PWG4 Analysis Status: jet analysis in AliRoot

Magali Estienne

ALICE Offline Week
24/06/2009 – CERN

Magali.Estienne@subatech.in2p3.fr

Modules available for Jet Studies

- JETAN:
  - Responsible: **Andreas Morsch**
  - Tools for jet reconstruction in the central barrel
    - Jet finding using charged particles only, using neutral particles, different jet finders, di-jet studies...
  - More recent modifications concern the neutral part (updated), the jet finders and the background subtraction tools
- PWG4/JetTasks:
  - Responsible: **Christian Klein-Bösing**
  - Takes output from JETAN analysis does further analyses with them and tracks
    - Jet spectrum, unfolding, UE, High p_T tracks QA...
- On the to do list, a major change:
  - JETAN is needed at some point to be in line with the structure in other PWGs
    => Probably moved to the directory PWG4/JetTasks.
  - Implementation in two different directories becomes more and more complicated.

JETAN structure

✓ Modifications for jet finding using neutral cells

✓ Correction for over-estimation of the deposited energy in the detectors

✓ EMCal geometry issue

✓ New jet finders available

✓ Tools for background studies

✓ Optimisation to fill signal and background information at the same time

✓ Store several jet finder branches in `aod.root`

✓ New data members in `AliAODJets`

✓ Specific user AOD for background `AliAODPWG4JetEventBackground`

Reader part: `fMomentumArray` vs `fUnitArray`

- 2 ways to store the input information for jet finding:
  - Using charged particles only: **`TClonesArray`** of **`TLorentzVector`** filled.
    - To one charged track corresponds on **`TLorentzVector`** (Px,Py,Pz,E).
  - Using charged tracks & neutral cells: **`TClonesArray`** of **`AliJetUnitArray`** filled.
    - To one given (η,ϕ) position corresponds one **`AliJetUnitArray`**
- An **`AliJetUnitArray`** can contain several charged tracks and neutral cells.
  - Created and initialized one time at the beginning of the analysis and re-initialized at each event thanks to a TRefArray
  - One can store the following information from an **`AliESDTrack`** or an **`AliEMCalDigit`**:
    - η, ϕ, Unit ID, Track ID, detector flag, p_T cut flag, vector < vector (Px,Py,Pz) >, etc.
  - Charged tracks stored in fUnitArray in the class **`AliJetFillUnitArrayTracks (TTask)`**, neutral cells stored in fUnitArray in the class **`AliJetFillUnitArrayEMCalDigits (TTask)`**
  - Relation UnitID $\Leftrightarrow$ (η,ϕ) position & other functions obtained with the **`AliJetGrid`** class
- Modifications needed:
  - Change **`Int_t TrackID`** to **`vect<Int_t trackID>`** not only for one track but for the list of tracks $\Rightarrow$ Needed for background subtraction or fragmentation function studies.
  - **`vector<vector>`**: structure not allowed by the ALICE coding convention.
  - Will be modified by storing a reference to the tracks that belong to a given (η,ϕ) position and same for neutral cells.

Reader part: AliJetESDReader/AliJetAODReader

- Both fMomentumArray & fUnitArray implemented at the level of **AliJetESDReader**:
  - Fills particle and neutral cell information from **AliESDTrack** and **AliESDCaloCluster**.
  - fUnitArray needs to be implemented in **AliJetAODReader** in order to look for jets using filtered track information from standard AODs. One also needs track references as in ESD Reader => on going activity.
- How to call the different functionalities ?
  - In the macro **ConfigJetAnalysis*.C**, the user can chose the object he wants to use to fill the particle information to give to the jet finder.
    - NB: the use of fMomentumArray or fUnitArray imposes the version of the jet finder to be used.
  - In the **ConfigJetAnalysis*.C**, it is set at the level of the **AliJetESD(AOD)ReaderHeader**:

```
AliJetESDReaderHeader *jrh = new AliJetESDReaderHeader();
// Detector options: 0 = Charged particles only (fMomentumArray)
//                   1 = Charged particles only (fUnitArray)
//                   2 = Neutral cells only (fUnitArray)
//                   3 = Charged particles + neutral cells (fUnitArray)
jrh→SetDetector(0);
```

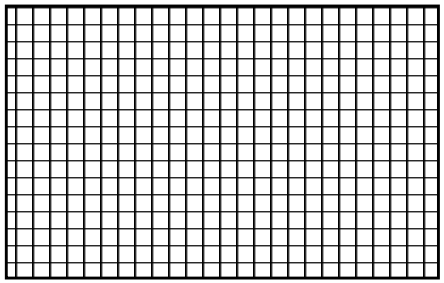
- Same implementation in AliJetAODReader to come soon

