Rethinking I/O: Using HPC resources within HEP

Jim Kowalkowski

Scalable I/O Workshop

23 Aug 2018

What we have been challenged with

- Greatly increase usage of HPC resources for HEP workloads
  - After all, many more compute cycles will be available in HPC than anywhere else.
- Can we …
  - Provide for large-scale HEP calculations
  - Demonstrate good resource utilization
  - Use tools available on HPC systems (we believe this is a practical decision)
- Scalable I/O has been one of the major concerns

Efforts have been underway to tackle challenges

- Big Data explorations (SCD)
  - New (to HEP) methods for performing analysis on large datasets
  - Began with Spark, moved to python/numpy/pandas/MPI
- HDF for experimental HEP (Fermilab LDRD 2016-010)
  - Organizing data for efficient access on HPC systems (HDF)
  - Organizing programs for efficient analysis of data with Python/numpy/MPI
- HEP Data Analytics on HPC (SciDAC grant)
  - Collaboration between DOE Office of High Energy Physics and Advanced Scientific Computing Research (ASCR supports the major US supercomputing facilities)
  - Physics analysis on HPC linked to experiments (NOvA, DUNE, ATLAS, CMS)

Questions to be addressed

- How ought data be organized and accessed?
  - Assuming usual HPC facility with a global parallel file system
  - A deeper memory hierarchy than we are used to
- How should the applications be organized?
  - Is our current programming model appropriate?
  - How do we achieve necessary parallelism?
  - What libraries should we be using?
- How will the operating systems and run-time environment affect our computing operations and software?
  - Are the software build and deployment tools we have in place adequate?
  - What if we could analyze an entire dataset all at once?
  - Can we benefit from tighter integration of workflow and application?

Plan of attack

- Choose representative problems to solve
  - NOvA analysis workflows, LArTPC processing, generator tuning
- Choose toolkits and libraries that could help
  - HDF5
  - ASCR data services geared towards HPC
  - Python with numpy, MPI, and Pandas
  - Container technology
  - DIY as a solution for data parallelism
- Facilities to be used initially:
  - NERSC Cori (KNL and Haswell)
  - ALCF Theta

Guiding principles, requirements, and constraints

- We aim to greatly reduce the time it takes to process HEP data.
- We need to redesign our workflows and code to take full advantage of HPC systems.
  - to use well-established parallel programming tools and techniques
  - to make sure these tools and techniques are sufficiently easy to use
  - We need programs that are adaptable to different “sizes” of jobs (numbers of nodes used) without changing the code.
  - We want data designed for partitioning across large machines.
- We want to partition data (and processing) by things that are meaningful in the *problem domain* (events, interactions, tracks, wires, . . .), not according to *computing model artifacts* (e.g. Linux filesystem files).
  - parallelism *implicit*

Experimental contexts for our work

- LArTPC wire storage, access, and processing
- Event selection for neutrino analysis
- Object store for physics data
- Physics generator data access

LArTPC wire storage, access, and processing

Noise removal from LArIAT waveforms

- LArIAT is a LArTPC (Liquid Argon Time Projection Chamber) test beam experiment
- Converted all LArIAT raw data sample to one HDF5 file
  - Started with 200K art/ROOT data files
  - ~42 TB of digitized waveforms (4.2 TB compressed)
  - 15,684,689 events.
  - Waveform data from u and v wireplanes (240 wires per plane, 3072 samples per wire)
- Reorganized the data using HDF to be more amenable for parallel processing
- Processing the entire LArIAT raw data sample
  - First step of reconstruction is noise reduction using FFTs

Example MPI code: processing many events at one time

```
# first and last are calculated by library code, to tell
# this function call what part of the data set it is to
# work on.
adc_data = adcdataset[first:last] # read block of array
adc_floats = adc_data.astype(float)
# view data as an array of wires, rather than as events
adc_floats.shape = (nevts*WIRES_PER_PLANE, SAMPLES_PER_WIRE)
waveforms = transform_wires(adc_floats) # real work done here
# view the data as events again
waveforms.shape(nevts, WIRES_PER_PLANE, SAMPLES_PER_WIRE)
```

