Femtocode: querying HEP data

Jim Pivarski

Princeton University – DIANA

April 17, 2017

Reminder of motivation

(I last talked about this on December 12.)

Query systems

In some industries, it is commonplace to query petabytes of data in real time, usually with SQL. (Ibis, Impala, Kudu, Drill, etc.)

Reminder of motivation

(I last talked about this on December 12.)

Query systems

In some industries, it is commonplace to query petabytes of data in real time, usually with SQL. (Ibis, Impala, Kudu, Drill, etc.)

For us, this would mean being able to do final analysis directly on collaboration-shared Analysis Object Datasets (AODs), without managing private skims.

Reminder of motivation

(I last talked about this on December 12.)

Query systems

In some industries, it is commonplace to query petabytes of data in real time, usually with SQL. (Ibis, Impala, Kudu, Drill, etc.)

For us, this would mean being able to do final analysis directly on collaboration-shared Analysis Object Datasets (AODs), without managing private skims.

However, these systems don't deal (well) with rich objects, like arbitrary-length lists of jets containing lists of tracks.

Reminder of motivation

(I last talked about this on December 12.)

Query systems

In some industries, it is commonplace to query petabytes of data in real time, usually with SQL. (Ibis, Impala, Kudu, Drill, etc.)

For us, this would mean being able to do final analysis directly on collaboration-shared Analysis Object Datasets (AODs), without managing private skims.

However, these systems don't deal (well) with rich objects, like arbitrary-length lists of jets containing lists of tracks.

Femtocode

I'm developing a query system whose performance permits real-time analysis, but is capable of complex manipulations, such as filtering tracks, picking pairs to compute invariant masses, etc.

Three interrelated parts

Language/compiler

- As familiar to the user as possible (objects, nested loops).
- But constrained to allow restructuring for fast execution (map/filter/reduce instead of for loops, total-functional).
- Extra-strength type system to eliminate runtime errors.

Execution engine

- Operate on contiguous columns of leaf data, rather than objects. Restructuring objects $\rightarrow$ changing arrays of integers.
- No memory allocation at runtime, vectorizable loops.
- JIT-compiled. CPU target for now, but GPU is possible.

Distributed server

- Vending machine: queries go in, histograms (etc.) come out.
- Referential transparency eliminates the need for "sessions."

Language: working example

```
pending = session.source("ZZ_13TeV_pythia8")
    .define(mumass = "0.105658")      # chain of operations on source
    .toPython(mass = """
muons.map(mu1 => muons.map({mu2 =>    # doubly-nested loop over muons
  p1x = mu1.pt * cos(mu1.phi);
  p1y = mu1.pt * sin(mu1.phi);        # shares scope with other steps
  p1z = mu1.pt * sinh(mu1.eta);       # in the chain (see "mumass")
  E1 = sqrt(p1x**2 + p1y**2 + p1z**2 + mumass**2);

  p2x = mu2.pt * cos(mu2.phi);
  p2y = mu2.pt * sin(mu2.phi);
  p2z = mu2.pt * sinh(mu2.eta);
  E2 = sqrt(p2x**2 + p2y**2 + p2z**2 + mumass**2);

  px = p1x + p2x; py = p1y + p2y;
  pz = p1z + p2z; E = E1 + E2;

  # "if" is required to avoid sqrt(-x)
  if E**2 - px**2 - py**2 - pz**2 >= 0:
    sqrt(E**2 - px**2 - py**2 - pz**2)
  else:
    None    # output type is nullable
}))
""").submit()                         # asynchronous submission to
final = pending.await()               # watch result accumulate
```

Yes, we see the Z peak.