Reducing Disk Footprint and I/O Load in GlueX Software

Jon Zarling

7/20/21

Motivation

- GlueX builds against ROOT 6.08.00
  - Released about 4.5 years ago
  - A number of improvements available since then
- Storage in data trees
  - TLorentzVectors are highly inefficient
- Compression + I/O time:
  - LZ4 reads through trees much faster
  - ROOT 6.24.00 or greater

Dataset Used For Benchmarks

- Data trees for $\eta \rightarrow 3\pi^0$
- Spring 2018 ver14
- 1.6 TB initial size

Disk Footprint: 10,000 events

Look at the disk footprint from running DSelector and saving “flat” tree output

Selection	Size (MB)	Percent From Previous Step	Percent of Initial
All events	148	100%	100%
dSaveDefaultFlatBranches=false • save basics only: event, run, beam, 4-vectors, invariant masses • Skips: tracking info, vertices, other subdetector info	94	64%	64%
Save 4-vectors as doubles instead of TLorentzVectors	45	48%	30%
Cut on $\chi^2/\mathrm{NDF} < 10$	0.756	17%	0.51%
Save only unique 6γ combos (i.e. stop worrying about how they arrange into 3 π^0s)	0.528	70%	0.36%

Compression Comparison

Run a DSelector to bring to reasonable file size

- Cut on $\chi^2/\mathrm{NDF} < 10$
- Tighter π^0 cuts
- …

- ZLIB (ROOT 6.08.00 standard): 9.39 GB
- LZ4 (ROOT 6.24.00 standard): 11.08 GB
- No compression at all: 15.5 GB

- Why did ROOT change to be less disk-space efficient?
  - I/O speed improves by a large factor

I/O Tests

- 37.7 million entries
  - (recall for flat trees 1 entry = 1 combo)
- Compare C++ ROOT for different compression and python alternatives

uproot: python-based alternative

Uproot:

- Standalone module for parsing ROOT format files
- NO ROOT dependency!
- Places

C++ ROOT code

```
TFile* f_in = new TFile(str_infile);
f_in->cd();
TTree* t_in = (TTree*)f_in->Get("eta_3pi0_Tree");

Double_t eta_mass_kin=0;                t_in->SetBranchAddress("eta_mass_kin",&eta_mass_kin);
Double_t accidweight=0;                 t_in->SetBranchAddress("accidweight",&accidweight);

Long64_t nentries = t_in->GetEntries();
TH1F* h_eta_mass_kin = new TH1F("h_eta_mass_kin","h_eta_mass_kin",1000,0.3,0.9);

for(Long64_t i =0; i<nentries; ++i) {
    if(i%10000==0) cout << "i: " << i << endl;
    t_in->GetEntry(i);
    h_eta_mass_kin->Fill(eta_mass_kin,accidweight);
}

TFile* outfile = new TFile( "ROOT_output.root", "RECREATE" );
outfile->cd();
h_eta_mass_kin->Write();
outfile->Close();
```

uproot

```
file = uproot.open(args[0])
eta_mass_kin_np = file["eta_3pi0_Tree/eta_mass_kin"].array(library="np")
accidweight_np  = file["eta_3pi0_Tree/accidweight" ].array(library="np")

h_eta_mass_kin = TH1F("h_eta_mass_kin","h_eta_mass_kin",1000,0.3,0.9)
h_eta_mass_kin.FillN(len(eta_mass_kin_np),eta_mass_kin_np,accidweight_np)

f = TFile.Open("uproot_output.root","RECREATE")
f.cd()
h_eta_mass_kin.Write()
f.Close()
```

I/O Comparisons

Run from ifarm1901
User time from /bin/usr/time

Compression Type	ROOT C++ compiled*	uproot
LZ4	106 s	3.28 s
ZLIB	158 s	9.11 s
uncompressed	96 s	3.19 s

*ROOT code precompiled – time listed does not include compilation time

Conclusion

- Either ROOT::Math::LorentzVector or four doubles could reduce our footprint by about 50%
  - ROOT::Math:: LorentzVector could probably to wrapped into DLorentzVector (replacing TLorentzVector as currently implemented)
- Fiducial kinfit cuts could reduce even more (but might need analyzer input)
- Using LZ4 compression (ROOT 6.24.00+) increases size by 15-20%
  - But can parse 1.5-2× faster
- Working on a wrapper for uproot in flat tree format, stay tuned!