INFN BOLOGNA
Istituto Nazionale di Fisica Nucleare
Sezione di Bologna

First NAIA Meeting:
NAIA User Experience

Alejandro Reina Conde, INFN Sezione di Bologna
Alberto Oliva, INFN Sezione di Bologna

Wednesday 21th of December, 2022

Pass7 CIEMAT analysis: Ntuples

CIEMAT ntuples are organized as usual in two Trees: RTI and DST.

DST Tree is NOT separated in different branches.

```
 TFile*         He.23_20200119.MC/C.B1215/c12.pl1.l1.612000.4_01/604601543.00000001.root
  KEY: TTree    DST;1   DST
  KEY: TTree    RTI;1   RTI
root [2] DST->Print()
******************************************************************************
*Tree    :DST       : DST                                                    *
*Entries :     8545 : Total =        27299922 bytes  File  Size =    7603638 *
*        :          : Tree compression factor =   3.59                       *
******************************************************************************
*Br    0 :Data      : Status/l:Time/D:TimeError/D:RTITime/i:Run/I:Event/I:    *
*         | EventError/i:Tag/I:ThetaM/F:PhiM/F:ThetaS/F:PhiS/F:CutOff/F:     *
*         | Zenith/F:LiveTime/F:DaqEventLength/i:DaqReplyCodeErrors/i:       *
*         | DaqJinfRoomErrors/i:JINJStatus[4]/i:JLength[24]/i:JRoomError/i:  *
*         | SubdetectorPattern/I:PhysicsPattern/I:AntiPattern/I:             *
*         | SubdetectorPatternRebuilt/I:PhysicsPatternRebuilt/I:             *
*         | TriggerRebuildRet/I:SABeta/F:SAN/I:SAInvBetaErr/F:SABetaChi2/F:  *
*         | SAQ/F:SAQLayers/I:SAT0/F:SABetaClusters/I:SAQTop/F:SAQBottom/F:  *
*         | SAOuterPosition[2][3]/F:SALargerCharge[2]/F:                     *
*         | SALargerChargePosition[2][3]/F:SALargerChargeH[2]/F:             *
*         | SALargerChargeYJ[2]/F:TofHClusters[4]/I:TofHQ2[4]/F:SAclsdt[4]/F:*
*         | SAclsed[4]/F:SAclsn[4]/I:EcalShowers/I:EcalTofDist/F:            *
*         | EcalEnergy/F:EcalEnergyTruncated/F:EcalEnergyC/F:EcalEnergyD/F:  *
*         | EcalEnergyE/F:EcalEnergyA/F:EcalEnergy2017/F:EcalBDT/F:          *
*         | EcalBDTCHI2/F:EcalEnergyFractionLastLayer/F:EcalCoo[3]/F:        *
*         | EcalTheta/F:EcalPhi/F:EcalRigidityEstimate[2]/F:EcalApex[2]/F:   *
*         | EcalCharge/F:EcalEnergyTotLayer[4]/F:EcalEnergyMaxLayer[4]/F:    *
*         | SAEcalEnergyD/F:SAEcalEnergyTotLayer[4]/F:                       *
*         | SAEcalEnergyMaxLayer[4]/F:EcalKShowers/I:EcalKEnergyFast/F:      *
*         | EcalKEnergy3D/F:EcalKLkl[2]/F:EcalKCoo[3]/F:EcalKTheta/F:        *
*         | EcalKPhi/F:Beta/F:BetaPattern/I:TRDLklHits/I:TRDLklPath/F:       *
*         | TRDLklAmpl/F:TRDLkl[3]/F:TRDTofMatchChi2[2]/F:TRDTrackCoo[3]/F:  *
*         | TRDTrackTheta/F:TRDTrackPhi/F:TRDTrackChi2/F:TRDTrackQ/F:        *
*         | TRDTrackHits/I:logProbHadron/F:logProbLeptonRightCharge/F:       *
*         | logProbLeptonWrongCharge/F:TRDGamma/F:TrQ[4]/F:TrQH[4]/F:        *
*         | TrQYJ[4]/F:TrQLayer[9]/F:TrQHLayer[9]/F:TrQYJLayer[9]/F:         *
*         | TrQStatusLayer[9]/I:TrHitCoo[9][3]/F:TrPlaneHits[6]/I:           *
```

Pass7 CIEMAT analysis: Reading Ntuples

For reading the ntuples we use the TSelector:

"A TSelector object is used by the TTree::Draw, TTree::Scan, TTree::Process to navigate in a TTree and make selections."

