Simplified Columnar File Conversions with:

hep▸convert

Zoë Bilodeau[1], Jim Pivarski[1]

[1]Princeton University

PRINCETON UNIVERSITY

Support for this work was provided by NSF cooperative agreements OAC-1836650 and PHY-2323298 (IRIS-HEP).

Problem: Time Spent on Columnar File Conversions

- Unnecessary time and energy from physicists to convert between file formats
- Even basic conversions require multiple lines of code, multiple file I/O packages
  - There are a number of common modifications that take extra time
- Many users are writing very similar code

What is hepconvert?

- High-level Python converter between **ROOT**, **Parquet**, (and eventually) and **HDF5**

- Uses common I/O packages
  - Uproot
  - Awkward
  - h5py
  - Dask-awkward

Quick, Simple File Conversions

- Main goal of hepconvert is **convenience**
- Blocks of code -> single function call
  - One package
  - Memory management and compression handled
  - Parameters for customization
- User input oriented

Features of hepconvert:

- Features added at **user request**
- Converters between Parquet and ROOT
- Common file manipulations
  - Add/remove data
  - Hadd-like functionality
  - Change compression
  - Regroup data
- Address common issues
- CLI

Memory Management: Batches

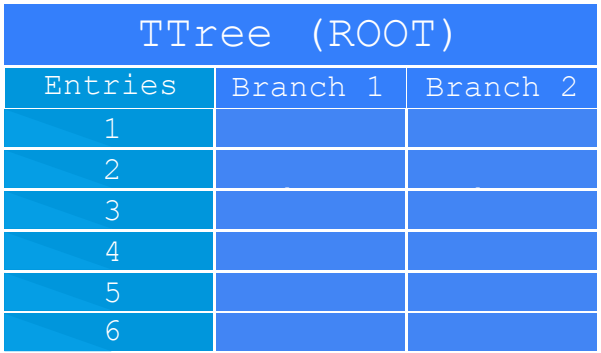
- For large files, it is necessary to read and write data in batches
- Can take time depending on file structure and I/O package;
  - Each “batch” is a different structure
  - Always require multiple lines of code/loops

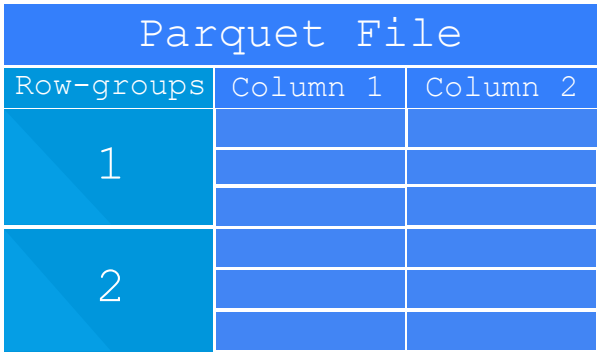
TTree (ROOT)		
Entries	Branch 1	Branch 2
1		
2		
3		
4		
5		
6		

Parquet File		
Row-groups	Column 1	Column 2
1		
2		

Memory Management: Batches

- Each hepconvert function automatically reads and writes in batches
  - (with the exception of add_histograms)
  - ROOT files over > 100MB
  - Parquet files with > 1 row-group
- Can choose step size when reading ROOT files
  - Entry size: 100
  - Data size: “100MB”





Work with ROOT files:

- Pure Python; users don't need ROOT
- Writing capabilities of Uproot
  - Currently works with **flat TTrees, NanoAOD-like** files
  - parquet_to_root() puts data in one TTree
- When possible, groups branches to avoid duplicate counters

Note

The small but growing list of data types can be written as TTrees is:

- dict of NumPy arrays (flat, multidimensional, and/or structured), Awkward Arrays containing one level of variable-length lists and/or one level of records, or a Pandas DataFrame with a numeric index
- a single NumPy structured array (one level deep)
- a single Awkward Array containing one level of variable-length lists and/or one level of records
- a single Pandas DataFrame with a numeric index

Parquet to ROOT

- One Parquet file -> one TTree
  - Soon adding merge_parquet; could merge data from multiple Parquet files to one TTree
- Writing capabilities of Awkward Array
  - Compression settings and many other options available

Parquet file to ROOT file:

```
>>> hepconvert.root_to_parquet("out_file.parquet", "in_file.root")
```

