Experiment	**ATLAS**	**CMS**	**DUNE**	**EDM4hep**	**NOvA***
General Overview					
Speaker	Scott Snyder	Matti Kortelainen	Mike Kirby	Benedikt Hegner	Marc Paterno
Talk Link (Indico)	https://indico.fnal.gov/event/57595/contributions/256583/attachments/162731/215145/2023-01-10-edm.pdf	https://indico.fnal.gov/event/55536/	https://indico.fnal.gov/event/58260/	https://indico.fnal.gov/e/55542	https://indico.fnal.gov/event/58962/contributions/262454/attachments/165673/220182/DataOrganizationForParallelProcessing.pdf
Github Link:	https://gitlab.cern.ch/akraszna/asyncgaudi.git		–	https://github.com/key4hep/EDM4hep	https://github.com/art-framework-suite/hep-hpc
Languages	C++/CUDA	C++/CUDA/ALPAKA	C++/CUDA	C++/python	python (PandAna*)
Storage Support	ROOT	ROOT	ROOT/HDF5	ROOT/LCIO	HDF5
Data Model (CPU)	xAOD	Arrays (std::vector<Foo>)	art	POD (Plain old Data)	CAF (Common Analysis Format)
Data Model (GPU)	xAOD-Like	SoA, ASoA	Array of simple types, Follow developments on ATLAS, CMS and other projects	POD (Plain old Data)	CAF (reorganized) (Columnar to be read with pandas)
Detailed Summary Link (Google Doc)	Link	Link	Link	Link	Link

Table I: A brief overview of the talks given by various speakers from various experiments

Talk Series

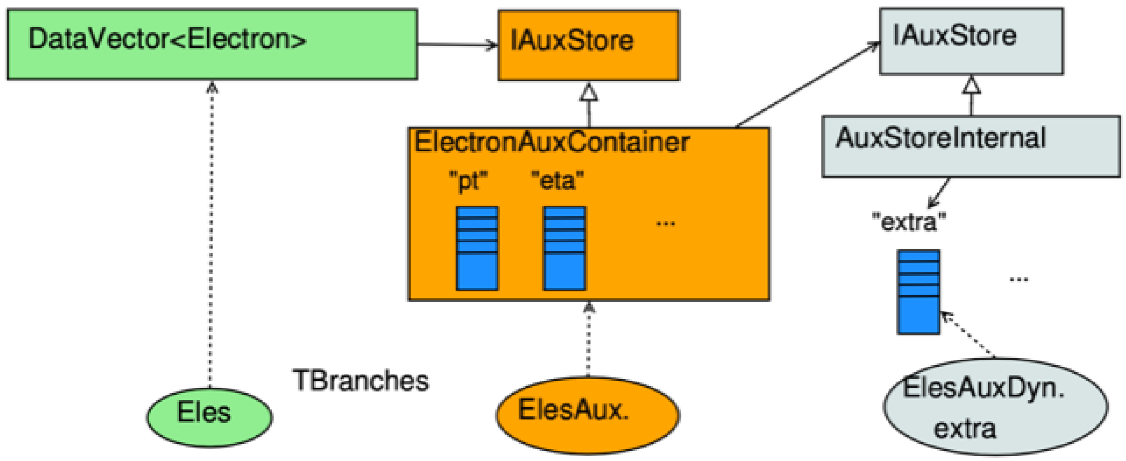
As a part of survey work, 5 talks from various experiments and projects were organized to understand the ongoing efforts to make the HEP data GPU friendly: 3 talks were given by the representatives of the experiments (CMS,DUNE and ATLAS); 2 talks were related to the general effort for the HEP experiments.

Data Model Design

Based on the survey, most of the effort in making the data GPU/HPC friendly has focused on preserving the core structure of the current data layouts. In the case of ATLAS, the data model is based on xAOD. CMS uses vector-like containers to store arrays of objects and the DUNE experiment uses the "art framework". The DUNE-DAQ writes the raw detector data into the HDF5 format as arrays of simple types which can facilitate the offloading of the data into the GPUs downstream of the computational workflow. EDM4hep data model is designed with the consideration of future collider experiments. NOvA converts it's final ntuples into the HDF5 datasets with the data represented as columnar tables that is optimized for parallel I/O and can be offloaded into the GPUs. Unlike ATLAS, CMS and DUNE efforts, where the focus is on development of data model that can be integrated into their computational workflow, the NOvA experiment takes the existing object oriented data, reorganizes into tabular format and writes them into HDF5 format that can then be used to perform the physics analysis.

ATLAS Experiment

The data model used by the ATLAS experiment is the xAOD data model. A typical example of the xAOD data model is shown in the flow diagram below.



