EPJ Web of Conferences **251**, 03033 (2021)
CHEP 2021
https://doi.org/10.1051/epjconf/202125103033

PandAna: A Python Analysis Framework for Scalable High Performance Computing in High Energy Physics

Micah Groh[1,2,*], *Norman* Buchanan[1], *Derek* Doyle[1], *James B.* Kowalkowski[3], *Marc* Paterno[3], and *Saba* Sehrish[3]

[1]Department of Physics, Colorado State University, Fort Collins, Colorado 80523, USA
[2]Department of Physics, Indiana University, Bloomington, Indiana 47405, USA
[3]Fermi National Accelerator Laboratory, Batavia, Illinois 60510, USA

Abstract. Modern experiments in high energy physics analyze millions of events recorded in particle detectors to select the events of interest and make measurements of physics parameters. These data can often be stored as tabular data in files with detector information and reconstructed quantities. Most current techniques for event selection in these files lack the scalability needed for high performance computing environments. We describe our work to develop a high energy physics analysis framework suitable for high performance computing. This new framework utilizes modern tools for reading files and implicit data parallelism. Framework users analyze tabular data using standard, easy-to-use data analysis techniques in Python while the framework handles the file manipulations and parallelism without the user needing advanced experience in parallel programming. In future versions, we hope to provide a framework that can be utilized on a personal computer or a high performance computing cluster with little change to the user code.

1 Introduction

High-Energy Physics (HEP) experiments continue grow in size and complexity, requiring the employment of sophisticated analytical and computational tools. Datasets are approaching the exabyte-scale leading to challenges in data handling and process distribution for analysis programs. These challenges are requiring HEP experiments to migrate to using High Performance Computing (HPC) facilities, tools, and techniques in order to efficiently perform physics analyses.

Experiment data are organized into a hierarchy of indices, run, subrun, and event, where different runs and subruns index the running conditions of the experimental apparatus and events can be full detector readouts with associated data products describing the event. Each event is independent from one another allowing for parallelization to be achieved by distributing subsets of a full dataset among a number of processes. Analysis applications read and operate on properties of the event and results are filled into histograms. Processed data are then aggregated by adding histograms. Scalability is limited in this approach by the total number of files in a dataset ($\mathcal{O}(10,000)$).

*e-mail: micah.groh@colostate.edu

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).

This paper introduces PandAna, an analysis framework built upon modern HPC tools and techniques. Though designed for the NOvA experiment, PandAna is an analysis framework designed to handle all types of HEP data. PandAna uses the HDF5 [1] file format, widely used for HPC applications, to support efficient and scalable storage. Python libraries such as *h5py* [2] and *mpi4py* [3] are utilized internally to provide easy-to-use parallel I/O and processing capabilities that remove scalability limitations of traditional HEP analyses without parallel programming experience. The PandAna interface is portable and platform-agnostic to support analysis applications run on personal and HPC machines. Within the last year, NOvA has used PandAna for basic data selection for machine learning particle identification algorithms.

1.1 Neutrino Experiments

The study of neutrinos is one of the main drivers of research in high energy physics [4]. They are the least understood particle in the standard model and could answer questions about the asymmetry between matter and antimatter. Neutrino experiments employ massive detectors which produce multi-petabyte datasets which capture the interactions of the neutrinos on different nuclear targets. The datasets are analyzed and used to extract fundamental properties of the neutrino such as neutrino oscillation parameters or the structure of neutrino interaction models.

The NOvA experiment has been operating two liquid scintillator-based neutrino detectors since 2014 [5]. The first is a 300 ton near detector located underground, on-site at Fermi National Accelerator Laboratory and the second is a 14 kt far detector on the surface in Northern Minnesota, USA. A beam of primarily muon neutrinos from the accelerator is sent through both detectors. The two detectors have produced in excess of 24 PB of data including the simulated datasets of the detectors. Combining these data NOvA has made precision measurements of neutrino oscillation parameters [6] and the near detector is capable of precision neutrino interaction cross section measurements [7].

