Accelerating Uproot with Awkward Forth

Aryan Roy
Manipal Institute of Technology

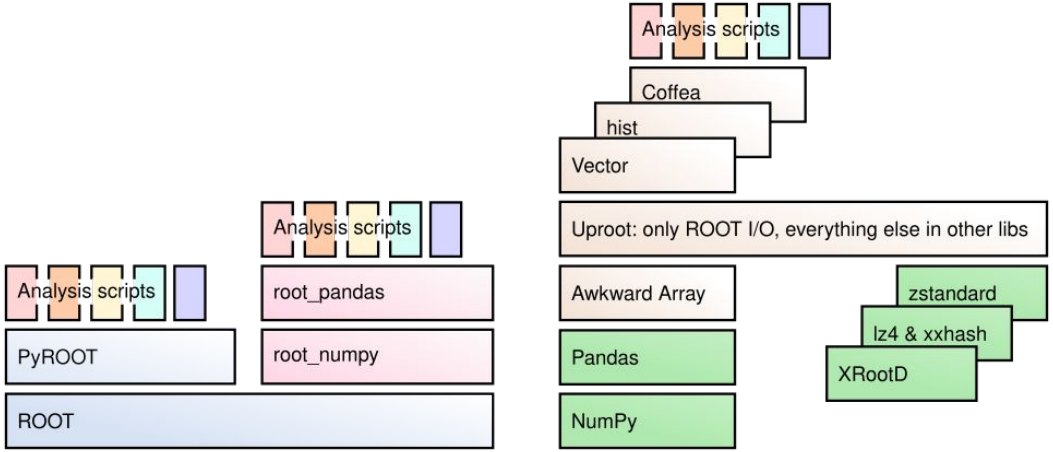
Dr. Jim Pivarski
Princeton University

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

PRINCETON UNIVERSITY

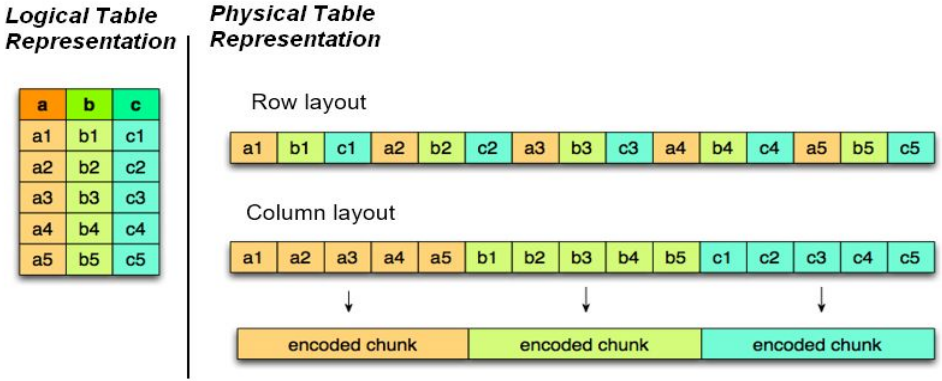
Uproot: ROOT I/O in Python

- Uproot is a library for reading and writing ROOT files in Python and NumPy.
- Uproot is pure Python, not dependent on ROOT.
- PyROOT and root_numpy depend on ROOT.



But, isn't Python slow?

- ROOT TTrees have column layout (numeric data and ragged arrays) and row layout (everything else) .
- Python implementation is slow, except for columnar data where it can cast a whole block of data as arrays, achieving the objective in O(1) time.
- For record-oriented data, we cannot do better than O(n), however, O(n) in a compiled language is much better than O(n) in Python.



Minimizing Dependencies

- Problem: Python is slow, but compilation toolchains (like Cling/LLVM) are heavy dependencies.
- Idea: interpreted languages can be fast if specialized.
- AwkwardForth [arXiv:2102.13516] is a Domain Specific Language (DSL) for file I/O into Awkward Arrays, based on Forth (an old programming language).

