Fermilab U.S. DEPARTMENT OF ENERGY Office of Science

Ntuple: Tabular Data in HDF5 with C++

Chris Green and Marc Paterno

HDF5 Webinar, 2019-01-24

Origin and motivation

- Particle physics analysis often involves the creation of *Ntuples*, tables of (usually complicated) columnar data.
- Meaning of “row” flexible, but consistent within a table (“collision,” “time period,” “track,” *etc.*).
- Historical use of domain-specific analysis and visualization tools with specific binary formats: PAW (Fortran) -> ROOT (C++).
- Recent trend toward general systems like R and Python (Jupyter, pandas, *etc.*).
- Increasing focus on HPC systems, MPI.
- HDF5 would seem to be a natural choice for a flexible scientific data format.

Origin and motivation

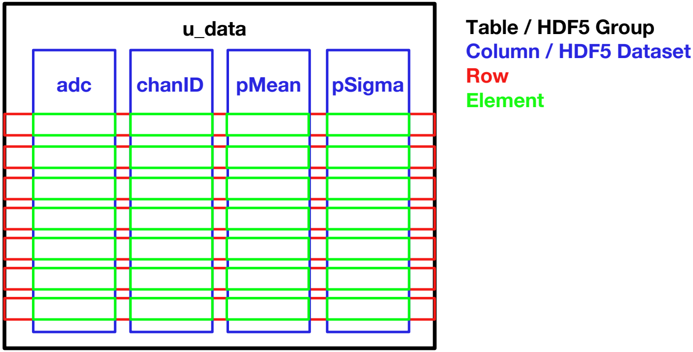
- Our scientists want to code algorithms, not data handling:
  - HDF5 C++ interface insufficiently flexible for our purposes (*e.g.* not recommended for parallel I/O).
  - C interface requires lots of error code checking, and can seem arcane to new users.
- Need something that makes the basic trivial and the complex tractable, and reduces boilerplate as much as possible.

Case study: LArIAT waveform analysis

- LAr "Time Projection Chamber:" time-sampled wires in two orientations (*u*, *v*).
- 2×240 wires, 3072 samples per wire per time slice ("event").
- $\approx 15.5 \times 10^6$ events.
- Saved data: event ID triplet, ADC sample data for *u* and *v* planes, wire #, pedestal (μ, σ).

Ntuple overview

- “Modern” C++ (>=11). Current is C++17.
- Table => Group, Column => Dataset.



- Columns are fixed-size arrays of basic types, fixed- or variable-length strings.
- User inserts data row-wise.

Ntuple overview

- For each column, user will specify:
  - Name
  - Basic element type (*e.g.* `short`)
  - Rank (dimension) of column entries *e.g.* 2
  - Extents of column entries *e.g.* 240×3072
- `Ntuple` is a C++ template: column element type and rank fixed at compile-time; names and extents at table construction time.
- Datasets have one more rank than the user specifies for the column: this is extensible, and represents the column entries in each row. For example, our dataset of shorts has extents $(N, 240, 3072)$, where N is the number of rows written so far.
- Rows are written in buffered groups.
- System for writing data only: files can be read with standard tools such as `h5py`.
- Exception-safe, featuring resource cleanup.

Simple things are very simple

```
// Define the ntuple.
Ntuple<int, double> nt("simple.hdf5", // File.
                       "data",        // Table name (group).
                       {"A", "B"}     // Column names.
                      );

// Data to store.
std::array<int, 3> idata { 1, 2, 3 };
std::array<double, 3> ddata { 4.5, 5.5, 6.5 };

// Insert the data for 3 rows.
for (auto i = 0; i < 3; ++i) {
 nt.insert(idata[i], ddata[i]);
}
```

Simple things under the covers, tho' ...

```
H5Fcreate(); // File create.
H5Gcreate2(); // Group create.
// Dataset A  (repeat from here for dataset B) ...
H5Pcreate(); // Properties for dataset A.
  H5Pset_chunk(); // Chunking required.
  H5Pset_deflate(); // Compression is default.
H5Screate_simple(); H5Dcreate2(); // Create dataset A.
// Per write operation:
  // Extend dataset for write:
  H5Dget_space(); H5Sget_simple_extent_dims(); H5Dset_extent();
  // Memory dataspace:
  H5Dget_space(); H5Sselect_hyperslab(); H5Screate_simple();
  H5Dwrite(); // Write to dataset A.
```

- All the `H5Xclose()` calls elided for "brevity."

Complicated things are still fairly simple

```
auto u_data =
  make_ntuple({"lariat-ish.hdf5", "u_data"},
              make_column<unsigned short, 2>("adc",
                 {240, 3072},
                 {PropertyList{H5P_DATASET_CREATE}
                  (&H5Pset_shuffle)
                  (&H5Pset_deflate, 7u)}),
              make_column<unsigned short>("chanID", 240),
              make_column<float>("pMean", 240),
              make_column<float>("pSigma", 240));
auto v_data =
  make_ntuple({u_data.file(), "v_data"}, ...);
// Insert data for one u row (_e.g._):
u_data.insert(u_adc.data(), chanIDs.data(), // STL vectors.
              chanMean, chanSigma); // C-style arrays.
```

Reading with Python is trivial

```python
from h5py import File
with File("lariat-ish.hdf5", "r") as infile:
    u_adc_data = infile["/u_data/adc"][slice_first:slice_last]
    # Analysis ...
```

Details

- `template <typename...Cols> class Ntuple`: variadic template for arbitrary number of columns.
- Helper functions embody more flexibility: `make_ntuple()`, `make_column()`, `make_scalar_column()`
- Tables can be inserted into an existing HDF5 file anywhere in the existing group hierarchy.
- Filters can be applied (and chained) and other properties customized as necessary.
- Endianness can be specified.
- Data organization: multi-dimensional column entries are contiguous, row-major.
- Exception-safe, including resource cleanup.

Performance notes

- Buffering / chunking, *etc.* is configurable to reduce dataset extension operations.
- Recommended buffer size is an integral number of chunks.
- Files produced this way have been used successfully in a 76.8Kproc Python run utilizing MPI I/O.
  - Read and decompress 42TiB of LArIAT waveform data in $< 20s$.

Other library features

- Resource-managing classes for common HDF5 entities:
  - Dataspace
  - Datatype
  - File
  - Group
  - PropertyList
- Ability to add attributes to datasets and groups.
- Ntuple file concatenation utility (MPI I/O available with recent HDF5).
- Error handling: allow exceptions to be thrown safely on HDF5 errors:
  - within the library with a single configuration call;
  - using a “thin-thick” wrapper function for direct HDF5 calls.

https://bitbucket.org/fnalscdcomputationalscience/hep_hpc