The DUNE experiment is a future neutrino oscillation experiment utilizing four liquid argon time projection chambers (LArTPC) totalling 40 kt [8]. LArTPCs have spatial resolution of about 1 mm compared to about 1 cm for the scintillator detector used by NOvA. In addition, the beam will be upgraded to be more than three times as powerful. The DUNE experiment requires new improvements in analysis software and tools and utilization of advances in HPC to process and analyze the detector data. The new tools will allow DUNE to resolve the question of the neutrino mass hierarchy and set precise constraints on all neutrino oscillation parameters.

1.2 Experimental Needs

These experiments store data from the detectors in units or events determined by the experiment software and data acquisition. These might be a spill of neutrinos from a beam or the crossing of two particle beams. Detector data and reconstructed quantities from these events are stored as tabular data within files to be analyzed. These events can then be selected based on computed information from the reconstructed quantities and used to fit physics models.

For example, the NOvA experiment groups hits within the detector into "slices" based on their spatial and temporal position in the detector readout. These slices contain a single particle interaction and make the base unit of the neutrino analyses. Reconstruction tools are then used to identify useful quantities of the slice like the energy or scores for various particle types. In some cases, the reconstruction tools identify multiple objects relevant to the event such as tracks for the individual particles in the interaction. A set of selection criteria are then

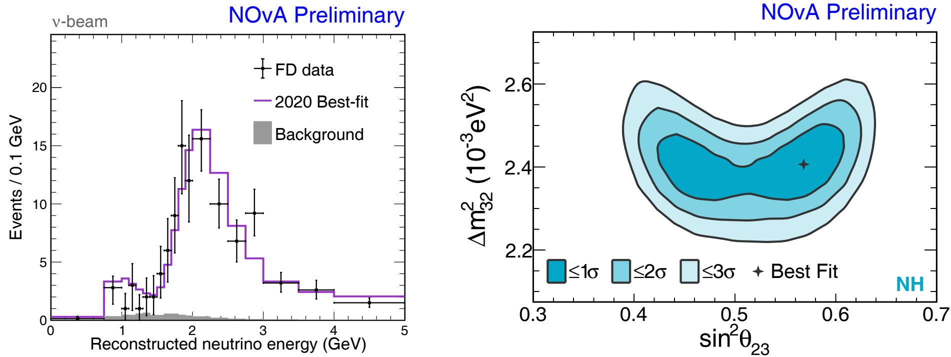

Figure 1. The NOvA oscillation analysis selects muon and electron neutrinos from the cosmic ray and beam backgrounds. Neutrino oscillation parameters can be measured by fitting the selected data to the neutrino oscillation model. The ν_μ data and fit are shown on the left and the resulting contours for the atmospheric parameters are shown on the right. Figures from Ref. [9].

used to identify the signal muon and electron neutrinos of interest. These data are then fit to the neutrino oscillation model and the parameters can be extracted. The measurements of NOvA's most recent analysis are shown in Fig. 1.

These analyses are traditionally done using frameworks based on ROOT [10]. In a ROOT analysis, data is read one event at a time and a decision is made on whether the selection criteria are met based on all the quantities related to the event.

The proposed tool in this paper uses modern HPC tools with easy-to-use libraries in Python to provide an alternative framework for data analysis. The most important feature is that tabular data is analyzed by column: a given quantity for all data is analyzed together, then, selection criteria are determined based on all quantities collectively.

2 PandAna Framework

PandAna is an analysis framework developed in python for the tidy-data model, which is essentially a new name for an old concept, the data matrix. Experiment datasets are stored as tabular data in HDF5 [1] files. Different types of objects are stored in different tables. Each row of a table corresponds to the observation of a single object of a given type (*e.g.* a slice, or a track, or a vertex). Within the tables, the data are stored as arrays corresponding to the columns of the table.

