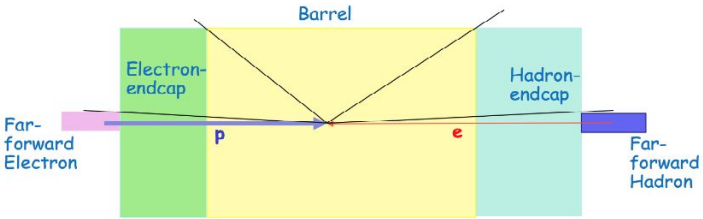
ATHENA: Hadron Encap Studies using Uproot

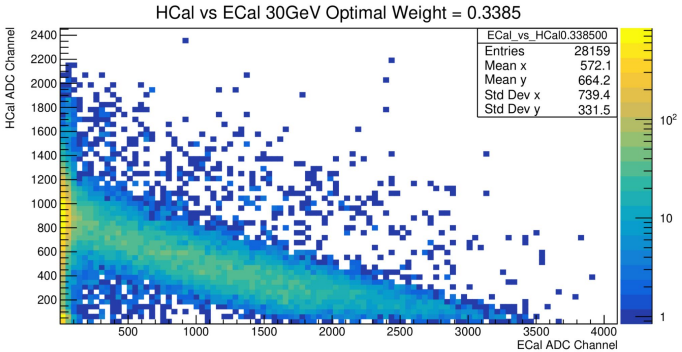
Ryan Milton
UCLA
In collaboration with: Zhiwan Xu, Oleg Tsai, Huan Z. Huang, ATHENA

September 15, 2021

Introduction

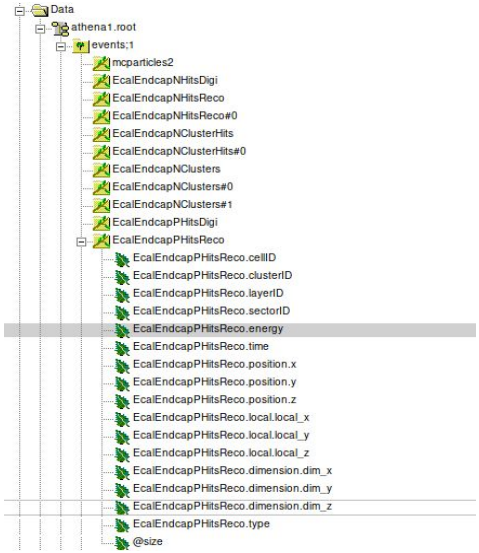
- Studying the energy deposition and resolution of hadron endcap in DD4hep simulations
- Measure the energy of incoming particles by absorbing particle showers
  - Electromagnetic cascades captured by ECal (front)
  - Hadronic cascades measured with HCal (behind)
- Combine ECal and HCal energies to get total energy deposition on event-by-event basis
- Used to obtain energy resolution of detector
  - How well particle energies can be separated
- Analyzed simulation data using Uproot





Using Uproot for DD4hep files

- Lots of .root data files, each containing a TTree containing many TBranches, each with multiple TLeafs
  - Only want small subset of data
  - Would be tedious and slow to read in everything and pick out what you actually want
- To solve this, can use uproot.concatenate() to easily read and combine the data of interest
- Say we're interested in the following:
  - Data files athena1.root and athena2.root
  - Within those, want energy from reconstructed hits in ECal
  - Have a big TTree with lots of TBranches and TLeafs
  - Only need one of those TLeafs
- How to accomplish this using uproot.concatenate()?



uproot.concatenate()

```
files = ['/home/ryan/Data/athena1.root:events', '/home/ryan/Data/athena2.root:events']
EcalP_events = ur.concatenate(files, ["EcalEndcapPHitsReco.energy"], library = 'np')
```

- Combines the EcalEndcapPHitsReco.energy leaf from each data file and stores the data in a numpy array
- Can also combine multiple leafs together:

```
endcap_events = ur.concatenate(files, ["EcalEndcapPHitsReco.energy", "HcalEndcapPHitsReco.energy"],
library = 'np')
```

