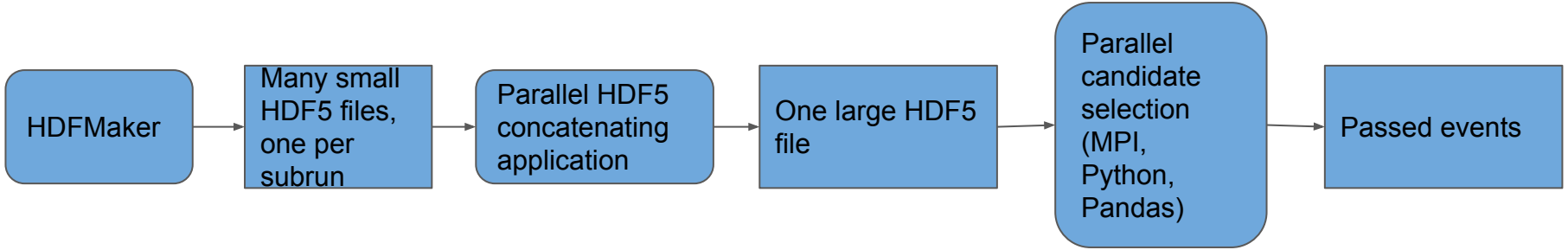
Our use of HDF5 in HEP

Some background

We have been exploring the use of HDF5 for HEP processing for a couple of years now, as part of CMS big data project, an LDRD project and a SciDAC4 project. In the SciDAC4 project, HEP data analysis on HPC, we

- **have provided support for storing NOvA's CAF equivalent data in HDF5**
  - NOvA has been writing HDF5 analysis ntuples in production for more than a year
  - NOvA collaborators have been using for more than a year
- **are working on a data-parallel package, PandAna, to read in HDF5 data and easily express user-defined cuts**
  - Users report faster development cycle compared to C++ (no compilation, fast exploration)
  - Users report 5-100 times *faster* than compiled C++/ROOT code, for various analyses

NOvA processing



Multidimensional data

- We organize our multidimensional (n-tuple) data to be able to write analysis code that is easy and scalable.
- A new name for a standard method for organizing these data: data matrix.
- Each variable is represented as a column and each observation as a row.
- Related to Boyce-Codd 3rd normal form.
- We use a data matrix (table) for each set of related observations; analysis code sees pandas.DataFrame objects.

NOvA data

Table 1

NOvA data table organization with one entry per slice.

run	subrun	event	sub-event	distallpngtop	... 35 more ...
433	61	6124	35	nan	
433	61	6124	36	-0.7401	
433	61	6124	37	nan	
433	61	6125	1	nan	
433	61	6125	2	423.633	
433	61	6125	3	-2.8498	

Table 2

NOvA data table organization with one entry per vertex.

run	subrun	event	sub-event	vtxid	npng3d	... 6 more ...
433	61	6124	35	0	0	
433	61	6124	36	0	1	
433	61	6124	36	1	1	
433	61	6124	36	2	5	
433	61	6125	1	0	1	
433	61	6125	3	0	0	

HDF5

- HDF5 is a file format designed to store large amounts of data. – It is supported by the HDF Group [https://www.hdfgroup.org]
  – It is widely available at HPC centers, and easily installable on laptops.
  – It supports (MPI) parallel IO, and has special drivers tuned for parallel filesystems.
- It has two very important abstractions:
  datasets, which are multidimensional arrays (like numpy) of homogeneous types, and groups, which are containers of datasets and other groups.
- We use it to store various tables: a table corresponds to a group;
- a column corresponds to a dataset in a group;
  all datasets is a group have the same number of entries, but they can have different types

StandardRecord

```
class StandardRecord
{

public:
  StandardRecord();
  ~StandardRecord();

  SRHeader          hdr;    ///< Header branch: run, subrun, etc.
  SRSpill           spill;  ///< Beam spill branch: pot, beam current, etc.
  SRSlice           slc;    ///< Slice branch: nhit, extents, time, etc.
  SRTrackBranch     trk;    ///< Track branch: nhit, len, etc.
  SRVertexBranch    vtx;    ///< Vertex branch: location, time, etc.
  SRMichelE         me;     ///< Michel electron branch
  SREnergyBranch    energy; ///< Energy estimator branch
  SRIDBranch        sel;    ///< Selector (PID) branch
  SRTruthBranch     mc;     ///< Truth branch for MC: energy, flavor, etc.
  SRParentBranch    parent; ///< True parent branch for matching, e.g. MRCC
  SRTrainingBranch training; ///< Extra training information for prototyping PIDs etc.
};
```

The StandardRecord is the primary top-level object in the Common Analysis File trees, complex but common.

