Awkward plans

Jim Pivarski

Princeton University – IRIS-HEP

July 10, 2019

Context

Awkward-array is the end of a long chain of ideas about columnar analysis:

2016 Femtocode was originally envisioned as a combined language, columnar processor, and distributed processing system: a full-service package for physics.

- To be more realistic, I separated the language and columnar processing from the distributed processing, leaving the latter to others.

2017 Shredtypes → Quiver (for Arrow) → OAMap were iterations on expressing an abstract type system with an internal columnar representation.

- Uproot 2 had a "minimal columnar data type": a `JaggedArray` class.

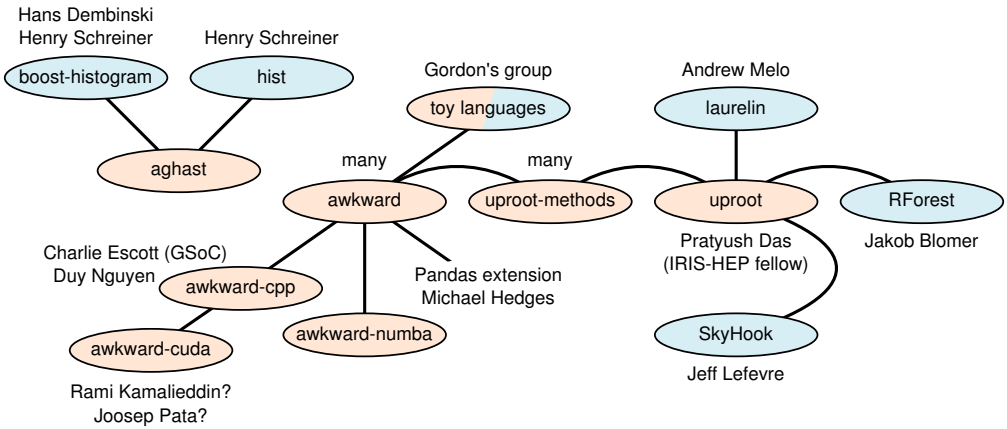
2018 Users found `JaggedArrays` useful despite the fact that it doesn't hide the columnar representation (this was a surprise to me).

- Awkward-array built the abstract type system to columnar representation in the other direction: bottom-up, from physical arrays to abstract types.

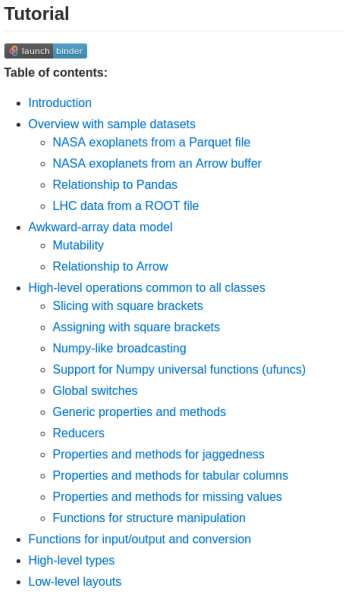
2019 Awkward-array is the first in this series to be widely used; I'm getting a lot of feedback from Coffea and uproot 3 users.

Up to now: one specification, many implementations

Four of my "ongoing projects" are different implementations of awkward, all to adhere to the `specification.adoc` (which is now out of date).



Just finished a homogenization/documentation campaign



Tutorial

launch binder

Table of contents:

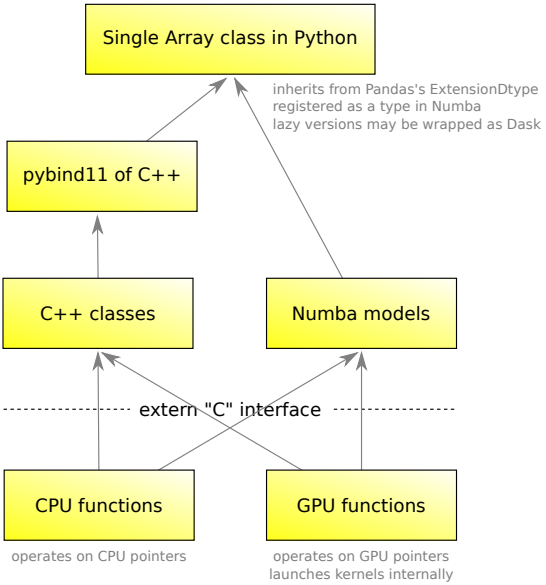
"Homogenization" because a common user complaint is that methods defined on one class (e.g. `JaggedArray`) don't work on another class (e.g. `MaskedArray` of `JaggedArray`).

While documenting (100 pages of examples), I ensured that a suite of "high-level" methods does something meaningful on every array class.

