Columnar Analysis at ATLAS

Nils Krumnack (Iowa State University) on behalf of the ATLAS computing activity
Introduction

• traditional analysis does one event at a time
  ‣ read event data for one event
  ‣ do all processing on that event
  ‣ fill histograms and n-tuples
  ‣ move on to the next event

• columnar analysis does an event loop per operation
  ‣ read a few variables/columns for many/all events
  ‣ do a single (or a couple) operation(s) on those variables
  ‣ can fill histograms, define new variables, etc.
  ‣ move on to the next operation

• will typically do multiple batches of events
  ‣ usually too many events to load all at once
  ‣ allows to split jobs up for a batch system
  ‣ for best performance need to run batches of operations at once
Data Layout

• for columnar analysis process events in batches
  ‣ batches presumed large enough to ignore per-batch overhead
  ‣ want calls to CP tools/algorithms to process entire batches

• data for each variable is a contiguous array in memory, e.g.
  ‣ pt for object 1, followed by pt for object 2…
  ‣ all pts for event 1, followed by all pts for event 2…

• objects are identified by index in array
  ‣ with later events having higher offsets

• event boundaries/offsets are available as a separate array

• **need to know needed columns ahead of time**
  ‣ mostly for loading data into memory

• assume I can pass all buffers by name from python to C++ tools
  ‣ both for input and output variables
Data Layout 2

- each variable has its own column
- can have columns for multiple object types (jets, muons…)
- all (used) columns loaded into memory at the same time
- all variables for one object type share an offset map
- there is a separate offset map for each object type
- all object types have the same events in the same order
CAna Advantages

- hope for a number of advantages
- actual advantages may vary
- still in the early prototyping stage

- easier to teach
- less boilerplate code
- analysis code less spread out/coupled
- automatically disable calculations not used for current study
- can be used with notebooks
- better code performance
- more productive in daily use
- integrate better with industry and ML tools
- experience more relevant for students leaving the field
CAAna at ATLAS

• can use CAAna at ATLAS already
  ▪ analysis generally done on user-generated n-tuples
  ▪ n-tuples incorporate corrections, systematics, etc.
  ▪ n-tuples normally readable without extra software
  ▪ can do analysis in either event-wise or columnar analysis

• not what this talk is about

• main goal: run directly on centrally produced PHYSLITE files
  ▪ incorporate object corrections (Jana Schaarschmidt’s talk)
  ▪ need "on-the-fly" calculation of systematics, scale factors, etc.
  ▪ code needs to be callable from CAAna frameworks

• requirement: performance competitive with n-tuple analysis
  ▪ otherwise users may continue using n-tuples
  ▪ also: don’t have infinite resources for analysis
  ▪ currently 1-2 orders of magnitude slower
CP Tools

• mentioned CP tools in earlier talk
  ‣ all analysis recommendations/recipes provided via CP tools
  ‣ very successful system for event-wise analysis
  ‣ CP tools shared with production and online

• can not break CP tool infrastructure
  ‣ want to maintain ability to do event-wise analysis
  ‣ should also not duplicate recommendation code
  ‣ can rewrite the tools and infrastructure though
  ‣ at the very least will need to wrap existing tools

• will need a fair rework of current CP tools for CAna
  ‣ need to get a lot faster to meet performance goals
  ‣ want to be as fast or faster than reading results from disk
  ‣ need across the board improvements for that
  ‣ already found substantial improvement potential for some tools
Types of Tools

• most tools involve a simple lookup from histogram
  ‣ used for scale factors and most systematics
  ‣ conceptually all very similar
  ‣ inherently fairly simple and fast

• most other tools involve fairly simple calculations
  ‣ e.g. object selection tools
  ‣ fairly easy to transcribe to most formalisms
  ‣ can incorporate some into PHYSLITE production

• a few tools involve fairly complex calculations
  ‣ e.g. full reconstruction of missing transverse energy
  ‣ generally need custom C++ implementations
  ‣ process full events instead of single objects
  ‣ harder to integrate with columnar analysis
Proposed Solutions

