

Efficient Data Layout For Mass Spectrometry Data (OpenMS)

18.04.2022

Amit Kumar Mishra

2nd Year B.Tech(ECE)

Faculty of Engineering and Technology, JMI

Okhla, New Delhi - 110025



Overview


Adapting core data structure of OpenMS from AOS(Array of Structs) to SOA(Struct of Arrays) using zero cost abstraction.

Personal Background

I am Amit Kumar Mishra, 2nd year B. Tech (Electronics and Communication Engineering) student at Faculty of engineering and technology, Jamia Millia Islamia.

I like programming because of its nature, how it enables us to create what we want to get done, gives exactly what we have delivered, no more no less.

In my free time, I enjoy tinkering with linux and customising (or ricing) it. The open source nature of linux and distros on top of it have amazed me so much when you don't get



anything bloated and can have the best performance of your hardware which can be felt on specially low specs. The whole idea of open source creates a feeling of gratefulness and creates an appeal to give back or carry forward this tradition of knowledge sharing with whomever who keeps the interest in it and wants to know more about it.

Helping people from the narrow and deep region of knowledge in wide possibilities to those who may be putting their share from some completely different region of specialisation. The resulting trust between communities from different fields can help in moving together.

I know c++, python and javascript. Have explored a wide range of technologies and frameworks. I know git and bash-scripting.

About the Project

Abstract

Efficiency of algorithms can't be overlooked especially when there are really huge computation demands. AOS data layout, however being readable, adds a vectorization efficiency overhead as multiple data have to be moved with a single instruction. On the other hand, using SOA data layout will give multiple instances of data sitting adjacently on memory and single instruction for single data movement which will reduce the load.

This problem raises two choices - either readability (offered by the object oriented design of AOS) or performance (SOA).

And even if we gave up on readability for the sake of performance that will demand lots of refactorisation and almost rewriting the whole code base with reimplementations of algorithms.

Modern C++ and libraries like Intel SDLT can offer a solution for it so that we don't have to rewrite or reimplement all the code but a small refactorisation can give the benefits of SOA especially on cache performance, hence, zero cost abstraction.

Why are we doing what we are doing? Why would we go to a data-oriented design approach?

The purpose of all the programs and the parts of the programs, is to transform data from one form to another. If the program or programmer doesn't understand the data then maybe we didn't understand the problem. Problems can be understood better by understanding the data transforms behind. Different data means different problems and different solutions will be required to solve it. Not understanding the cost of the problem may result in an inefficient solution and without understanding hardware, cost can't be

reasoned. Every small component has a finite range of performance characteristics. And only after considering those characteristics, a cost efficient solution can be created.

Benefits to the community

OpenMs is an open source framework for computational mass spectrometry. It deals with the development of **computational** methods for the automated analysis of **MS** data. It features a wide range of algorithms and data structures to process and analyse mass spectra. Creating an effective data layout will improve the performance and reduce the computational overhead.

Adapting the mass spectra of OpenMS to SOA data layout will make it resource efficient. Doing this with zero cost abstraction using modern C++ will allow for writing generic and efficient code regardless of real memory data layout underneath.

Problems

Doing it from scratch can cost a complete rewrite of the program and we never want to do that. Moreover, we may sometimes need to compare the performance difference between different implementations with changing data and structure of the program. Saving our energy to fulfil the required we would need to solve few problems

- i) To have API based access to AOS and SOA memory layouts which quickly allow switching between AOS and SOA.
- ii) To get access to iterators in a resulting non-standard container.
- iii) Dereferencing the iterator in such a way so that it behaves like an object and its data members or member functions can be accessed using the dot ('.') operator.
- iv) To have access to function calls which are optimised **by compilers** so that our abstraction is really zero cost (i.e. doesn't take any intervention for different layout access).

Solutions

i) To have API based access to AOS and SOA memory layouts which quickly allow switching between AOS and SOA.