In this model, the object pointers are used to access the variables related to the data object. The object pointers are stored separately in std::vector<T*> like containers (DataVector<Electron> in above example). The variables themselves are stored in the contiguous arrays.

Transformation to make Data GPU Friendly

Data Model:

Since the data itself is represented as contiguous arrays, it can be copied into the GPUs explicitly.
In the case of nested arrays and 2-D vectors, ATLAS is thinking about the use of flat arrays since the Aux variables are already represented as contiguous arrays of fixed size. Use array::(begin,end) like indexing functionalities to keep track in the flattened 1D space. From the user's perspective, these flat arrays will look like a 2D “std::vector”.

Memory Management

Memory management is not implemented yet but the plan is to utilize “std::pmr” libraries to manage memory resources and pass them to the Auxcontainer constructors.

Compatibility with the ROOT

ROOT 6.26 seems to support backward and forward compatibility making it possible to implement memory allocator related libraries. However there are still ATLAS specific compatibility issues to resolve.

Persistency

xAOD data models remain intact or minimally modified and will be persisted as they are now in the ROOT.

CMS Experiment

CMS data models are stored as “std::vector<foo>” like containers where foo is a template for a class. In their model, a data product can reference other data products. Similarly the elements of different containers (or collections) might be associated with one another.

Transformation to make data GPU Friendly

Data Model

The CMS event data models are “std::vector<Foo>” like containers. The GPU friendly version of these data structures are the SoA with runtime-determined memory allocation to minimize the memory footprint. The data structure is given by a Layout class and the memory allocation is handled by a separate Alpaka<Buffer> library. The interaction with the Layout members is carried out with a “View” object.

The CMS SoA layout class supports arrays of “simple types" (size defined during the run time), Eigen vectors or matrices (size defined during compile time) and scalars of “simple types”.

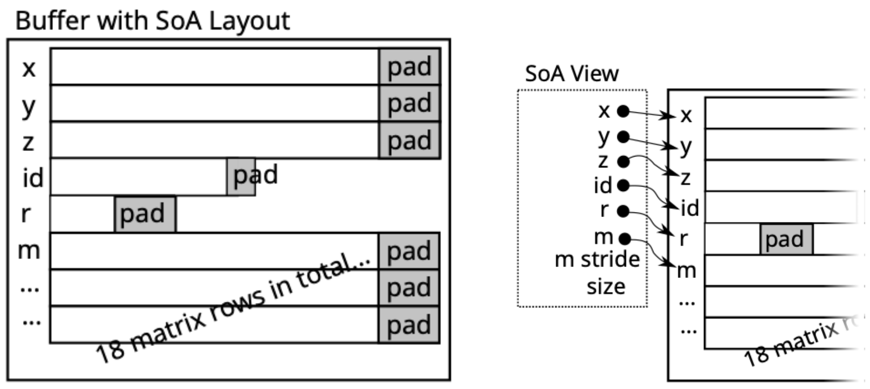


Figure: SoA Layouts in the Memory (left) and the schematic of a View object interaction with the SoA elements (right). Left figure shows the padding implementation for caching alignment.

Memory Management

The Layout or the View objects do not own memory and hence cannot be offloaded into the GPUs directly. Instead, the event data products (which own the memory) are passed into the device (GPUs) and host (CPUs) using the “Portable Collection” class template. Through this class template, “alpaka::Buf<...>” (alpaka buffer) is used to allocate the memory used for the layout object instead.

Note that the ATLAS is thinking about using std::polymorphic for memory allocation while CMS is using a custom class called “Portable Collection” which (de)allocates host and device memory separately.

Persistency

Data is persisted as “Portable Host Collection” in the ROOT. Layer object defines the serialization of SoA. The Serialization information should be added in the ROOT dictionary definition XML files. Plans and discussion with ROOT experts are underway to minimize the amount of information needed to add in the XML files.

DUNE Experiment

Unlike ATLAS or CMS, the DUNE experiment is still being constructed and a lot of computing infrastructure is in the R&D stage. Actual data taking will start from 2031 AD. The tests are being done by running the proto type LArTPCs that will eventually become part of the DUNE far detector.

