FCCAnalyses*: tools and algorithms for analysis

Clement Helsens CERN-EP
Virtual FCC week, June 2021

Outline
• Introduction
• Overall Structure
• Workflow with one example

* This is not specific to FCC, but to the input data format
Introduction

FCCAnalyses:  https://github.com/HEP-FCC/FCCAnalyses/

- Set of tools to for data analysis, based on LHC experience
  - Originally developed for FCC-hh physics studies
  - Was based on a CMS python analysis framework (event loop slow)
  - Was used to produced FCC-hh physics analyses published in the CDR

- Transitioned to RDataFrame
  - State of the art in HEP (considering it is ROOT based)
  - Achieved much higher performances
  - Fully compatible with EDM4hep
Common software?

- **FCCAnalyses**
  - Is a common tool for analyzing large amount of data using RDataFrame
  - Is composed of a library of C++ analysers and python configuration files
    - C++ analysers are developed in common
    - Python code specific to the analysis, define analysers, output variables, input samples, etc…
  - Use the produced ROOT ntuples
    - To produce, with common tools, final variables for analysis, for MVA training, and for plotting

- **EDM4Hep**
  - EDM4Hep is the common event data model
  - Will be used for the very long term for any simulation
    - better to start using it now!
Overall structure for case studies - 1

- Set of tools to help processing the output of ‘simulation’
  - Agnostic to the type of simulation but specific reader functions are required
  - Build a common set of utility functions, algorithms for common use
  - Users to test their algorithms locally, or in dedicated repositories before publishing them

**Analysis configuration**

4 python scripts to configure:
1. Samples to run over (common production, EDM4Hep)
2. Functions/algorithm to call to produce variables of interests in a flat ntuple
3. Event selection and histograms definition using the flat ntuples
4. Plotting configuration

https://github.com/HEP-FCC/HEP-FCC/FCCeePhysicsPerformance/ Analysis configuration part close to the case studies and analysis specific code (like fitting, specific calculations, etc...)

Overall structure for case studies - 2

- Analysis configuration -> in FCCAnalyses-config
  - 4 python scripts to configure:
    1. Samples to run over (common production, EDM4Hep)
    2. Functions/algorithm to call to produce variables of interests in a flat ntuple
    3. Event selection and histograms definition using the flat ntuples
    4. Plotting configuration
Overall structure for case studies - 3

- **Code specific to the case study -> in dataframe/analyzers**
  - Code your ideas here before pushing them for more general usage
  - Could be very specific to a case study
  - But a case study could also be just fine using all the functionalities/algorithms that are available in the main library
Foresee overall structure for case studies

**Analysis configuration**
4 python scripts to configure:
1. Samples to run over (common production, EDM4Hep)
2. Functions/algorithm to call to produce variables of interests in a flat ntuple
3. Event selection and histograms definition using the flat ntuples
4. Plotting configuration

**Analysis Algorithms**
Dedicated C++ algorithms not available in the common library

Analysis configuration part close to the case studies and analysis specific code (like fitting, specific calculations, etc…)

https://github.com/HEP-FCC/FCCAnalyses/
Centrally installed on cvmfs and accessible by common setup. But totally OK to use private version.

https://github.com/key4hep/k4Analysis
Centrally installed on cvmfs and accessible by common setup. But totally OK to use private version.

---

**Common utility functions, algorithm, dedicated to FCC**
C++ library

**Common utility functions, algorithm, of interest for any k4Analyses using EDM4Hep**
C++ library
Workflow with one Example
B0 -> mumu  (took me ~45min to produce signal events and write the code, 1h to run it)

Produce signal events with EventProducer (not covered in this talk)

```
python bin/run.py --FCCee --reco --send -p p8_ee_Zbb_ecm91_EvtGen_Bd2MuMu -n 100000
--type p8 -N 10 --condor -q microcentury --version spring2021 --detector IDEA
```
Common samples (official productions) are available on eos and can be accessed through a database. The sample name is the “key”
But still possible to run with your own files
B0 -> mumu github
B0 -> mumu