The collection of tables are related to each other by index columns, in a manner similar to the Boyce-Codd third normal form [11], so that the user can identify which track belongs to which vertex, and which vertex belongs to which slice.

At analysis time, the data is read into *numpy* [12] arrays and then used to construct *pandas* [13] dataframes with a different dataframe for each group. A dataframe is a two-dimensional data structure with rows and columns and are a high performance, easy-to-use data structure for data analysis. Rows in the dataframes correspond to individual observations and columns correspond to the properties of each observation. As the dataframe is constructed, columns are set as the dataframe index to uniquely identify each row.

With the dataframes in hand, the data can be analyzed, one column at a time, making use of efficient, vectored operations available in *pandas* and *numpy*.

2.1 Dependencies

The PandAna framework was designed to take advantage of efficient tools developed for Python. The *h5py* library provides a pythonic interface for interacting with HDF5 files. The *mpi4py* library allows for parallel computation within a Python program. As data is read from the HDF5 files, it is divided evenly amongst each rank of the MPI program such that each rank analyzes a unique subset of the data.

The goal of the PandAna toolkit is that interactions with *h5py* and *mpi4py* are handled internally to the framework. Users interact primarily with *numpy* arrays and *pandas* dataframes. When developing an analysis in PandAna, the user reads a set of dataframes and then useful quantities can be computed. The construction of the dataframes is handled by the framework.

Importantly, no experiment-specific compiled libraries are necessary when developing an analysis in PandAna. This allows for greater mobility in code execution; an analysis can be developed on experiment-provided machines and then easily ported to a personal computer or a HPC cluster.

2.2 Proxy DataFrame

The proxy dataframe is a special class which inherits from the *pandas* dataframe class used to reduce the memory requirements and computation time of the workflow. First, the framework user defines the total set of computations they want executed. Empty proxy dataframes are used to evaluate the computations, but each time a dataframe is requested, the necessary group and datasets within the HDF5 files are stored in a cache. Since the data is empty and no I/O is performed, this process takes negligible time. Using the cache, the proxy dataframes are filled with just the columns which will be needed for the users analysis, reducing the time spent in file I/O, and then the computations are actually performed on the complete dataframes.

The second function of the proxy dataframe is to yield reduced datasets. Each time the user defines an additional selection criteria, the set of event indices which pass that criteria are returned. Before any additional computations are done, the indices are used to filter rows in all proxy dataframes.

By constraining the number of columns in each dataframe, and only performing computations on events which pass previous selection criteria, significant speedup in the workflow is achieved.

3 PandAna Analysis

PandAna provides four main tools to aid in the development of an analysis: *loader* which contains the list of files to be analyzed, *var* which is a computed quantity, *cut* which defines a selection criteria, and *spectrum* which contains resulting dataframes.

The *loader* is constructed with a list of HDF5 files to be analyzed. Within the framework, the *loader* handles the file I/O and the construction of dataframes. No manipulations are done by the user.

A variables or *var* is a quantity which can be computed from data within the HDF5 files represented by a dataframe. This can be either a single dataset read directly from the files or a more complicated function involving several datasets. In cases where two quantities will be compared to one another, a *var* can return a dataframe with multiple columns. An example of a *var* used by the NOvA experiment for the number of hits in a slice is shown below.

```
def kSliceHits(tables):
    NHitDF = tables['rec.slc']['nhit']
    return NHitDF
kSliceHits = Var(kSliceHits)
```

Here, *tables* is a python dictionary of dataframes, representing tabular data. The 'rec.slc' group represents data pertaining to slice level quantities which are grouped into a dataframe. The 'nhit' column of the dataframe is returned by the *var*. The final line uses the defined function to create an object of the *var* class.