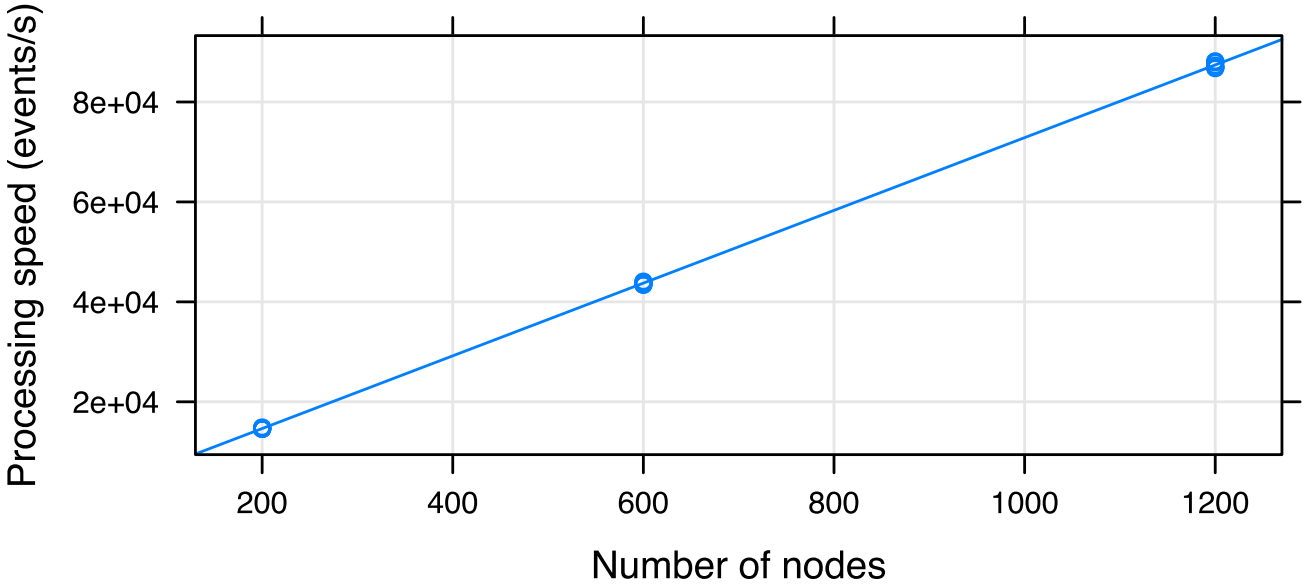
Parallelism is *entirely implicit*, and entirely data parallel.

Example code: processing many wires

- All the real work is done in the numpy library, implemented in C.
  - The library ***can*** use multithreading, or *vectorization*, to get the most performance from the hardware.
- The script that launches the application specifies how many processes to use:
  - mpirun -np 76800 python process_lariat.py **<**filename**>**
  - This starts 76800 communicating parallel instances of our program — equivalent to running 76800 jobs all at once.

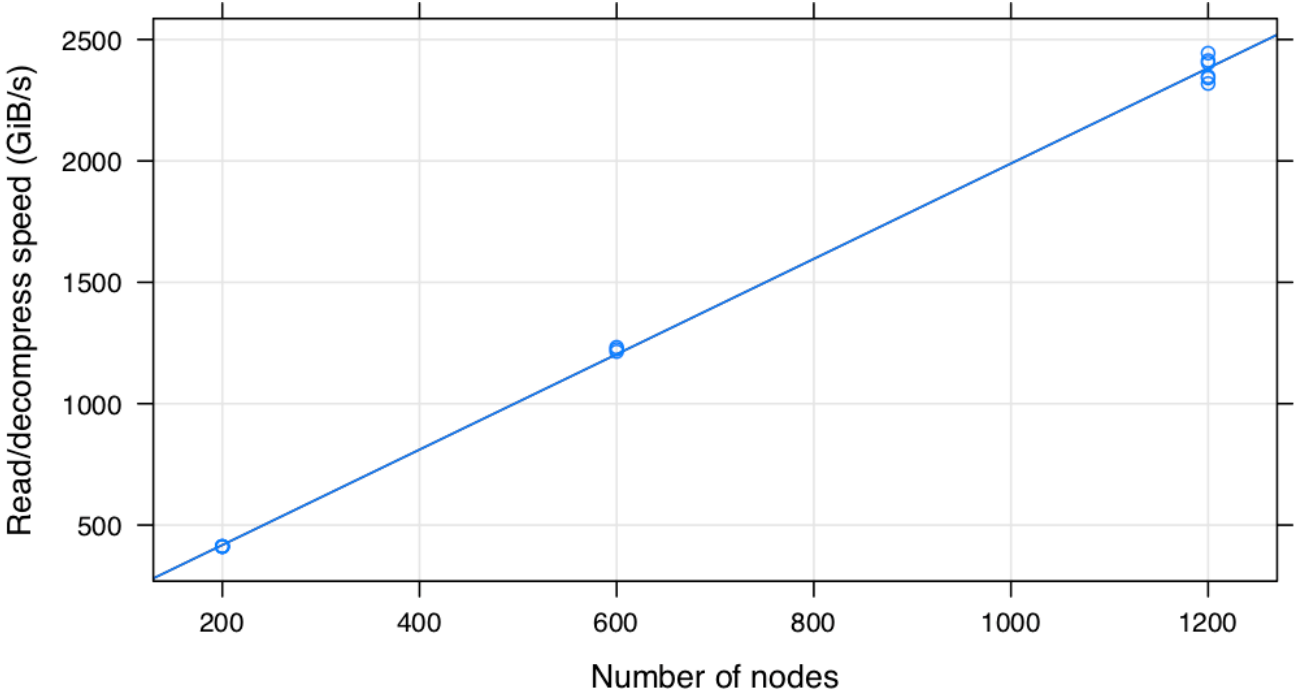
```
def transform_wires(wires):
    ftrans = numpy.fft.rfft(wires, axis=1)
    filtered = THRESHOLDS * ftrans
    return numpy.fft.irfft(filtered, axis=1)
```

