Introduction to LHCb software and DaVinci (Run 1+2)

LHCb Starterkit 2022

Miguel Fernández Gómez — November 30, 2022

Introduction

- My name is Miguel Fernández, I am a 2nd year PhD student from Santiago de Compostela.
- During my first year, I've worked on $K^0_{S(L)} \to 4\mu$ (LHCb-PAPER-2022-035) and I'm also working on $\eta^{(\prime)} \to \mu^+\mu^-$ and $\eta^{(\prime)} \to \mu^+\mu^-\pi^+\pi^-$ decays (in development).
- This lesson will explore some basic LHCb software terms and will (likely) be your first introduction to creating ntuples for your analyses.
- I've had to create the ntuples for the above analyses from scratch…
- But that's as far as my experience goes
- I am no expert, so we'll try to learn together
- No question is too stupid, we are here to learn!

Materials

- We will be following these lessons from the “First Analysis Steps” section of the Starterkit page (https://lhcb.github.io/starterkit-lessons/first-analysis-steps/README.html):
    - An introduction to LHCb Software
    - Running a minimal DaVinci job locally
    - Fun with LoKi functors
    - TupleTools and branches
    - How do I use DecayTreeFitter?
- It’s a lot!
- I’ve also reused and repurposed some of the content there, as well as slightly expanded on it. This is where you can find my repository, with the lessons and all the code:

  https://gitlab.cern.ch/femiguel/davinci-lessons
- The lessons there and the code, as well as this presentation, go well beyond today’s lesson and are intended as supplementary material
- Let’s start!

LHCb Software

- Running a particle physics detector involves a lot of challenges
- Focus today: software-based challenges
- Gaudi framework was designed by LHCb to address many of them
- It allows to analyze events one by one, instead of holding all of them in memory at once → **EventLoop**
- It organizes the event information so that it is easily accessible → **Transient Event Store**
- We can also apply algorithms and tools to perform certain tasks, apply filters, perform calculations, and more → **Option files**

Running LHCb software

- One of the main projects based on Gaudi used by LHCb is DaVinci, which is used for ntuple-making, for instance (our goal today).
- On a practical matter, DaVinci is essentially a collection of tools and algorithms inside of an environment
- To start an environment in LHCb software, we use the key term `lb-run`
- We will be using version v46r4 of DaVinci
- To initialise this, we simply type:

```
$ lb-run DaVinci/v46r4 gaudirun.py
```

NOTE: This symbol is not to be typed. It is just to indicate that we are running this command on our terminal

Running DaVinci

- gaudirun.py is a script that sets up Gaudi's EventLoop
- So far, nothing else has been done, as we haven't specified any algorithms to run.
- This is done through python modules called option files. Assume we create an options file named options1.py with some algorithms to run. To execute it, we simply type on our terminal:

```
$ lb-run DaVinci/v46r4 gaudirun.py options1.py
```

- We will be doing this in a few minutes
- To avoid typing such a long command every time, we can also set ourselves in the DaVinci environment by typing:

```
$ lb-run DaVinci/v46r4 $SHELL
```

- To run options1.py now, we simply type:

```
$ gaudirun.py options1.py
```

Which DaVinci version should we run?

- The formal answer is to use the last version available. In our case, this is v46r4.
- Versions above v50 are Run 3-oriented, while versions below v50 are Run 1&2-oriented
- My personal preference is to use the DaVinci version used for the stripping
- We can access this, for instance, by scrolling down through the Python file we downloaded from the bookkeeping, and reading the commented parts right before the LFN paths.
- Nonetheless, it is crucial to be consistent with the DaVinci version throughout the analysis.

Can we get to it already?

- Here we are! We are now set to for running our first DaVinci job

- We will be studying the decay $D^{\star+} \to D^0(\to K^+K^-)\pi^+$, using the Monte Carlo DST file we downloaded in the previous lesson

- We want to create a ROOT file where we store the events inside this DST that pass our stripping line, D2hhPromptDst2D2KKLine

- A **stripping line** is a collection of filters and selections that allow us to narrow down our decay search

