Student Project: Efficient Algorithm Implementation in C++ for large-scale Biomedical Data

The group is involved with the development of analysis methods for large-scale biomedical data. While small and low-dimensional data sets can be processed with Python or R based implementations, do high-dimensional data scenarios and large-scale biobanks require advanced implementations of statistical machine learning algorithms for with a backend or complete implementation in C++.
In these implementations, efficiency is required with regards to algorithmic operations, but also efficient memory handling/memory mapping and online processing.
Moreover, we still aiming for frontends in R and Python, at which analysts are comfortable working with.
A major task of this role will be the creation of a benchmark study for established linear algebra libraries, to help us pick the best performing tools for the future development of algorithmic implementations, plus the search through scientific literature and manuals.
Additionally, we aim at recurrent feedback and presentations about your progress.

Tools for algorithm development for very high-dimensional data

- **Optimized Libraries**: Utilizing optimized linear algebra libraries like Eigen or Armadillo can accelerate computations on high-dimensional data by leveraging efficient algorithms and hardware optimizations (e.g., SIMD instructions).

- **Sparse Data Structures**: Within high-dimensional data, many features may be zero or sparse. Using sparse data structures such as compressed sparse row (CSR) or compressed sparse column (CSC) formats for matrices can significantly reduce memory usage and computational complexity.

- **Parallelism and Multithreading**: Parallel processing and multithreading can improve the performance of algorithms for high-dimensional data by leveraging multiple CPU cores. Libraries like OpenMP and Intel Threading Building Blocks (TBB) provide easy-to-use parallel constructs for C++ programs.

- **Memory Management**: Efficient memory management techniques like memory pooling, smart pointers (e.g., `std::shared_ptr`, `std::unique_ptr`), and custom memory allocators can help manage memory efficiently and reduce overhead.

- **Streaming and Incremental Learning**: High-dimensional data may be too large to fit into memory at once. Streaming algorithms and incremental learning techniques allow processing data in batches or streams, reducing memory requirements and enabling real-time analysis.

- **Algorithm Design Patterns**: Design patterns such as strategy pattern, factory pattern, and decorator pattern can be useful for designing flexible and scalable algorithms.

- **Performance Profiling and Optimization**: To identify bottlenecks in your algorithms and optimize critical sections for better performance. Techniques such as loop unrolling, cache optimization, and algorithmic optimizations can help improve the efficiency of algorithms operating on high-dimensional data.

Reporting

- In-person or Zoom meeting: every 2 weeks – 1 hr. to discuss progress (dates to be agreed upon).
- Setup a GitHub and share your Code with us.
- Prepare an ongoing set of slides for your progress report: current insights, achievements, problems, figures, and plots. Discuss ideas with the audience. Will also serve as your documentation of your work.
- Outline how you want to continue.

Project Schedule (roughly – to be discussed and adjusted, Version 1.0):

Month	Topic
March	• Install C++ compiler and IDE. • Prepare IDE to work with. • Setup Git and GitHub (continuous process + Readme) and share it with us. • Refresh C++ and Git knowledge and work through tutorials (continue with them over the course of the project). Document your progress with a GitHub Repository.
April	• Research Study: Selection of candidate libraries for mathematical operations, familiarization, and demo. ▪ Are there any inbuild mathematical capabilities within the C++ standard template library (STL)? ▪ Which mathematical libraries are available in C++, what are their strengths and dependencies? ▪ What is their status of life, how well are they maintained, documented? ▪ Make suggestions which are worth investigating. Comparison Metrics? ▪ How to install them (OS), Docker images? ▪ Are their industry or community standards? Engagements? Commercial or open source? ▪ Compatibility with C++ Standard Template Library (STL)? ▪ Are there other benchmark studies available? ▪ Start building up some demo examples for a classical set of linear algebra operations, sparsity formats and tensor operations, if possible, a framework. ▪ Is their compatibility with Python/R or can we use them as C++ backend code for Python/R frontends? Performance overhead, language bindings, data interchange format? ▪ Evaluate your results with statistics and plots and document your results on some slides, presentation of intermediate results.
May	• Research Study: Benchmarking study for promising libraries. ▪ Evaluate time and memory performance for standard operations and algebra for low- and high-dimensional data regimes. ▪ Implement and compare two sparse selectors, such as OMP, MP, WMP, Thresholding, or others. ▪ Use C++ Code as backend and develop some frontend in Python or R to make use of your selectors there. Or familiarize with existing candidates s.a., Rcpp, Cpp11, Cython, and make your results applicable from R/Python. ▪ Evaluate your results with statistics and plots and document your results on some slides, presentation of intermediate results.
June	• Presentation and discussion of research results in front of the group and colleagues.

