Physics Analysis Software Framework for Belle II

Marko Starič

Belle II collaboration
Jožef Stefan Institute, Ljubljana

CHEP 2015
Belle II experiment

SuperKEKB accelerator
- asymmetric $e^+e^-$ collider (4 GeV / 7 GeV)
- nano-beam optics
- luminosity $40 \times$ KEKB ($8 \times 10^{35}$ cm$^{-2}$s$^{-1}$)

Belle II detector
- tracking and vertexing
- hadron and lepton ID
- $\gamma$ and $K_L$ detection
Belle II experiment

Broad physics program including

- B physics (rare decays, CPV)
- Charm physics (mixing, CPV, new resonances)
- Tau physics (LFV)

Physics analysis software must provide

- Exclusive and inclusive decay reconstructions
- Reconstruction of recoil particles
- Flavor tag
- Continuous background suppression
- Event pre-selection (skim) / output to micro dst
Belle II software framework (Basf2)

- Flexible framework build of
  - modules, which process data
  - data store, to share data between modules
- Modules are organized in a user-defined chain using python script
- Data store consists of a Root-storable objects
  - StoreArrays and StoreObjPtrs
  - relations between elements of different data objects
- The content of the data store (or its part) can be read from or written to a Root file at any point in the chain using dedicated modules
Basf2 ingredients

Modules

- A module can be put into path many times; each time a new instance is created.
- Data processing of a module can be steered by module parameters; parameters can be of types
  - int, float, std::string
  - std::tuple constructed of above types
  - std::vector of all above types

Data store objects

- template classes representing single object (StoreObjPtr) or an array of objects (StoreArray)
- multiple data store objects of a given type can be created
- distinguished by their names

Basf2 provides all ingredients to build a framework for the physics analysis.
Physics Analysis Framework

Data model

- **class Particle**: a common representation of a reconstructed particle
  - final state particle \((e, \mu, \pi, K, p, \gamma, K_L)\)
  - pre-reconstructed V0 particle \((K_S, \Lambda)\)
  - reconstructed in decay (via combinations)
  - internally holds Lorentz vector, vertex, error matrix, relation to daughter particles, PDG code, and some other informations.

- **StoreArray<Particle>**: array of all reconstructed particles
  - a work space
  - modules can append particles to this array
  - modules can modify particles in this array (vertex fit!)
  - modules can select particles from this array

- **class ParticleList**: list of particles of a given type (PDG code)
  - internally holds a vector of pointers (indices) to appropriate elements of StoreArray<Particle>
  - particle lists: single Data store objects distinguished by their names
  - name composed of a standard particle name and a label: \(pi+:slow\)
The basic modules

- **ParticleLoader**
  - appends particles constructed from mdst objects (reconstructed tracks, ECL clusters $K_L$, V0) to the work space

- **ParticleSelector**
  - selects particles from the work space (makes particle list) or
  - applies selection criteria to the list (by removing unselected)
  - selection criteria (Boolean expression) are given via module parameter

- **ParticleCombiner**
  - makes combinations from any number of input particle lists
  - appends combined particles to a work-space
  - creates particle list of combined particles

- **VertexFitter**
  - performs all kinds of vertex fits on particles from a given list
  - updates successfully fitted particles (on the work space) and connects (via framework relations) the vertex object or
  - removes badly fitted from the list
Other modules

- a module for best candidate selection
- a module for MC truth matching
- a module for continuum suppression
- TMVA teacher and expert modules
- a module for flavor tagging
- a module which builds and connects the rest-of-event to a particle
- a ntuple maker module (flat ntuple)
- ...

Additional data objects

- Vertex, RestOfEvent, FlavorTagInfo, EventExtraInfo, ...
- can be linked to particles via framework relations
Python steering

- Commands that represent dedicated actions are defined using python functions. Example:

```python
def reconstructDecay(decayString, cut, path=analysis_main):
    combiner = register_module('ParticleCombiner')
    combiner.param('decayString', decayString)
    combiner.param('cut', cut)
    path.add_module(combiner)
reconstructDecay('D0 -> K- pi+', '1.8 < M < 1.9')
```