A *cut* is a dataframe of boolean values defining whether an event passes a given selection criteria. *Cuts* can be created from a *var* or using a defined function in the same style as a *var*, but a *cut* should return boolean values. *Cuts* define the selection criteria for the proxy dataframe index filtering. When two *cuts* are combined together the second will only be evaluated on indices that pass the first *cut*. An example of a *cut* used by the NOvA experiment requiring that all tracks in the slice stop before the edge of the detector is shown below.

```
def kTracksContained(tables):
    TrackDF = tables['rec.trk.kalman.tracks'][['start.z', 'stop.z']]
    TrackDF = TrackDF.max(axis=1)
    TrackDF = TrackDF < 1200
    TrackDF = TrackDF.groupby(level=KL).agg(np.all)
    return TrackDF
kTracksContained = Cut(kTracksContained)
```

Here, *TrackDF* is a two column dataframe with the start and stop coordinates of each track. Note that tracks are a reconstructed object and there could be more than one per slice. A condition is set requiring the larger of the two coordinates to be less than 1200 cm, the edge of the active region of the detector. A reduction is performed, grouping together all tracks belonging to the same event and requiring them all to satisfy the condition. The resulting dataframe will have a single column with one boolean for each event.

Finally, a spectrum is the basic PandAna object for creating dataframes for the user. It is constructed with a *loader*, a *cut*, and a *var* where the *cut* defines some selection criteria for events in the final dataframe and the *var* is the quantity of interest. At construction time, the cache for the proxy dataframes are constructed from the operations used in the *cut* and *var* functions. Once all desired spectra are created, a call to *loader* evaluates all associated spectra in sequence.

When writing functions for *cuts* or *vars*, built in functions within *numpy* and *pandas* should be used when they are available. These Python libraries provide a wide variety of vector operations which cover most common manipulations a user may want to do. When more complicated operations are needed, *numba* [14] should be used to accelerate the manipulation. *Numba* is a python compiler for vector and numerical operations. Numba can compile complicated operations on numpy arrays into just-in-time optimized machine code.

An example of a distribution created using these tools for the NOvA experiment is shown in Fig. 2.

The standard NOvA event selection utilizes a framework built around ROOT. In parallel evaluation, each file is distributed to each rank. The result of each rank must be combined in a separate process. This method is limited to parallel evaluation up to the number of individual files being analyzed. PandAna has no such limitation and anlyses can be run in parallel up to the total number of events being analyzed. Comparing an event selection of data in 2000 files, representing a small subsample of the complete NOvA dataset of roughly 10 million events,

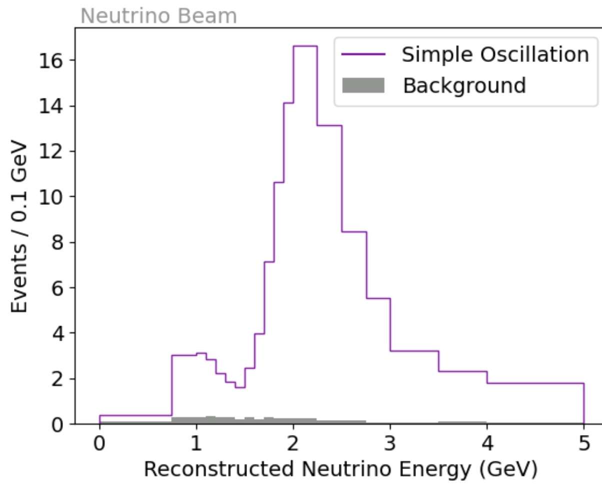

Figure 2. An approximate recreation of the NOvA reconstructed energy distribution in Fig. 1 using the PandAna framework.

the ROOT analysis took 44 seconds to run distributed across 2000 MPI ranks, not including the additional time to combine the results. The PandAna analysis took a comparable time to run on only 256 MPI ranks still allowing for further scaling with more ranks.