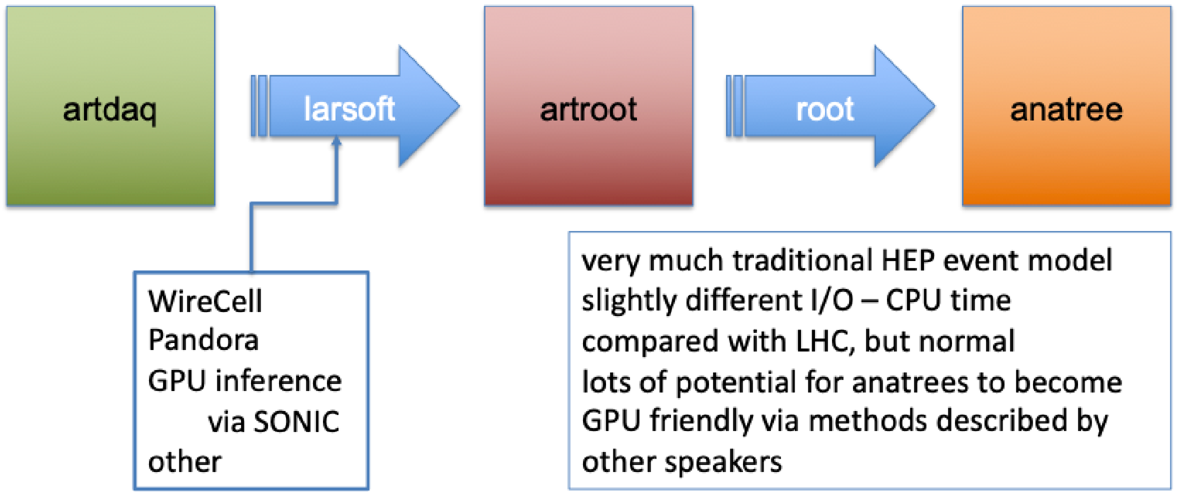

artdaq
larsoft
artroot
root
anatree
WireCell
Pandora
GPU inference
via SONIC
other
very much traditional HEP event model
slightly different I/O – CPU time
compared with LHC, but normal
lots of potential for anatrees to become
GPU friendly via methods described by
other speakers

Figure: (Proto) DUNE has the opportunity to explore the utilization of GPUs in its workflows from raw DAQ objects (art objects) to the final ana tuples (simple TTrees).

Transformation to make GPU Friendly Data model

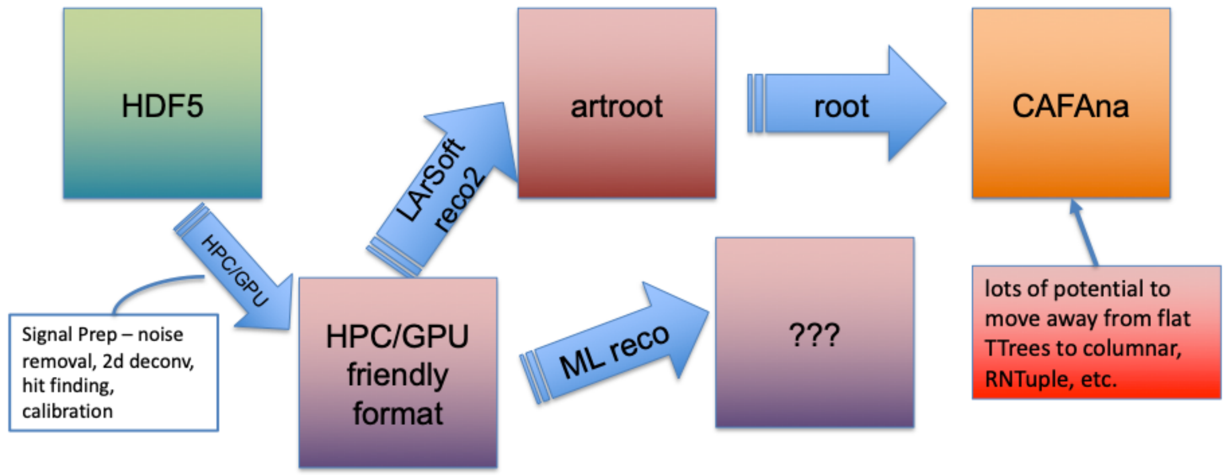

HDF5
HPC/GPU
Signal Prep – noise removal, 2d deconv, hit finding, calibration
HPC/GPU friendly format
LArSoft reco2
artroot
root
CAFAna
ML reco
???
lots of potential to move away from flat TTrees to columnar, RNTuple, etc.

Figure: (Proto) DUNE is exploring and experimenting with the workflow that uses GPUs in various stages.

Currently, the DUNE writes the DAQ data into the HDF5 on a trigger by trigger basis (1 data set per trigger with the metadata that links to information related to channel, APA etc). The goal is to keep the DAQ data design friendly to offload into the GPU/HPC directly. Currently, the DAQ data is basically arrays of simple types with relevant metadata and attributes written as HDF5 objects. The motivation for adopting/testing with HDF5 format for DAQ data is to allow ML/AI reconstruction at the reconstruction level. Similarly, at the ntuple level, the data is stored in simple TTrees allowing it to move ahead to columnar or RNTuple format in the future. Because DUNE is years away from collecting the data, it plans to leverage the progress made in ATLAS, CMS and other projects in this field.

Memory Management

—---

Persistency

ROOT (TTrees, RNtuple*), HDF5