- Decay string in this example defines input and output particle lists
  - charge conjugate states are implicitly included
- Selection criteria given as the second argument is parsed to a C++ representation during module initialization
  - variable names are replaced during parsing with function pointers
  - Boolean expression is per event evaluated using recursion
- Additional variables can be easily defined by user
Steering example: reconstruction of $D^{*+} \rightarrow D^0\pi^+, \ D^0 \rightarrow K^-\pi^+$

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from basf2 import *
from modularAnalysis import *
from stdLooseFSParticles import *

inputMdst('DstarSignalMC.mdst.root') # define mdst input file

stdVeryLoosePi()  # make lists of very loosely selected pions (pi+:all, pi-:all)
stdLoosePi()      # make lists of loosely selected pions (pi+:loose, pi-:loose)
stdLooseK()       # make lists of loosely selected kaons (K+:loose, K-:loose)

# reconstruct D0 -> K- pi+ + cc
reconstructDecay('D0 -> K-:loose pi+:loose', '1.7 < M < 2.0 and p_CMS > 2.2')
vertexKFit('D0', 0.001)
applyCuts('D0', '1.81 < M < 1.91')

# reconstruct D*+ -> D0 pi+ + cc
reconstructDecay('D*+ -> D0 pi+:all', 'Q < 0.05')
vertexKFit('D*+', 0.001)
applyCuts('D*+', 'Q < 0.02 and p_CMS > 2.5')

outputUdst('recDstar.udst.root', ['D*+'])  # write selected events to micro dst

process(analysis_main)  # process events
```
Conclusions

- An overview of the physics analysis software framework at Belle II has been given.
- The framework utilizes the Basf2 software framework, and uses python for steering.
- Although the framework is still under development, a user can already perform most of the physics analysis steps like decay reconstructions, vertex fits, tag the flavor of a $B$ meson and perform TMVA-based continuum suppression.
Backup: cpu usage

running steering example from slide 10 for 1000 events

<table>
<thead>
<tr>
<th>Name</th>
<th>Calls</th>
<th>Time(s)</th>
<th>Time(ms)/Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>RootInput</td>
<td>1001</td>
<td>0.25</td>
<td>0.25 +/- 0.36</td>
</tr>
<tr>
<td>Progress</td>
<td>1000</td>
<td>0.01</td>
<td>0.01 +/- 0.01</td>
</tr>
<tr>
<td>Gearbox</td>
<td>1000</td>
<td>0.01</td>
<td>0.01 +/- 0.00</td>
</tr>
<tr>
<td>ParticleLoader_pi+:all</td>
<td>1000</td>
<td>0.17</td>
<td>0.17 +/- 0.04</td>
</tr>
<tr>
<td>ParticleLoader_pi+:loose</td>
<td>1000</td>
<td>0.16</td>
<td>0.16 +/- 0.04</td>
</tr>
<tr>
<td>ParticleLoader_K+:loose</td>
<td>1000</td>
<td>0.15</td>
<td>0.15 +/- 0.04</td>
</tr>
<tr>
<td>ParticleCombiner_D0 -&gt; K-:loose pi+:loose</td>
<td>1000</td>
<td>0.03</td>
<td>0.03 +/- 0.03</td>
</tr>
<tr>
<td>Geometry</td>
<td>1000</td>
<td>0.01</td>
<td>0.01 +/- 0.00</td>
</tr>
<tr>
<td>ParticleVertexFitter_D0</td>
<td>1000</td>
<td>0.14</td>
<td>0.14 +/- 0.15</td>
</tr>
<tr>
<td>ParticleSelector_applyCuts_D0</td>
<td>1000</td>
<td>0.01</td>
<td>0.01 +/- 0.00</td>
</tr>
<tr>
<td>ParticleCombiner_D*_+ -&gt; D0 pi+:all</td>
<td>1000</td>
<td>0.02</td>
<td>0.02 +/- 0.01</td>
</tr>
<tr>
<td>ParticleVertexFitter_D*_+</td>
<td>1000</td>
<td>0.11</td>
<td>0.11 +/- 0.14</td>
</tr>
<tr>
<td>ParticleSelector_applyCuts_D*_+</td>
<td>1000</td>
<td>0.01</td>
<td>0.01 +/- 0.00</td>
</tr>
<tr>
<td>RootOutput</td>
<td>1000</td>
<td>0.70</td>
<td>0.70 +/- 1.69</td>
</tr>
</tbody>
</table>

Total                                             | 1001  | 2.00    | 2.00 +/- 1.77 |