Jet Reconstruction with Julia

Atell-Yehor Krasnopolski
Kyiv National Taras Shevchenko University, Ukraine

June 2022

Mentors: Benedikt Hegner (CERN), Graeme A Stewart (CERN).

Project Description

Motivated by the increasing interest in Julia [1] as an alternative/additional language for high-energy physics (HEP), where C++ and Python are currently the main languages, this project is aimed at continuing this investigation by re-implementing some important algorithms for HEP in Julia in order to assess developer convenience and performance compared to standard implementations in C++. Ideally, this could prove that Julia offers the convenience features of Python, but the optimal runtime speed of C++ on the tasks related to the HEP field.

The Anti-k_t Algorithm

When a high-energy particle passes through detectors like ATLAS and CMS at CERN, it results in showers of secondary particles that form a cascade (a jet), which is then captured in the calorimeters of these experiments. It is extremely important to reconstruct the energy of the primary particle from the calorimeter measurements. Such a task is called jet reconstruction. [2] The anti-k_t algorithm [3] is a jet reconstruction algorithm which has been proven to be a crucial one. It is both IRC-safe and soft-resilient, those are useful properties that no previously invented algorithm has possessed simultaneously. That is why we are interested in implementing exactly this algorithm.

Project Goals

The task is to write the anti-k_t jet reconstruction algorithm in Julia. Of course, this includes implementing the necessary data structures and time assessments. To ensure correctness, a comparison with the FastJet C++ package [4, 5] (which is the golden standard) will be done **after** implementing the Julia version from scratch. The speeds of the two codes will then be benchmarked on different platforms, for both serial and multi-threaded running; the convenience for the developer and code maintainability will be assessed. During the initial development, however, no inspiration from the C++ implementation should be drawn, so there is no bias and as much gain from Julia as possible.

Proposed Timeline

Weeks 1-2

Understand the Anti-k_k jet reconstruction algorithm by actively reading the main paper [3] and other helpful HEP-related materials such as "Towards Jetography" by Gavin P. Salam [2]. Resolve all the newly aroused and previously highlighted questions by asking mentors. Get stronger intuition about the underlying physics. Try solving a small toy problem with this algorithm by hand (with a calculator) as well as using pseudo-code. These weeks are focused on theoretical explorations and possibly testing some ideas for the implementation.

Weeks 3-6

By the end of this period, there should be a working prototype (or even several versions/branches of it) in the GitHub repository. That is, there should be code that 1) defines all the necessary data structures to handle the data and describe the needed physics; 2) defines a Julia function with at least one method or any other callable object representing the anti-k_t algorithm which takes in the data (a collection of points in $\mathbb{R}^4$, representing 3-dimensional Cartesian coordinates

and energy) and outputs the clustered jets; 3) has a simple test that can later be used to compare the results and the execution time to the FastJet's ones.

Week 7

Compare Julia's and FastJet's outputs on the same tasks, not looking into the details of how FastJet works yet. Fix possible differences in the outputs of the C++ and Julia implementations if they are caused by errors in the code. "Polish" the code and make it more "Julia-like" (add vectorisation and code accelerators like `@inbounds`, `@fastmath`, etc, use functional programming techniques). Compare the polished version against FastJet once again and save the results.

Weeks 8-10

Start developing a version of the algorithm that is optimised to analyse larger problems with more particles (and create better tests to represent this type of problem). Read the FastJet code for the anti-k_t algorithm and look at the optimisation methods used there. Implement the relevant ones in Julia. Keep track of different branches, as it is likely that not all of FastJet's tricks will help us achieve better performance with Julia.

Weeks 11-12

Prepare the final version of the code. Write alternative aliases and interfaces for it and make improvements to usage convenience – ideally, the finalised implementation should utilise Julia's multiple dispatch paradigm and work equally for input data points represented as Julia's native vectors `Vector{T} where T<:AbstractFloat` and as specially designed structures like `LorentzVector` from the `LorentzVectorHEP.jl` [6]. Write documentation, restructure, and compile the Julia module. Assess the performance of the compiled module. Make the final comparison with FastJet on multiple tests and add visualisation.

References

[1] Marcel Stanitzki and Jan Strube. Performance of julia for high energy physics analyses. *Computing and Software for Big Science*, 5(1), apr 2021.

[2] Gavin P. Salam. Towards jetography. *The European Physical Journal C*, 67(3-4):637–686, may 2010.

[3] Matteo Cacciari, Gavin P Salam, and Gregory Soyez. The anti-ik/isubit/i/subjet clustering algorithm. *Journal of High Energy Physics*, 2008(04):063–063, apr 2008.

[4] Matteo Cacciari and Gavin P. Salam. Dispelling the jet-finder. *Physics Letters B*, 641(1):57–61, sep 2006.

[5] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. FastJet user manual. *The European Physical Journal C*, 72(3), mar 2012.

[6] Jerry Ling JuliaHEP. Lorentzvectorhep.jl github repository.