Getting Started

Git & GitHub

- https://www.youtube.com/watch?app=desktop&v=RGOj5yH7evk
- https://connkat.medium.com/setting-up-multiple-ssh-keys-on-one-computer-75f068d972d9
- https://www.atlassian.com/git/tutorials/install-git
- Optional: https://git-scm.com/book/en/v2

Code Editor/Integrated Development Environment (IDE)

- Visual Studio Code (VSC) (recommended): https://code.visualstudio.com/
  - https://www.youtube.com/watch?app=desktop&v=WPqXP_kLzpo
  - https://www.youtube.com/watch?app=desktop&v=VqCgcpAypFQ
  - https://www.youtube.com/watch?app=desktop&v=ORrELERGIHs
  - https://www.youtube.com/watch?v=YjhkcvS1xKU
  - Recommended extensions:
    - C/C++ Extension Pack (franneck94), C/C++ Runner (franneck94), Coding Tools Extension Pack (franneck94), Prettier – Code formatter, Material Icon Theme
- Visual Studio (Visual C++): https://visualstudio.microsoft.com/

Install C++ Compiler

- **Windows :** minGW64 via MSYS2 (https://www.msys2.org/) or https://sourceforge.net/projects/mingw-w64/ or you use Cygwin (https://www.cygwin.com/)
- **Mac :** XCode ; alternative : Linux Virtual machine https://www.macobserver.com/analysis/5-ways-to-write-c-code-on-your-mac/
- **Linux :** https://www.cyberciti.biz/faq/howto-compile-and-run-c-cplusplus-code-in-linux/

C++ Programming Course

- Run C++ in VSC: https://www.youtube.com/watch?v=3-9sObAg6R0
- Setup VSC: https://www.youtube.com/watch?v=YgKnzIV4uME
- C++ FULL COURSE For Beginners (10 hours): https://www.youtube.com/watch?v=GQp1zzTwrIg
- C++ Programming Course – Beginner to Advanced (31 hours): https://www.youtube.com/watch?v=8jLOx1hD3_o
- Very good channel: CodeBeauty https://www.youtube.com/@CodeBeauty/playlists
- C++ Full Course (FOUR HOUR All-in-One Tutorial for Beginners) https://www.youtube.com/watch?v=9Myk2vcK8s8&list=PL_c9BZzLwBRIwEnaRU2LC4GkqxUbD0Auc
- C++ Programming All-in-One Tutorial (10 Hours) https://www.youtube.com/watch?v=_bYFu9mBnr4&list=PL_c9BZzLwBRJ55lLw8PdPlTVbIIlPKfX5

Optional content:

- Concurrency and multi-threading: https://www.youtube.com/watch?v=xPqnoB2hjjA
- Modern cpp concurrency https://www.youtube.com/watch?v=Fn0xBsmact4&list=PLvv0ScY6vfd_ocTP2ZlicgqKnvq50OCXM

Optional look up references:

- https://en.cppreference.com/w/
- https://www.boost.org
- https://en.wikiversity.org/wiki/C%2B%2B
- Professional C++ (2021), Gregoire Marc, 5th edition, *John Wiley & Sons, Inc.*
- StackOverflow
- ChatGPT, Claude

Optional but paid (~10 €) courses from Udemy:

- Frank Mitropolous: https://www.udemy.com/course/beginning-c-plus-plus-programming/?couponCode=ST12MT030524
- Jan Schaffranek (German): https://www.udemy.com/course/der-komplettkurs-zur-modernen-c-programmierung/?couponCode=ST12MT030524

C++ libraries for mathematical algorithm development

Eigen: A C++ template library for linear algebra. It provides efficient implementations of common linear algebra operations such as matrix and vector operations, matrix decompositions, and solving linear systems of equations.

