Experience with julia in high-energy physics

Joosep Pata

joosep.pata@cern.ch, @jpata
ETH Zurich, CMS experiment
July 18, 2016 @ DIANA meeting

My background

- Julia user since late 2013, currently inactive (C++ & Python due to group constraints)
- **mostly in trenches of CMS/LHC physics analysis** (Higgs searches) & object identification (machine learning for b-tagging)
- Some HPC & distributed computing work with the Swiss National Supercomputing Centre
- Author of ROOT.jl, first try at interfacing ROOT & julia, some minor HEP packages for julia
- Advocating julia:
  - most recently at the ROOT Users Workshop ‘15 (got ROOT t-shirt for best talk)
  - CMS computing meeting
  - at ETH Zurich: astrophysics and particle physics groups
  - among other LHC PhD students & analysis folks with a notebook

ROOT + julia

- started ROOT.jl in early 2014 in order to read my analysis n-tuples (pure binary data, no schema evolution)
- observation: we mostly use only a few classes from ROOT: I/O (TFile, TTree), histograms (TH*)
- julia ccall to C shims around ROOT C++ classes, blind void pointers to stack objects
- autogenerated shims using Clang.jl from ROOT headers
- No type hierarchy, difficult to extend, no templating
- ROOT6 bundles an old LLVM that ~~clashes with~~ used to clash with julia LLVM without recompiling the julia interpreter
- Baseline in order to provide a julia layer on top of usual HEP tasks
- Tests of base functionality passing with ROOT6, used for analysis in arXiv: 1511.02138

Writing a TTree

```
using ROOT

tf = TFile("test.root", "RECREATE")
ttree = TTree("my_tree", "My Tree”)

x = Float64[0.0]
px = convert(Ptr{Void}, pointer(x))
br = Branch(ttree, "x", px, "x/D")

nevents = 100

for i=1:nevents
    x[1] = i
    Fill(ttree)
end

Write(tf)
Close(tf)
```

Reading a TTree

```
tf = TFile("test.root")
ttree = Get(tf, "my_tree")

nevents = GetEntries(ttree)

br = GetBranch(ttree, “x")
x = Float64[0.0]
px = convert(Ptr{Void}, pointer(x))
SetAddress(br, convert(Ptr{Void}, px))

total = 0.0

for i=1:nevents
    GetEntry(ttree, i-1)
    total += x[1]
end
```

ROOT histograms

```
hi = TH1D("my_hist", "My Hist", Int32(10), -30.0, 30.0)

n=1000000

for i=1:n
     Fill(hi, randn())
end

#need to specify also the final edge of the last bin
bx = Float64[0.0, 1.0, 2.0]
by = Float64[0.0, 1.0]

hi2 = TH2D("my_hist2", "My Hist", Int32(2), pointer(bx), Int32(1), pointer(by))
Fill(hi2, 0.0, 0.0, 0.5)
```

julia + ROOT.jl can be used to write code that “looks familiar to python”

via Cxx.jl

- using julia v0.5.0-dev+5453, Cxx 68f295bc, libLLVM-3.7.1
- Enable RTTI data structures [Cxx.jl#104]
- Load vanilla ROOT v6 libaries, headers
- Ignore ROOT LLVM warnings about symbol clashes (still relevant?)
- Finally able to call ROOT+Cling directly with vanilla julia+Cxx!
- Two parallel C++ interpreters (Cling vs Cxx) with competing state: how far can it go?

just got this working today!

```
using Cxx
const ROOT_PATH = "/Users/joosep/Documents/root-build/"
addHeaderDir(joinpath(ROOT_PATH, "include"), kind=C_System)
for lib in ["Core", "RIO", "Hist"]
    Libdl.dlopen("$ROOT_PATH/lib$lib.so", Libdl.RTLD_GLOBAL)
end

cxx"""#include <TFile.h> """
cxx"""#include <TH1D.h> """

tf = @cxxnew TFile(pointer("asd.root"), pointer("RECREATE"))
th = @cxxnew TH1D(pointer("h"), pointer("h"), 100, -1.0, 1.0)

for i=1:100000
    @cxx th->Fill(randn())
end

@cxx tf->Write()
@cxx tf->Close()
```

One could now wrap ROOT in a standard way with Cxx.jl!