Processing speed for full analysis being done



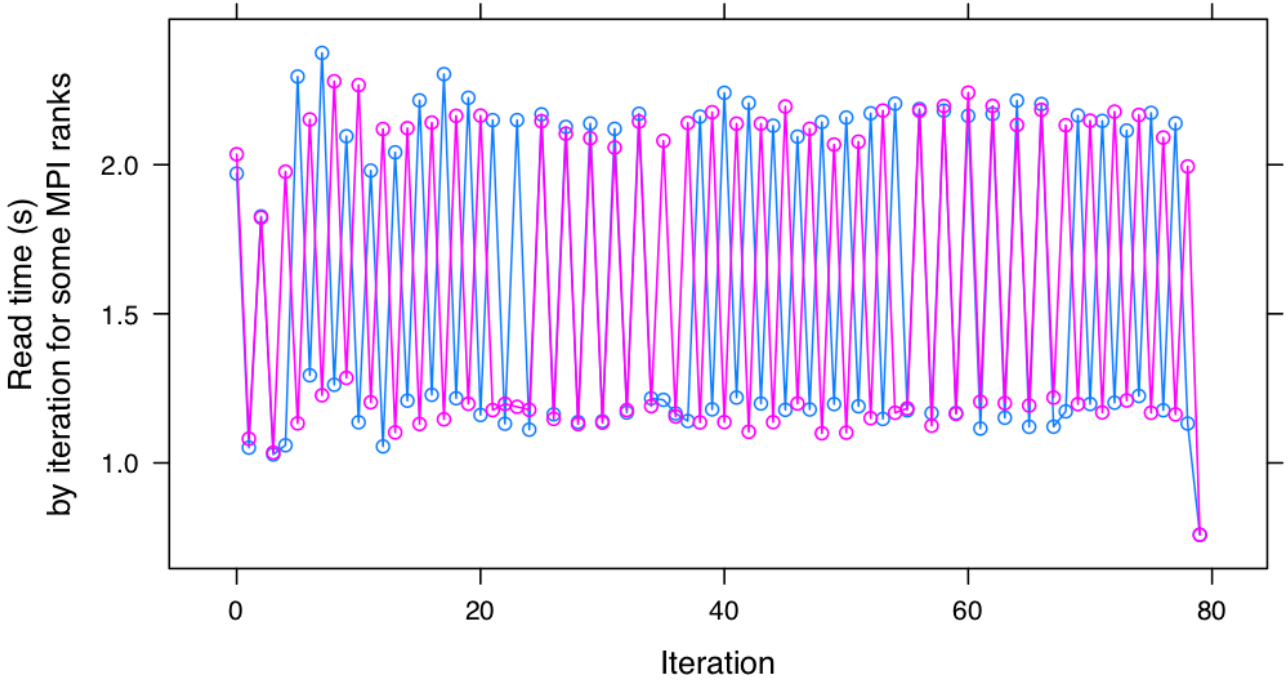
- Entire LArIAT dataset processed in three minutes (at 1200 nodes)
- Shows perfect scaling

Read speed – how does the I/O scale?



