CMS Feedback on RNTuple

Matti Kortelainen

RNTuple Format and Feature Assessment

6 November 2023

CMS data model

- Data products produced by CMSSW algorithms are serialized+stored using ROOT
  - Event, LuminosityBlock, and Run data are stored in separate TTrees
  - Also various framework metadata is stored (forming the “EDM format”)
- Main data tiers: RAW, AOD, MiniAOD, NanoAOD
  - All in “EDM format”, plus NanoAOD as a “flat ntuple” (and RNTuple prototype)
- In principle nearly anything serializable by ROOT is allowed, except
  - Should have no raw pointers (some exceptions, listed later)
  - No pointers to other data products
    - We have our own implementation of “persistable reference to other data product”
- In practice we have mostly (nested) `std::vector`’s of things
  - Plus some `std::set/std::map/std::unordered_map`
- All data types are wrapped in `edm::Wrapper<T>`
  - ROOT gets to know concrete type, framework uses base class pointer in many places

Dynamic polymorphism

- CMS has several data types that rely on dynamic polymorphism that are widely used
  - “Widely” meaning both data tiers (AOD, MiniAOD, special skims, AlCa) and places in code (thousands)
- CMS wants to eventually move to simpler data types.
  - Migration appears to be hard but doable by Run 4, but with large uncertainties
  - **We need help from ROOT team to support a reasonable transition**
    - We need to strive for gradual transformation
    - Need TTree to support the same data types that we would use in RNTuple
- E.g. `std::variant` initially looks like a plausible direct replacement, but details make it difficult to use in all cases
  - Increases coupling, have class hierarchy of O(100) classes

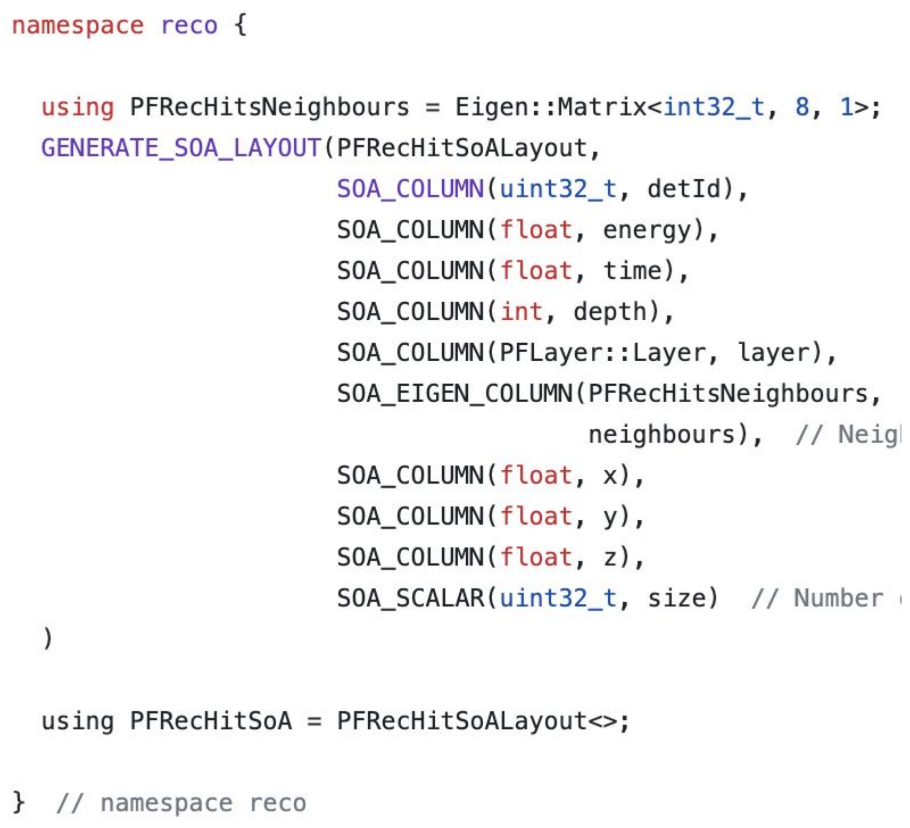
`std::set` and `std::map`

- Currently `std::set` and `std::map` are being used in many places
  - Also some `std::unordered_map`
- Moving to sorted `std::vectors` should be technically feasible
  - But need to stay backwards compatible

