Investigate HDF5 as intermediate event storage for HPC processing

Saba Sehrish for the HEP-CCE IOS team
11/04/2020 HEP-CCE All-hands meeting

What are we doing?

- The focus of this activity is to explore the use of HDF5 files for writing HEP data products that have already been serialized using ROOT serialization.
- We are interested in developing an experiment-independent approach.
- We are currently using a multi-threaded testing framework developed as part of the CCE project to work on the use of HDF5.
- This work is the first attempt to write intermediate output in HDF5 style.
- We have demonstrated the efficient and high performing data access and hence subsequent analysis by using HDF5 representation of the *analysis-ready* data in SciDAC (HEP on HPC) project.

Why HDF5?

- HDF5 (Hierarchical Data Format) is a portable, self-describing file format designed to store large amounts of data
  - It is maintained by the HDF Group [https://www.hdfgroup.org]
  - It is widely available at HPC centers, and easily installable on laptops
  - It supports parallel IO using MPI, and has special drivers tuned for parallel file systems at HPC centers
- A few key abstractions are:
  - *datasets*, which are multidimensional arrays of homogeneous types,
  - *groups*, which are containers of datasets and other groups, and
  - *attributes*, which are small metadata objects to describe groups and datasets
- Allows efficient columnar data access for the “required” data products

What HEP data looks like?

- Different levels of data aggregation: Run/lumi or subrun/event/data products
- Data products are complex data structures representing physics objects
- Each event consists of several different data products and data product size varies from event to event.
- In the current Framework, for non-ROOT format data is serialized before writing to the file and for ROOT, serialization is internal to the writing process.

Mapping HEP data to HDF5

There is no direct mapping to HDF5 way of organizing data. But ...

- map a subrun/lumi to a group and add lumi and run number as attributes
- map each data product to multiple data sets; first dataset for data product itself, and the second dataset for event ID, and another for offset or size.
  - Different options explored
- read and write performance will depend on access patterns

Use 1D variable string dataset

- 1D dataset of variable length string per serialized data product
  - There are performance, accuracy and space concerns associated with converting to string and storing them

An example layout of a data product for 4 events using 1D dataset of strings shown in green color. The event ID and offset of each row have to be stored in separate datasets to indicate the (variable) length of each data product.

Dimension: 4 x 1 for all three data sets

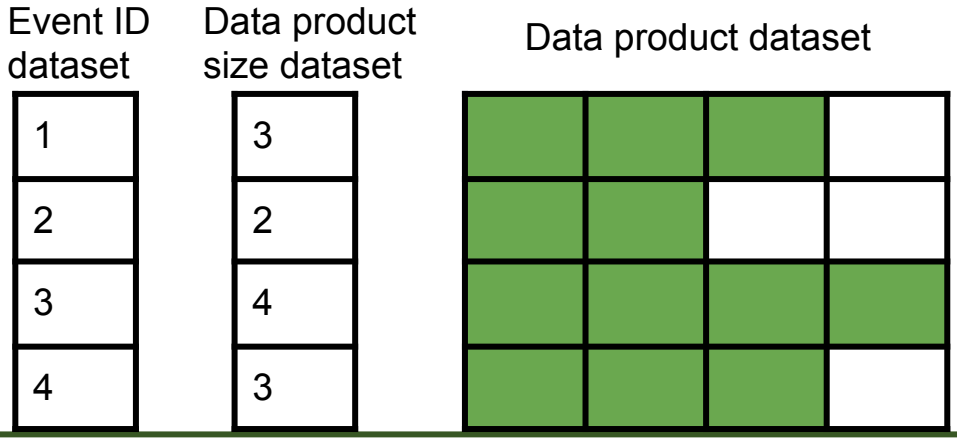
Use 2D dataset of chars

A 2D dataset of chars with the higher dimension of fixed size (maximum size of the data product), need to store size in a separate dataset

- more space, extra calculation for maximum data product size and resize calls

An example to layout a data product for 4 events. We create a 4x4 2D dataset, but it can be seen that not all data products are of size 4 as shown by green boxes. The size of each row have to be stored in separate datasets to indicate the length of each data product.

Dimension: 4 x 1 for first 2 data sets and 4x4 for the last one

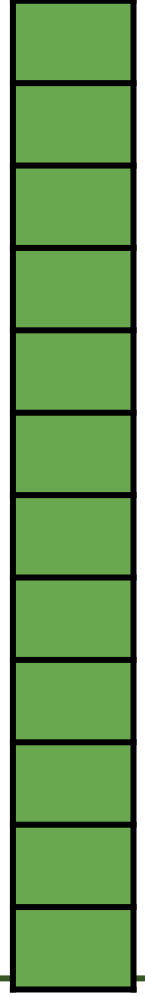


Using 1D dataset of chars

This is our current implementation strategy.

- one group per subrun,
- run number added as a group attribute,
- 1D dataset for event IDs per group,
- 1D dataset of chars for data products per different data product type, and
- a corresponding 1D dataset for size/offset per event.