- Read + decompression speed for the whole application
- Nearly perfect strong scaling

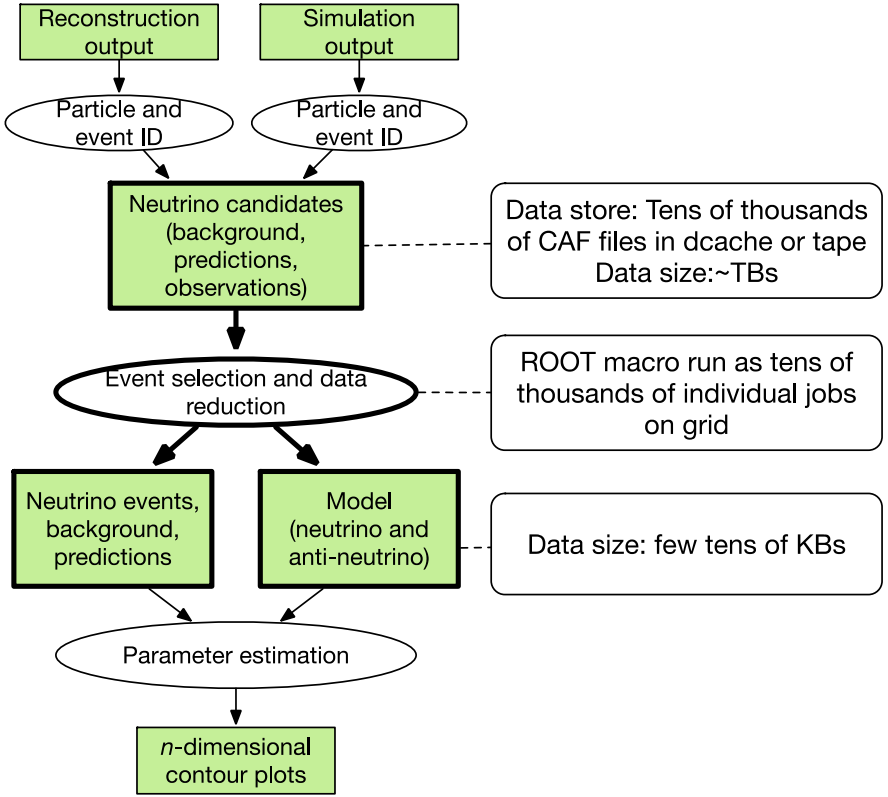
We should be able to do better …



- Different colors correspond to different ranks in the application
- Slower iterations within the application are twice as slow as faster ones (81 iterations in whole run)

Event selection for neutrino analysis

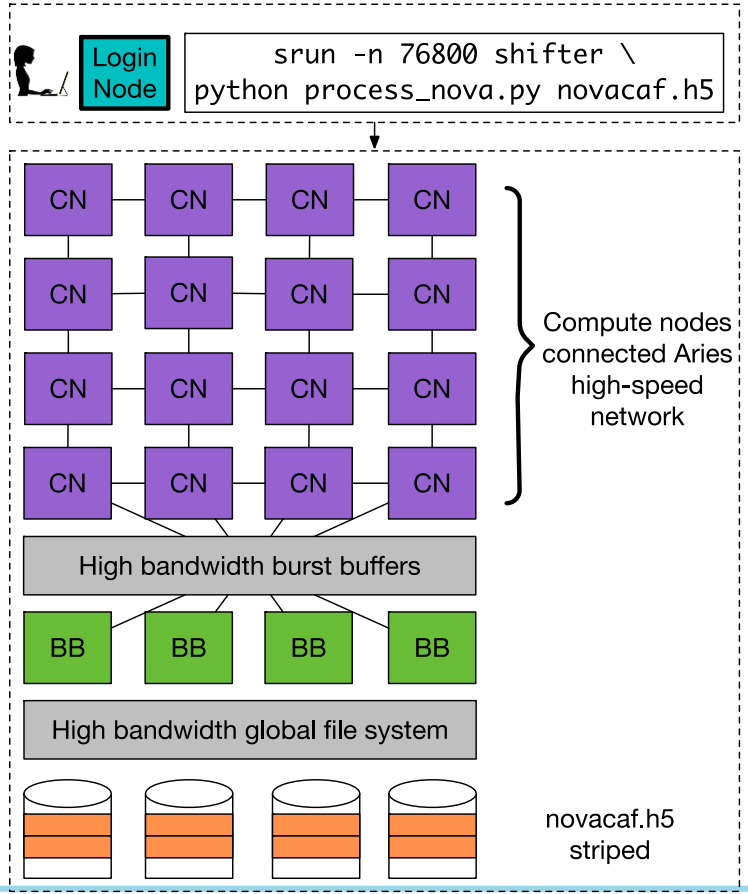
Traditional solution for oscillation parameter measurement