SoA data structures

- CMS uses Structure-of-Arrays data structures when interacting with GPUs
  - Want to have a single memory block for all the data in the SoA data structure
- CMS' current SoA data structure can be persisted with `TTree`, but is awkward
  - Requires duplicating nontrivial, error prone snippets in the selection XML files
- We want to have a better mechanism to serialize and store SoAs
  - Preferably in a way that CMS can specify the allocation strategy
- Example in the following slides
  - More details in E. Cano ACAT 2022

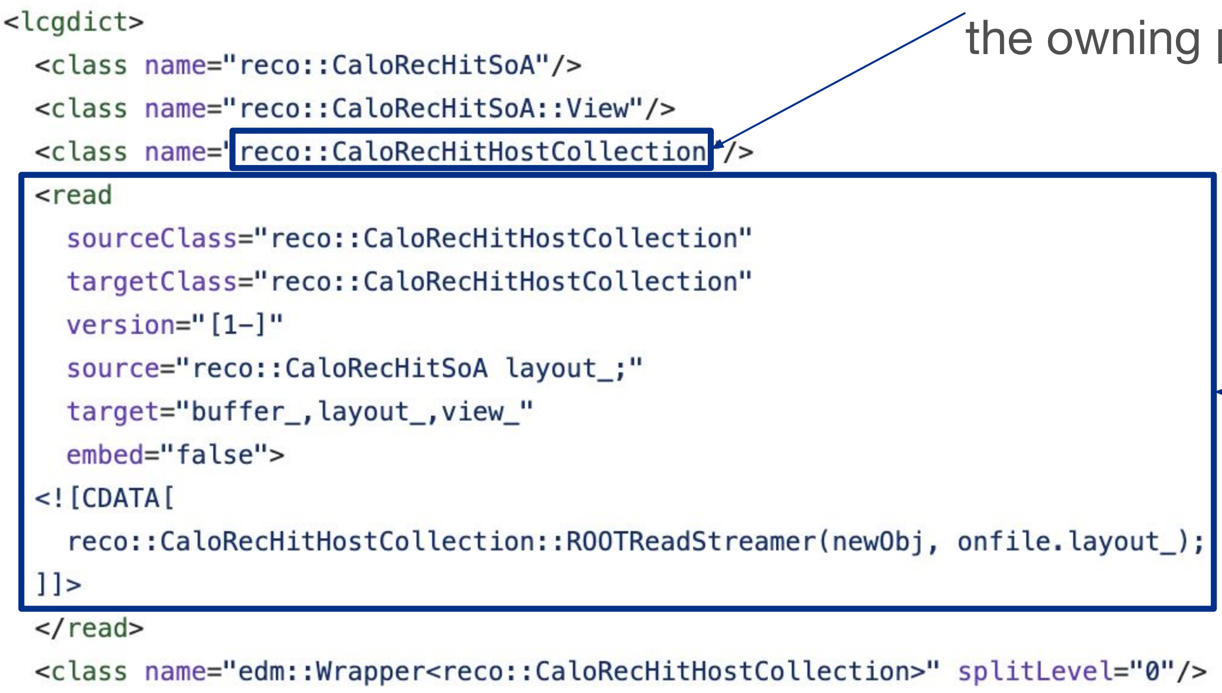
SoA example



```
namespace reco {

  using PFRecHitsNeighbours = Eigen::Matrix<int32_t, 8, 1>;
  GENERATE_SOA_LAYOUT(PFRecHitSoALayout,
                      SOA_COLUMN(uint32_t, detId),
                      SOA_COLUMN(float, energy),
                      SOA_COLUMN(float, time),
                      SOA_COLUMN(int, depth),
                      SOA_COLUMN(PFLayer::Layer, layer),
                      SOA_EIGEN_COLUMN(PFRecHitsNeighbours,
                                       neighbours),  // Neig
                      SOA_COLUMN(float, x),
                      SOA_COLUMN(float, y),
                      SOA_COLUMN(float, z),
                      SOA_SCALAR(uint32_t, size)  // Number
  )

  using PFRecHitSoA = PFRecHitSoALayout<>;

}  // namespace reco
```

- *Layout* specifies how the memory block is interpreted
  - Can contain scalars, columns, and Eigen vector/matrix
  - Padding at the end of each column to match alignment
- Memory ownership is handled separately
- Want the columns to be visible as columns in TTree/RNTuple