Produce signal events with EventProducer (not covered in this talk)

```bash
python bin/run.py --FCCee --reco --send -p p8_ee_Zbb_ecm91EvtGen_Bd2MuMu -n 100000 --type p8 -N 10 --condor -q microcentury --version spring2021 --detector IDEA
```

Run first stage selection (run over $10^9$ Z->bb, and 100k B0->MuMu)

```bash
python examples/FCCee/flavour/Bd2MuMu/preSel.py
```
```python
# def run(self):
    df2 = self.df

    # Aliases for # in python
    .Alias("MCRecoAssociations0", "MCRecoAssociations#0.index")
    .Alias("MCRecoAssociations1", "MCRecoAssociations#1.index")
    .Alias("Particle0", "Particle#0.index")
    .Alias("Particle1", "Particle#1.index")

    # Build MC Vertex
    .Define("MCVertexObject", "myutils::get_MCVertexObject(Particle, Particle0)")

    # Build Reco Vertex
    .Define("VertexObject", "myutils::get_VertexObject(MCVertexObject, ReconstructedParticles, EFlowTrack_1, MCRecoAssociations0, MCRecoAssociations1)")

    # Build PV var and filter
    .Define("EVT_hasPV", "myutils::hasPV(VertexObject)")
    .Define("EVT_NtracksPV", "myutils::get_PV_ntracks(VertexObject)")
    .Define("EVT_Nvertex", "VertexObject.size()")
    .Filter("EVT_hasPV==1")
```
```python
# Build new RecoP with PID
.ReDefine("RecoPartPID", "myUtils::PID(ReconstructedParticles, MCRecoAssociations0, MCRecoAssociations1, Particle)")

# Build new RecoP at vertex
.ReDefine("RecoPartPIDatVertex", "myUtils::get_RPCatVertex(RecoPartPID, VertexObject)"

# Build B0 -> MuMu candidates
.ReDefine("Bd2MuMuCandidates", "myUtils::build_Bd2MuMu(VertexObject, RecoPartPIDatVertex)"

# Filter B0 -> MuMu candidates
.ReDefine("EVT_NBd2MuMu", "float(myUtils::getFCCAnalysesComposite_N(Bd2MuMuCandidates))" .Filter("EVT_NBd2MuMu==1")

# Get the B0 -> MuMu candidate mass
.ReDefine("Bd2MuMu_mass", "myUtils::getFCCAnalysesComposite_mass(Bd2MuMuCandidates)"
```

**PID/B0 -> mumu**
```python
### Build new RecoP with PID ###
.
Define("RecoPartPID", "myUtils::PID(ReconstructedParticles, MCRecoAssociations0, MCRecoAssociations1, Particle)")

### Build new RecoP with PID at vertex ###
.
Define("RecoPartPIDatVertex", "myUtils::get_RP_atVertex(RecoPartPID, VertexObject)")

### Build B0 -> MuMu candidates ###
.
Define("Bd2MuMuCandidates", "myUtils::build_Bd2MuMu(VertexObject, RecoPartPIDatVertex)")

### Filter B0 -> MuMu candidates ###
.
Define("Evt_Pass2MuMu", "float(myUtils::getFCCAnalysesComposite_N(Bd2MuMuCandidates))")
.Filter("Evt_Pass2MuMu,Mu==1")

### Get the B0 -> MuMu candidate mass ###
.
Define("Bd2MuMu_mass", "myUtils::getFCCAnalysesComposite_mass(Bd2MuMuCandidates)")
```
# Build new RecoP with PID

```python
.Define("RecoPartPID", "myUtils::PID(ReconstructedParticles, MCRecoAssociations0, MCRecoAssociations1, Particle)"
```

# Build new RecoP with PID at vertex

```python
.Define("RecoPartPIDatVertex", "myUtils::get_RP_atVertex(RecoPartPID, VertexObject)"
```

# Build B0 -> MuMu candidates

```python
.Define("Bd2MuMuCandidates", "myUtils::build_Bd2MuMu(VertexObject, RecoPartPIDatVertex)"
```