Reader part: **AliJetGrid**

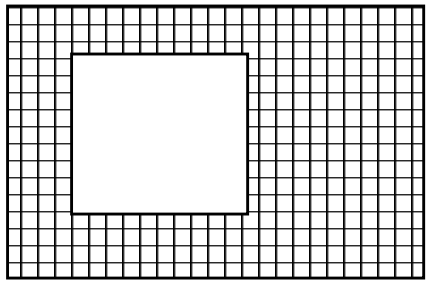
- Grid definition:
  - If options 1, 2 or 3 are chosen, one has to define a grid for the fUnitArray definition and initialization using the class **AliJetGrid**.
  - The grid also allows to have a quick access to a given (η,ϕ) position in the unit object and vice-versa.
  - Grid also used to deal with dead zones (not commented here).
  - Two different types of grid can be set in the macro ConfigJetAnalysis*.C:

```
AliJetGrid *grid1/2 = new AliJetGrid(N_bin_φ, N_bin_η, φ_min, φ_max, η_min, η_max);
grid1→SetGridType(0);   // Complete (η,φ) grid rectangle
grid2→SetGridType(1);   // (η,φ) grid minus a rectangle inside
```

Type 0:



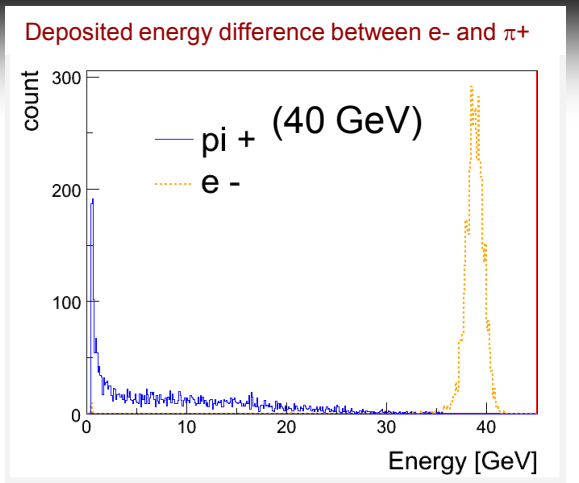
Type 1:



- Grid initialization in the ConfigJetAnalysis*.C file:

```
AliJetESDReader *eh = new AliJetESDReader();
eh→SetReaderHeader(jrh);    // Set the esd reader header
eh→SetTPCGrid(grid2);       // Set the TPC grid
eh→SetEMCalGrid(grid1);     // Set the EMCal grid
```

Reader part: Hadron and electron corrections



- Over-estimation of hadron and electron energy (counted twice).
- Two different approaches for correction:
  statistical (MIP) or on a case-by-case basis (hadronic or electron)
  - The corrections are applied when the tracks (in **AliJetFillUnitArrayTracks**) or the neutral cells (in **AliJetFillUnitArrayEMCalDigits**) are filled in the fUnitArray.
  - For MIP correction, correct all charged tracks from an average contribution extracted from simulation. (η,p_T,E_{depo_mean}) parametrization stored in **AliJetHadronCorrection** class. Performed in the class **AliJetFillUnitArrayTracks.**
  - For hadronic or electron correction (*« à la STAR »*), use track matching and subtract a fraction of the deposited energy in the calorimeter from a given track momentum (new class to be implemented ? In progress). Performed in **AliJetFillUnitArrayEMCalDigits.**
- How to ask for these corrections to be applied ? => In the ConfigJetAnalysis*.C

```
AliJetHadronCorrectionv1 *hadcorr = AliJetHadronCorrectionv1::Instance();
AliJetESDReader *eh = new AliJetESDReader();
eh→SetReaderHeader(jrh);              // Set the esd reader header
eh→SetApplyMIPCorrection(kTRUE);      // Chose or not MIP correction
eh→SetHadronCorrector(hadcorr);       // Set the mapping for MIP correction
eh→SetApplyFractionHadronicCorrection(kFALSE); // Chose or not hadronic correction
eh→SetApplyElectronCorrection(kTRUE); // Chose or not electron correction
```

EMCal Geometry in independent library

- Presently, one temporary class to define the EMCal geometry and functions:
  - **AliJetDummyGeo** => not flexible as it has to be modified each time **AliEMCalGeometry** changes
- AliEMCalGeometry modified in order to make the EMCal geometry tools independent from the rest of aliroot:
  - New classes for geometry in EMCAL Module:
    - **AliEMCalGeometry:** modified and inherits from **AliEMCalGeoUtils**
    - **AliEMCalEMCGeometry:** contains the EMCal geometry parameters for its initialization
    - **AliEMCalGeoUtils:** contains main transformations and functions
  - Classes still under testing
    - Functions called by the jet finder work properly.
    - Some bugs still need to be fixed at the level of the simulation/reconstruction.
- New library to be called by the user analysis: **libEMCalGeoUtils.so**
  - To be tested
  - One needs to load libCDB.so which only depends on libSTEERBase.so
  - Open issue on CAF with the calibration classes which need to have access to the OCDB (accessible on the grid through the URI alien://) ⇒ alien access now on CAF

Finder part: New jet finder implementations

- Jet finders available:
  - Statistical jet finder:
    - Deterministic annealing - Charged only: **AliDAJetFinder**.
  - Cone jet finders:
    - Pxcone – Charged only: **AliPxconeJetFinder** (old implementation)
    - UA1 - Charged: **AliUA1JetFinder**, **AliUA1JetFinderV1** and charged + neutral: **AliUA1JetFinderV2**.