Julia interfaces to ROOT

- Example of workflow of using julia in HEP

Derive a histogram with errorbars from StatsBase AbstractHistogram

```
using StatsBase

type ErrorHistogram{T<:Real, N, E} <: AbstractHistogram{T, N, E}
    edges::E
    weights::Array{T,N}
    weights_sq::Array{T,N}
    closed::Symbol
    …
end

Hist1D = ErrorHistogram{Float64, 1, Tuple{Vector{Float64}, }}
```

jpata/Histograms.jl

fill, convert to ROOT, store or share

```
using Histograms

h1 = ErrorHistogram(linspace(0, 1, 11))
for i=1:n
    push!(h1, randn())
end

using Histograms

tf = ROOT.TFile("tfile.root", "RECREATE")
th::TH1D = to_root(h, “hist1")
Write(th)
Close(tf)

hd = load_hists_from_file("tfile.root")
h = hd["hist1"]
```

jpata/ROOTHistograms.jl

CMSSW.jl

- CMSSW is a complex C++ & python ecosystem for the whole CMS online & offline software

- Crucially, stores and handles physics objects with schema (e.g. pat::Jet, but also std::vector<float>) in ROOT files using ROOTs class evolution mechanism in EDM (Event Data Model) files

- use explicit C shims to get and copy std:: class data from EDM

```
const mu_eta = Source(
    InputTag(:goodSignalMuonsNTupleProducer, :Eta, :STPOLSEL2), Handle(Vector{Cfloat})
)

const cos_theta = Source(
    InputTag(:cosTheta, :cosThetaLightJet, :STPOLSEL2), Handle(Float64)
)
…
ev = Events(testfile)
#Move to the i-th event (! means that the parameters are modified in-place, as is the case for Events)
to!(ev, 100)

mueta::Vector{Cfloat} = ev[mu_eta]
ct::Float64 = ev[cos_theta]
```

Opening up HEP to julia

- On top of language, how to benefit from julia ecosystem (e.g. JuliaStats)?
- Most HEP codes are built on an event loop with row-by-row aggregation of statistics (loop & fill)
- JuliaStats etc seem more based on in-memory column-based operations
- Need layers on ROOT (e.g. DataFrame-esque TTree) or new Julia packages (e.g. array-backed histograms with weights) **and** appropriate Abstract interfaces in julia-world
- Conversion semantics between julia & ROOT objects: Histogram, DataFrame
- All are straightforward to develop but need a beachhead effort

Wishlist

- **~stable distribution of julia 0.5 with Cxx.jl, Gallium.jl easily usable (without containers) on SLC6-7 machines**
  - possibly in /cvmfs distributed file system via CERN or the experiments
- hiding LLVM symbols in julia [#12644] to prevent clashes with ROOT & Cling: *maybe no longer relevant post LLVM3.6?*
- **Cxx.jl compatible with ROOT** (at least I/O part), possibly via Keno's CxxROOT.jl: *if fine with two mismatched C++ interpreters, proof-of-concept already working*
- new ROOT.jl package using Cxx.jl tools, no C shims: needs a few people
- Base HEP functionality in julia (Lorentz vectors, weighted n-dimensional histograms) natively or via sanitized ROOT wrappers: community effort instead of my naive implementations.

Observations

- Overall, significant interest in trying julia from LHC people who need to get work done: PhD students & professors alike
- Needs to be relatively stable (month-to-month) and work out of the box on conventional HEP stack to get traction
- Some quirks (e.g. 1-based indexing) need explaining, but rationality prevails
- If julia was in /cvmfs, all LHC physicists would be able to evaluate and poke around
- Python only now entering adoption phase in offline analysis (write once, run once kind of code; time uncritical)
- ROOT-folks (Axel Naumann, Pere Mato Vila) seem mostly open to another kid on the block

Summary

- ROOT & julia runs, but needs some package work on top of a stable-ish Cxx.jl
- there is lots of interest and potential at the LHC for dynamic computing (analysis switching to python+ROOT)
- DIANA-HEP seems ideally positioned to establish julia use in HEP
- I'm mostly busy with my PhD and actual analysis for the next 1.5 years, little real time to invest in packages