# Filter B0 -> MuMu candidates

```python
.Filter("Evt_NBd2MuMu", "float(myUtils::getFCCAnalysesComposite_N(Bd2MuMuCandidates))"
```

# Get the B0 -> MuMu candidate mass

```python
.Define("Bd2MuMu_mass", "myUtils::getFCCAnalysesComposite_mass(Bd2MuMuCandidates)"
```

# select branches for output file

```python
branchList = ROOT.vector('string')()
for branchName in ["Bd2MuMu_mass"]:
    branchList.push_back(branchName)
```

df2.Snapshot("events", self.outname, branchlist)
B0 -> mumu candidate building

```cpp
// select candidates with exactly 2 muons and charge 0
if (nobj_Bd!=2) continue;
if (charge_Bd!=0) continue;

// build a composite vertex
FCCAnalysesComposite2 comp;
comp.vertex = i;
comp.particle = build_tlv(recop, vertex.at(i).reco_ind);
comp.charge = charge_Bd;

// add the composite vertex to the collection
result.push_back(comp);
}```
Run pre-selection

```python
#python examples/FCCee/flavour/Bd2MuMu/preSel.py

from config.common_defaults import deffccdicts
import config.runDataFrameBatch as rdf
import os

basedir=os.path.join(os.getenv('FCCDICTSDIR', deffccdicts), '') + "yaml/FCCee/spring2021/IDEA/
outdir="/eos/experiment/fcc/ee/analyses/case-studies/flavour/Bd2MuMu/flatNtuples/spring2021/Batch/"
NUM_CPUS=8
output_list=[]

fraction=1.

inputana="/afs/cern.ch/user/h/helsens/FCCsoft/HEP-FCC/FCCAanalyses/examples/FCCee/flavour/Bd2MuMu/analysis.py"
process_list=['p8_ee_Zbb_ecm91']
myana=rdf.runDataFrameBatch(basedir,process_list, outlist=output_list)
myana.run(ncpu=NUM_CPUS,fraction=fraction, chunks=50, outDir=outdir, inputana=inputana)

process_list=['p8_ee_Zbb_ecm91_EvtGen_Bd2MuMu']
import config.runDataFrame as rdf2
myana=rdf2.runDataFrame(basedir,process_list, outlist=output_list)
myana.run(ncpu=NUM_CPUS, fraction=fraction, outDir=outdir)
```
B0 -> mumu

Produce signal events with EventProducer (not covered in this talk)

```
python bin/run.py --FCCee --reco --send -p p8_ee_Zbb_ecm91EvtGen_Bd2MuMu -n 100000 --type p8 -N 10 --condor -q microcentury --version spring2021 --detector IDEA
```

Run first stage selection (run over $10^9$ Z->$bb$, and 100k B0->MuMu)

```
python examples/FCCee/flavour/Bd2MuMu/preSel.py
```

Run final selection and make histograms

```
python examples/FCCee/flavour/Bd2MuMu/finalSel.py
```
```python
#python examples/FCCee/flavour/Bd2MuMu/FinalSel.py

from config.common_defaults import defccdicts
import sys, os
import ROOT

###Input directory where the files produced at the pre-selection level are
baseDir = "/eos/experiment/fcc/ee/analyses/case-studies/flavour/Bd2MuMu/flastNtuples/spring2021/Batch/"

###Link to the dictionary that contains all the cross section informations etc...
procDict = os.path.join(os.getenv("FCCEMCXS_DIR", defccdicts), "") + "FCCEee_procDict_spring2021_2021.json"

process_list=["p0_e+Zbb_eag91_EvtGen_Bd2MuNu",
              "p0_e+Zbb_eag91"
             ]

define_list={}

###Dictionary of the list of cuts. The key is the name of the selection that will be added to the output file
cut_list = {"sel0":"Bd2MuMu_mass>8"}