- Simple to access individual events:

```
print(EcalP_events["EcalEndcapPHitsReco.energy"][2])
```

```
[4.0466310e-03 3.2867431e-03 3.7445067e-03 3.1463623e-03 3.2150268e-03
 1.0971070e-03 6.0821534e-03 3.9489744e-03 3.1875609e-03 3.1677247e-03
 3.7338256e-03 3.1753541e-03 3.3599853e-03 1.2413025e-02 1.7280579e-02
 4.0130614e-04 7.6293945e-05 4.5928956e-04 2.1209716e-04 9.0026857e-05
 5.1269529e-04 1.9989014e-04 8.5449217e-05 3.5400392e-04 9.5779421e-03
 1.2313842e-03 7.9040526e-04 6.6238404e-03 1.1785889e-02 9.0896608e-03
 2.6931763e-02 4.8034666e-03 4.0817261e-03 2.6611327e-03 1.6723633e-03
 3.3996582e-03 1.7379761e-03 8.5449219e-04 6.4605712e-03 3.6071776e-03
 3.8497925e-03 2.1820069e-04 1.9683837e-04 1.5605164e-02 6.5353392e-03
 6.5002439e-04 1.5370178e-02 5.5084226e-04 6.4697268e-04 2.4581910e-03
 6.9885253e-04 6.3629152e-04]
```

Structure of resulting arrays

- Now have an array containing M arrays, where M is the number of TLeafs chosen
  - Each of the M arrays contain N arrays, where N is the number of events
  - Each of the N arrays contain the event data
- Can also read in data from XRootD server

```
files.append('root://sci-xrootd.jlab.org//osgpool/eic/ATHENA/RECO/master/SINGLE/pi+/1GeV/3to50deg/pi+_1GeV_3to50deg.0001.root:events')
```

- Seems like a lot to keep track of at first, but simple once you get used to it!
- Easy conversion from .root files to Numpy arrays
  - Especially useful for newcomers since Python knowledge is common
  - Can then use standard Numpy array operations and functions like np.histogram()

Incident particle angles

- Initial studies using single particle events, i.e. one particle hitting endcap
- Generated data use varying particles and energies within an angle range
  - Can filter on the angle to study specific range
- mcparticles2 branch contains simulation information like momentum and pdgID
- Can get the angle of each event using the following:

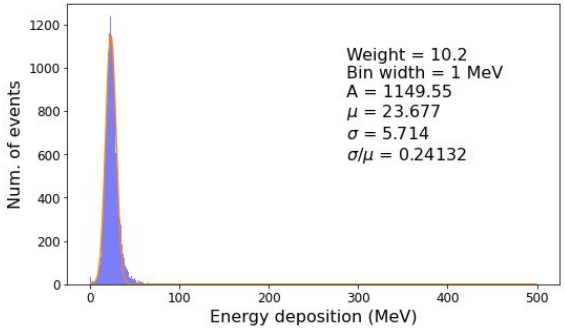
```
full_events = ur.concatenate(files, ["mcparticles2.ps.x", "mcparticles2.ps.y", "mcparticles2.ps.z", "mcparticles2.pdgID"], library = 'np')
num_events = len(full_events['mcparticles2.pdgID'])

for i in range(num_events):
    for j in range(len(full_events["mcparticles2.pdgID"][i])):
        if full_events["mcparticles2.g4Parent"][i][j] != 0: # Condition for initial/incident particle
            continue
        px = full_events["mcparticles2.ps.x"][i][j]
        py = full_events["mcparticles2.ps.y"][i][j]
        pz = full_events["mcparticles2.ps.z"][i][j]
        p = np.sqrt(px*px + py*py + pz*pz)
        theta = np.arccos(pz/p)*180/np.pi
        theta_list.append(theta)
```

- Now have a list of angles for each event and can easily select certain angles with if statements!

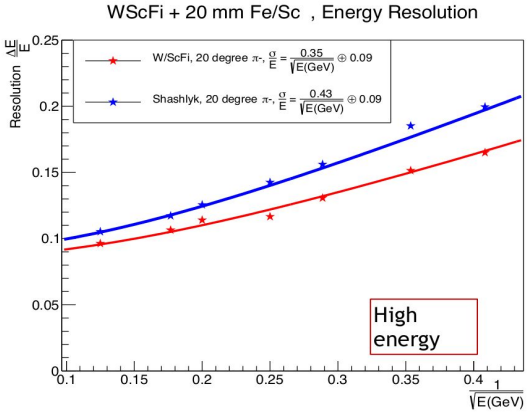
Combining Energy Depositions

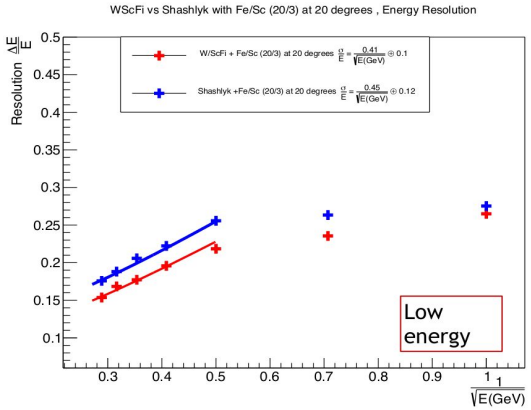
- To obtain resolution, need total energy deposit in the endcap for each event
  - Need total from both ECal and HCal
  - Can loop over each event and just use np.sum()!
- Could use a simple sum: $E_{dep} = E_{ECal} + E_{HCal}$
- For best energy resolution, we use a weighted sum: $E_{dep} = E_{ECal} / w_0 + E_{HCal}$
  - w_0 is an Ecal weighting constant
- Can then form a histogram using np.histogram() or matplotlib.pyplot.hist()