The Header of the TSelector should include the variables in the same order of the root file.

TSelector allows you to fill many histograms at the same time by looping just one time over all the Tree entries.

```
class DST : public TSelector {
public :
   TTree          *fChain;   //!pointer to the analyzed TTree or TChain

   // Declaration of leaf types
   ULong64_t       Data_Status;
   Double_t        Data_Time;
   Double_t        Data_TimeError;
   UInt_t          Data_RTITime;
   Int_t           Data_Run;
   Int_t           Data_Event;
   UInt_t          Data_EventError;
   Int_t           Data_Tag;
   Float_t         Data_ThetaM;
   Float_t         Data_PhiM;
   Float_t         Data_ThetaS;
   Float_t         Data_PhiS;
   Float_t         Data_CutOff;
   Float_t         Data_Zenith;
   Float_t         Data_LiveTime;
   UInt_t          Data_DaqEventLength;
   UInt_t          Data_DaqReplyCodeErrors;
   UInt_t          Data_DaqJinfRoomErrors;
   UInt_t          Data_JINJStatus[4];
   UInt_t          Data_JLength[24];
   UInt_t          Data_JRoomError;
   Int_t           Data_SubdetectorPattern;
   Int_t           Data_PhysicsPattern;
   Int_t           Data_AntiPattern;
   Int_t           Data_SubdetectorPatternRebuilt;
   Int_t           Data_PhysicsPatternRebuilt;
   Int_t           Data_TriggerRebuildRet;
   Float_t         Data_SABeta;
   Int_t           Data_SAN;
```

Pass7 CIEMAT analysis: The Goods and the Bads

GOODS

1. It is intuitive, better for low C++ knowledge, everything is one large TSelector.

2. Ntuple production is really fast. Ones could get all the 10 years Data in 1 or 2 days (LXPLUS CERN).

BADS

1. Almost impossible to understand the Ntuples variables without help:
   - There is no documentation.
   - Names are not clear.
   - Ones should go to the Ntuple production code, check with what you are filling a variable, and then go to gbatch (AMS production code).

2. Analysis Code is messy, everything is one large TSelector. Worse than other possibilities if you have some C++ understanding.

3. Ntuple code production is totally managed by Jorge, which means depending on him for every change.

Pass8 NAIA analysis: Implementing new features

During the first months of working with NAIA I have participated in the implementation of the new features needed for pass8 (Qi Yan code…):

- V6 Alignment: TrTrackResiduals, PartialTrTrackResiduals, TrTrackFitPos, PartialTrTrackFitPos…
- New pass8 track reconstruction: AMSPlane, track->Interpolate…
- StandAlone variables: TOF charges with no rigidity correction, Tracker charges with no beta correction…

For implementing a new feature ones should:

- Open an issue
- Create a new branch for that feature.
- Implement it in the new branch.
- Test if it is working as intended.
- Commit and ask for merge with the master branch (or bugfix/new release ones)

Pass8 NAIA analysis: Reading Ntuples

For NAIA Alberto and myself decided to build all our code from the scratch.

This time we are using a different approach:

- There is no more a huge one TSelector.
- Everything is separated in its own class (BetaEfficiency, TrackEfficiency, Counts, Purity…)
- All these classes are grouped in 2 main groups:
- Flux, where we have all the needed classes for the estimation of the flux (efficiencies, rate…)
- Ancillaries, which are classes that have elements of the analysis which are needed to estimate just one time (Charge calibration, Rigidity Estimators…)

In this way we try to save time, while in the past we were producing everything all the time with the TSelector

Pass8 NAIA analysis: Running time problem (I)

- One possible solution is to Skim the files. For example applying ChargeGreaterThan2 Masks.