What AwkwardForth Looks Like

```
>>> import awkward as ak
>>> vm = ak.forth.ForthMachine32("""
: fibonacci    ( n -- nth-fibonacci-number )
  dup
  1 > if
    1- dup 1- recurse
    swap recurse
    +
  then
;
20 0 do
  i fibonacci
loop
""")
>>> vm.run()
>>> vm.stack
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]
```

Maximizing speed

In an informal study, it was found that Python took on average 900 ns per instruction, compared to 5 ns for AwkwardForth on the same machine.

Like Python and Java, AwkwardForth instructions are turned into bytecode to be interpreted by a VirtualMachine. But…

- Python checks types at runtime, AwkwardForth has only one type (integers).
- Python and Java follow object pointers at runtime, AwkwardForth has only one data structure (a stack of integers).
- AwkwardForth is a very minimal language.

I started this project by writing an Avro file reader with AwkwardForth.
It's 8× faster than fastavro Python package.

The Implementation

The AwkwardForth code generation is interwoven with the current Python implementation because the type-dependent code for ROOT TTrees already exists. Python and AwkwardForth generation alternate line by line to ensure that they can be maintained together.

This is **meta-programming:** Python code that generates AwkwardForth code.

```
length = cursor.field(chunk, _stl_container_size, context)
if helper_obj.is_forth():
    key = forth_obj.get_keys(1)
    form_key = f"node{key}-offsets"
    helper_obj.add_to_header(f"output node{key}-offsets int64\n")
    helper_obj.add_to_init(f"0 node{key}-offsets <- stack\n")
    helper_obj.add_to_pre(
        f"stream !I-> stack\n dup node{key}-offsets +<- stack\n"
    )
```

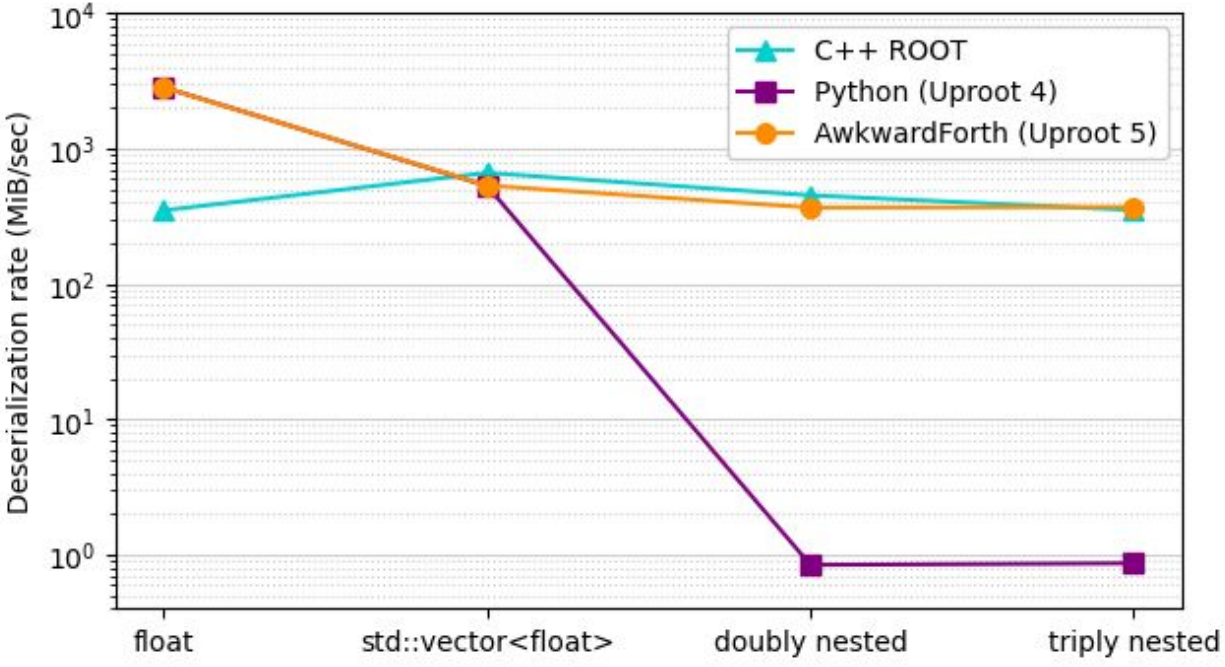
Then it gets more complicated…

ROOT specifies the data types in a file using TStreamerInfo, so even the Python code needs to be generated on the fly from this data.