High-level organization of processing

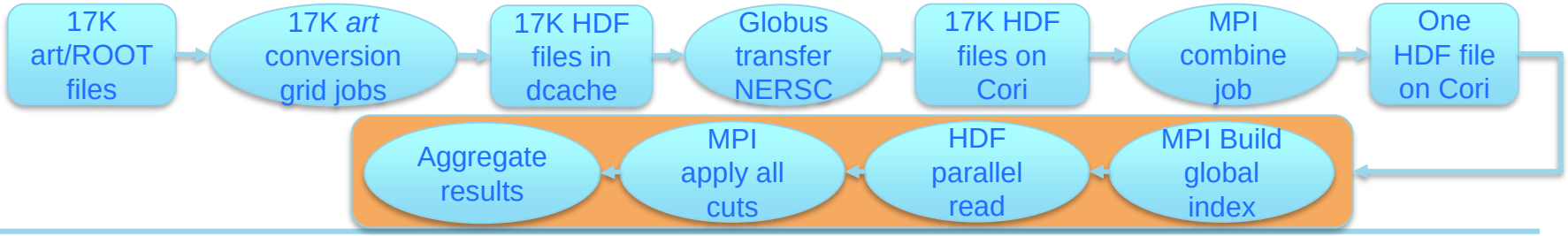
- We want to minimize …
  - *Reading*
  - *Communication* and *synchronization* between ranks
- We organize the data into a single HDF5 file, containing many different tables
  - some tables have one entry per slice
  - some have a variable number of entries per slice
- We want to process all data for a given *slice* in a single rank.
  - the *slice* is NOvA's "atomic" unit of processing, like a collider *event*.
  - for data that represent per-slice information, this is trivial
  - for other data, we need to do some work to ensure each rank has the correct data.

HPC solution



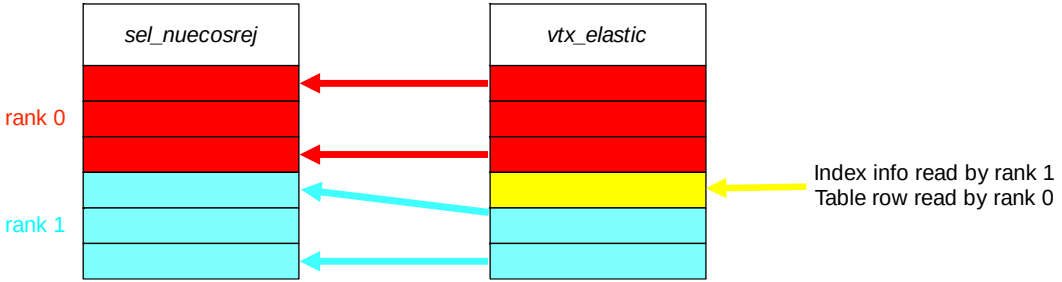
Parallel event pre-selection

- Current situation
  - NOvA slice data held in 17K ROOT files across
  - ~27 million events are reduced to tens using ROOT macros applying physics “cuts”
- New method
  - Data prepared for analysis using workflow shown below
  - End state: >50 groups (tables), each with many attributes



Distributing and reading information

- Each rank reads its “fair share” of index info from each table.
  - identifies which rank should handle which event, for most even balance
  - identifies range of rows in table that correspond to each event (all slices)



- Event “ownership” information distributed to all ranks
  - this assures no further communication between ranks is needed while evaluating the selection criteria on a slice-by-slice basis.
  - perfect data parallelism in running all selection code
- Each rank reads *only* relevant rows of relevant columns from relevant tables
  - all relevant data read by some rank
  - no rank reads the same data as another

Example selection code

```
def kNueSecondAnaContainment(tables):
    df = tables['sel_nuecosrej']
    return (df.distallpngtop > 63.0) & \
           (df.distallpngbottom > 12.0) & \
           (df.distallpngeast > 12.0) & \
           (df.distallpngwest > 12.0) & \
           (df.distallpngfront > 18.0) & \
           (df.distallpngback > 18.0)
```

- Selection can be done on multiple columns of a table.
- Logical operations are connected by & operator.
- Data parallelism is totally *implicit.*
- Returns an array with one logical value per *slice*.

```
def vtxelasticzCut(tables):
    df = tables['vtx_elastic']
    df['good'] =
        (df.vtxid == 0) & (df.npng3d > 0)
    KL = ['run', 'subRun', 'event', 'slice']
    return df.groupby(KL)['good'].agg(np.any)
```

- *vtx_elastic* table has one entry per *vertex;* may be more than 1 per slice.
- *groupby* combines results for all vertices in one slice.
- Returns an array with one logical value per *slice*.

Current status

- NOvA has taken ownership of our HDF “ntuple” production code
  - They will use this in their own future production.
  - Especially interested in using for machine learning; many tools work with HDF5 files.
- We will be comparing performance with C++-MPI implementation.
- Integration with larger workflow that is also part of the SciDAC project
  - use of changes in event selection criteria to evaluation systematic uncertainties in the mixing parameter measurements
  - one integrated MPI program, to take best advantage of HPC platform.

