RNTuple: A CMS Perspective

Nick Smith, Christopher Jones, Matti Kortelainen *on behalf of CMS Collaboration*

CHEP 2024

23 October 2024

What is RNTuple?

- New I/O format for ROOT
  - Will replace TTree for HL-LHC
    - They will still support reading from TTrees, but not writing to them
- Standard format design
  - More fully split than TTree
    - will split containers within containers which TTree cannot do
- Is in an beta release now
  - CMS has been giving feedback to the developers
  - ROOT wants a 1.0 release end of this year
    - will be ‘final’ storage format with backwards compatibility guarantee
- See plenary talk on Wednesday

Changing CMSSW to be able to use RNTuple

RNTuple Preferred Characteristics of Stored Classes

- TTree's goal was to support nearly all C++ data objects
  - RNTuple's preferred support is for a smaller subset of C++
- No use of polymorphism
  - e.g. no holding a pointer to a base class
- No recursive class dependencies
  - e.g. no class which holds a std::vector of its own type
- No storage of *bare* pointer
  - std::unique_ptr is fine

CMS' Data Model

- The in memory and storage data models are the same
  - in memory data products are serialized/deserialized directly into/out of storage
  - transient data members might not be stored if they can be regenerated later

- Data products must own all memory to which they refer
  - no shared memory
  - no pointers to memory owned elsewhere

- Cross data product reference are handled by CMS specific smart pointers
  - These are capable of reading back a data product from storage on demand
  - The storage representation is an index used by CMSSW

- No other restrictions made on data products

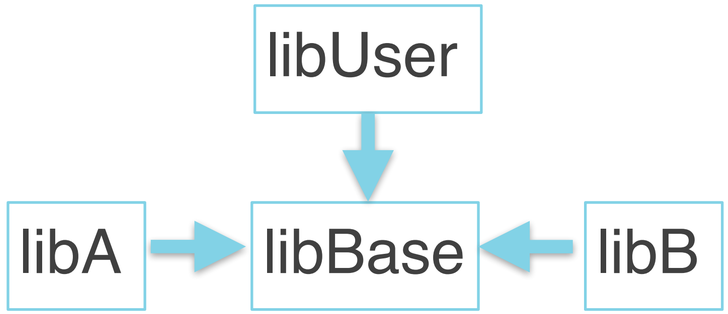
Last Year's Plan: Conform to RNTuple Preferences

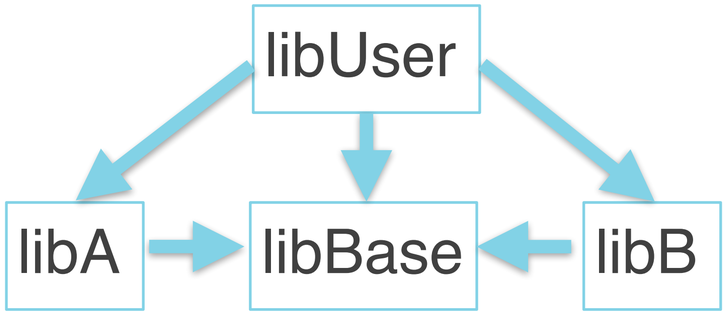
- Remove uses of std containers that were not planned to be supported
  - e.g. std::map
- Survey uses of polymorphism in data products
  - Remove cases where not needed
  - Use std::variant to handle other cases
- Wanted changes to be done adiabatically
  - Want TTree version to also be able to store changed types
  - Want ROOT schema evolution to allow reading old class implementations stored in TTree

Plan Collides with Reality

- Problems removing unnecessary polymorphism
  - Removed an unnecessary base class which used polymorphism
    - ROOT schema evolution unable to handle the change

- Using std::variant lead to problematic library design
  - Code using the type became explicitly dependent on all possible sub-classes





Fermilab

ROOT Team to the Rescue

- Discussed problems with the ROOT team

- ROOT team allowed unsplit fields
  - These use the algorithms already used by TTree
  - Supports all class designs that TTree supports

New Plan

- All presently stored classes can be stored in RNTuple
  - Made possible by unsplit fields

- Classes to be unsplit can be marked in files used to generate dictionaries
  - Changes already done in CMSSW
    - 44 C++ classes were marked to be unsplit
  - Also possible to specify for each data product separately in job configuration

- No further changes to CMS data products are necessary to use RNTuple
  - Optimizing some classes for better storage could be a worthwhile goal

Preliminary Performance Measurements

Testing Procedure

- Read data from a standard TTree based TFile
  - Contains data for CMS' second smallest file format: MiniAOD
  - 84,000 events in 4.7GB file
- Have prototype components that can read/write RNTuple TFiles
  - have various options to control performance
- Testing procedure
  - Read the MiniAOD file
  - Write either TTree or RNTuple based file containing full content of the input

File Size

- TTree Standard
  - Most data products are unsplit, except those which are smaller when split
- RNTuple Partially Split
  - Looked data product by data product to determine if smaller split or unsplit
  - Picked best split level for data products that made the most difference
    - 7 out of 120 data products were smaller unsplit

