All about AwkwardForth

Jim Pivarski, Aryan Roy

Princeton University, Manipal Institute of Technology

March 6, 2023

Introduction: what is AwkwardForth?

AwkwardForth is an internal DSL within Awkward Array for accelerating sequential (non-columnar) data-ingest processes that must be discovered at runtime, avoiding a JIT-compilation toolchain as a dependency because Awkward Array is a foundational library.

Introduction: what is AwkwardForth?

AwkwardForth is an internal DSL within Awkward Array for accelerating sequential (non-columnar) data-ingest processes that must be discovered at runtime, avoiding a JIT-compilation toolchain as a dependency because Awkward Array is a foundational library.

It has a stable API because an external package, Uproot, uses it heavily.

Introduction: what is AwkwardForth?

AwkwardForth is an internal DSL within Awkward Array for accelerating sequential (non-columnar) data-ingest processes that must be discovered at runtime, avoiding a JIT-compilation toolchain as a dependency because Awkward Array is a foundational library.

It has a stable API because an external package, Uproot, uses it heavily.

Because of all the qualifications in its justification, I doubt other projects would ever need it.

Conclusions

Conclusions

- You probably don't need it.

BACKUP

Uproot-Awkward problem (for more-complex-than-jagged-arrays)

uproot

Awkward Array

Specializes in ROOT file (de)serialization.

Python-only.

Should know nothing about ROOT files.

Has a compiled C++ part.

Uproot-Awkward problem (for more-complex-than-jagged-arrays)

Specializes in ROOT file (de)serialization.

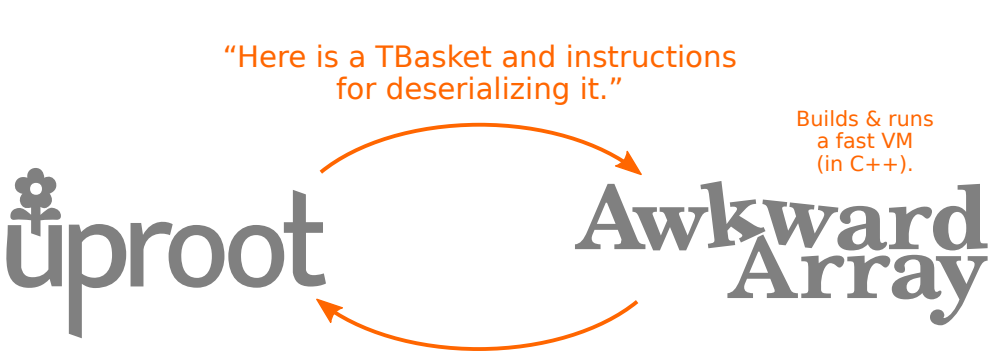
Python-only.

Should know nothing about ROOT files.

To build *fast* VMs.

Has a compiled C++ part.

Uproot-Awkward problem (for more-complex-than-jagged-arrays)



Specializes in ROOT file (de)serialization.

Python-only.

Should know nothing about ROOT files.

To build *fast* VMs. Has a compiled C++ part.

We need a deserialization language; why Forth?

Forth was promoted in the early 1980's as an alternative to compiled languages and (slow) BASIC. For example, it was the only language that could be written on the first Macintosh and produce applications that look and feel like compiled ones.

What Forth looks like (good for code generation!)

```
: fibonacci    ( pops n -- pushes nth-fibonacci-number )
  dup
  1 > if
    1- dup 1- fibonacci
    swap fibonacci
    +
  then
;

( pushes [0 1 1 2 3 5 8 13 21 34 55 89 144 233 377] onto the stack )
15 0 do
  i fibonacci
loop
```

Almost no syntax; each whitespace-separated word performs an action on a stack of integers. This stack is the only data. Apart from the ability to define new words and the lack of jumps, it's essentially an assembler for a virtual stack-based machine.