###Dictionary for the output variable/histograms. The key is the name of the variable in the output files. "name" is the name
variables = {
    "EVT_CandMass" :{"name":"Bd2MuMu_mass","title":"mass [GeV]","bin":300,"xmin":0,"xmax":6.},
    "EVT_CandMass_zoom" :{"name":"Bd2MuMu_mass","title":"mass [GeV]","bin":100,"xmin":5.,"xmax":6.}
}

###Number of CPUs to use
NUM_CPUS = 4

###This part is standard to all analyses
import config.runDataFrameFinal as rdf
myana=rdf.runDataFrameFinal(baseDir,procDict,process_list,cut_list,variables,defines=define_list)
myana.run(ncpu=NUM_CPUS, doTree=True)
```
B0 -> mumu

Produce signal events with EventProducer (not covered in this talk)

```bash
python bin/run.py --FCCee --reco --send -p p8_ee_Zbb_ecm91_EvtGen_Bd2MuMu -n 100000 --type p8 -N 10 --condor -q microcentury --version spring2021 --detector IDEA
```

Run first stage selection (run over $10^9$ Z->bb, and 100k B0->MuMu)

```bash
python examples/FCCee/flavour/Bd2MuMu/preSel.py
```

Run final selection and make histograms

```bash
python examples/FCCee/flavour/Bd2MuMu/finalSel.py
```

Produce the plots

```bash
python config/doPlots.py examples/FCCee/flavour/Bd2MuMu/plots.py
```
Run plots

```python
import ROOT

# global parameters
intnum1 = 150000000. # in pb-1
ana_tex = "Z \rightarrow \mu\nu q\bar{q}(q)"
phoVersion = "3.4.2"
energy = 91.0
collider = "FCC-ee"
inputDir = "/eos/experiment/fcc/ee/analyses/case-studies/leflavour/BD2MU/flatNtuples/spring21/8inch/
formats = ["png", "pdf"]
yaxis = ["lin", "log"]
stacksig = ["nostack"]
outdir = "plots_Bd2MuMu/"

variables = ["EVT_CandMass", "EVT_CandMass_zoom"

scaleSig=1.

## dictionary with the analysis name as a key, and the list of selections to be plotted for this analysis.
## The name of the selections should be the same than in the final selection
selections = {}

selections["Bd2MuMu"] = ["sel0"]

extralabel = {}
extralabel["sel0"] = "Selection: M B\rightarrow\mu\nu\mu\nu\mu=1"

colors = {}
colors["Z_bb"] = ROOT.kBlue
colors["Z_Bd"] = ROOT.kRed

plots = {}
plots["Bd2MuMu"] = {
    "signal": ["Z_Bd": ["p8_e11_ee_Zbbar_ecm91_EvtGen_Bd2MuMu"],
               "backgrounds": ["Z_bb": ["p8_e11_ee_Zbb_ecm91"]]
}

legend = {}
legend["Z_Bd"] = "Z\rightarrow\mu\nu\mu\nu\mu"
legend["Z_bb"] = "Z\rightarrow\mu\nu\bar{\mu}\bar{\nu}"
```
Candidate mass plots

FCC-ee Simulation (Delphes)

$\sqrt{s} = 91.0$ GeV
$L = 150$ ab$^{-1}$

$Z \rightarrow q\bar{q}$

Selection: $N B \rightarrow \mu\mu = 1$
Summary

• Easy to use analysis code to produce root ntuples based on RDataFrame
• Extremely powerful and flexible
• Already actively used: flavour, higgs and top physics at FCC-ee
• Large scale usage with Bc->tauNu analysis (~$10^{10}$ events, 12h processing on batch with 9000HS06, not sure the fraction of it I got though)
• Already contains lot’s of functionalities
  • Jet clustering, vertexing, thrust/sphericity axis, simple PID (more accurate to come), etc…

• Started to move FCCAnalyses from HEP-FCC to Key4Hep/k4Analyses
  • https://github.com/key4hep/k4Analysis
  • New schema needs to be defined in the next weeks (where the various codes have to be)
  • Split C++ and python