NAIA User Experience in Perugia

NAIA Meeting,
Bologna, 22-12-2022

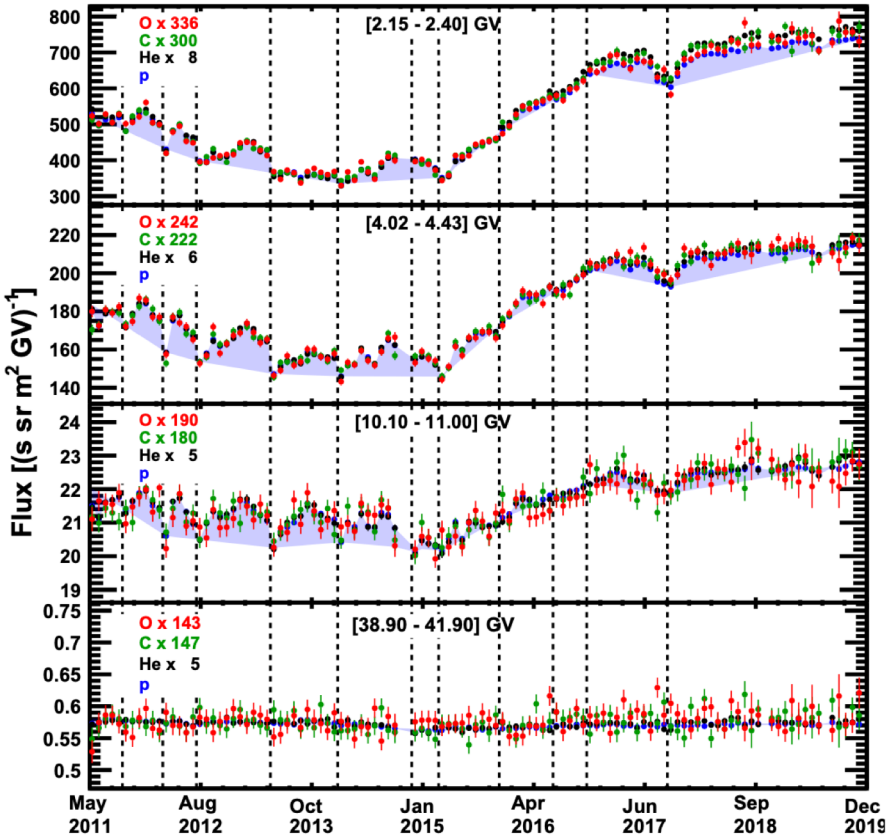
Federico Donnini
(INFN Sez. Perugia)

Analysis

Isotropic Differential Flux for each ***Bartel Rotation*** **(BR)**, starting from May 2011 up to May 2021(135 Bartels Rotations):

$$\Phi_i^{BR} = \frac{N_i^{BR}}{A_i^{BR}\,\varepsilon_i^{BR}\,T_i^{BR}\,\Delta R_i}$$

- **Isotropic Differential Flux** *(m^2 sr s GV)$^{-1}$*
- **Number of events:** PG selections, 10 years of Pass8 Data
- **Effective Acceptance:** B1236.6_02 (B, C, O)
- **Trigger Efficiency**
- **Exposure Time**
- **Bin width:** bins from 1.9 GV to 60 GV



NAIA containers

Container type	Name	Description
Header	header	Contains simple information like run number, run tag, event number and UTC time.
EventSummary	evSummary	Contains some aggregated variables to roughly describe the event.
DAQ	daq	Contains variables describing the status of the AMS DAQ system for the event.
TofBase	tofBase	Contains basic Tof variables that are accessed most frequently
TofPlus	tofPlus	Contains additional Tof variables that are accessed less frequently
TofBaseStandalone	tofBaseSt	Contains basic Tof variables that are accessed most frequently (no-tracker reconstruction)
TofPlusStandalone	tofPlusSt	Contains additional Tof variables that are accessed less frequently (no-tracker reconstruction)
EcalBase	ecalBase	Contains basic Ecal variables that are accessed most frequently
EcalPlus	ecalPlus	Contains additional Ecal variables that are accessed less frequently
TrTrackBase	trTrackBase	Contains basic Track variables that are accessed most frequently
TrTrackPlus	trTrackPlus	Contains additional Track variables that are accessed less frequently
SecondTrTrackBase	secondTrTrackBase	Contains basic Track variables for the most energetic track in the event after the primary one
TrTrackBaseStandalone	trTrackBaseSt	Contains basic Track variables reconstructed withouth using any Tof information
TrdKBase	trdKBase	Contains basic TRD variables that are accessed most frequently
TrdKBaseStandalone	trdKBaseSt	Contains basic TRD variables that are accessed most frequently (no-tracker reconstruction)
RichBase	richBase	Contains basic RICH variables that are accessed most frequently
RichPlus	richPlus	Contains additional RICH variables that are accessed less frequently
UnbExtHitBase	extHitBase	Contains basic variables for unbiased external hits
MCTruthBase	mcTruthBase	Contains basic MC truth variables that are accessed most frequently
MCTruthPlus	mcTruthPlus	Contains additional MC truth variables that are accessed less frequently