Adapted for data-ingest

```
0 offsets0 <- stack              ( offsets start at zero )
0 offsets1 <- stack
0 offsets2 <- stack

begin
  byte_offsets i-> stack         ( get a position from the byte offsets )
  6 + data seek                  ( seek to it plus a 6-byte header )
  data !i-> stack                ( get the std::vector size )
  dup offsets0 +<- stack         ( add it to the offsets )
  0 do                           ( and use it as the loop counter )
    data !i-> stack              ( same for the inner std::vector )
    dup offsets1 +<- stack
    0 do
      data !i-> stack            ( and the innermost std::vector )
      dup offsets2 +<- stack
      data #!f-> content         ( finally, the floating point values )
    loop
   loop
again                            ( ends with a "seek beyond" exception )
```

Summary of the argument

1. Uproot and Awkward need a way to talk to each other (neither are human).

Summary of the argument

1. Uproot and Awkward need a way to talk to each other (neither are human).
2. Uproot expresses a procedure to turn bytes into Awkward Arrays.

Summary of the argument

1. Uproot and Awkward need a way to talk to each other (neither are human).
2. Uproot expresses a procedure to turn bytes into Awkward Arrays.
3. Awkward needs to execute this procedure quickly.

Summary of the argument

1. Uproot and Awkward need a way to talk to each other (neither are human).
2. Uproot expresses a procedure to turn bytes into Awkward Arrays.
3. Awkward needs to execute this procedure quickly.
4. We don't consider bundling a JIT-compiler into Awkward to be an option.

Summary of the argument

1. Uproot and Awkward need a way to talk to each other (neither are human).
2. Uproot expresses a procedure to turn bytes into Awkward Arrays.
3. Awkward needs to execute this procedure quickly.
4. We don't consider bundling a JIT-compiler into Awkward to be an option.
5. Forth is an *interpreted* language on a virtual machine, like Python, but it's so stripped down that it doesn't have most of the things that make Python slow. AwkwardForth instruction $\sim$5 ns, Python instruction $\sim$900 ns.

Summary of the argument

1. Uproot and Awkward need a way to talk to each other (neither are human).
2. Uproot expresses a procedure to turn bytes into Awkward Arrays.
3. Awkward needs to execute this procedure quickly.
4. We don't consider bundling a JIT-compiler into Awkward to be an option.
5. Forth is an *interpreted* language on a virtual machine, like Python, but it's so stripped down that it doesn't have most of the things that make Python slow. AwkwardForth instruction $\sim$5 ns, Python instruction $\sim$900 ns.
6. Bundling AwkwardForth in Awkward Array adds a few kilobytes without any dependencies. (Sssh! No one needs to know!)

The language was implemented 2 years ago

Computer Science > Programming Languages

[Submitted on 24 Feb 2021]

AwkwardForth: accelerating Uproot with an internal DSL

Jim Pivarski, Ianna Osborne, Pratyush Das, David Lange, Peter Elmer

File formats for generic data structures, such as ROOT, Avro, and Parquet, pose a problem for deserialization: it must be fast, but its code depends on the type of the data structure, not known at compile-time. Just-in-time compilation can satisfy both constraints, but we propose a more portable solution: specialized virtual machines. AwkwardForth is a Forth-driven virtual machine for deserializing data into Awkward Arrays. As a language, it is not intended for humans to write, but it loosens the coupling between Uproot and Awkward Array. AwkwardForth programs for deserializing record-oriented formats (ROOT and Avro) are about as fast as C++ ROOT and 10–80× faster than fastavro. Columnar formats (simple TTrees, RNTuple, and Parquet) only require specialization to interpret metadata and are therefore faster with precompiled code.

Comments: 11 pages, 2 figures, submitted to the 25th International Conference on Computing in High Energy & Nuclear Physics
Subjects: **Programming Languages (cs.PL)**; High Energy Physics - Experiment (hep-ex)
Cite as: arXiv:2102.13516 **[cs.PL]**
(or arXiv:2102.13516v1 **[cs.PL]** for this version)
https://doi.org/10.48550/arXiv.2102.13516
Related DOI: https://doi.org/10.1051/epjconf/202125103002

Submission history

From: Jim Pivarski [view email]
[v1] Wed, 24 Feb 2021 18:49:10 UTC (157 KB)

Uproot was updated to use it last summer (version 5.0)

Preview

Using a DSL to read ROOT TTrees faster in Uproot

Aryan Roy and Jim Pivarski

Uproot reads ROOT TTrees using pure Python. For numerical and (singly) jagged arrays, this is fast because a whole block of data can be interpreted as an array without modifying the data. For other cases, such as arrays of std::vector<std::vector<float>>, numerical data are interleaved with structure, and the only way to deserialize them is with a sequential algorithm. When written in Python, such algorithms are very slow.