- We now create a new file named ntuple_options.py and write the following:

```python
from Configurables import DecayTreeTuple

## Specify the stream and stripping line
stream = "AllStreams"
line   = "D2hhPromptDst2D2KKLine"

## We create the DecayTreeTuple object, and indicate the Input
## (i.e., the TES location where the desired candidates may be)
## as well as the decay descriptor
dtt = DecayTreeTuple("TupleDstToD0pi_D0ToKK")
dtt.Inputs = ["/Event/{0}/Phys/{1}/Particles".format(stream, line)]
dtt.Decay  = "[D*(2010)+ -> (D0 -> K- K+) pi+]CC"
```

Configuring DaVinci

- Before we run the code, we need to specify some DaVinci attributes like the year of release, MC tags, etc. We include:

```
from Configurables import DaVinci

DaVinci().UserAlgorithms += [dtt]
DaVinci().InputType       = "DST"
DaVinci().TupleFile       = "DVntuple.root"
DaVinci().PrintFreq       = 1000
DaVinci().DataType        = "2016"
DaVinci().Simulation      = True
DaVinci().Lumi            = not DaVinci().Simulation
DaVinci().EvtMax          = -1
DaVinci().CondDBtag       = "sim-20170721-2-vc-md100"
DaVinci().DDDBtag         = "dddb-20170721-3"
```

Configuring DaVinci (2)

- **UserAlgorithms** indicates the algorithm to be run over the events.
- **InputType** should be 'DST' for DST files (and 'MDST' for microDST).
- **TupleFile** is the name of the output ROOT file, where the TTree will be stored.
- **PrintFreq** is the frequency of events with which DaVinci will print the status.
- **DataType** is the year of data-taking this corresponds to.
- **Simulation** is either True when dealing with Monte Carlo files, or False when using LHCb-taken data.
- **Lumi** is set to True if we want to store information on the integrated luminosity.
- **EvtMax** is the number of events to run over, where -1 indicates to run over all events.
- **CondDBtag** and **DDDBtag** are the exact detector conditions that the Monte Carlo was generated with. It contains, for instance, information on the magnet polarity.

Running the code

- Finally, we just need to indicate DaVinci what's the data we're running
- We give it the path to the DST file we downloaded. Assuming it's on the same directory as ntuple_options.py, we simply type:

```
from GaudiConf import IOHelper

IOHelper().inputFiles([
        "./00070793_00000001_7.AllStreams.dst"
], clear=True)
```

- We can now run the file:

```
$ lb-run DaVinci/v46r4 gaudirun.py ntuple_options.py
```

Quick note on how to obtain database tags

- There are several ways to do this. Here is a bit of a tedious version that is explained in the Starterkit lessons
- We get the Production ID from the bookkeeping location. In the case of `/lhcb/MC/2016/ALLSTREAMS.DST/00070793/0000/00070793_00000002_7.AllStreams.dst`, the production ID is `00070793`.
- We go to the transformation monitor and insert this number in the field ProductionID(s)
- Right click -> Show request -> Right click -> View
- Inside “Step 1” we get both tags

An alternate way to obtain the tags

- We can also read the tags directly from the downloaded DST file.
- However, there is even a way to access them without downloading the DST files.
- In the terminal, outside of the DaVinci environment, simply type:

```
$ lb-dirac dirac-bookkeeping-decays-path <eventtype>
```

- where `<eventtype>` is the number we used yesterday to find our DST inside the bookkeeping (more on what it means in today's session). In the case of our decay, it is `27163002`

Getting more out of our DST

- We have now run our first DaVinci job and created a very basic ROOT output file.
- The question is now how can we get more information out of it, fully exploring DaVinci's potential.
- We currently only have information on the mother particle.
- For any other particle we want information on, we add a ^ symbol in the Decay parameter:

```
dtt.Decay  = "[D*(2010)+ -> ^(D0 -> ^K- ^K+) ^pi+]CC"
```

- We can also customize the names of the branches through the `addBranches` function.

```
from DecayTreeTuple.Configuration import addBranches

dtt.addBranches({"Dstar" : "[D*(2010)+ -> (D0 -> K- K+) pi+]CC",
                 "D0"    : "[D*(2010)+ -> ^(D0 -> K- K+) pi+]CC",
                 "Kminus": "[D*(2010)+ -> (D0 -> ^K- K+) pi+]CC",
                 "Kplus" : "[D*(2010)+ -> (D0 -> K- ^K+) pi+]CC",
                 "pisoft": "[D*(2010)+ -> (D0 -> K- K+) ^pi+]CC"})
```

Let’s take a break!