Object store for physics data

HEPnOS: Fast Event-Store for HEP (on HPC)

Goals:

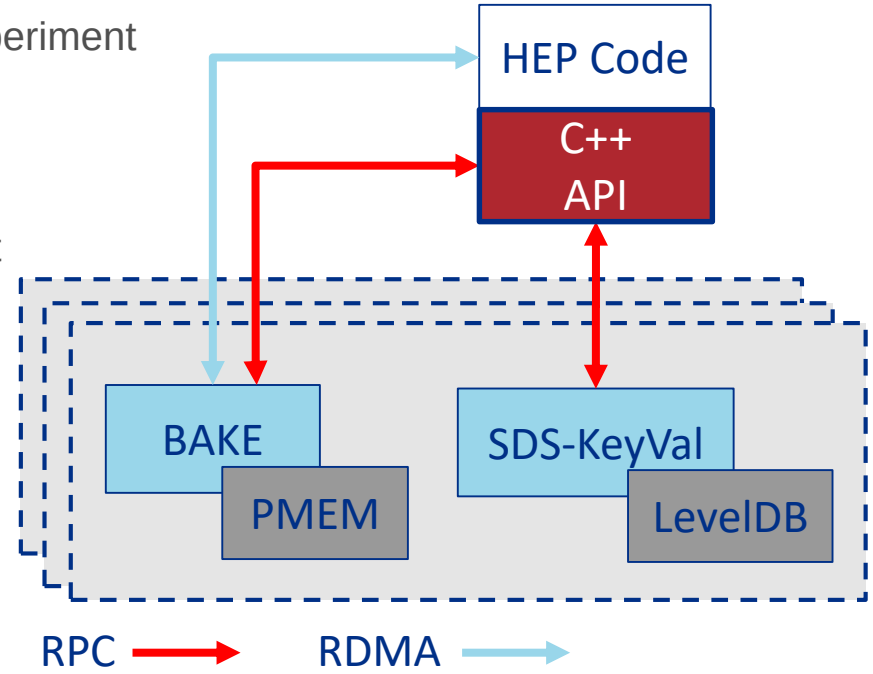
- Manage physics event data from simulation and experiment through multiple phases of analysis
- Accelerate access by retaining data in the system throughout analysis process
- Reuses components from Mochi ASCR R&D project

Properties:

- Write-once, read-many
- Hierarchical namespace (datasets, runs, subruns)
- C++ API (serialization of C++ objects)

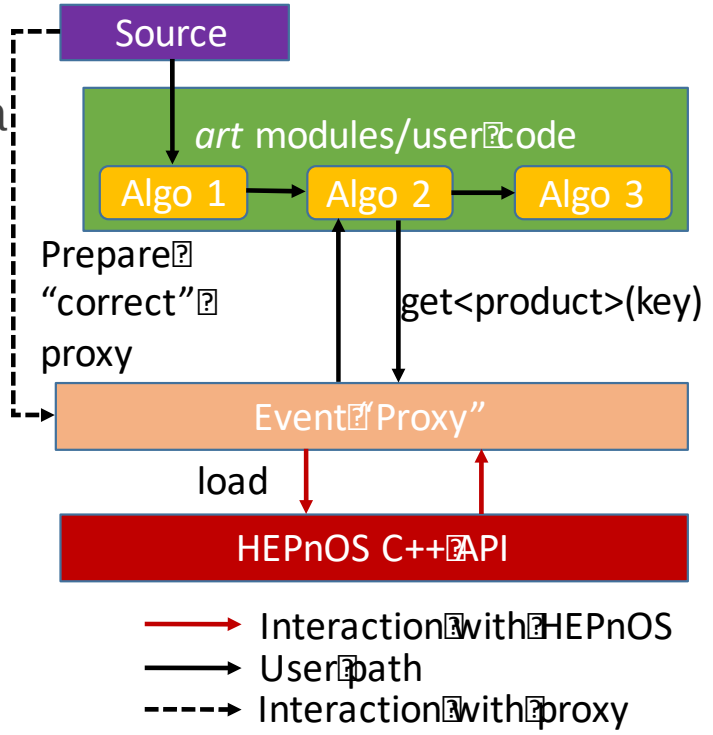
Components:

- Mercury, Argobots, Margo, SDSKV, BAKE, SSG
- **New code: C++ event interface**
  Map data model into stores