Template-metaprogramming can help us in solving this problem. Two helper classes corresponding to both memory layouts can be created in order to generate container types suitable for AOS or SOA layout. Depending on the tag template argument, tuple of vectors or vector of tuples can be created.

To get specialised behaviour for specialised data types template specialisation would be helpful.

```
template <template <typename...> class Container, DataLayout TDataLayout,
        typename TItem>
```

ii) To get access to iterators in a resulting non-standard container.

A workaround for handling iterators in non-standard containers can be by creating `begin()` and `end()` methods which will return first and last elements of the container respectively. For example, consider a container, say `BaseContainer`, so iterators can be defined as following

```
iterator begin() { return iterator(this, 0); }
iterator end() { return iterator(this, size()); }
```

where `this` will be referring to `BaseContainer<TContainer, TDataLayout, TItem> *`

iii) Dereferencing the iterator in such a way so that it behaves like an object and its data members or member functions can be accessed using the dot ('.') operator.

An iterator would be required to define such that it behaves like a STL iterator with having reference to the container. This will allow us to get the operator which will statically dispatch the operations according to the container's data layout. C++17 guaranteed copy elision would play an important role by not instantiating any data layout at the declaration of operator and will be referencing only when dereference operator is called **inside Iterator class** where data layout is already predefined.

iv) To have access to function calls which are optimised by compilers so that our abstraction is really zero cost (i.e. doesn't take any intervention for different layout access).

To access one of the member functions of the concerned object, we might create a tuple each time and throw it right away. We may rely on compiler's flags like `-O3` for our abstraction to be zero cost. All the accesses should be optimised as a result of compiler optimizations.

Pre-GSOC Contributions to OpenMS

This was my first time with Qt Framework and to get myself familiarised with it and the repository, I wanted to get my hands dirty on it. I noticed several build warnings. Hence I created [multiple issues](#)¹ for them and was later assigned to fix them too myself. Most of them have been resolved with various [pull requests](#)² till now. To make myself more comfortable with OpenMS, I will read documentation for developers, wikis and go through code bases.

Timeline

Community Bonding Period (20 May - 12 Jun)

In this period I will increase familiarity with community, mentors and codebase. I will discuss the project implementation and will take their opinions and feedback on it. I will discuss benchmarking, performance benefits and other pros and cons mentors can see as long time contributors. Understanding how and where OpenMs is used will be another area of interest during that period.

A large part of time will be devoted to gaining experience understanding the implementation of switching memory layouts with zero cost abstraction. I will constantly communicate with mentors and will stay in touch with the OpenMS community and will make sure I clear my doubts regarding any issues I face at that time.

By the end of this period my aims are to:

1. Learn thoroughly about implementation of different memory layouts with zero cost abstraction.
2. Understand its applicability in OpenMS by deeply familiarising myself with its codebase.
3. Finalising which memory layouts will be most effective for different parts of the framework.

Coding Period (13 Jun)

1. Implementing helper classes for corresponding container types as a vector of tuples or a tuple of vectors.
2. Implementing static member functions inside the class to interact with containers.

¹ <https://github.com/OpenMS/OpenMS/issues?q=is%3Aissue+author%3AAmit0617+>

² <https://github.com/OpenMS/OpenMS/pulls?q=is%3Apr+author%3AAmit0617+>

- 
3. Implementing template specialisations for AOS.
 4. Implementing template specialisations for SOA with `std::reference_wrapper` decorator to have lvalue reference access to its elements.

Phase 1 Evaluation

(25 July - 29 July)

Work Period (25 July)

1. Implementing type alias using helper class.
2. Implementing iterator for non-standard containers.
3. Defining our concerned "object" for which different data layout is implemented.
4. Testing and verifying performances of both memory layouts using compiler optimisation.
5. Verifying whether the compiler optimises it with zero cost abstraction.

Phase 2 Evaluation

(5 Sept - 12 Sept)

GSoC Period Ends
