Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

Declarative paradigms for analysis description and implementation

Alberto Annovi[1], Tommaso Boccali[1], Paolo Mastrandrea[1], Andrea Rizzi[1,2], Francesco Vaselli[1,3]

Motivation

The progress in the research in High Energy Physics (HEP) requires, among its main ingredients, the acquisition, storage and analysis of larger and larger data samples, pushing for the adoption and development of state-of-the-art computing technologies.

The adoption of approaches able to exploit the new hardware architectures plays a pivotal role in boosting data processing speed, resources optimisation, analysis portability and analysis preservation.
The scientific collaborations in the field of HEP (e.g. the Large Hadron Collider experiments, the next-generation neutrino experiments, and many more) devote increasing resources to the development and implementation of bleeding-edge software technologies, pushing the reach of the single experiment and the whole HEP community.

The introduction of declarative paradigms in the analysis description and implementation is gaining interest and support in the main collaborations [1]. This approach can simplify and speed-up the analysis description phase, support the portability of an analysis among different datasets/experiments, and strengthen the preservation of the results.
Furthermore, this approach - providing a deep decoupling between the analysis algorithm and back-end implementation - is a key element for present and future processing speed.

Paradigms for analysis description and implementation

Main paradigm approaches — Wikipedia [2]

There are two main approaches to programming:

- Imperative programming – focuses on how to execute, defines control flow as statements that change a program state.
- Declarative programming – focuses on what to execute, defines program logic, but not detailed control flow.

Imperative paradigms have been preferred for analysis description and implementation due to a more straightforward application for “simple” tasks and linear/serial computing tools. What has changed in the last decade?

- hardware: parallelism/multithreading
- software: more expressive programming languages (Python, C++ 17/20/23)
- tasks: increased complexity, increased data size (analyses, combinations)

Benefits of a (more) declarative paradigm:

- **Deeper decoupling** between algorithm and implementation
  - Faster analysis development
  - Wider portability of an analysis (different datasets/experiments)
  - Stronger preservation of the results
- **Better scaling** of development and preservation for increasing complexity of the algorithms and size of the data
- Better support for automatic (technical) optimisation
- Improve the support for **parallelization** of the tasks (multithreading/GPU)
- More flexibility: e.g. different backend processors

Data-format interface

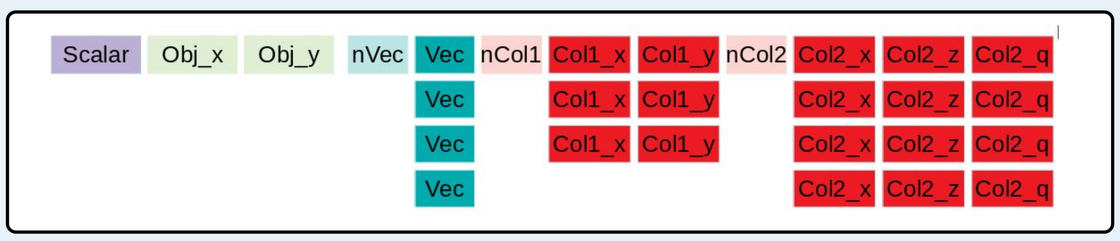
In principle 3 equivalent data-formats are involved in an analysis process:

A. data-format used inside the framework for variables manipulation
B. data-format used in the description of the algorithm by the user
C. data-format used in the encoding of the input data to be processed

While A and B can easily be unified for most applications, C is experiment-dependent: the introduction of a **translation** step is needed in order to provide

- extended analysis portability among datasets/experiments
- robust support for data-format evolution

Moving towards columnar analysis: the storage of event data as Structure of Arrays (SoA) is generally preferred to the Array of Structures (AoS) due to better memory performance and better support to parallelization and multi-threading.



SoA data representation for nanoAOD [8] format developed by the CMS experiment

Declarative analysis framework

Main features required for a declarative analysis framework:

- full set of declaration instructions able to describe the analysis algorithms
- consistent and portable representation of the analysis
- interface with backend processors (e.g. ROOT[3], RDataFrame[4], PyROOT[5], NumPy[6], …)

Features under development:

Extension to multiple input data-formats:

- translation layer between analysis description and backend processor
  - e.g. porting of analysis developed on **nanoAOD** [8] (CMS) to **PHYSLITE** [9] (ATLAS)

Extension to full analysis chain:

- full set of declaration able to describe all analysis/combination tasks

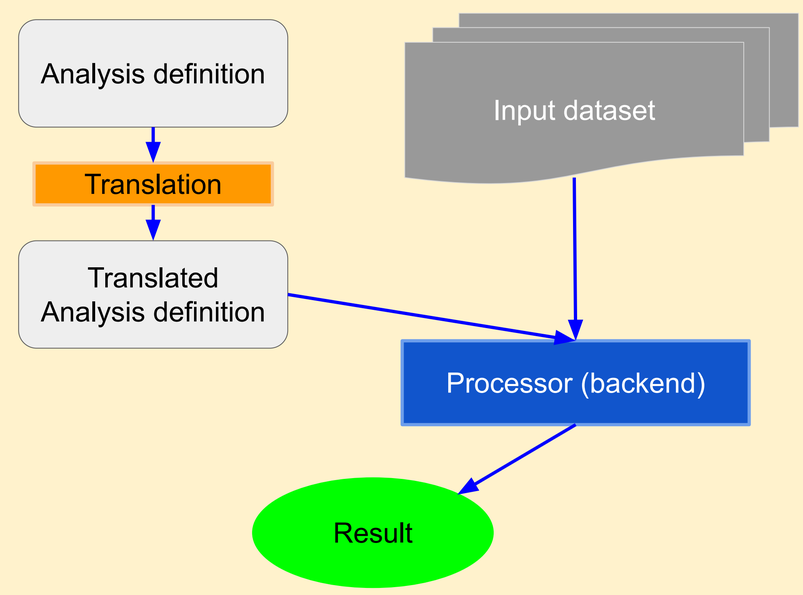
Working example: the *Natural Analysis Implementation Language* (NAIL) [7], an embedded declarative language, developed in the context of the CMS collaboration, implemented in Python and based on nanoAOD [8] data format. NAIL allows to specify the event-by-event processing actions in a declarative form. Analysis variations for optimizations and systematic uncertainties evaluation are automatically derived from the event processing computational graph. Currently ROOT's RDataFrame is used as backend for a concrete implementation of the event processing, supporting parallelization and lazy evaluation.

Support to different input-data formats: extends portability and preservation

Multiple input-data formats

A translation layer between the analysis declaration and the backend processing can provide support for:

- input dataset encoded in different formats for the same analysis
- transparent support for data-format evolution for the same input dataset



Extension to full analysis chain: extend advanced platform integration and acceleration, portability and preservation

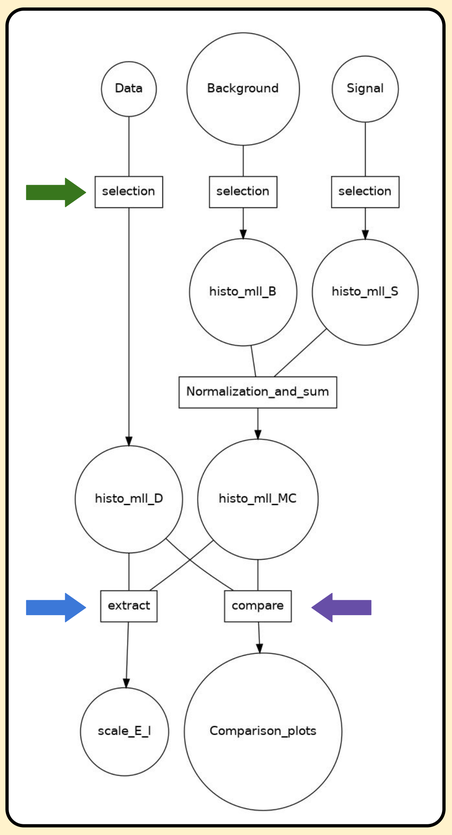
Full analysis chain

The extraction of a valuable result from an input dataset is usually based on procedures which require multiple steps, each processing different kind of information and different algorithms.

Full tracking of the processed steps, via hashing of the processing steps, provide support for optimal segmentation of the tasks, optimization of the computing resources, optimization of the development time and result preservation.

Example steps for the application of a declarative paradigm to a full-chain simplified energy calibration procedure:

- Sample preparation
- **Event-by-event processing** (e.g. NAIL)
- Snapshot/data reduction
- **Combination / comparison of distributions**
- **Statistical analysis / Extraction of results**
- Selective / incremental execution
  - for both development and production phases



References

1. https://indico.cern.ch/event/769263/
2. https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms
3. https://root.cern.ch//
4. https://root.cern/doc/master/classROOT_1_1RDataFrame.html
5. https://uproot.readthedocs.io/en/latest/
6. https://numpy.org/
7. https://github.com/arizzi/nail
8. https://iopscience.iop.org/article/10.1088/1742-6596/1525/1/012038
9. https://cds.cern.ch/record/2857821

1: INFN - Istituto Nazionale di Fisica Nucleare
2: University of Pisa
3: Scuola Normale Pisa

This work is supported by ICSC – Centro Nazionale di Ricerca in High Performance Computing, Big Data and Quantum Computing, funded by European Union – NextGenerationEU