UnROOT: an I/O library for the CERN ROOT file format written in Julia

Tamás Gál[1, 2], **Jerry (Jiahong) Ling**[3], **and Nick Amin**[4]

1 Erlangen Centre for Astroparticle Physics **2** Friedrich-Alexander-Universität Erlangen-Nürnberg **3** Harvard University **4** University of California, Santa Barbara

DOI: 10.21105/joss.0XXXX

Software

- Review
- Repository
- Archive

Editor: Editor Name

Submitted: 01 January XXXX
Published: 01 January XXXX

Summary

`UnROOT.jl` is a pure Julia implementation of CERN ROOT (Brun & Rademakers, 1997) files I/O (`.root`) that is fast, memory-efficient, and composes well with Julia's high-performance iteration, array, and multi-threading interfaces.

Statement of need

The High-Energy Physics (HEP) community has been troubled by the two-language problem for a long time. Often, physicists would start prototyping with a `Python` front-end which glues to a `C/C++/Fortran` back-end. Soon they will hit a task which is extremely hard to express in columnar (i.e. "vectorized") style, a type of problems which are normally tackled with libraries like `numpy` (Harris et al., 2020) or `pandas` (The pandas development team, 2020). This usually leads to either writing `C++` kernels and interface them with `Python`, or porting the prototype to `C++` and start to maintain two code bases including the wrapper code. Both options are engineering challenges for physicists who usually have no or little background in software engineering. Many steps of this process are critical, like identifying bottlenecks, creating an architecture which is both performant and maintainable at the same time while still being user-friendly and logically structured. Using a `Python` front-end and dancing across language barriers also hinders the ability to parallelize tasks that are conceptually trivial most of the time.

`UnROOT.jl` attempts to solve all of the above by choosing Julia, a high-performance language with simple and expressive syntax (Bezanson et al., 2017). Julia is designed to solve the two-language problem in general. This has been studied for HEP specifically as well (Stanitzki & Strube, 2021). Analysis software written in Julia can freely escape to a `for-loop` whenever vectorized-style processing is not flexible enough, without any performance degradation. At the same time, `UnROOT.jl` transparently supports multi-threading and multi-processing by simply providing data structures which are a subtype of `AbstractArray`, the built-in abstract type for array-like objects, which allows to interface with array-routines from other packages easily, thanks to multiple dispatch, one of the main features of Julia.

Features and Functionality

The `ROOT` dataformat is flexible and mostly self-descriptive. Users can define their own data structures (C++ classes) which derive from `ROOT` classes and serialise them into directories, trees and branches. The information about the deserialisation is written to the output file (therefore: self-descriptive) but there are some basic structures and constants needed to bootstrap the parsing process. One of the biggest advantages of the ROOT data format is the ability to store jagged structures like nested arrays of structs with different sub-array lengths. In high-energy physics, such structures are preferred to resemble e.g. particle interactions and detector responses as signals from different hardware components, combined into a tree of events.

UnROOT.jl understands the core structure of ROOT files, and is able to decompress and deserialize instances of the commonly used TH1, TH2, TDirectory, TTree etc. ROOT classes. All basic C++ types for TTree branches are supported as well, including their nested variants. Additionally, UnROOT.jl provides a way to hook into the deserialisation process of custom types where the automatic parsing fails. By the time of writing, UnROOT is already used successfully in the data analysis of the KM3NeT neutrino telescope (Adriá n-Martínez et al., 2016) and the CMS detector (Ehatäht, 2020).

Opening and loading a TTree lazily – i.e. without reading the whole data into memory – is simple:

```
julia> using UnROOT

julia> f = ROOTFile("test/samples/NanoAODv5_sample.root")
ROOTFile with 2 entries and 21 streamers.
test/samples/NanoAODv5_sample.root
   Events
      "run"
      "luminosityBlock"
      "event"
      "HTXS_Higgs_pt"
      "HTXS_Higgs_y"
      ...

julia> mytree = LazyTree(f, "Events", ["Electron_dxy", "nMuon", r"Muon_(pt|eta)$"]
 Row    Electron_dxy       nMuon    Muon_eta           Muon_pt
        Vector{Float32}    UInt32   Vector{Float32}    Vector{Float32}

 1      [0.000371]         0        []                 []
 2      [-0.00982]         2        [0.53, 0.229]      [19.9, 15.3]
 3      []                 0        []                 []
 4      [-0.00157]         0        []                 []
        ...
```

As seen in the above example, the entries in the columns are multi-dimensional and jagged. The LazyTree object acts as a table which suports sequential or parallel iteration, selections and filtering based on ranges or masks, and operations on whole columns:

```
for event in mytree
    # ... Operate on event
end

Threads.@threads for event in mytree # multi-threading
    # ... Operate on event
end

mytree.Muon_pt # a column as a lazy vector of vectors
```

The Journal of Open Source Software

The `LazyTree` is designed as `<: AbstractArray` which makes it compose well with the rest of the Julia ecosystem. For example, syntactic loop fusion[1] or Query-style tabular manipulations provided by packages like `Query.jl`[2] without any additional code support just work out-of-the-box.

