It was originally designed for particle physics data analysis and contains several features specific to this field, but it is also used in other applications such as astronomy and data mining.

   ROOT is optimized for describing small matrices and vectors and **It can be used only in problems when the size of the matrices is known at compile time**, like in the tracking reconstruction of physics experiments.

   The Matrix Class of ROOT**(SMatrix)**

   The template class **ROOT::Math::SMatrix** represents a matrix of arbitrary type with nrows x ncol dimension. Only limited linear algebra functionality is available for Smatrix.

The **ROOT::Math::SMatrix** class has following 4 template parameters, which define at compile time, its properties:

1. type of the contained elements, T, for example float or double;
2. number of rows;
3. number of columns;
4. representation type. This is a class describing the underlined storage model of the Matrix. Presently exists only two types of this class:
    1. **ROOT::Math::MatRepStd** - for a general *n* rows x *n* cols matrix.
    2. **ROOT::Math::MatRepSym** - for a symmetric matrix of size NxN. This class is a template on the contained type and on the symmetric matrix size N. It has as data member an array of type T of size N*(N+1)/2. It does not provide the complete linear algebra functionality but the basic matrix and vector functions such as matrix-matrix, matrix-vector, vector-vector operations, plus some extra functionality for square matrices, like inversion and determinant calculation.

Criticisms of ROOT include its difficulty for beginners, as well as various aspects of its design and implementation. Frequent causes of frustration include extreme code bloat, heavy use of global variables, and a perverse class hierarchy. But as of now, Root is mainly used for plotting and data analysis at CERN. Issues like class structure issues, functionality issues, general design issues. Also, **not all matrix and vector operations are implemented, unlike Eigen or CLHEP.**

2. **The CLHEP project** - A Class Library for High Energy Physics .It is C++ library that provides utility classes for general numerical programming, vector arithmetic, geometry, pseudorandom number generation and specifically targeted for high energy physics simulation and analysis software. It contains vector and matrix classes for different dimensions, linear algebra functions, and geometry packages. All classes are connected to a generic interface for easy use.

   The Event Data Model(which analyzes the Raw Data produced during a collision), as well as the algorithmic codes, were based on CLHEP. After thorough research, it was realized that CLHEP was one of the bottlenecks in the ATLAS software. So ATLAS switched to EIGEN algebra library.

3. **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. 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.

Solution

The current need is a Symmetric Matrix class just similar to the **Eigen:: Matrix.** A working implementation of symmetric matrices in Eigen, 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 operations related to the Matrix using optimized subroutines similar to **BLASS**
- 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()
    - .noalias()
    - static methods such as **Zero()** and **Constant()**
    - **SymMat:: Random()** : like the Gaussian, Laplacian
- Add different types of Symmetric matrices like skew, Symmetrizable, providing more space efficiecny 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 accessors, '*', '+', '-' e.t.c
    - Explore the possibility of using efficient algorithms for different functions like

    - Optimize the matrix multiplication by using the **GEMM(General Matrix Multiplication)** routine of **BLASS** , by deviding the work in parllel task, by C++ pointer Math
    - Basic matrix multiplication (O(n^3))
    - *Strassen algorithm* **O((n^(2.8074))**, *Karatsuba algorithm* for multiplication
    - Using multithreading() -> This reduced the time by 27% when 4 threads were used
    - 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.
- 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.
- Comparision 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, 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 efficent 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 Task3**
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 mebers.

Refrences

[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 April14, 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