  This is one day:
  -rw-r--r-- 1 reinacon ams 11G dic 7 17:31 IonSkimming_2435_00.txt.root
  The files are too heavy even after skimming (only applied GT2 Masks)

- For comparison this is the total size of CIEMAT and NAIA Carbon Inner-L1 Focus MonteCarlo. Even when the MonteCarlo versions are different, it is a factor 10.

```
CIEMAT
ls -lh /eos/ams/user/o/oliva/He.23_20200119.MC/C.B1215/c12.pl1.l1.612000.4_01
total 144G
ls -lh /eos/ams/user/o/oliva/He.23_20200119.MC/C.B1215/c12.pl1.l1.612000.4_01.1/
total 206M

NAIA
ls -lh /storage/gpfs_ams/ams/groups/AMS-Italy/ntuples/v1.0.0/C.B1236/c12.pl1.l1.612000.6_02/
total 1,6T
```

- Managing these huge files means larger resources usage for a single job → condor user_priority becomes lower with less jobs → larger processing time (From 1-2 days with CIEMAT to > 4 days for NAIA, for a full 10 years data production).

Pass8 NAIA analysis: Running time problem (II)

```
/storage/gpfs_ams/ams/groups/AMS-Italy/ntuples/v1.0.0/ISS.B1236/pass8/1326499497.root
LooperEvent::Process-Processed 0 entries of 495978
```

```
 if ((!isMC)&&(element>2)&&
     (!event_base.CheckMask(NAIA::Category::ChargeGT2_Tof))&&
     (!event_base.CheckMask(NAIA::Category::ChargeGT2_Trk))&&
     (!event_base.CheckMask(NAIA::Category::ChargeGT2_Tof_St))&&
     (!event_base.CheckMask(NAIA::Category::ChargeGT2_Trk_St))) return;
   else if ((!isMC)&&(element==2)&&
     (!event_base.CheckMask(NAIA::Category::Charge2_Tof))&&
     (!event_base.CheckMask(NAIA::Category::Charge2_Trk))&&
     (!event_base.CheckMask(NAIA::Category::Charge2_Tof_St))&&
     (!event_base.CheckMask(NAIA::Category::Charge2_Trk_St))) return;
```

With the CheckMask

real	**3m32.118s**	**real**	**4m2.669s**
user	**0m43.178s**	**user**	**0m42.800s**
sys	**0m0.812s**	**sys**	**0m0.791s**

Without the CheckMask

real	**10m3.386s**	**real**	**8m51.537s**
user	**8m18.718s**	**user**	**8m22.669s**
sys	**0m2.815s**	**sys**	**0m3.080s**

Pass8 NAIA analysis: Running time problem (III)

pass8/1326499497.root
1,6G Before Skimming
136M After Skimming

I SKIM the file by applying the four GT2 Masks showed before.

495978 entries in the chain

real	**3m55.125s**	**real**	**4m50.041s**
user	**3m41.478s**	**user**	**3m42.794s**
sys	**0m1.320s**	**sys**	**0m2.321s**

Then I run over this skimmed file w/o the Masks.

With the Masks:		**Without the Masks:**	
real	**0m40.665s**	**real**	**0m44.441s**
user	**0m33.783s**	**user**	**0m39.532s**
sys	**0m0.501s**	**sys**	**0m0.557s**

Conclusion

- **A completely new analysis code has been implemented using NAIA. A lot of work has been done, but it is still to be finished.**

- **NAIA effort comes a bit late, but it is already showing its goods, and should save time for everyone in the future: for analysis discussions, for ntuples productions and for results crosschecks among the different groups.**

- **The highest priority for me is to solve the processing time.**

One of these things is not like the other. Real refers to actual elapsed time; User and Sys refer to CPU time used *only by the process*.

•**Real** is wall clock time - time from start to finish of the call. This is all elapsed time including time slices used by other processes and time the process spends blocked (for example if it is waiting for I/O to complete).

•**User** is the amount of CPU time spent in user-mode code (outside the kernel) *within* the process. This is only actual CPU time used in executing the process. Other processes and time the process spends blocked do not count towards this figure.

•**Sys** is the amount of CPU time spent in the kernel within the process. This means executing CPU time spent in system calls *within the kernel,* as opposed to library code, which is still running in user-space. Like 'user', this is only CPU time used by the process. See below for a brief description of kernel mode (also known as 'supervisor' mode) and the system call mechanism.

User+Sys will tell you how much actual CPU time your process used. Note that this is across all CPUs, so if the process has multiple threads (and this process is running on a computer with more than one processor) it could potentially exceed the wall clock time reported by Real (which usually occurs). Note that in the output these figures include the User and Sys time of all child processes (and their descendants) as well when they could have been collected, e.g. by wait(2) or waitpid(2), although the underlying system calls return the statistics for the process and its children separately.