Comparison with existing software

This section focusses on the comparison with other existing ROOT I/O solutions in the Julia universe, however, one honorable mention is `uproot` (Pivarski et al., 2021), which is a purely Python-based ROOT I/O library and played (still plays) an important role for the development of `UnROOT.jl` as it is by the time of writing the most complete and best documented ROOT I/O implementation.

- `UpROOT.jl` is a wrapper for the aforementioned `uproot` Python package and uses `PyCall.jl`[3] as a bridge, which means that it relies on `Python` as a glue language. In addition to that, `uproot` itself utilises the C++ library `AwkwardArray` (Pivarski et al., 2018) to efficiently deal with jagged data in `ROOT` files. Most of the features of `uproot` are available in the Julia context, but there are intrinsic performance and usability drawbacks due to the three language architecture.
- `ROOT.jl`[4] is one of the oldest Julia `ROOT` packages. It uses C++ bindings to directly wrap the `ROOT` framework and therefore is not limited ot I/O. Unfortunately, the `Cxx.jl`[5] package which is used to generate the C++ glue code does not support Julia 1.4 or later. The multi-threaded features are also limited.

Conclusion

`UnROOT.jl` is an important package in high-energy physics and related scientific fields where the `ROOT` dataformat is established, since the ability to read and parse scientific data is certainly the first mandatory step to open the window to a programming language and its package ecosystem. `UnROOT.jl` has demonstrated tree processing speeds at the same level as the `C++ ROOT` framework in per-event iteration as well as the Python-based `uproot` library in chunked iteration.

References

Adriá n-Martínez, S., Ageron, M., Aharonian, F., Aiello, S., Albert, A., Ameli, F., Anassontzis, E., Andre, M., Androulakis, G., Anghinolfi, M., Anton, G., Ardid, M., Avgitas, T., Barbarino, G., Barbarito, E., Baret, B., Barrios-Martí, J., Belhorma, B., Belias, A., … Zúñiga, J. (2016). Letter of intent for KM3NeT 2.0. *Journal of Physics G: Nuclear and Particle Physics*, *43*(8), 084001. https://doi.org/10.1088/0954-3899/43/8/084001

Bezanson, Jeff., Edelman, Alan., Karpinski, Stefan., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

[1] https://julialang.org/blog/2017/01/moredots/
[2] https://github.com/queryverse/Query.jl
[3] https://github.com/JuliaPy/PyCall.jl
[4] https://github.com/JuliaHEP/ROOT.jl
[5] https://github.com/JuliaInterop/Cxx.jl

The Journal of Open Source Software

Brun, R., & Rademakers, F. (1997). ROOT: An object oriented data analysis framework. *Nucl. Instrum. Meth. A*, *389*, 81–86. https://doi.org/10.1016/S0168-9002(97)00048-X

Ehatäht, K. (2020). NANOAOD: a new compact event data format in CMS. *EPJ Web Conf.*, *245*, 06002. https://doi.org/10.1051/epjconf/202024506002

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Pivarski, J., Osborne, I., Ifrim, I., Schreiner, H., Hollands, A., Biswas, A., Das, P., Roy Choudhury, S., & Smith, N. (2018). *Awkward array* (Version 1.9.0rc4) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.6522027

Pivarski, J., Schreiner, H., Smith, N., Burr, C., Kalinkin, D., Stark, G., Hartmann, N., Davis, D., O'Neil, R., Novak, A., Greiner, B., Stanislaus, B., ChristopheRappold, Deaconu, C., Cervenkov, D., Rübenach, J., Bendavid, J., Lieret, K., Peresano, M., … Held, A. (2021). *Scikit-hep/uproot4: 4.1.3* (Version 4.1.3) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.5539722

Stanitzki, M., & Strube, J. (2021). Performance of julia for high energy physics analyses. *Computing and Software for Big Science*, *5*(1), 1–11.

The pandas development team. (2020). *Pandas* (latest) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3509134