Our first use of HEPnOS

- Make high-volume reconstructed physics object data available to analysis workflows
  - Use existing *art* framework and gallery library
  - Starting point: Use actual LArSoft *Tracks*, *Hits*,and *Associations* from ProtoDUNE simulation
- Allow HPC facility services to distribute data at any scale, using existing HEP abstractions
  - Runtime ROOT File I/O replacement using HEPnOS
  - Include all levels (or layers) of data aggregation with metadata
- Data distribution and data parallelism implicit to user



Event currently interacts with *art*/ROOT File

Current status and future direction

- Prototype test programs are running:
  - one to read from existing art/ROOT data files, and to write to the new data store
  - one to read from the new data store, and verify the integrity of the data
- We are using Docker containers for easy portability of development environment
  - Some of us develop on macOS laptops, others on a variety of Linux installations
  - We will deploy to NERSC (through Shifter) and ALCF (through Singularity)

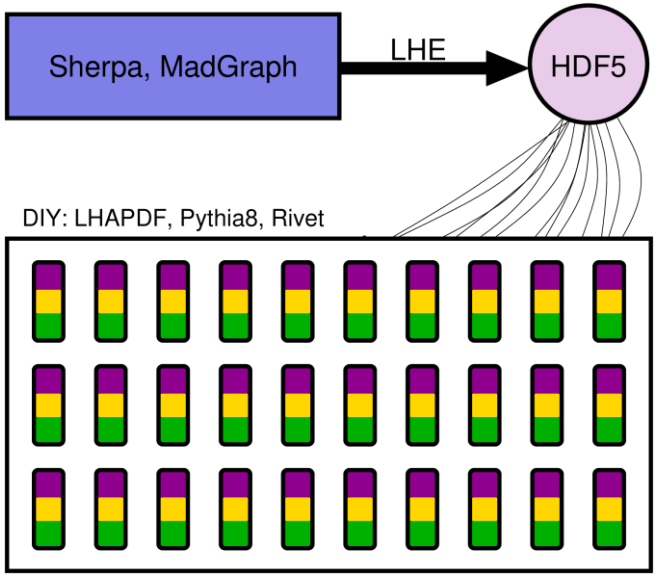
- The dataset (description and name) is included in the namespace
  - Interesting to have direct access to any part from any process on any node
  - Opens up new workflow possibilities
  - Can readily represent and access things below the event, such as NOvA slices

Physics generator data access

Matrix element (ME) calculations and physics generators

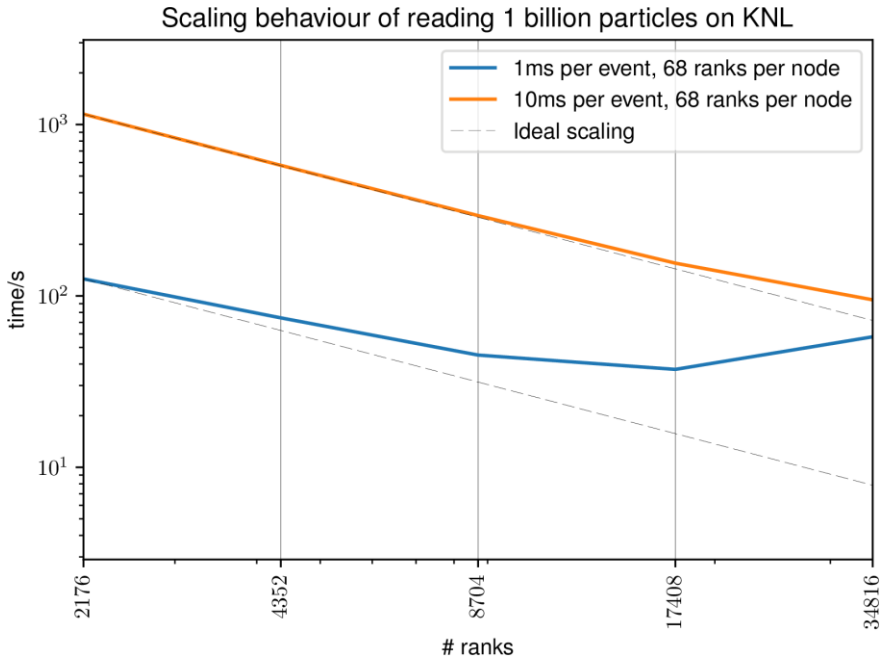
- Adequate predictions of many observables require input from matrix element generators
  - e.g. Getting angular distribution of jets correct
- ME generators (Sherpa, MadGraph) are used to generate high-multiplicity parton-level events
  - LHE description is the typical representation
  - Used as input to Pythia8 to get fully simulated events
- XML-based LHE data format is unsuitable for HPC
- Task is to write LHE data in HDF5 instead
  - We can accumulated all the XML data into one HDF5 file
  - Also working on writing HDF5 directly from Sherpa
- Will work seamlessly with our new DIY-based generator applications that tie together Pythia8, LHAPDF, and Rivet