- several proposed solutions for columnar tools
  - change existing tool to work directly in RDataFrame (RDataFrame can handle ATLAS EDM)
  - wrap existing tools for columnar analysis (make EDM objects wrap data columns)
  - use ServiceX to create n-tuples on-the-fly (straightforward, prototype exists)
  - extend CP tools with separate columnar mode (see later slides)
  - use correctionlib [initially developed at CMS] (allows abstract description of object corrections)
  - rewrite tools in numpy
  - some combination of the above
Design Criteria

• main criteria for choosing solutions:
  ‣ support both columnar environments: uproot & RDataFrame
  ‣ also support event-wise analysis and production jobs
  ‣ support calculations that operate one-event-at-a-time
  ‣ need C++ implementations for at least some calculations
  ‣ match/exceed performance of reading from n-tuple
  ‣ minimize rewriting of tools
• most solutions fail on one or more points
  ‣ no solution is fulfilling all requirements
• can have different solutions for different classes of tools

• investigating two solutions right now:
  ‣ correctionlib: well-established at CMS for single object tools
  ‣ triple-use tools: custom solution to match above criteria
Triple-Use Tools

• current recommendations are provided via dual-use CP tools
  ‣ can be used in both the analysis and production framework
  ‣ uses conditional compilation to select one or the other
• plan to extend that to triple-use tools
  ‣ add columnar analysis as a third usage mode
  ‣ provide zero-overhead, vectorizable access to data columns

• design centers heavily around data handles
  ‣ handles are member objects for each CP tool
  ‣ all data access happens through handles
    • regular mode: handles access event data from whiteboard
    • columnar mode: data columns get loaded into handles
  ‣ handles also declare inputs/outputs
• objects identified via ObjectId objects
  ‣ regular mode: pointer to EDM objects
  ‣ columnar mode: simple index into data column
Mock Code

- simplified mock-up of what code would look like
- C++ side inspired by current tool design
- buffers managed purely on the python side
- python buffers get connected to C++ handles

```cpp
class MuonTool {
    ObjectHandle muons;
    ReadHandle pt;
    ReadHandle eta;
    WriteHandle output;

    void apply (MuonId muon) {
        output[muon] = calc (pt[muon], eta[muon]);
    }
    ...
};

import uproot as up
tool = MuonCPTool ()
for batch in up.iterate_batches(...):
    for var in tool.inputs() :
        tool.setBuffer (var, batch[var]...)
    for var in tool.outputs() :
        buffer = ...
        tool.setBuffer (var, buffer)
    tool.apply_batch()

# user code here
```
Anticipated Rollout

• mostly talked about plans today
  ‣ have a prototype for triple-use tools
• did benchmarking for correctionlib and triple-use tools, results:
  ‣ correctionlib is not faster than existing CP tool
  ‣ no overhead passing data from python into triple-use tools
• ultimate goal: run directly on PHYSLITE
  ‣ requires full suite of tools available
  ‣ requires all tools to be fast enough
  ‣ allows elimination of user n-tuples
• stepping stone: run on user n-tuples
  ‣ can be done one tool at a time
  ‣ allows to gain experience with simple tools first
  ‣ replace n-tuple variables with on-the-fly calculations
  ‣ immediate benefit: smaller user n-tuples
Summary & Outlook

• columnar analysis still in early stages at ATLAS
  ‣ can do CAna on current user n-tuples
  ‣ hope to replace n-tuple variables with "on-the-fly" calculations

• ultimate goal: running directly on PHYSLITE files
  ‣ would avoid the need for intermediate n-tuples
  ‣ aim to be ready for Run 4
  ‣ partial solutions would already be useful for n-tuple analysis

• main issue: how to do "on-the-fly" calculations
  ‣ need columnar interface for CP tools
  ‣ need substantial performance improvements
  ‣ need some rework of algorithms and PHYSLITE content
  ‣ prototyping and benchmarking multiple solutions
  ‣ still in the early stages