For this analysis all the containers are used, with ecception of containers storing TRD, RICH and ECAL informations)

NAIA::RTIInfo tree

In NAIA the RTI database is converted to a TTree, useful to compute the LiveTime

Old code

- Loop over rootfiles
- take UTC time from first and last event
- Loop over seconds

```
AMSSetupR::RTI *rti = new AMSSetupR::RTI();
AMSSetupR::RTI::UseLatest();

// Loop over rootfiles
for (auto rootfile : inputFiles) {

std::unique_ptr<AMSChain> chain(new AMSChain());
chain->AddFile(rootfile.c_str());

AMSEventR *event = NULL;

UInt_t starttime = chain->GetEvent(0)->UTime();
UInt_t stoptime = chain->GetEvent(nEntries - 1)->UTime();

//Loop over seconds
for (UInt_t isec = starttime; isec < stoptime; isec++) {

AMSEventR::GetRTI(*rti, isec);

// RTI cuts
```

NAIA

- Define the NAIAChain
- Access RTI tree
- Loop over seconds

```
// NAIA chain
NAIA::NAIAChain chain;
for (const auto &filename : fileList) {
chain.Add(filename);
}
chain.SetupBranches();

// get RTI tree
TChain *rti_chain = chain.GetRTITree();
NAIA::RTIInfo *rti_info = new NAIA::RTIInfo();
rti_chain->SetBranchAddress("RTIInfo", &rti_info);

// loop over RTI
for (unsigned long long isec = 0; isec < rti_chain->GetEntries();
++isec) {

rti_chain->GetEntry(isec);

// RTI cuts
```

SetupBranches

Old code

The SetBranchAddress was done manually, requiring hundreds of lines of code

```
TChain* chain = new TChain("IonsEvSel");
chain->Add( infilename );
```

```
// List of branches
TBranch *b_Run;
TBranch *b_RunID;
TBranch *b_EventNo;
TBranch *b_UTime;
TBranch *b_PhysBPatt;
TBranch *b_IsUnbiased;
TBranch *b_IsPhysics;
TBranch *b_ntracks;
TBranch *b_iTrTrack;
. . .
```

```
// Declaration of leaf types
Int_t Run;
Int_t RunID;
Int_t EventNo;
Int_t UTime;
Int_t PhysBPatt;
Bool_t IsUnbiased;
Bool_t IsPhysics;
Int_t ntracks;
Int_t iTrTrack;
. . .
```

```
// Declaration of leaf types
chain->SetBranchAddress("Run", &Run, &b_Run);
chain->SetBranchAddress("RunID", &RunID, &b_RunID);
chain->SetBranchAddress("EventNo", &EventNo, &b_EventNo);
chain->SetBranchAddress("UTime", &UTime, &b_UTime);
chain->SetBranchAddress("PhysBPatt", &PhysBPatt, &b_PhysBPatt);
chain->SetBranchAddress("IsUnbiased", &IsUnbiased, &b_IsUnbiased);
chain->SetBranchAddress("IsPhysics", &IsPhysics, &b_IsPhysics);
chain->SetBranchAddress("ntracks", &ntracks, &b_ntracks);
chain->SetBranchAddress("iTrTrack", &iTrTrack, &b_iTrTrack);
. . .
```

NAIA

In NAIA the SetBranchAddress is done in one line of code,
just calling the SetupBranches() method

```
NAIA::NAIAChain chain;
for (const auto &filename : fileList) {
chain.Add(filename);
}

chain.SetupBranches();
```

NAIA::Category

In NAIA it's possible to discard uninteresting events, by using the NAIA::Category

```
auto chrono_start =
std::chrono::system_clock::now();

// event loop
for (NAIA::Event &event : chain) {

// selections

}

auto chrono_end =
std::chrono::system_clock::now();
std::chrono::duration<double> time_spent =
chrono_end - chrono_start;
```

Output without NAIA::Category cut:
Processed 622923 entries, 104 selected for IL1, and 16 for FS, in 88.6 seconds

```
//----- setup categary
NAIA::Category cat = NAIA::Category::ChargeGT2_Trk |
NAIA::Category::ChargeGT2_Tof;

logger->debug("Enabling masks for charge > 2");

auto chrono_start = std::chrono::system_clock::now();

// event loop
for (NAIA::Event &event : chain) {

// check charge with TOF or Tracker
if (!MatchAnyBit(event.header->Mask(), cat))
continue;
nSel_Category++;

// selections

}

auto chrono_end = std::chrono::system_clock::now();
std::chrono::duration<double> time_spent = chrono_end -
chrono_start;
```

Output with NAIA::Category cut:
Processed 622923 entries, 27438 passed category cut, 104 selected for IL1, and 16 for FS, in 10.33 seconds

