An Interactive and Comprehensive Working Environment for High-Energy Physics Software with Python and Jupyter Notebooks

CHEP 2016 – San Francisco, United States of America

Nils Braun¹, Thomas Hauth¹, Christian Pulvermacher², Martin Ritter³
¹ Institute for Experimental Nuclear Physics (IEKP), KIT, Germany
² KEK, Tsukuba, Japan
³ Ludwig-Maximilians-University, Munich, Germany
SuperKEKB and the Belle II Experiment

- KEKB was an electron–positron collider in Tsukuba/Japan which studied the decay of B mesons at the Y(4S) resonance
- Discovery of CP violation in the B system by the Belle experiment at KEKB and by the BaBar experiment at SLAC
  - Nobel Prize in Physics to Kobayashi and Maskawa in 2008
- The SuperKEKB collider and the Belle II detector will build on the previous success:
  - Study B meson system in far greater precision
  - Probe for new physics in a wide range of interesting topologies

<table>
<thead>
<tr>
<th></th>
<th>KEKB</th>
<th>SuperKEKB</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instantaneous Luminosity</td>
<td>$2 \times 10^{34}$ cm$^{-2}$s$^{-1}$</td>
<td>$8 \times 10^{35}$ cm$^{-2}$s$^{-1}$</td>
<td>40</td>
</tr>
<tr>
<td>Integrated Luminosity</td>
<td>1 ab$^{-1}$</td>
<td>50 ab$^{-1}$ (projected)</td>
<td>50</td>
</tr>
<tr>
<td>Runtime</td>
<td>1998 to 2010</td>
<td>start in 2017</td>
<td>-</td>
</tr>
<tr>
<td>Detector</td>
<td>Belle</td>
<td>Belle II</td>
<td>-</td>
</tr>
<tr>
<td>Raw Data</td>
<td>1 PB</td>
<td>100 PB (projected)</td>
<td>100</td>
</tr>
</tbody>
</table>
The basf2 Framework

- Event processing framework for Belle II written in C++11
- Modules use a common data store to read event data and write back results
- All processing steps for recorded and simulated events are implemented in basf2:
  Event generation, simulation, digitization, HLT, reconstruction and analysis
- Important libraries are bundled into *externals*: ROOT, gcc, Geant4, …

Python is a first-class citizen in our framework:

- Steering files connect modules to paths and are written in Python 3
- Modules can be written in C++ and Python
- Framework functionality is exported to Python via *boost::python*
- User classes and all objects within the **DataStore** are available in Python via **PyROOT**
Jupyter Notebooks

Jupyter inherits all benefits available to Python in our framework, plus:

- **Notebooks**
  - Save expressions and corresponding results in one place
  - Include comments, documentation, pictures, drawings, \LaTeX, videos
  - Send notebooks, including all results, to someone else (use-cases: software examples, bug reports)
  - Perform analyses interactively

- **Clickable widgets in HTML and JavaScript**

- **Sections of a notebook can be executed individually**

- **Tab-completion and syntax highlighting**

- **Server–client structure**
  - Access the Jupyter service from your home computer, smartphone, tablet, etc. but run the calculations on a high-performance machine
  - No need to rely on X forwarding or other technologies

- **Many data science tools with Jupyter integration: ROOT, matplotlib, pandas**

- **Not only for Python (Haskell, Julia, C++, ROOT, Terminal)**
In [1]: import math
   print(math.pi)

3.141592653589793
Under the Hood: Integrating basf2

We developed the Python library hep_ipython_tools [1] which simplifies the integration of HEP Frameworks with jupyter notebooks.

Core component for seamless Jupyter integration of basf2:

**Process handler for background framework execution**

- Creates a separate worker process for basf2
- Transfers path configuration and starts processing
- Monitors running framework process
- Installs a message queue between jupyter and basf2 processes to transfer status information (current event number, performance statistics etc.)
- Can support multiple basf2 Instances to concurrently scan a parameter space
- Implementation is generic and can be easily adapted to support other frameworks
Better User Experience: Widgets

- Jupyter Widgets are graphical extensions to notebooks to better view use-case specific contents
- Written in Python and JavaScript, running interactively in the user’s browser
- Allows to use rich library ecosystem of Python and web-development world (jQuery, HTML5, CSS etc.)

We developed a set of Jupyter widgets to improve the user experience of basf2 in Notebooks