An example to layout a data product for 4 events in 1D dataset. The event ID and either size/offset of each row are stored in separate datasets to indicate the length of each data product.



HEP-CCE

Event ID dataset

1
2
3
4

Data product dataset

Dimensions: 12 x 1 for data product dataset and 4x1 for the other two

Data product size/offset dataset

3/0
2/3
4/5
3/9

An example HDF5 file

- One group named *Lumi*
- Two attributes; one for lumi number and one for run number
- One dataset for EventIDs; not captured in this screenshot
- Two dataset for each product in the event

```
HDF5 "out.h5" {
GROUP "/" {
   GROUP "Lumi" {
      ATTRIBUTE "lumisec" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SCALAR
      }
      ATTRIBUTE "run" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SCALAR
      }
      DATASET "BranchListIndexes" {
         DATATYPE  H5T_STD_I8LE
         DATASPACE  SIMPLE { ( 162 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "BranchListIndexes_sz" {
         DATATYPE  H5T_STD_U64LE
         DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "EventAuxiliary" {
         DATATYPE  H5T_STD_I8LE
         DATASPACE  SIMPLE { ( 1215 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "EventAuxiliary_sz" {
         DATATYPE  H5T_STD_U64LE
         DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "EventProductProvenance" {
         DATATYPE  H5T_STD_I8LE
         DATASPACE  SIMPLE { ( 378 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "EventProductProvenance_sz" {
         DATATYPE  H5T_STD_U64LE
         DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "EventSelections" {
         DATATYPE  H5T_STD_I8LE
         DATASPACE  SIMPLE { ( 369 ) / ( H5S_UNLIMITED ) }
      }
```

Current status and next steps

- Chris Jones developed a mini test Framework which can “process/serialize” events concurrently and has different modes of input and output.
  - Please see Chris’s talk for more details.
- We have implemented the HDFOutputer for the test Framework
- We used HighFive C++ HDF5 library
  - H5CPP, hephpc_toolkit
- Able to run with a realistic CMS RECO file, and immediately caught a few performance issues;
  - Use chunking: What chunk size to use?
  - Use batch writes: How many data products to accumulate before writing?
- HDFSource to be able to read in HDF5 files via framework

Plan

- Do we need to look at different layout?
  - One variation of the last option was suggested on HighFive forum, i.e. to consider using Compressed Row Sparse matrix approach.
  - Compound data types instead of breaking down data products into multiple datasets

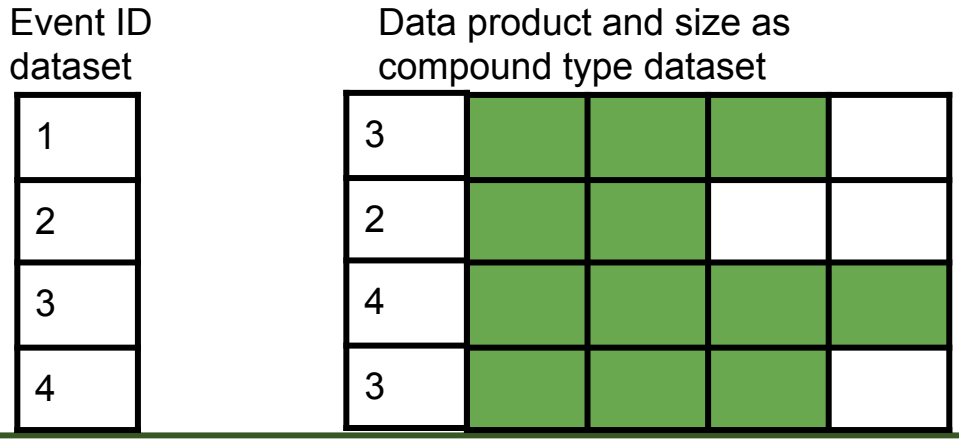
A variation: using 1d dataset of compound datatype

A compound datatype representing each data product will consist of:

- a data array of chars of fixed size (maximum data product size)
- size_t (actual size of the data product)

An example to layout a data product for 4 events. We create a 1D dataset of compound type.

Dimension: 4 x 1 for both data sets



Plan

- Do we need to look at different layout?
  - One variation of the last option was suggested on HighFive forum, i.e. to consider using Compressed Row Sparse matrix approach.
  - Compound data types instead of breaking down data products into multiple datasets.
- HDF5 tuning
  - Chunking: Looked at some prelim numbers, using 128 chunk size, which seems to work fine
  - Asynchronous I/O: Use a background thread to perform I/O
- Compression
  - In both serial and parallel mode, and combined with chunking
- Using node-local storage for storing intermediate HDF5 files
- Parallel IO (using multi-process MPI-based writes)
- Multi-threaded HDF5
  - There is a feature branch available with simple read/write patterns, which we have in our use case, that we should look into
- Explore direct storage access from GPUs