SRHeader

```
class SRHeader
  {
  public:
    SRHeader();
    ~SRHeader();

    unsigned int   run;       ///< run number
    unsigned int   subrun;    ///< subrun number
    int            cycle;     ///< MC simulation cycle number
    int            batch;     ///< MC simulation batch number
    unsigned int   evt;       ///< ART event number, indexes trigger windows.
    unsigned short subevt;    ///< slice number within spill
    bool           ismc;      ///< data or MC?  True if MC
    Det_t          det;       ///< Detector, ND = 1, FD = 2, NDOS = 3
    bool           blind;     ///< if true, record has been corrupted for blindness
    bool           filt;      ///< if true, record has ben filtered

    unsigned short dibfirst;  ///< first diblock in detector configuration (1-14)
    unsigned short diblast;   ///< last diblock in detector configuration (1-14)
    unsigned short dibmask;   ///< diblock mask (bitfield, lowest bit = diblock 1)
    unsigned short maskstatus;///< 0 no mask found in DB, 1 mask used ok, 2 masking turne
wrong in this case.

    unsigned short year;      ///< year of spill
    unsigned short month;     ///< month of spill
    unsigned short day;       ///< day of spill within month
    unsigned short doy;       ///< day of spill within year
    unsigned short hour;      ///< hour of spill
    unsigned short minute;    ///< minute of spill
    unsigned short second;    ///< second of spill

    float          unixtime;  ///< unix time of spill

    float subevtstarttime;    ///< time of beginning of slice within spill [ns]
    float subevtendtime;      ///< Slice end time  [ns]
    float subevtmeantime;     ///< Slice mean time [ns]

    unsigned int  nbadchan;   ///< Number of bad channels in a subrun. Ignores channels i
    unsigned int  ntotchan;   ///< Total number of channels in the analysis masked region

    unsigned short gain;      ///< Global gain setting of the detector
    bool           finetiming;///< Is fine timing enabled in this run?

    void setDefault();
  };
```

Each variable is represented as a column.

```
ssehrish@cori01:~/pandana_stuff> h5ls /global/cscratch1/sd/ssehrish/pandana_input/nd_165/nd_165_files_with_evtseq.h5caf.h5/rec.hdr
batch                    Dataset {2308113/Inf, 1}
blind                    Dataset {2308113/Inf, 1}
cycle                    Dataset {2308113/Inf, 1}
day                      Dataset {2308113/Inf, 1}
det                      Dataset {2308113/Inf, 1}
dibfirst                 Dataset {2308113/Inf, 1}
diblast                  Dataset {2308113/Inf, 1}
dibmask                  Dataset {2308113/Inf, 1}
doy                      Dataset {2308113/Inf, 1}
evt                      Dataset {2308113/Inf, 1}
evt.seq                  Dataset {2308113/Inf, 1}
filt                     Dataset {2308113/Inf, 1}
finetiming               Dataset {2308113/Inf, 1}
gain                     Dataset {2308113/Inf, 1}
hour                     Dataset {2308113/Inf, 1}
ismc                     Dataset {2308113/Inf, 1}
maskstatus               Dataset {2308113/Inf, 1}
minute                   Dataset {2308113/Inf, 1}
month                    Dataset {2308113/Inf, 1}
nbadchan                 Dataset {2308113/Inf, 1}
ntotchan                 Dataset {2308113/Inf, 1}
run                      Dataset {2308113/Inf, 1}
second                   Dataset {2308113/Inf, 1}
subevt                   Dataset {2308113/Inf, 1}
subevtendtime            Dataset {2308113/Inf, 1}
subevtmeantime           Dataset {2308113/Inf, 1}
subevtstarttime          Dataset {2308113/Inf, 1}
subrun                   Dataset {2308113/Inf, 1}
unixtime                 Dataset {2308113/Inf, 1}
year                     Dataset {2308113/Inf, 1}
```

rec.hdr is a group in our HDF5 file representing SRHeader class, and a dataset each variable.

SRVertexBranch

```
/// Vectors of reconstructed vertices found by various algorithms
class SRVertexBranch
  {
  public:
    SRVertexBranch();
    ~SRVertexBranch();

    SRElastic     elastic;  ///< Single vertex found by Elastic Arms


    std::vector<SRHoughVertex> hough;     ///< Vector of vertices found by HoughVertex
    size_t                     nhough;    ///< Number of vertices in HoughVertex (hough.size())

    std::vector<SRVertexDT>    vdt;       ///< Vector of vertices found by VertexDT
    size_t                     nvdt;      ///< Number of vertices in VertexDT (vdt.size())

    void fillSizes();

  };

} // end namespace
```