4 Future Work

Physicists want to concentrate on their analysis, not on programming. More than most languages, Python makes this possible. But Python is often viewed as relatively slow and thus unsuitable for dealing with large data volumes. PandAna makes use of efficient and widely-used libraries, such as *pandas* and *numpy* for both expressiveness and speed. Our future development of PandAna will extend the system to allow efficient use of high performance computing systems for handling of multi-terabyte data samples.

We are in the process of integrating MPI, the Message Passing Interface, into the PandAna framework. MPI is the dominant parallel programming model in HPC; every HPC center has an MPI implementation optimized for the network interconnects available within the facility. MPI also works on laptops and can be readily deployed on grid nodes and computer clusters. The programming model uses multiple operating system processes communicating using messages, rather than shared-memory threads. Because Python does not support the use of multiple simultaneously active threads, MPI is an ideal candidate for leveraging many cores for a single program. *mpi4py* is the popular Python module that implements MPI. In our use, analysis code (almost) never makes calls to the MPI library (and mostly doesn't need to know that it exists); the parallelism in an application is almost entirely implicit.

Each HPC installation has a global parallel filesystem, and HDF5 has efficient drivers to take advantage of all popular parallel filesystems. To take advantage of the resulting high-bandwidth reading capabilty of an HPC system, we are currently evaluating a concurrent reading method for HDF5 files which is dependent on our schema. Our method divides the rows of the tables as evenly as possible among the processes (ranks, in MPI terms), taking care of reading all the data from a given slice (regardless of the table from which it is read) in

a single process, and that no data are duplicated between processes. We use library functions provided by the MPI implementation to coalesce the final data produced by all the processes.

5 Acknowledgements

We thank the NOvA collaboration for the use of its simulation analysis files in the development of this framework. Particular thanks to NOvA collaborators Nitish Nayak, Fernanda Psihas, and Karl Warburton.

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, grant "HEP Data Analytics on HPC", No. 1013935.

References

[1] The HDF Group (2000-2010), http://www.hdfgroup.org/HDF5

[2] A. Collette, https://www.h5py.org

[3] L. Dalcan, R. Paz, M. Storti, Journal of Parallel and Distributed Computing **65**, 1108 (2005)

[4] S. Ritz et al., *Building for Discovery: Strategic Plan for U.S. Particle Physics in the Global Context*, https://science.energy.gov/~/media/hep/hepap/pdf/May-2014/FINAL_P5_Report_Interactive_060214.pdf (2014)

[5] D.S. Ayres et al. (NOvA), *The NOvA technical design report*, FERMILAB-DESIGN-2007-01 (2007)

[6] M.A. Acero et al. (NOvA), Phys. Rev. Lett. **123**, 151803 (2019), [arXiv]1906.04907

[7] M.A. Acero et al. (NOvA), Phys. Rev. D **102**, 012004 (2020), [arXiv]1902.00558

[8] B. Abi et al. (DUNE), JINST **15**, T08008 (2020), [arXiv]2002.02967

[9] A. Himmel, *New Oscillation Results from the NOvA Experiment* (2020), the XXIX International Conference on Neutrino Physics and Astrophysics

[10] R. Brun, F. Rademakers, Nucl. Instrum. Meth. A **389**, 81 (1997)

[11] M. Arenas, *Boyce-Codd Normal Form* (Springer US, Boston, MA, 2009), pp. 264–265, ISBN 978-0-387-39940-9, https://doi.org/10.1007/978-0-387-39940-9_1245

[12] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith et al., Nature **585**, 357 (2020)

[13] Wes McKinney, *Data Structures for Statistical Computing in Python*, in *Proceedings of the 9th Python in Science Conference*, edited by Stéfan van der Walt, Jarrod Millman (2010), pp. 56 – 61

[14] S.K. Lam, A. Pitrou, S. Seibert, *Numba: A LLVM-Based Python JIT Compiler*, in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* (Association for Computing Machinery, New York, NY, USA, 2015), LLVM '15, ISBN 9781450340052, https://doi.org/10.1145/2833157.2833162