Calculating resolution

- Fit distribution with Gaussian
  - Can use Scipy to fit
- Resolution = ΔE/E = σ/μ from fit
- Can be found for different geometries, particles, energies, and other parameters
- Group also did previous studies using Geant4





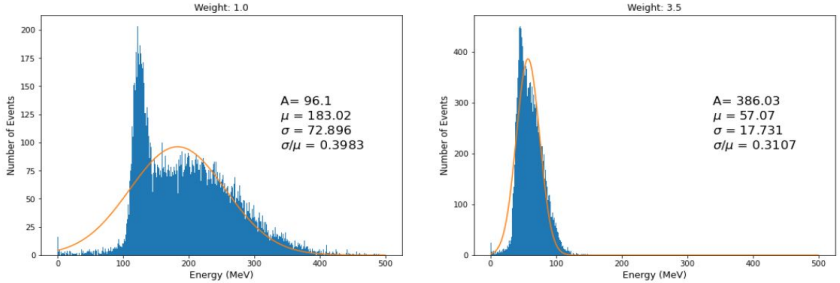
Plots from Zhiwan Xu

More on ECal weighting

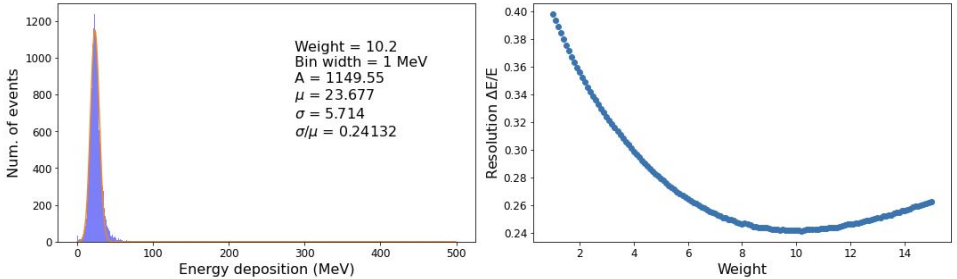
- Deposition from ECal and HCal parts is added with a weight w_0
- Optimal value of w_0 is found by minimizing energy resolution as a function of w_0 at each incident energy
- Accounts for differences in ECal and HCal
- w_0 is value between 1 and some larger number (e.g. 15)
  - Vary in small steps (e.g. 0.1)
  - Fill multiple histogram with sets of energy deposition, each weighted by different value
  - Fit each one with a Gaussian and select weight that gives minimum resolution
  - Choose this weight as the optimal weight and use this as the final resolution number
- Note: For particles that produce purely electromagnetic showers (e.g. electron, photon), weighting is not needed
  - Majority of energy deposited in ECal, so do not need to account for differences between ECal and HCal with weight

Weighting Example

- Set of data from an example endcap calorimeter
  - Results and data not important -- just for illustrative purposes
- Can loop through all the weights, fit the histogram and calculate the resolution, then store the resolutions
  - Optimal weight gives the minimum resolution



Examples of fitting at w_0=1.0 and w_0 = 3.5. There are histograms like these for each weight value.



On the left is the optimal weight, w_0 = 10.2, since it gives the min. resolution, as seen on the right.

What's next

- Optimize the endcap implementation in DD4hep simulations
- Want to produce resolution results from generated DD4hep data files
- Plot vs 1/sqrt(E) and compare to group's previous results

Conclusion

- Studying hadron endcap implementation in DD4hep
  - Producing resolution results to try to reproduce group's previous results
  - Further work will help with proposal for EIC
- Uproot can greatly ease the process of reading the generated .root files
- Allows for easy file combination
- Connects data to well-known libraries like Numpy

Thank you!

BACKUPS

Detector geometries

- Different endcap geometries
  - HCal longer than ECal and can be segmented longitudinally
  - ECal can be shashlyk (tile-based) or have scintillating fibers
  - HCal divided into 36 towers
- On the right: Geant4 visualization of W/ScFi geometry
  - Top: Visualization of a run (fibers not drawn)
  - Bottom: Fibers at front in yellow