SoA example (2)



```xml
<lcgdict>
  <class name="reco::CaloRecHitSoA"/>
  <class name="reco::CaloRecHitSoA::View"/>
  <class name="reco::CaloRecHitHostCollection"/>
  <read
    sourceClass="reco::CaloRecHitHostCollection"
    targetClass="reco::CaloRecHitHostCollection"
    version="[1-]"
    source="reco::CaloRecHitSoA layout_;"
    target="buffer_,layout_,view_"
    embed="false">
  <![CDATA[
    reco::CaloRecHitHostCollection::ROOTReadStreamer(newObj, onfile.layout_);
  ]]>
  </read>
  <class name="edm::Wrapper<reco::CaloRecHitHostCollection>" splitLevel="0"/>
```

Class containing both the layout and the owning pointer (Alpaka buffer)

Serialization is done through the non-owning Layout

Concurrency

- Event-level concurrency is perfectly scalable for CMS
  - CMS prefers to have one CPU thread per concurrent event
    - Framework scales perfectly up to at least thousands of concurrent events, I/O does not
      - C. Jones CHEP 2023
- We want storage that can scale with concurrent events
- TTree parallelizes along branches
  - But branches have very unequal read/write times, in practice we end up being dominated by a few
  - As far as we can see, we gain about 2x speedup (before hitting Amdahl's law)
- We would like to see the concurrency used in I/O to line up with event-level concurrency
  - E.g. asynchronous API, or thread-safe/efficient API

Concurrency (2)

- We would like to be able to pass arbitrary data down into the IO read rules from the equivalent for `TBranch::GetEntry()` function call
  - CMS' version of "`TRef`" (persistent reference to other data product) relies on a pointer to the "Event". Right now we have to pass it down via a thread_local variable. If the actual IO read rule gets run in a different thread than the one calling the "`GetEntry()`", this functionality breaks.
  - More general, e.g. in schema evolution, there can easily be cases where passing arbitrary data to the IO read rules would be extremely useful

Comments on miscellaneous features

- CMSSW does not use `TRef`
- CMS' data types do not use (networks of) raw pointers, except in
  - HepMC
  - `TH1[SIFD]`
  - Serialization of the SoA
  - (there may be more corner cases)
- CMS' persistent data types do not use `std::shared_ptr`
- Some CMS data types use multidimensional C-arrays
- CMSSW framework doesn't depend `TTree::Draw()`
  - I would imagine the proposed separate `ROOT::Plot()` functionality would be sufficient for users
  - What about `TTree::Scan()` like functionality?

Questions on future plans

- What about `std::unordered_set` and `std::unordered_map`?
- What are the plans for reading/writing Events concurrently?
- Are there plans for direct input/output to GPU memory?
- What is the plan for schema evolution support?
- Will ROOT's standalone serialization API continue to be supported?
- Are there plans for TTreeCache-equivalent for RNTuple?
- To what degree will TTree writing be supported after RNTuple is deployed?
  - We assume reading TTree will be supported ~forever
- What are the plans for low-precision floats/ints? (e.g. float16, int4)
- Are there plans for `std::span` or `std::mdspan`?
- What about interoperability with other languages such as Python? E.g. storing a `dict` in RNTuple?

CMS needs in ROOT in order to move to RNTuple

- Need to be able to create a Field from `std::type_info` and/or class name
- Need to be able to pass the data to/from the Field via `void const*`
  - Simplifies a lot how framework deals with data types
    - Note that `std::any` would likely not work
  - Framework guarantees the type safety for user code
- Support for schema evolution
  - We would like to see ROOT to preserve the name of an inline namespace
    - Inline namespaces may be a useful way to deal with library evolution
- Long and wide stress-testing to iron out (rare) bugs
  - We have decades of experience with `TTree`
- Test suite that covers corner and error cases

Strawman timeline for the feature needs

- **Q1 2024**: Need support for std::variant in TTree to help the transition to RNTuple
  - CMS needs an evolutionary path towards RNTuple migration
  - We want to decouple the data type migration from TTree-to-RNTuple migration
- **Q2 2024**: Need to be able to create Field from `std::type_info`/class name, and fill it via `void const*`
- A possible strawman timeline towards Run 4
  - **Q3 2024**: Need production version of ROOT with "NanoAOD-complete" RNTuple
  - **2025**: First CMS large-scale RNTuple-NanoAOD production
  - **Q2 2026**: Need production version of ROOT with complete RNTuple
  - **Q1 2027**: CMSSW release for 2027 data challenge
    - Want to use RNTuple as the file format in this challenge
    - Need all AOD and MiniAOD data types to be compatible with RNTuple