We solve this problem by writing the same logic in a language that can be executed quickly. AwkwardForth is a Domain Specific Language (DSL), based on Standard Forth with I/O extensions for making Awkward Arrays, and it can be interpreted as a fast virtual machine without requiring LLVM as a dependency. We generate code as late as possible to take advantage of optimization opportunities. All ROOT types previously implemented with Python have been converted to AwkwardForth. Double and triple-jagged arrays, for example, are 400x faster in AwkwardForth than in Python, with multithreaded scaling up to 1 second/GB because AwkwardForth releases the Python GIL. We also investigate the possibility of JIT-compiling the generated AwkwardForth code using LLVM to increase the performance gains. In this paper, we describe design aspects, performance studies, and future directions in accelerating Uproot with AwkwardForth.

Comments: 6 pages, 3 figures; submitted to ACAT 2022 proceedings

Categories

Categories

Primary: High Energy Physics - Experiment (hep-ex)
Cross lists (optional): Performance (cs.PF) Remove

---- Choose archive ---- --- Choose Subject Class ---- Add

The language is fully documented (so it's a stable API)

AwkwardForth virtual machines

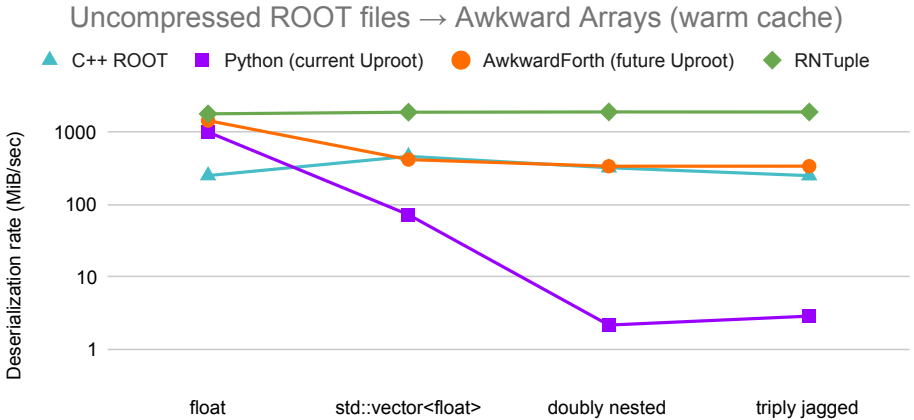
Introduction

AwkwardForth is a subset of standard Forth with some additional built-in words. It is a domain specific language for creating columnar Awkward Arrays from record-oriented data sources, especially for cases in which the deserialization procedure for the record-oriented data is not known until runtime. Typically, this is because the data has a type or schema that is discovered at runtime and that type determines how bytes of input are interpreted and in what order. This does not apply to columnar data sources, such as Apache Arrow, Parquet, or some ROOT data, such as numerical types (like `int` or `float`) and jagged arrays of numbers (like `std::vector<int>`). It does apply to record-oriented sources like ProtoBuf, Avro, and complex types in ROOT TTrees, such as `std::vector<std::vector<int>>` or unsplit classes. Note that ROOT's new RNTuple is entirely columnar.