ROOT to Parquet

- One Parquet file -> one TTree
  - Soon adding merge_parquet; could merge data from multiple Parquet files to one TTree
- Writing capabilities of Awkward Array
  - Compression settings and many other options available

ROOT file to Parquet file:

```
>>> hepconvert.root_to_parquet("out_file.parquet", "in_file.root")
```

Awkward Feature: Iterative Writing to Parquet Files

- Re-implemented **ak.to_parquet_row_groups()**
- Writes data to parquet files in batches (row-groups)
- Pass data as an iterable over data rather than array

```
ak.to_parquet_row_groups(
    (i for i in f[tree].iterate(step_size=step_size,)),
)
```

Copy (and modify) ROOT Files

```
>>> hepconvert.copy_root("out_file.parquet", "in_file.root")
```

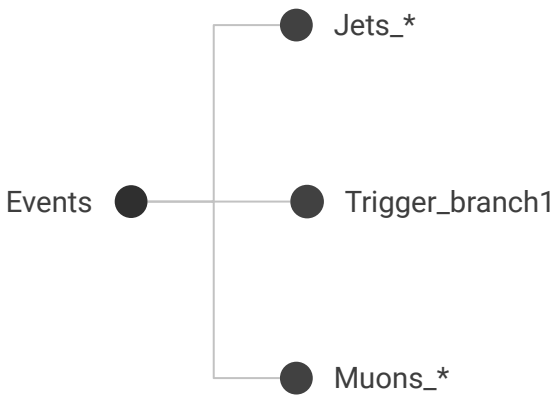
- Why include this? The additional features!
  - Automatically groups branches to avoid duplicate counter branches when writing with Uproot
    - Instead of manually choosing and grouping branches with ak.zip()
  - Branch-skimming, TTree removal, Branch removal
    - Wildcarding supported
  - Can either write to a new file or return a writable uproot object in memory to then work with
  - Change compression type
  - Run from command-line

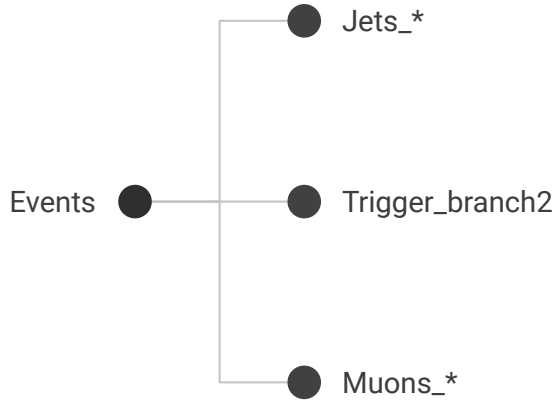
Merging TTrees and Histogram Summing (hadd-like)

- `add_histograms()`:
  - Sums contents of histograms in many files
  - Writes to a new file
- `merge_root()`:
  - Merges like TTrees, sums histograms from many files
  - Branch skimming, branch slimming, cuts, etc.
  - Customizable parameters similar to hadd
    - union, append, same_names
- Not dependent on ROOT!

Uproot Feature: Add Branches to an Existing TTree

- **Goal:**
  - `uproot.add_branches('tree', {branch1: data, branch2: data})`
- **Relation to hepconvert: merge_root()**
  - Addresses common issue with CMS data
    - Users wanted to merge NanoAOD files with mismatched branches
    - Can backfill with booleans
- Problems: making it inherently as robust/flexible as possible





Uproot Feature: Adding Branches to an Existing TTree

Current state:

- Can copy and write TBranches and TBranchElements of common data types
- Can copy data even if Uproot cannot write it (ex. a vector of vectors)
- File contents are never read into memory

Addressing Robustness:

- Rewrites TTree metadata
  - Can only handle most recent ROOT versions (generally after 2017)
- Copies old branches; copying process does not depend on branch type/content

Command-Line Interface

- Many functions are more useful in the command line
- All functions implemented
  - Most options work; check the docs!
- Implemented with Python Package Click
- Brief example

Jupyter Notebook Demo

Role of User Input

- Features added at **user request**
  - All features so far were at user request
- What relevant tasks are users spending time doing
- User interaction is necessary to make this a useful tool

Ideas or feedback?

https://github.com/scikit-hep/hepconvert/issues

Mattermost:

CMS Coffea Users channel

Slack:

PyHEP2024

IRIS-HEP: awkward-dask, awkward-uproot

Thank you!