This is **meta-metaprogramming:** Python that generates Python that generates AwkwardForth.

```
            read_members.append(
                """
        if context.get('speedbump', True):
            cursor.skip(1)
            if helper_obj.is_forth():
                helper_obj.add_to_pre('1 stream skip \\n')
""".strip(
                    "\n"
                )
            )
```

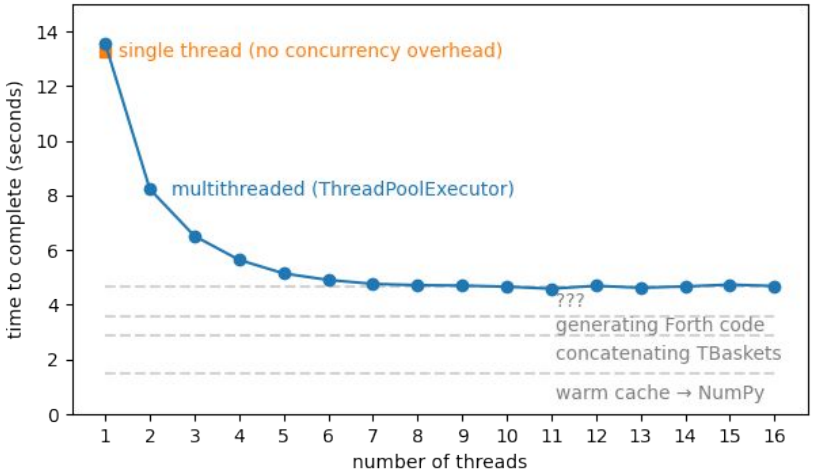
The Final Implementation

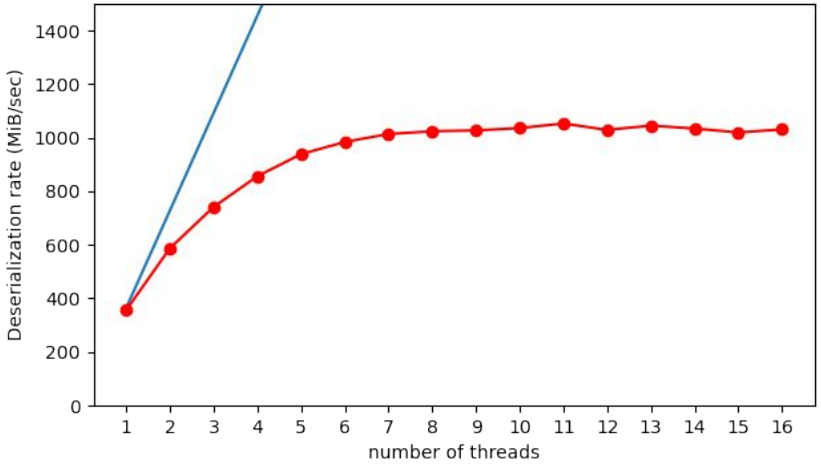


Record-oriented data types are 400× faster!

Tapping Into Multithreading

Uproot 4 (pure Python) doesn't scale at all due to the GIL.





Uproot 5 (AwkwardForth) scales but the end to end process has a serial part.

Conclusion

- We achieved the predicted performance (400× faster!) for an AwkwardForth based ROOT TTree reader.
- The new reader is extremely fast without having to install a compiler.
- The AwkwardForth code generation involves meta-programming and meta-meta-programming.
- Part of Uproot 5, due to be released in early December!