Accelerating Uproot with Awkward Forth

Aryan Roy
IRIS-HEP Fellow
Manipal Institute of Technology

Dr. Jim Pivarski
Mentor
Princeton University
Uproot: ROOT I/O in Python

- Uproot is a library for reading and writing ROOT files in Python and NumPy.
- It differs from PyROOT and root_numpy in that it is independent of ROOT.
But, isn’t Python slow?

- ROOT TTrees have columnar data (numeric data and ragged arrays) and record-oriented data (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).
Maximizing speed

In a 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 follows 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, the gold standard for reading Avro files.
Record-oriented data types are 400× faster

AwkwardForth paper [arXiv:2102.13516] (prediction)

Uncompressed ROOT files → Awkward Arrays (warm cache)

The final implementation

![Graph showing deserialization rate for different data types and implementations]
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.
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.

```python
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')
    "\n"
)"
```
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.
- Merged into main!