NAIA::Category

```
auto chrono_start = std::chrono::system_clock::now();

// event loop
for (NAIA::Event &event : chain) {

auto Rigidity_IL1 = event.trTrackBase-
>Rigidity[fitType][NAIA::TrTrack::Span::InnerL1];
auto Rigidity_FS = event.trTrackBase-
>Rigidity[fitType][NAIA::TrTrack::Span::FullSpan];

// check charge with TOF or Tracker
if (!MatchAnyBit(event.header->Mask(), cat))
continue;
nSel_Category++;

// selections

}

auto chrono_end = std::chrono::system_clock::now();
std::chrono::duration<double> time_spent = chrono_end -
chrono_start;
```

Output: *Processed 622923 entries, 27438 passed category check, 104 selected for IL1, and 16 for FS, in 79.5 seconds*

```
auto chrono_start = std::chrono::system_clock::now();

// event loop
for (NAIA::Event &event : chain) {

// check charge with TOF or Tracker
if (!MatchAnyBit(event.header->Mask(), cat))
continue;
nSel_Category++;

auto Rigidity_IL1 = event.trTrackBase-
>Rigidity[fitType][NAIA::TrTrack::Span::InnerL1];
auto Rigidity_FS = event.trTrackBase-
>Rigidity[fitType][NAIA::TrTrack::Span::FullSpan];

// selections

}

auto chrono_end = std::chrono::system_clock::now();
std::chrono::duration<double> time_spent = chrono_end -
chrono_start;
```

Output: *Processed 622923 entries, 27438 passed category check, 104 selected for IL1, and 16 for FS, in 10.3 seconds*

NAIA Selections Library (NSL)

In NAIA it's possible to include a common selections library: NSL

With NSL it's possible to:

- Access to different standard selections
- Create you own selections

```
// NSL headers
#include <NSL/AllSelections.h>


auto fitType = NAIA::TrTrack::Fit::GBL;
auto InnerSpan = NAIA::TrTrack::Span::InnerOnly;
auto InnerL1Span = NAIA::TrTrack::Span::InnerL1;

//----- defining selection

NSL::Selection triggerSel =
NSL::Selections::Trigger::HasPhysicsTrigger();

// event loop
for (NAIA::Event &event : chain) {

  //----- Apply selection
  if (!triggerSel(event)) {
  continue;

}
```

```
// our headers
#include "NSL/Trigger/HasPhysicsTrigger.h"

// c++ headers
#include <memory>

NSL::Selections::Trigger::HasPhysicsTrigger::HasPhysicsTrigger() {
m_matcher = std::make_shared<boolMatcher>([](Event &event) {
event.evSummary->RestorePhysBPatt(true);
return event.evSummary->IsPhysicsTrigger();
});
}
```

NAIA Selections Library (NSL)

With NSL it's possible to concatenate different selections

```
// NSL headers
#include <NSL/AllSelections.h>


auto fitType = NAIA::TrTrack::Fit::GBL;
auto InnerSpan = NAIA::TrTrack::Span::InnerOnly;
auto InnerL1Span = NAIA::TrTrack::Span::InnerL1;

//----- defining IL1 selection

NSL::Selection innerL1Sel;

innerL1Sel = NSL::Selections::Trigger::HasPhysicsTrigger() &&
NSL::Selections::InnerTracker::HitPattern() &&
NSL::Selections::Track::ChiSquareLessThan(10.0f, NAIA::TrTrack::Side::Y, fitType, InnerSpan) &&
NSL::Selections::Track::HitCut(1) &&
NSL::Selections::Track::ChiSquareLessThan(10.0f, NAIA::TrTrack::Side::Y, fitType, InnerL1Span) &&
NSL::Selections::Track::L1NormResidualLessThan(10.0f, fitType);

. . .

// event loop
for (NAIA::Event &event : chain) {

  //----- Applying IL1 selections
  if (!innerL1Sel(event)) {
  continue;

}
```

NAIA Selections Library (NSL)

With NSL it's possible to make requests before/after each selection

```
// NSL headers
#include <NSL/AllSelections.h>

auto fitType = NAIA::TrTrack::Fit::GBL;
auto InnerSpan = NAIA::TrTrack::Span::InnerOnly;

// define lambda function to plot counts after each cut
auto counters = new TH1D("counters", "", ncuts, -0.5, ncuts - 0.5);
auto fill_counters = [&counters, &icut, &labels](Event &event) {
counters->Fill(icut);
icut++;
};

//----- defining IL1 selection
NSL::Selection innerL1Sel;

innerL1Sel = NSL::Selections::Trigger::HasPhysicsTrigger().AddPostHook(fill_counters) &&
NSL::Selections::InnerTracker::HitPattern().AddPostHook(fill_counters) &&
NSL::Selections::Track::ChiSquareLessThan(10.0f, NAIA::TrTrack::Side::Y, fitType, InnerSpan).AddPostHook(fill_counters);
. . .

// event loop
for (NAIA::Event &event : chain) {

  //----- Applying IL1 selections
  if (!innerL1Sel(event)) {
  continue;

}
```