The Easy Forth one-page tutorial is an excellent introduction to the idea of Forth. In a nutshell, whereas functional programming strives for pure functions with no side effects, Forth operations consist purely of side effects: every operation changes the state of the machine, whether the global stack of integers, global variables, or in the case of AwkwardForth, positions in input buffers and data written to output buffers. It has almost no syntax, less even than Lisp, in that it consists

Fixing Uproot's problem with non-columnar deserialization

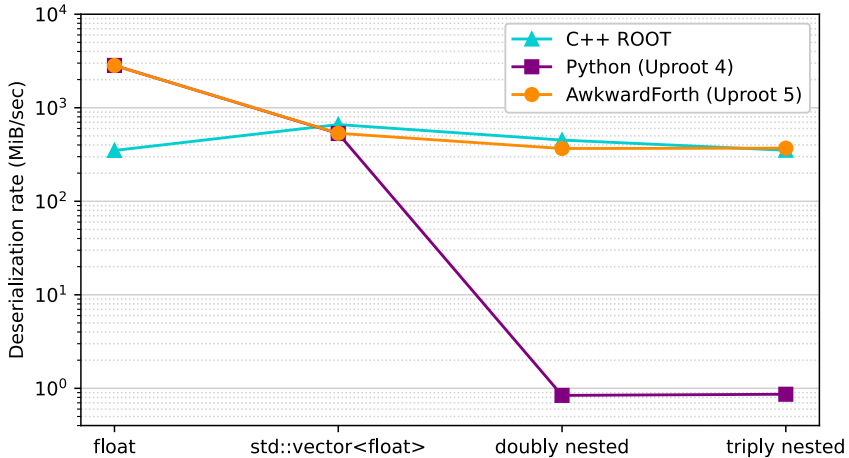
This is the *predicted* performance (2 years ago, in proof-of-concept demo).



Float and vector-of-float are columnar (and the purple point is 5× too low, a *mistake*).
Vector-of-vector-of-float and above are non-columnar.

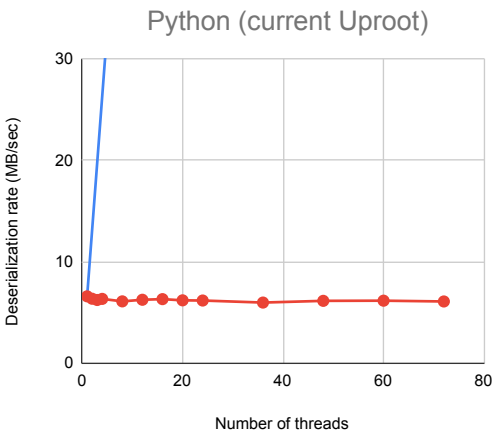
Fixing Uproot's problem with non-columnar deserialization

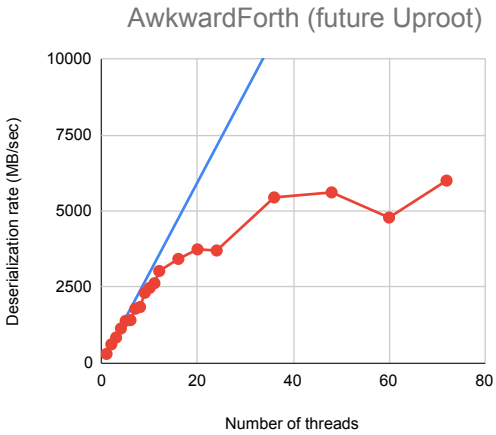
This is the *actual* performance (last summer, in Uproot itself).



Scaling: AwkwardForth releases the Python GIL

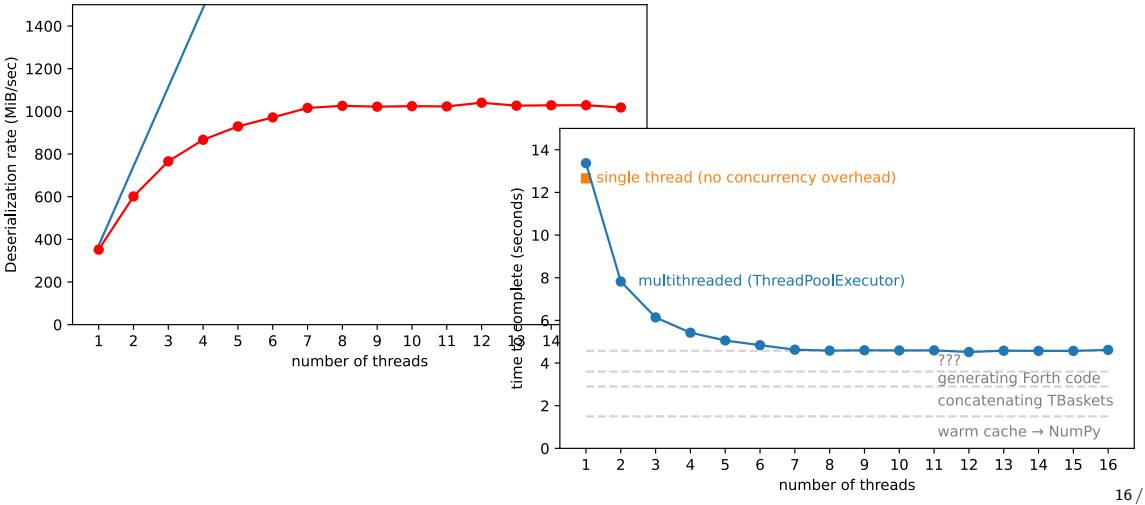
This is the *predicted* performance (2 years ago, in proof-of-concept demo).