Format	File	Size	Relative Size
TTree	**Standard**	4.69GB	100.0%
	Fully Split	4.8GB	102.4%
RNTuple	**Fully Split (standard)**	4.6GB	**98.1%**
	Fully Unsplit	5.16GB	**110.0%**
	Partially Split	4.39GB	**93.7%**

Memory Performance

- Monitor calls to new and delete
  - continuously track sums of new and delete and record the max difference of the two
- Run job without output to get baseline memory
- RNTuple output uses the partial split setting

Job	Max Memory	Output Overhead	Relative Overhead
No output	1.34GB	0GB	
TTree standard	2.33GB	0.99GB	1.00
RNTuple	3.03GB	1.69GB	**1.70**

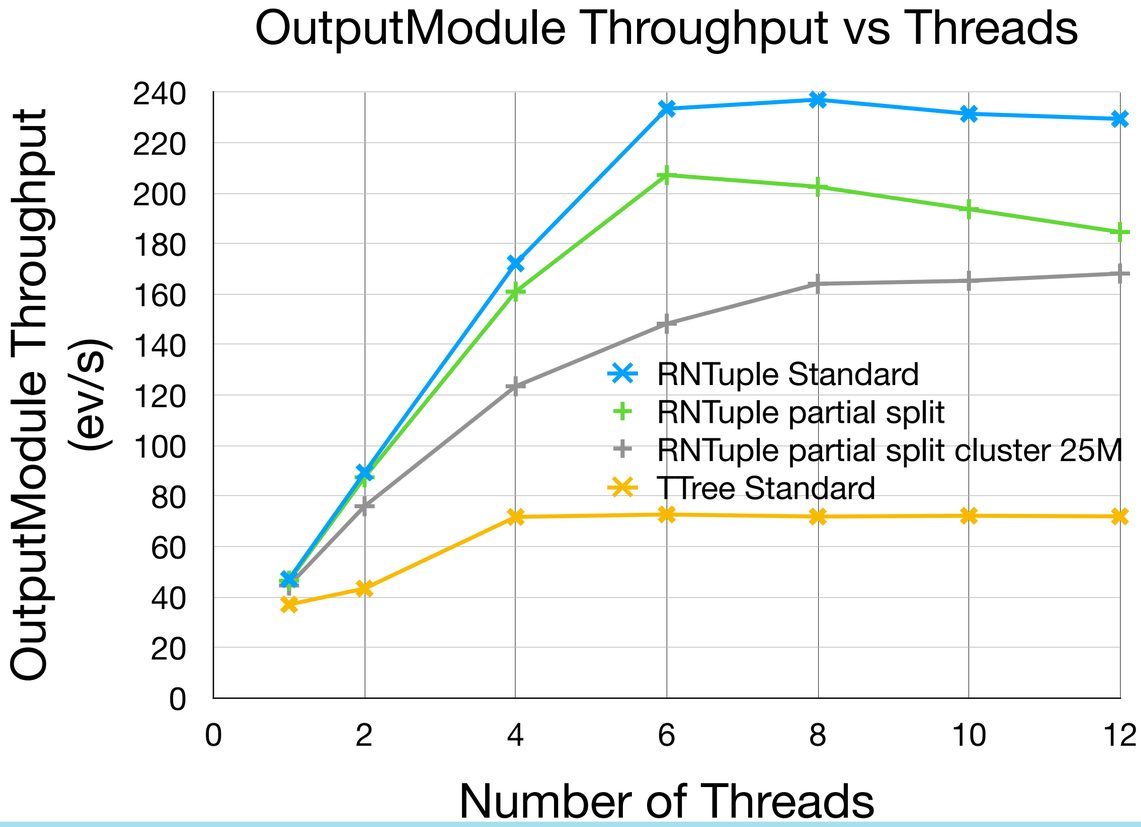
Memory Performance Improvements

- RNTupleWriter takes options to adjust memory usage
  - `SetApproxZippedClusterSize`
    - used by ROOT to decide when to write cluster to file
    - default is $50*10^6$ bytes
- Found changing to 25M bytes to give a good operating point
  - same file size

Job	Max Memory	Output Overhead	Relative Overhead
No output	1.34GB	0GB	
TTree standard	2.33GB	0.99GB	1.00
RNTuple	3.03GB	1.69GB	**1.70**
ZipCluster 25M	2.49GB	1.15GB	**1.16**

Threading Performance

- Ran the test jobs at different thread counts
  - Number of concurrent events always kept at 1
  - Use only time spent in the code that interacts with output file (OutputModule)



Fermilab

Conclusion

- CMS can store its data in RNTuple
  - Thanks to added unsplit option

- Relative performance compared to TTree is still under study
  - File size reduction are modest: ~6%
  - Concurrency improvements are substantial: ~2-3x faster at 8 threads
  - Memory gain is controllable : 16% more memory required
  - Need to study effect of RNTuple on CMS' other file formats