Generator data available for any size study



- Organization of running Pythia8 on HPC facilities

- I/O will not be an issue



Summary

- This work may be useful to inform other projects that are ongoing or starting
- NOvA has embraced the work we are doing here
  - Took ownership of the HDF writer module
- Converting full HEP experiment datasets has been very difficult
  - Heavily influenced by ROOT tree structure (NOvA tuple organization)
  - Complexity of data structures (LArSoft RawDigits class and others)
  - Some data structures have been reorganized (vectorization becomes straightforward)
- Python was excellent for prototyping; further work is needed to determine if we are getting the best performance possible.
  - Pandas provides a powerful set of abstractions for analysis tasks
  - Comparisons of C++ and python/pandas forthcoming

Future directions

- Performance studies and tuning will continue
- Upcoming C++ codes will be using DIY, and possibly additional HPC-centric workflow tools such as Decaf
- Will be working with HEPCloud on integrating dataset handling.
- Looking into
  - Adding Summit as a platform
  - Making sure that analysis involving heterogeneous computing is handled
- As we more calculation towards reconstruction activities in the framework, the same techniques will be applied:
  - read the right information into memory,
  - use vectorized libraries for high-level operations on the data,
  - use the network to round up results that are distributed around the system

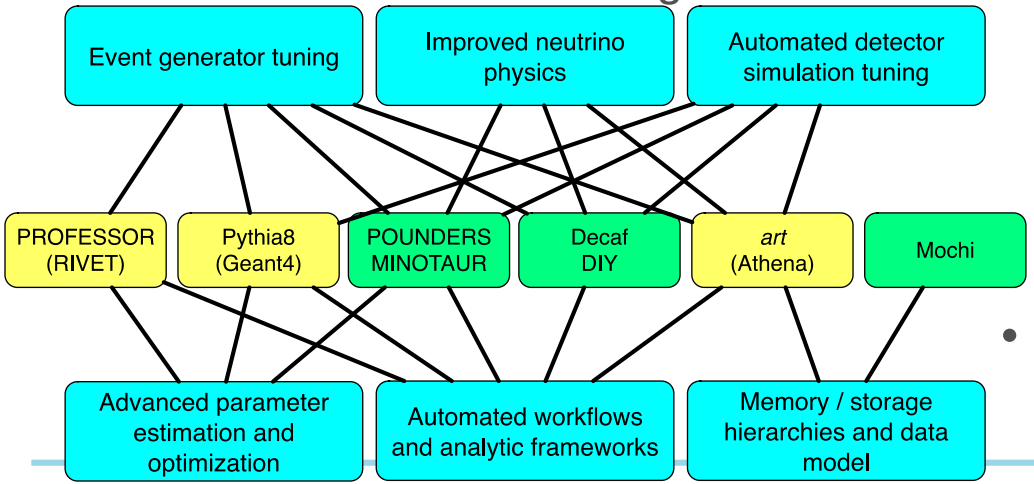
Acknowledgements

- People involved in one or more of these projects
  - M. Paterno (FNAL), T. Peterka (ANL), R. Ross (ANL), S. Sehrish (FNAL)
  - C. Green (FNAL), H. Schulz (University of Cincinnati), B. White (FNAL)
  - N. Buchanan (CSU, NOvA/DUNE), P. Calafiura (LBNL, LHC-ATLAS), Z. Marshall (LBNL, LHC-ATLAS), S. Mrenna (FNAL, LHC-CMS), A. Norman (FNAL, NOvA/DUNE), A. Sousa (UC, NOvA/DUNE)
  - A. Austin (ANL), S. Calvez (Colorado State University), P. Carns (ANL), P.F. Ding (FNAL), M. Dorier (ANL), D. Doyle (Colorado State University), X. Ju (LBNL), R. Latham (ANL), S. Snyder (ANL)

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program.

HEP Data Analytics on HEP: Goals

- Extend physics reach of LHC and neutrino experiments
  - Event generator tuning
  - Neutrino oscillation and cross-section measurements
  - Detector simulation tuning



- Transform how these physics tasks are carried out through ASCR math and data analytics
  - High-dimensional parameter fitting,
  - Workflows supporting automated optimization
  - Distributed dataset management storage and access (*in situ*) for experiment data
  - Introduction of data-parallel programming within analysis procedures
- Accelerate HEP analysis on HPC platforms

http://computing.fnal.gov/hep-on-hpc/