Progress Bar

Status: finished

85 % Remaining time: 2.04 s

30 % Remaining time: 26.92 s
Better User Experience: Widgets

Collection Viewer

Log Parser
Create path

In [3]: # Create BASF2 path and fill it with BASF2 modules
path = create_path()
    importData(filename, path=path)
loadRunBox(path=path)
# Load final state particles
fillParticleList(mu+, 'mu1 > 0.5', path=path)
fillParticleList(pi+, 'pico1 > 0.5', path=path)
# Reconstruct our decay using some soft cuts on the invariant mass
reconstructDecay(J/psi -> mu+ mu-, '2 < M < 4', path=path)
reconstructDecay(KS0 -> pi+ pi-, '0.1 < M < 1', path=path)
reconstructDecay(B0 -> J/psi K S0, '4 < M < 6', path=path)
buildRestOfEvent(B0 sig, path=path)
# Fit Vertex
fitVertex(J/psi1, conf_level=0.8, path=path)
fitVertex(K S0, conf_level=0.8, path=path)
fitVertex(B0 sig, conf_level=0.8, path=path)
# Match MC Truth
matchMCTruth(J/psi, path=path)
matchMCTruth(K S0, path=path)
matchMCTruth(B0 sig, path=path)
# Apply Flavour Tagging and Tag-Side Vertexing
TagFlavour(B0 sig, 'breco', path=path)
# Write out information
variablesToTuple(mu+, ['isSignal', 'mu1', 'chiProb', 'charge'], filename='muons.root', path=path)
variablesToTuple(pi+, ['isSignal', 'pico1', 'chiProb', 'charge'], filename='piions.root', path=path)
variablesToTuple(J/psi, ['isSignal', 'M', 'distance', 'chiProb'], filename='Jpsis.root', path=path)
variablesToTuple(KS0, ['isSignal', 'M', 'distance', 'chiProb'], filename='Kshors.root', path=path)
variablesToTuple(B0 sig, ['isSignal', 'M', 'Mbc', 'deltaE', 'distance', 'chiProb', 'Delta', 'MCtagFlavor'], filename='Bbars.root', path=path)

Process path

In [5]: # Run BASF2 path in the background on the data using ipython_handler
calculation = ipython_handler.run_command(process(path)
calculation.start()}
calculation.wait_for_end(False)
Full Analysis Example

Read input files

```
In [6]: Jpsi = root_pandas.read_root('Jpsi.root')
Kshorts = root_pandas.read_root('Kshorts.root')
B0s = root_pandas.read_root('B0s.root')
B0s.describe()
```

```
Out[6]:
```
<table>
<thead>
<tr>
<th></th>
<th>isSignal</th>
<th>M</th>
<th>Mbc</th>
<th>deltaE</th>
<th>distance</th>
<th>chiProb</th>
<th>DeltaT</th>
<th>MCTagBFavor</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td></td>
<td>37777.000000</td>
<td>37777.000000</td>
<td>37777.000000</td>
<td>37777.000000</td>
<td>37777.000000</td>
<td>37777.000000</td>
<td>37777.000000</td>
</tr>
<tr>
<td>mean</td>
<td>0.117029</td>
<td>4.831285</td>
<td>5.028489</td>
<td>0.251113</td>
<td>1.872794</td>
<td>0.189709</td>
<td>6.236745</td>
<td>-4.287980</td>
</tr>
<tr>
<td>std</td>
<td>0.321459</td>
<td>18.471546</td>
<td>0.582552</td>
<td>38.040882</td>
<td>8.319394</td>
<td>0.302396</td>
<td>752.426514</td>
<td>510.447388</td>
</tr>
<tr>
<td>min</td>
<td>0.000000</td>
<td>1.213630</td>
<td>0.000000</td>
<td>-3.947679</td>
<td>0.000476</td>
<td>0.000000</td>
<td>-31209.353516</td>
<td>-511.000000</td>
</tr>
<tr>
<td>25%</td>
<td>0.000000</td>
<td>4.193513</td>
<td>5.008841</td>
<td>-0.796031</td>
<td>0.026089</td>
<td>0.000000</td>
<td>-5.144518</td>
<td>-511.000000</td>
</tr>
<tr>
<td>50%</td>
<td>0.000000</td>
<td>4.496523</td>
<td>5.141731</td>
<td>-0.536930</td>
<td>0.088729</td>
<td>0.000221</td>
<td>-0.027970</td>
<td>-511.000000</td>
</tr>
<tr>
<td>75%</td>
<td>0.000000</td>
<td>5.079098</td>
<td>5.241263</td>
<td>-0.062886</td>
<td>0.590760</td>
<td>0.305392</td>
<td>3.952079</td>
<td>511.000000</td>
</tr>
<tr>
<td>max</td>
<td>1.000000</td>
<td>3556.052979</td>
<td>5.286911</td>
<td>5543.626953</td>
<td>270.880885</td>
<td>1.000000</td>
<td>31147.080078</td>
<td>511.000000</td>
</tr>
</tbody>
</table>
```

Do some plots

```
In [7]: p = b2plot.Distribution(figure=plt.figure())
p.set_plot_options({'linestyle': '-', 'color': 'red'})
p.add(Jpsi[(Jpsi.M > 2) & (Jpsi.M < 4) & (Jpsi.isSignal == 1)], 'M')
p.set_plot_options({'linestyle': '-', 'color': 'blue'})
p.add(Jpsi[(Jpsi.M > 2) & (Jpsi.M < 4) & (Jpsi.isSignal == 0)], 'M')
p.finish()
```

12 October 2016
The shown notebook was already successfully tested with students in a tutorial at KIT.
Analysis in the cloud (analysis as a service)

The notebooks can be used for outreach (e.g. tutorials in universities and schools)

Jupyterhub provides a jupyter notebook server with authentication, user management, distributed computation/cluster support.

Prototype and evaluation setup is running successfully on a machine at KIT

In the future, we hope that data centers offer to host this kind of services
Perform Python calculations with Jupyter notebooks to have all benefits of Python together with the interactivity.

The lightweight software layer provided by hep_ipython_tools allows a **seamless integration of HEP frameworks** (here basf2) with interactive Jupyter notebooks.

Notebooks can be used for:
- Interactive development of framework module algorithms
- Working on analyses with fast feedback via inlined plots
- Self-describing Notebooks for tutorials and outreach

Using Jupyter(hub) with basf2 is a full environment for physics analysis!

In the future: possibilities for interactive physics analysis via the web browser, centrally hosted at data centers.
References I

“HEP IPython Tools.”