Introduction to Coffea and its Applications to High Energy Physics

Daniel Ocampo Henao
Universidad de Antioquia

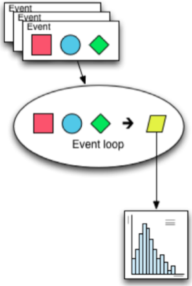
VII Uniandes Particle Physics School - Bogota - 07.12.2022

Event loop paradigm vs Columnar analysis

Text generated by ChatGPT. Image designed by Luigui.

Event loop paradigm

Traditionally HEP analysis are made using a event loop paradigm



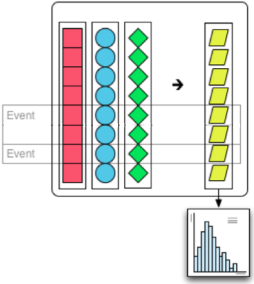
Event loop

- Load variables for a given event
- Compute observables
- Fill histograms
- Repeat

```
ROOT.gInterpreter.Declare('''
void compute(TH1D& roothist, TTree& roottree) {
    UInt_t nMuon;
    float Muon_pt[50];
    float Muon_eta[50];
    float Muon_phi[50];
    int32_t Muon_charge[50];

    roottree.SetBranchStatus("*", 0);
    roottree.SetBranchStatus("nMuon", 1);
    roottree.SetBranchStatus("Muon_pt", 1);
    roottree.SetBranchStatus("Muon_eta", 1);
    roottree.SetBranchStatus("Muon_phi", 1);
    roottree.SetBranchStatus("Muon_charge", 1);

    roottree.SetBranchAddress("nMuon", &nMuon);
    roottree.SetBranchAddress("Muon_pt", Muon_pt);
    roottree.SetBranchAddress("Muon_eta", Muon_eta);
    roottree.SetBranchAddress("Muon_phi", Muon_phi);
    roottree.SetBranchAddress("Muon_charge", Muon_charge);

    for (int index = 0; index < 100000; index++) {
        roottree.GetEntry(index);
        if (nMuon >= 2 && Muon_charge[0] + Muon_charge[1] == 0) {
            float mu1_pt = Muon_pt[0];
            float mu2_pt = Muon_pt[1];
            float mu1_eta = Muon_eta[0];
            float mu2_eta = Muon_eta[1];
            float mu1_phi = Muon_phi[0];
            float mu2_phi = Muon_phi[1];
            roothist.Fill(
                sqrt(2*mu1_pt*mu2_pt*(cosh(mu1_eta - mu2_eta) - cos(mu1_phi - mu2_phi)))
            );
        }
    }
}
''')
```

(Jim Pivarski's uproot-awkward tutorial)

Columnar analysis



Columnar

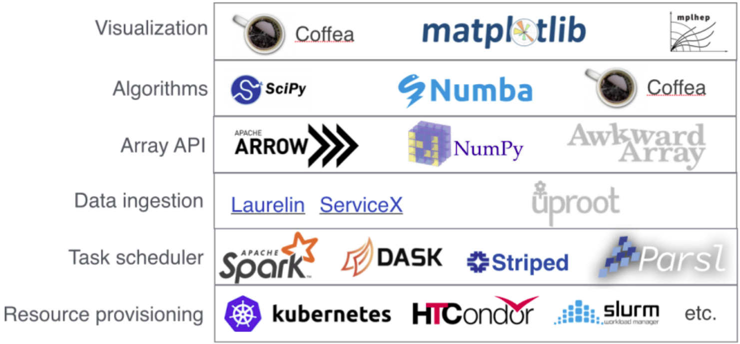
- Load variables of interest for all events into an array
- Compute observables evaluating array expressions
- Fill histograms with arrays

```python
# read data
muons = events.arrays(
    ["pt", "eta", "phi", "charge"],
    aliases={"pt": "Muon_pt", "eta": "Muon_eta", "phi": "Muon_phi", "charge": "Muon_charge"},
    array_cache=None,   # no cheating!
)

# compute
cut = (ak.num(muons.charge) >= 2) & (ak.sum(muons.charge[:, :2], axis=1) == 0)
mu1 = muons[cut, 0]
mu2 = muons[cut, 1]
h = hist.Hist.new.Reg(120, 0, 120, name="mass").Double()
h.fill(np.sqrt(2*mu1.pt*mu2.pt*(np.cosh(mu1.eta - mu2.eta) - np.cos(mu1.phi - mu2.phi))))
```