      Parameters: backMode, radius, E_T seed, Min jet E_T, p_T cut on charged particles, etc.
    - CDF - Charged only: **AliCDFJetFinder**.
    - SISCone - Charged + neutral: **AliSISConeJetFinder** – neutral part in progress.

      Fastjet package using SISCone plugin:
      Parameters: radius, overlap parameter, p_T min of proto jets (before split/merge), etc.

```
fastjet::JetDefinition::Plugin *plugin;
plugin = new fastjet::SISConePlugin(coneRadius,
overlapThreshold, nPassMax, ptProtoJetMin, caching);
// Set the esd reader header
```

  - Sequential jet finders:
    - FastJet - Charged + neutral: **AliFastJetFinder** – neutral part in progress.

      Parameters: radius, p_T min of jet, background tools (see after)

- Call in ConfigJetAnalysis*.C:

Algo = DA, Pxcone, UA1, CDF, SISCone, Fast
Vx = nothing, V1, V2

```
AliAlgoJetHeaderVx *jh = new AliUA1JetHeaderVx();
jh->Set…Options…
jetFinder = new AliJetFinderVx();
jetFinder->SetJetHeader(jh);
jetFinder->SetJetReader(eh);
```

Finder part: AliFastJetFinder / AliSISConeJetFinder

- In FastJet module, one can choose in ConfigJetAnalysisFastJet.C:
  - The algorithm he wants to run:

```
AliFastJetHeaderV1 *jh = new AliFastJetHeaderV1();
jh→SetAlgorithm(fastjet::kt_algorithm);          // Fast k_T
jh→SetAlgorithm(fastjet::antikt_algorithm);      // Anti k_T
jh→SetAlgorithm(fastjet::cambridge_algorithm); // Cambridge
```

  - The strategy (way of finding pairs of particles, neighbors to our particles):
    - For instance, for high multiplicity events, the NlnN strategy (which uses Voronoi diagrams to find pairs of particles) will be chosen

```
jh→SetStrategy(fastjet::Best); // standard value, automatically choose the
                               // best strategy following the number of
                               // particles in the event.
```

  - The recombination scheme:

```
jh→SetRecombScheme(fastjet::BIpt_scheme); // standard value, pt weighted
                                          // recombination of y and phi
                                          // (and summing of pt's)
```

- For SISCone, one can choose in ConfigJetAnalysisSISCone.C:
  - The overlap parameter (which determine to split or merge overlapping cones):

```
jh->SetOverlapThreshold(0.75); //standard value, recommended
```

  - The p_T min of protojets (stable cones before split/merge procedure) :

```
jh->SetptProtojetMin(4);
```

Finder part:
Tools for background studies

- Parameters for background studies in FastJet module:
  - Flag to select or not background subtraction.
  - Kind of area (passive, active …).
  - The η interval to study background.
  - Ghosts area.
  - p_T mean of ghosts.

- Background subtraction strategy available in UA1:
  - Cone method (BackgMode=2) = Standard (BackgMode=1)
  - Statistical method (BackgMode=4)
  - Ratio method (BackgMode=3)

- Background subtraction strategy available in FastJet (Under development):
  - Smaller R: done
  - Only charged track TPC + scaling for the neutral energy: in progress
  - Out-Of-Cone (as in UA1): in progress
  - Statistical background (parametrization): in progress

Finder part:
New tools for background subtraction in FastJet

- New classes for background subtraction using FastJet jet finder:
  - **AliFastJetInput** → fills the input_particles array needed by FastJet (array filled only once). Loop over the unitarray and fill 2 input_particles, one for all, one for charged tracks only.
  - **AliFastJetFinder** and **AliJetBkg** get the input_particles from **AliFastJetInput**
  - **AliJetBkg** calculates the bkg/area for the different bkg schemes enumerated in the previous slide for fastjet.

(Under development)

```
Bool_t AliJetFinder::ProcessEvent2() {
…
fLeading→FindLeading(fReader);
fInputFJ→SetHeader(fHeader);
fInputFJ→SetHeader(fHeader);
fInputFJ→SetReader(fReader);
fInputFJ→FillInput();

// Jets
FindJets();
fJetBkg->SetHeader(fHeader);
fJetBkg->SetReader(fReader);
fJetBkg->SetFastJetInput(fInputFJ);
fJetBkg->BkgFastJet();
fJetBkg->BkgChargedFastJet();
…
}
```

- To be generalized to the other jet finders ?