What if we want to add even more info?

- The key from now on is how can we make our data files more complete.
- TupleTools provide us with ways to add more information to our ntuples
- They are a collection of algorithms that select specific pieces of information
- Examples:
  - TupleToolKinematic fills the kinematic information of the decay
  - TupleToolPid stores DLL and PID information of the particle

Adding TupleTools

- There are different ways of adding TupleTools into our DecayTreeTuple object
- Simplest way:

```
dtt.ToolList = ["TupleToolEventInfo",
                "TupleToolANNPID",
                "TupleToolGeometry",
                "TupleToolKinematic"]
```

- To add more configuration:

```
track_tool         = dtt.addTupleTool("TupleToolTrackInfo")
track_tool.Verbose = True
```

- Alternative way:

```
dtt.addTupleTool("TupleToolPrimaries")
```

- Note: TupleToolProperTime needs the mother particle to come from the PV
- We can ensure that happens using a Configurable named CheckPV, or through the use of a LoKi__VoidFilter object.

How to require PV

- Easy way out:

```
from Configurables import CheckPV
DaVinci().UserAlgorithms += [CheckPV(), dtt]
```

- A more sophisticated approach:

```
from Configurables import LoKi__VoidFilter

pv = LoKi__VoidFilter("hasPV",Code="CONTAINS('Rec/Vertex/Primary')>0")
DaVinci().UserAlgorithms += [pv, dtt]
```

- Either object created needs to then be added to the `UserAlgorithms` before the `dtt` object.

- To make sure only the events that come from the PV will then be applied the DecayTreeTuple algorithms, we can use a `GaudiSequencer`.

```
from Configurables import LoKi__VoidFilter, GaudiSequencer

pv = LoKi__VoidFilter("hasPV",Code="CONTAINS('Rec/Vertex/Primary')>0")
gs = GaudiSequencer("myseq")
gs.Members += [pv, dtt]

DaVinci().UserAlgorithms += [gs]
```

LoKi functors

- LoKi functors are to DSTs what branches are to ROOT files
- They help us interact with the DST and apply cuts before creating ROOT files
- They are used, for instance, when writing stripping lines
- We can implement them easily into our DaVinci scripts to create new variables

```
mainTools = dtt.D0.addTupleTool("LoKi::Hybrid::TupleTool/MyCustomVars")
mainTools.Variables = {
        "MinDght_P": "PFUNA(AMINCHILD(P))",
        "MaxDght_P": "PFUNA(AMAXCHILD(P))",
}
```

Monte Carlo Decay Tree Tuples

- We've now seen the basics of how to access and treat the data that passes our stripping.
- But in Monte Carlo, it's often useful to also know how many events we have in total, for instance, to compute efficiencies.
- For this, we need to go back to the very basics, even before implementing the detector and the reconstruction.
- We use MC DecayTreeTuples, where we are accessing truth-level variables (variables that did not go through reconstruction, and instead were randomly generated by Pythia8, for instance).

```
from Configurables                 import MCDecayTreeTuple
from DecayTreeTuple.Configuration import addBranches

mctuple        = MCDecayTreeTuple("MCDecayTreeTuple")
mctuple.Decay  = "[D*(2010)+ => (D0 ==> K- K+) pi+]CC"
## Add here further configuration (branch names, TupleTools, etc.)

DaVinci().UserAlgorithms += [mctuple]
```

A good place to either stop or take another break!

Catalogs

- When working with DaVinci, in general, there is no need to download any DST files locally, as we can always access them directly through the grid
- To do this, we can use catalogs
- Let's create a new directory called `catalogs`, where we move our `*ALLSTREAMS.DST.py` file, and create a copy of it with a different name. For instance, `testcatalog.py`
- Opening `testcatalog.py`, let's remove all LFNs except for a couple, that we will use for testing
- We now run the following command:

```
$ lb-dirac dirac-bookkeeping-genXMLCatalog --Options=testcatalog.py
                        --Catalog=myCatalog.xml
```

Accessing catalogs from DaVinci

- Going back to our options file now, we can substitute the IOHelper lines that call to our DST file by the following:

```
from Gaudi.Configuration import FileCatalog

FileCatalog().Catalogs = [“xmlcatalog_file:/catalogs/mycatalog.xml”]
```