(Jim Pivarski's uproot-awkward tutorial)

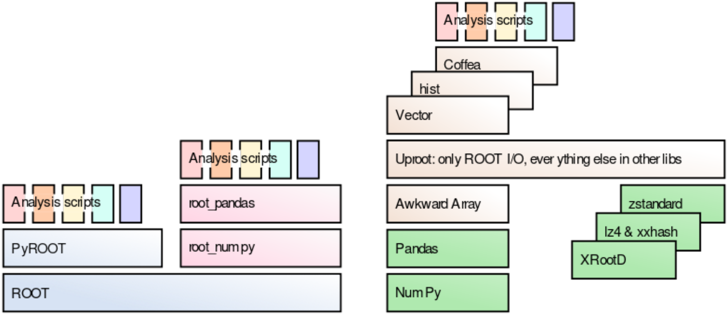
What is Coffea?

- Coffea provides basic tools and wrappers for enabling not-too-alien syntax when running columnar collider HEP analysis
- Coffea is part of the pythonic HEP ecosystem.



Uproot

- Uproot is a library for reading and writing ROOT files in pure Python
- Uproot uses Numpy to cast blocks of data from ROOT files as Numpy arrays.
- Primarily intended to stream data into machine learning libraries in Python.



Awkward

Awkward Array is a library for manipulating JSON-like data using NumPy-like idioms.

```
jets = ak.Array([
    [{"pt": [125, 210], "eta": [-1.2, 1.5], "phi": [1.2, 2.4]}],
    [{"pt": 50, "eta": 0.1, "phi": 1.5}],
    [{"pt": [223, 140, 234], "eta": [1.5, -0.5, 1.4], "phi": [2.3, 4.2, 3.4]}]
])
```

Arrays are dynamically typed, but operations on them are compiled and fast.

```
muon_pairs = ak.combinations(events.Muon, 2)

mu1, mu2 = ak.unzip(muon_pairs)

dimuon_mass = np.sqrt(
    2 * mu1.pt * mu2.pt * (np.cosh(mu1.eta - mu2.eta) - np.cos(mu1.phi - mu2.phi))
)
```

Corrections

- Scale factors and corrections are usually required in a typical HEP analysis.
- Coffea and Correctionlib provide well-structured data formats and evaluation tools suitable for use in python programs.

```
📈 Collisions17_UltraLegacy_goldenJSON (v0)
No description
Node counts: Category: 1, Binning: 3
┌──────────── ▶ input ────────────┐ ┌──────────── ▶ input ──────────┐
│ NumTrueInteractions (real)      │ │ weights (string)              │
│ Number of true interactions     │ │ nominal, up, or down          │
│ Range: [0.0, 99.0), overflow ok │ │ Values: down, nominal, up     │
└─────────────────────────────────┘ └───────────────────────────────┘
┌──────────── ◀ output ─────────────────┐
│ weight (real)                         │
│ Event weight for pileup reweighting   │
└───────────────────────────────────────┘
```

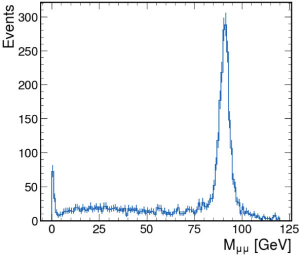
```python
# correction set
cset = correctionlib.CorrectionSet.from_file(
    get_pog_json(json_name="pileup", year="2017")
)

# scale factors
values = {}
values["nominal"] = cset["Collisions17_UltraLegacy_goldenJSON"].evaluate(
    NumTrueInteractions=ak.to_numpy(events.Pileup.nPU), weights="nominal"
)
values["up"] = cset["Collisions17_UltraLegacy_goldenJSON"].evaluate(
    NumTrueInteractions=ak.to_numpy(events.Pileup.nPU), weights="up"
)
values["down"] = cset["Collisions17_UltraLegacy_goldenJSON"].evaluate(
    NumTrueInteractions=ak.to_numpy(events.Pileup.nPU), weights="down"
)
```

Histograms

Hist is a powerful histogramming tool within the python HEP ecosystem for analysis based on boost-histogram

- N-dimensional histograms
- Discrete and dense axis: Regular, Boolean, Variable, Integer, IntCategory and StrCategory.
- Useful methods to transform and index histograms
- Plotting via matplotlib or mplhep: stacked and normalized plots, ratio plots, 2D plots, etc.

```python
import hist

h = hist.Hist(
    hist.axis.Regular(
        bins=120,
        start=0,
        stop=120,
        name="dimuon_mass",
        label="$M_{\mu\mu}$ [GeV]"
    )
)
h.fill(ak.flatten(dimuon_mass))
h.plot1d()
```



Processors

Coffea analyses are written in a “Processor” class.

