Fermilab | U.S. DEPARTMENT OF ENERGY | Office of Science

Some HEP experience with HDF5

Marc Paterno, Chris Green, Jim Kowalkowski and Saba Sehrish

HDF BOF @SC18

Fermilab

"Fermilab is America's particle physics and accelerator laboratory.

We bring the world together to solve the mysteries of matter, energy, space and time."

How we are currently using HDF5

- Most of our use is recent; HEP does not have a long history with HDF5.
- We use it to store *tabular* data (HEP calls them *ntuples*)
  - for interactive analysis by scientists
  - for input to machine learning algorithms
- We are exploring its use for large data sets
- HEP scientists program in (1) C++ and (2) Python
- We have provided some C++ library code to make creation of ntuple files simpler.
- Uptake in our community is encouraging:
  - The NOvA experiment is using this in large-scale data processing.
  - The multi-lab and multi-university SciDAC project “HEP Data Analytics on HEP” is making use of it in research.

In a Nutshell

- C++ template-based library for writing tabular data to HDF5.
  - Table -> HDF5 Group.
  - Column -> HDF5 Dataset.
  - Row element -> scalar or fixed size array of basic type (including variable and fixed-length strings).
- Requires C++11 and recent compiler (*e.g.* GCC newer than 6.4 verified); C++14 preferred for more intuitive syntax.
- Produces standard HDF5 files readable with *e.g.* `h5py` or C API.

Detail

- Table structure (column types and ranks) determined at compile-time; column names, extents at runtime.
- Row-based data entry: each column is an extensible dataset of row elements.
- Built on a “thick-thin” wrapper providing:
  - Exception-safe, resource-managed classes for HDF5 entities such as files, property lists, groups, datasets, and dataspaces.
  - Error handling via exceptions and an all-purpose call wrapper for HDF5 functions returning a possible error condition.
- Convenience functions `make_ntuple()`, `make_column()` and `make_scalar_column()` allow easy specification of column metadata such as filters.

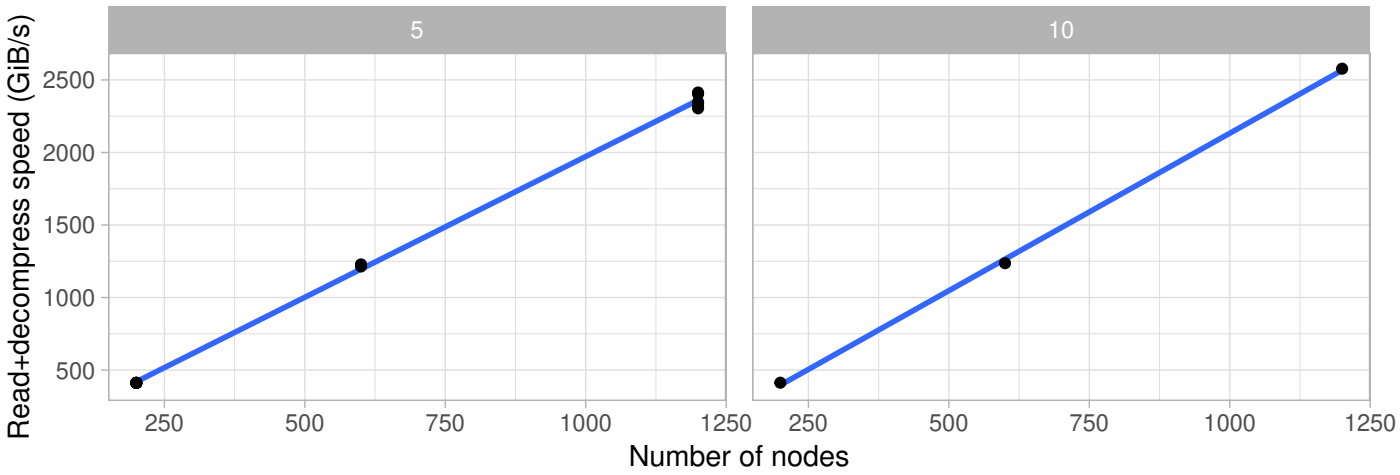
Simple things are very simple

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

Complicated things are fairly simple

```
auto data =
  make_ntuple({"less-simple.hdf5", "g1"},
              make_column<int, 2>("A", {2, 3}),
              make_scalar_column<double>("B",
               {PropertyList{H5P_DATASET_CREATE}
                 (&H5Pset_shuffle)
                 (&H5Pset_deflate, 7u)}));
int i1data[] = {  1,  2,  3,  4,  5,  6,
                  7,  8,  9, 10, 11, 12,
                 13, 14, 15, 16, 17, 18 };
double d1data[] = { 1.01, 2.02, 3.03 };
for (auto i = 0; i < 3; ++i) {
  // Array -> address, value -> scalar or address.
  data.insert(&i1data[i*2], d1data[i]);
}
```

We are pleased with the read performance



These results are running on Cori at NERSC, using 200/600/1200 KNL nodes, 64 MPI ranks per node, in a Python program, reading 42 TiB of LArIAT experiment raw data, compressed into a single 4.2 TiB file. Panels show performance with 5 and 10 TiB Burst Buffer allocations.

The addition we would like the most

- We store (redundantly) index information in our tables (e.g. run number, subrun number, event number to identify a row in a table).
- We would prefer to store the relevant data at a location defined by *run*, *subrun*, and *event* “coordinates” in a 3-d space (and we need 4-, 5-, .. n-D spaces).
- But only a few “cells” of such a structure would be filled.
- We really need *efficient sparse multi-dimensional arrays*.

What we are finding hard

- Concatenation of files is hard to make efficient (i.e, to scale well)
  - concatenation means combining the rows of tables in similarly-structured input files into a single outpt file.
- Parallel writing in Python with filters (e.g. compression) is not currently possible.
  - ... but that is `h5py`, not really HDF5