- https://eigen.tuxfamily.org/index.php?title=Main_Page

Boost.Math: A library as part of the Boost C++ libraries and provides a wide range of mathematical functions and constants. It includes functions for special functions, numerical integration, interpolation, random number generation, and more.

- https://www.boost.org/doc/libs/?view=category_math

GSL (GNU Scientific Library): GSL is a collection of numerical routines for scientific computing written in C. While it's not a C++ library per se, it can be used with C++ code through wrapper classes or directly. GSL provides functions for numerical integration, root finding, optimization, interpolation, and more.

- https://www.gnu.org/software/gsl/

Armadillo: Armadillo is a high-quality linear algebra library for C++ with a syntax like MATLAB. It provides easy-to-use classes and functions for matrix and vector operations, linear solvers, eigenvalue decomposition, and more.

- https://arma.sourceforge.net

NT2 (The Numerical Template Toolbox): Is a C++ library for numerical computing that focuses on performance and expression templates. It provides a wide range of functions for numerical computation and can leverage SIMD (Single Instruction, Multiple Data) instructions for parallelism.

- https://github.com/jtlap/nt2

BLAS (Basic Linear Algebra Subprograms) and **LAPACK (Linear Algebra Package)**: While not C++ libraries themselves, BLAS and LAPACK are widely used libraries for linear algebra operations. Many other libraries, such as Eigen and Armadillo, can be configured to use optimized BLAS and LAPACK implementations for better performance.

- https://www.netlib.org/blas/

ALGLIB: A cross-platform library for numerical analysis and data processing supporting C++, C#, Java, Python, Delphi, and several systems (Windows and POSIX, including Linux). Features are Classification/Regression, Statistics, Optimization, nonlinear solvers, Interpolation, and linear/nonlinear least-squares fitting, EVD/SVD, and FFT.

- https://www.alglib.net

Blaze: An open-source, high-performance C++ math library for dense and sparse arithmetic. With its state-of-the-art *Smart Expression Template* implementation **Blaze** combines the elegance and ease of use of a domain-specific language with HPC-grade performance, making it one of the most intuitive and fastest C++ math libraries available.

- https://bitbucket.org/blaze-lib/blaze/src/master/

Fastor: is a high-performance tensor (fixed multi-dimensional array) library for modern C++. It promises bare metal performance for small matrix/tensor multiplications, contractions, and tensor factorizations [LU, QR etc.]. **Compile time operation minimization** such as graph optimization, greedy matrix-chain products, and nearly symbolic manipulations to reduce the complexity of evaluation of BLAS or non-BLAS type expressions by orders of magnitude.
Explicit and configurable **SIMD vectorization** and support for **FPGAs, micro-controllers, and embedded systems**.
Light weight header-only library with no external dependencies offering **fast compilation times.**
Well-tested on most compilers including GCC, Clang, Intel's ICC and MSVC

- https://github.com/romeric/Fastor

XTensor: A C++ library meant for numerical analysis with multi-dimensional array expressions. It provides an extensible expression system enabling lazy broadcasting. An API following the idioms of the C++ standard library tools to manipulate array expressions and build upon XTensor.
Containers are inspired by NumPy, the Python array programming library. Adaptors for existing data structures to be plugged into our expression system can easily be written. It can be used to process NumPy data structures in place using Python's buffer protocol. Similarly, we can operate on Julia and R arrays. For more details on the NumPy, Julia and R bindings, check out the xtensor-python, xtensor-julia and xtensor-r projects respectively. XTensor requires a modern C++ compiler supporting C++14.

- https://github.com/xtensor-stack/xtensor

Awesome C++: A curated list of awesome C/C++ frameworks, libraries, resources, and shiny things: https://cpp.libhunt.com/libs
In particular: C++ Math libraries

- https://cpp.libhunt.com/libs/math

Cppreference.com: C++ Numeric library for common mathematical functions.
- https://en.cppreference.com/w/cpp/numeric/math