Accelerating Uproot with AwkwardForth

Aryan Roy **Mentor: Jim Pivarski**

CCE, B.Tech **Duration: 25th May 2022 to 23rd July, 2022**

Manipal Institute of Technology

Project Description

Reading data is an essential first step of any form of data analysis. This holds true for particle physics as well where a majority of the data is stored in the ROOT format. ROOT is an incredibly powerful open source data analysis framework which allows easy access to advanced analysis tools. The widespread use of ROOT has motivated the development of Uproot, a Python library that reads and writes data in the ROOT format. This fellowship will be concerned with accelerating the uproot package's ability to read ROOT files and make Awkward Arrays out of them.

The Uproot library can only read columnar data quickly, the other data types with a record-oriented layout are hundreds of times slower. One solution to this can be to generate sufficiently low level language code to cut down on the overhead time required for reading such record-oriented data. Currently, uproot generates Python code to read the data from ROOT files, which is inefficient. The current implementation generates code that can only deal with complex data as Python objects. To get Awkward Arrays, the library passes the Numpy objects through the "ak.from_iter" function to get an Awkward Array equivalent. This was a temporary feature that stuck around.

To overcome the shortcomings of the library, it will be modified to generate AwkwardForth code instead. AwkwardForth is a dialect of Forth, modified to facilitate easy deserialization of record-oriented data into its columnar counterpart. It will allow the library to read data in easily digestible forms, regardless of the original format. This would also enable the user to read the non-columnar data extremely fast when compared to the generated Python code.

The fellowship would involve understanding the mapping from ROOT C++ types into Awkward Arrays. Subsequently, the current solution will have to be studied in detail to create an AwkwardForth version. After the new code is being generated, extensive testing would have to be carried to ensure that the code correctly reads the data. Once all the previous steps are done, the current infrastructure will be dismantled and deprecated. A successful completion of all the steps mentioned here would result in a state-of-the-art ROOT format reader for Python.

Timeline

Week 1
Understanding the mapping from ROOT C++ types to Awkward Arrays. This would also include going over the current infrastructure and understanding how it deserializes ROOT files.

Week 2
The second week would involve coming up with the skeleton of the program. The Awkward Array library has awkward_forth methods, each of which will need a “form_and_forth” method. Initially these would be no-ops (raise NotImplementedError). As the fellowship progresses, I will figure out the scope of these methods along with the arguments that need to be passed to them. I will also come up with a “backdoor” option which would invoke some specific code instead of what normally happens. This would allow me to easily test the code without going through and breaking the existing workflows.

Week 3-6
Test driven development starts for the code that actually generates the AwkwardForth code. The development will start with relatively simpler cases such as std::vector<std::vector<float>> as described in arxiv paper. Starting the development from this point means that a direct comparison can be made with the performance as listed in the paper and also ensure that there are no mistakes.

Week 7
Once the test driven development has run its course and all the “form_and_forth” methods are filled, the process of removing the “backdoor” and other development infrastructure begins. Once that is done, I will start making the new methods the default implementation, rendering the old infrastructure “dead code”, i.e uncallable by any other function. At this stage I would go through the whole codebase of Uproot and remove any part of the code that has become “dead”

Week 8
The final week would involve writing the documentation for the new code. This could range from writing new tutorials, to writing comments in the codebase itself, depending on the nature of the new API.