HDF5 representation of SRVertex and SRElastic

```
ssehrish@cori01:~/pandana_stuff> h5ls /global/cscratch1/sd/ssehrish/pandana_input/nd_165/nd_165_files_with_evtseq.h5caf.h5/rec.vtx
cycle                    Dataset {2308113/Inf, 1}
evt                      Dataset {2308113/Inf, 1}
evt.seq                  Dataset {2308113/Inf, 1}
nelastic                 Dataset {2308113/Inf, 1}
nhough                   Dataset {2308113/Inf, 1}
nvdt                     Dataset {2308113/Inf, 1}
run                      Dataset {2308113/Inf, 1}
subevt                   Dataset {2308113/Inf, 1}
subrun                   Dataset {2308113/Inf, 1}
ssehrish@cori01:~/pandana_stuff> h5ls /global/cscratch1/sd/ssehrish/pandana_input/nd_165/nd_165_files_with_evtseq.h5caf.h5/rec.vtx.elastic
cycle                    Dataset {1913329/Inf, 1}
evt                      Dataset {1913329/Inf, 1}
evt.seq                  Dataset {1913329/Inf, 1}
rec.vtx.elastic_idx      Dataset {1913329/Inf, 1}
run                      Dataset {1913329/Inf, 1}
subevt                   Dataset {1913329/Inf, 1}
subrun                   Dataset {1913329/Inf, 1}
time                     Dataset {1913329/Inf, 1}
vtx.x                    Dataset {1913329/Inf, 1}
vtx.y                    Dataset {1913329/Inf, 1}
vtx.z                    Dataset {1913329/Inf, 1}
```

rec.vtx is a group in our HDF5 file representing SRVertexBrnach class, and rec.vtx.elastic SRElastic.

HDF5 representation of SRHoughVertex

```
ssehrish@cori01:~/pandana_stuff> h5ls /global/cscratch1/sd/ssehrish/pandana_input/nd_165/nd_165_files_with_evtseq.h5caf.h5/rec.vtx.hough
cycle                    Dataset {0, 1}
evt                      Dataset {0, 1}
evt.seq                  Dataset {0, 1}
rec.vtx.hough_idx        Dataset {0, 1}
run                      Dataset {0, 1}
subevt                   Dataset {0, 1}
subrun                   Dataset {0, 1}
time                     Dataset {0, 1}
vtx.x                    Dataset {0, 1}
vtx.y                    Dataset {0, 1}
vtx.z                    Dataset {0, 1}
```

rec.vtx.hough is a group in our HDF5 file representing SRHoughVertex.

Using HEP_HPC ntuple library to write HDF5 files

- We have a C++ library hep_hpc on BitBucket to help write HDF5 files.
- We have written an *art* module, HDFMaker, to be used in the NOvA workflow that also creates the CAF files.
  - https://cdcvs.fnal.gov/redmine/projects/novaart/repository/show/trunk/HDF5Maker
  - This module writes one HDF5 tabular file per job. The job is run on the Fermi grid nodes and one small HDF5 file corresponding to each *art*-ROOT input file is generated. This results in **thousands of small HDF5 files**.
- NOvA has the ownership of this module now, and their use of HEP_HPC ntuple library has evolved much.
- An example of using HEP_HPC code is available here: https://bitbucket.org/fnalscdcomputationalscience/hep_hpc/src/master/examples/make_ntuple_file.cc

Concatenating thousands of HDF5 files

The NOvA data consists of millions and millions of events that are grouped in hundreds to thousands of small HDF5 files.

- We have worked on a scalable parallel IO utility program to concatenate large number of HDF5 files.
  a. Parallelism beyond number of files
  b. Use striping and parallel IO for improved performance
- The utility is an MPI program, where input file(s) are evenly distributed among all MPI ranks.
- It has options for independent and collective modes for reads and writes
- The HDF5-related tuning includes adjusting metadata cache size to 128 MiB, collective metadata IO mode, in-memory IO, data chunk size and data storage layout adjustments.

PandAna Framework

- We are developing a framework, PandAna, to facilitate easy-to-use, scalable and high- performance analysis code.
- PandAna supports **parallel reading of HDF5 files,** which have our type of schema.
  - Each table has an additional column used to support load-balancing for parallel reading.
  - Each MPI process processes a portion of each table that is used.
- Many analyses do not use every table in a large dataset.
  - PandAna will read from only the tables used in your analysis program.
- Many analyses do not use every column in each table that is used.
  - PandAna will read only those columns that are actually needed.

More PandAna

- Analysis code is written (almost) exactly as in a serial program; parallelism is implicit.
- Analysis code sees a pandas.DataFrame for each table, carrying only columns that will be used.
  – All needed data is in some dataframe.
  – No data are duplicated between processes.
  – Data are distributed to assure that no event is split across processes.

PandAna is still under development.

Some near future plans

We will be working with DUNE, CMS and ATLAS

- DUNE has already adopted NOvA's CAF, and is looking to use PandAna
- Initiating discussions with Coffea project team
- ATLAS collaborators in our SciDAC project are also interested in using PandAna approach,
  - We have an example program that uses uproot to read a CMS *nanoaod* "flat ntuple" and write our style of HDF5 file.
  - The generic equivalent is a fairly obvious modification. Something like it could be written for an ATLAS equivalent of the *nanoaod*.