Scaling: AwkwardForth releases the Python GIL

This is the *actual* performance (last summer, in Uproot itself).



What if we compile it anyway?

Some users have Numba (LLVM) installed.

What if we convert AwkwardForth $\rightarrow$ Numba and let Numba compile it?

What if we compile it anyway?

Some users have Numba (LLVM) installed.

What if we convert AwkwardForth $\rightarrow$ Numba and let Numba compile it?

Proof-of-concept implementation, given this test problem:

Compute: $(x + 1) * (x - 2) + 3$
for N iterations (i.e. no data flow).

What if we compile it anyway?

Some users have Numba (LLVM) installed.

What if we convert AwkwardForth $\rightarrow$ Numba and let Numba compile it?

Proof-of-concept implementation, given this test problem:

Compute: $(x + 1) * (x - 2) + 3$
for N iterations (i.e. no data flow).

```
dup 1 + swap 2 - * 3 +
```

What if we compile it anyway?

Some users have Numba (LLVM) installed.

What if we convert AwkwardForth → Numba and let Numba compile it?

Proof-of-concept implementation, given this test problem:

Compute: $(x + 1) * (x - 2) + 3$
for N iterations (i.e. no data flow).

```
dup 1 + swap 2 - * 3 +
```

```
lea     eax, [rdi+1]
sub     edi, 2
imul    eax, edi
add     eax, 3
```

What if we compile it anyway?

Some users have Numba (LLVM) installed.

What if we convert AwkwardForth → Numba and let Numba compile it?

Proof-of-concept implementation, given this test problem:

Compute: $(x + 1) * (x - 2) + 3$
for N iterations (i.e. no data flow).

```
dup 1 + swap 2 - * 3 +
```

```
lea     eax, [rdi+1]
sub     edi, 2
imul    eax, edi
add     eax, 3
```

∼400 ms to compile, then 10× faster,
but only for arithmetically intensive code.

What if we compile it anyway?

Some users have Numba (LLVM) installed.

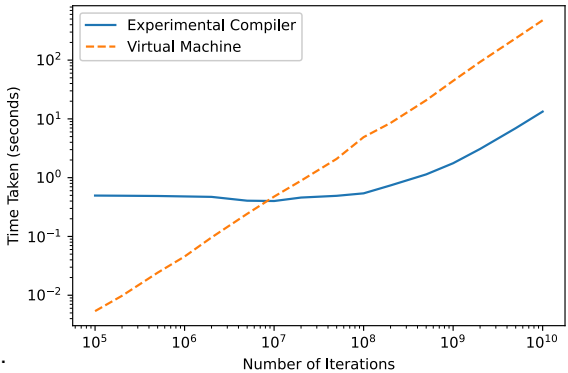
What if we convert AwkwardForth → Numba and let Numba compile it?

Proof-of-concept implementation, given this test problem:

Compute: $(x + 1) * (x - 2) + 3$
for N iterations (i.e. no data flow).

```
dup 1 + swap 2 - * 3 +
```

```
lea     eax, [rdi+1]
sub     edi, 2
imul    eax, edi
add     eax, 3
```



~400 ms to compile, then 10× faster,
but only for arithmetically intensive code.