Output AOD for jets and background

- Modification of **AliAODJet** with new data members:
  - Three new data members added: relative error of jet area, neutral fraction and jet trigger

```
virtual Bool_t IsTriggerEMCAL()
virtual Bool_t IsTriggeredTRD()
virtual Uchart_t Trigger()
virtual void SetEffArea(Double_t effACh,Double_t effANe,Double_t effAErrCh,Double_t effAErrNe)
virtual void SetTrigger(Uchar_t f)
virtual void ResetTrigger(Uchar_t f)
virtual Double_t ErrorEffectiveAreaCharged() const
virtual Double_t ErrorEffectiveAreaNeutral() const

// First only one bit for EMCal and TRD, leave space
// for more trigger types and/or other detectors
enum{kEMCALTriggered = 1<<0, kTRDTriggered = 4<<0};

private:
  Double32_t  fEffectiveAreaError[2]; // [0,1,10] relative error of jet areas, 10 bit precision
  Double32_t  fNeutralFraction;    // [0,1,12] Neutral fraction between 0 & 1 12 bit precision
  Uchar_t     fTrigger;            // Bit mask to flag jets triggered by a certain detector
```

- Specific user AOD: **AliAODPWG4JetEventBackground**
  - Contains the estimated background event by event from different schemes:
    - **kSmallR, kOnlyCharged, kOutOfCone, kStatistical, kMaxBackground.**
  - Plan to put it into new PWG4/base directory (libPWG4base). There will be more AOD additions for PWG4 which should not go in STEER.
  - Open issue: where should they go ? In STEER or PWG4 ?

Jet Analysis tasks in the analysis train

- Jet tasks in the train ConfigJetAnalysis*.C changed to AddTask*.C:
  - Jet finding: **`AliAnalysisTaskJet`** (in JETAN)
    - Choose the algorithm or input data AOD, ESD, MC, CDF, DA, FastJet, UA1, SISCone…:

      `ConfigJetAnalysis*.C` ⇒ `Changed to` **`AddTaskJets.C`**
  - Di-jet study: **`AliAnalysisTaskDijets`** (in JETAN)

    ⇒ **`AddTaskDiJets.C`**
  - Jet Spectrum: **`AliAnalysisTaskJetSpectrum`**, **`AliAnalysisHelperJetTask`** (in PWG4/JetTasks)

    ⇒ **`AddTaskJetSpectrum.C`**
  - Jet spectrum unfolding: **`AliJetSpectrumUnfolding`** (in PWG4/JetTasks)

    ⇒ **`AddTaskJetSpectrum.C ?`**
  - Underlying event: **`AliAnalysisTaskUE`** (in PWG4/JetTasks)

    ⇒ **`AddTaskUE.C`**
  - Simple AOD PID: **`AliAnaESDSpectraQA`**, **`AliAnalysisTaskPWG4PidDetEx`** (in PWG4/JetTasks)

    ⇒ **`AddTaskPWG4PidDetEx.C`**

- General example to run several jet finders at the same time and **putting them to different AOD branches**, with the JetSpectrum task reading back a selectable branch.
  - Example macro available in PWG4/JetTasks/macro/ **`AnalysisTrainCaf.C`**

FastJet on the Grid

- Fastjet headers in JETAN directory under JETAN/fastjet/
  - Not so convenient as it has to be modified with the fastjet package version used for the analysis
- On the Grid:
  - The following packages have been installed: FastJet, CGAL and Boost
  - Associated environment variables: FASTJET_ROOT, CGAL_ROOT and BOOST_ROOT
  - Before, FASTJET not defined by default on the Grid

  ⇒ necessity to use a specific JETAN.par file

  ⇒ necessity to load the above packages in the JDL
- NOW: Classes for the fastjet interface in AliRoot available in an independent library: libFASTJETAN.so
  - Compiled with the headers in JETAN/fastjet/ by default on the grid
  - No need anymore to use parfiles
  - Packages to be loaded in the JDL by the user only if it needs to run FastJet.
  - Test under progress !
- Example macros in the directory: /alice/cern.ch/user/m/morsch/fastjet/

Conclusion and do list

- Write AliJetAODReader using UnitArray for charged + neutral jet finding and include a Reference to tracks in the UnitArray + other technical points

- Recent developments for fastjet
=> universal structure of the jet finders to be recovered

- Move JETAN to PWG4/JetTasks

- Trigger issue at the level of ESDs and then AODs

- Specific PWGs AODs => where do they go ?

- Write a « how to » manual for jet finding