- **__init__**: Define an accumulator object (histogram, dictionary, DataFrame or array) that will be fill later.
- **process**: Implement the analysis (observables, regions, corrections, etc) and fill the accumulator object.
- **postprocess**: Manipulate the accumulator object.

```python
class Processor(processor.ProcessorABC):
    def __init__(self):
        dataset_axis = hist.Cat("dataset", "")
        # Split data into 50 bins, ranging from 0 to 100.
        MET_axis = hist.Bin("MET", "MET [GeV]", 50, 0, 100)

        self._accumulator = processor.dict_accumulator({
            'MET': hist.Hist("Counts", dataset_axis, MET_axis),
        })

    @property
    def accumulator(self):
        return self._accumulator

    def process(self, events):
        output = self.accumulator.identity()

        dataset = events.metadata["dataset"]
        MET = events.MET.pt

        output['MET'].fill(dataset=dataset, MET=MET)
        return output

    def postprocess(self, accumulator):
        return accumulator
```

Executors

The Processor class gets deployed on an executor, which chunks up input data and feeds it in.

```
import coffea.processor as processor

fileset = {'data': ['root://xcache//store/SomeData.root']}

output = processor.run_uproot_job(
    fileset=fileset,
    treename="Events",
    processor_instance=Processor(),
    executor=processor.dask_executor,
    executor_args={
        'client': client,
        'schema': processor.NanoAODSchema
    },
)
```

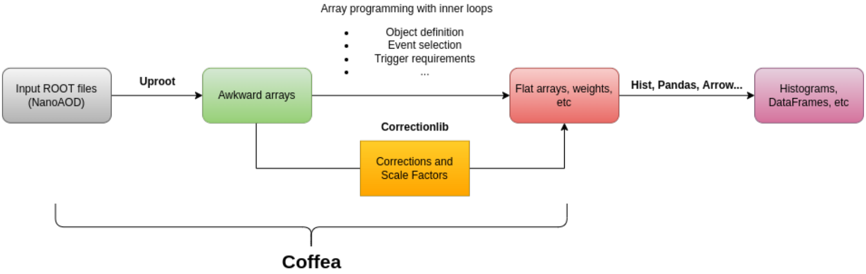
Local executors:

- iterative
- futures

Distributed executors:

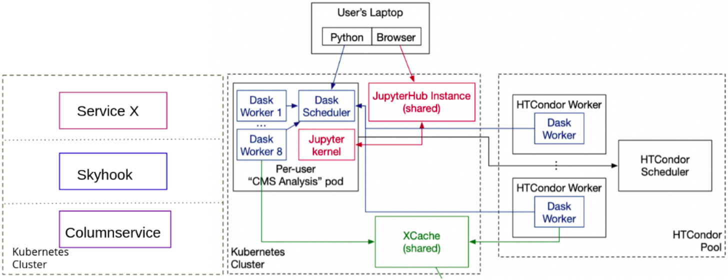
- dask
- parsl
- spark

Coffea workflow



Coffea-Casa

Coffea-casa is a prototype of an analysis facility that provides services for "low latency columnar analysis".



- Coffea-Casa uses JupyterLab to bring Jupyter Notebooks to the cloud
- Dask executor runs out of the box
- Tokens give access to CMS data without certificate set-up (There is an opendata instance for those outside of CMS; UChicago has an ATLAS instance)

Useful links

- Uproot: https://uproot.readthedocs.io/en/latest/
- Awkward: https://awkward-array.org/quickstart.html
- Hist: https://hist.readthedocs.io/en/latest/
- Correctionlib: https://cms-nanoaod.github.io/correctionlib/
- Coffea: https://coffeateam.github.io/coffea/
- Coffea-casa: https://coffea-casa.readthedocs.io/en/latest/index.html
- Scikit-HEP: https://scikit-hep.org/
- Mattermost channel: https://mattermost.web.cern.ch/lpcrun2discuss/channels/coffea-users


Thank you
24