Problems

1. Writing the same method on 14 classes $\times$ 4 implementations is a problem for developer FTEs now and maintainance later.
   (Remember, I haven't touched awkward-numba since February!)
2. Users need a better separation between high-level and low-level.
   - Structural classes, such as `ChunkedArray` of `VirtualArrays` of `X` to make array `X` lazy-loading, should be hidden.
   - Named methods, like `x.cross(y)` meaning "cross-join," compete for the same namespace with column names and domain-specific methods, like `x.cross(y)` meaning "3-D cross product."
3. Too many features are opt-in. Numba integration will fail and Pandas integration will have a subtle performance bug if the user doesn't **`import awkward.numba`** and **`import awkward.pandas`** first.

Solution to #1: implement at most twice



Put all CPU implementations behind a stateless **`extern "C"`** interface.

- C++ classes (e.g. `JaggedArray`) provide structure and ownership rules.
- Numba (e.g. `JaggedArrayModel`) extensions associate Python classes in `@numba.jit` functions with the same implementations.
- Structure classes mirrored between C++ and Python by pybind11.
- Structure classes hidden inside `Array`.

Maybe later write **`extern "C"`** functions for GPU as well.

Solution to #2a: separation between high-level and low-level

Single `Array` class with a high-level type; narrow access to underlying structure.

```
>>> myarray
<Array [[1.1, 2.2, 3.3], [], [4.4, 5.5]] at 0x7fce7170bcf8>
>>> print(awkward.arraytype(array))    # datashape.readthedocs.io
3 * var * float64
>>> array.struct
JaggedArray([3, 99, 0], [5, 99, 3], [4.4, 5.5, 1.1, 2.2, 3.3])
```

The user only finds out that it is a `JaggedArray`, and not a lazy `JaggedArray`, by digging into the structure classes provided by pybind11.

This also hides the `starts=[3, 99, 0]` and `stops=[5, 99, 3]`.

Solution to #2b

It would be easier to implement and a cleaner API to put structure-manipulation operations in free-standing functions like

```
awkward.cross([x, y, z])        # size of output known at the beginning
```

rather than

```
x.cross(y).cross(z)             # have to make intermediate arrays
```

Apart from a single `struct` attribute and unnamed magic like `__add__`, `__array_ufunc__`, etc., the `Array` class can be kept clear for columns and domain-specific methods:

```
mydata.btag                     # field names as attributes
lorentzVectors.boost(others)    # physics methods
vector3Ds.cross(others)         # reduced ambiguity
```

In short, everything after the dot would be physics.

Solution to #3

Originally, I didn't want to automatically register Numba types and automatically make awkward arrays descend Pandas's `ExtensionDtype` because importing these libraries would add to the startup time, even if not used.

	slow computer	medium computer	fast computer
`import pandas`	1.5 sec	0.35 sec	0.25 sec
`import numba`	1.0 sec	0.35 sec	0.20 sec
`pandas` and `numpy`	2.0 sec	0.60 sec	0.35 sec

"Slow computer" is 1.1 GHz, "medium" is 2.6 GHz. Most users will be fine.

Drawbacks of this plan

- The original Numpy-based implementation and partial Numba-based one will have to be scrapped in favor of C++.
- A major change in API, even a good one, will require roll-out: awkward 0.x $\rightarrow$ awkward 1.0.
- After Charle Escott's GSoC ends, I'm the only developer.
- I estimate about 6 months for this transition.

Drawbacks of this plan

- The original Numpy-based implementation and partial Numba-based one will have to be scrapped in favor of C++.
  It served us well. The feedback we got from getting something in front of users quickly was invaluable.
- A major change in API, even a good one, will require roll-out: awkward 0.x $\rightarrow$ awkward 1.0.
  Since we envisioned multiple awkward implementations, uproot already has hooks for switching between alternatives.
- After Charle Escott's GSoC ends, I'm the only developer.
  It's my time to learn C++11. :)
- I estimate about 6 months for this transition.
  We already know it has a userbase.

Not-originally-intended advantage

By putting the *primary* implementation in C++, awkward-array could be directly used by C++ projects.

- Easier to emit awkward arrays from ATHENA/CMSSW: e.g. for ServiceX.
- Could become a standard way to wrap C++ libraries for Pythonic analysis: e.g. `JaggedArray` of 4-vectors → FastJet → `JaggedArray` of jets.
- Actually write production code with awkward-arrays, to transparently replace CPUs with GPUs? (Giuseppe Cerati has been looking into this.)

My projected timeline

July: two more tutorials: CoDaS-HEP and DPF (a total of 7 this year!)

August: Charlie's GSoC project finishes: learn as much as we can about pybind11's strengths and weaknesses

August: I finish the prototype SQL-for-events toy language

September: Strange Loop and working on fundamentals of awkward 1.0

October: Still working on awkward 1.0 and maybe go to PyHEP

November: CHEP and should be getting usable prototypes of awkward 1.0 to Coffea for testing

December: Should be transitioning PyPI packages awkward0 and awkward

— — — then it's the year 2020 — — —