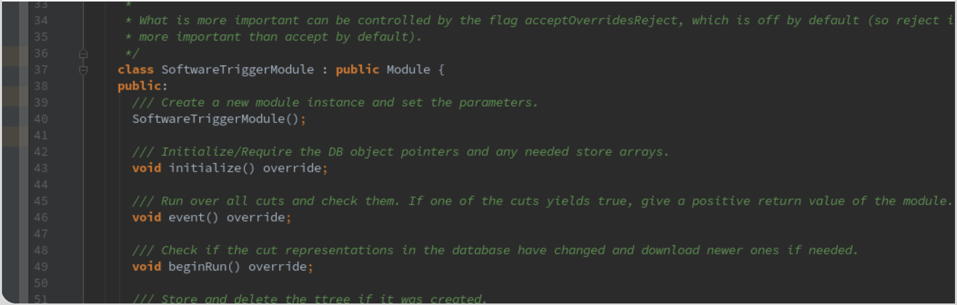
Karlsruher Institut für Technologie

Software Trigger Status and Next Steps

2017 Belle II TRG/DAQ Workshop

Thomas Hauth, Nils Braun | August 25, 2017

ETP - KIT



```
 *
 * What is more important can be controlled by the flag acceptOverridesReject, which is off by default (so reject i
 * more important than accept by default).
 */
class SoftwareTriggerModule : public Module {
public:
  /// Create a new module instance and set the parameters.
  SoftwareTriggerModule();

  /// Initialize/Require the DB object pointers and any needed store arrays.
  void initialize() override;

  /// Run over all cuts and check them. If one of the cuts yields true, give a positive return value of the module.
  void event() override;

  /// Check if the cut representations in the database have changed and download newer ones if needed.
  void beginRun() override;

  /// Store and delete the ttree if it was created.
```

Introduction

The Software Trigger Module

The software trigger module is a general module to perform cut decisions needed in the HLT software.

- Calculates predefined variables and applies predefined, downloaded cuts (from the database) on the event.
- Each event is classified as rejected or accepted.

Features of the Software Trigger Module

- Centralized framework for all cuts and decisions applied for the HLT software stack
- Database up- and download of the cut settings and tag names
- Versioning of all cuts through the condition database
- Easy to extend calculation framework for variables needed in the cuts
- Cuts based on the well-tested `GeneralCut` from the framework
- Easy to use python interface for quick cut development
- Nearly 100% unittest coverage

Relevant Software Trigger Channels

	Cross Section (nb)	Background
BB	1.1000	False
BB charged	0.5643	False
BB mixed	0.5357	False
$B \to J/\psi K_s ee$		False
$B \to \nu\nu$		False
$B \to \pi_0\pi_0$		False
$B \to \rho_0\gamma$		False
Continuum ($s\bar{s}$)	0.3800	False
Continuum ($d\bar{d}$)	0.4000	False
Continuum ($c\bar{c}$)	1.3000	False
Continuum ($u\bar{u}$)	1.6100	False

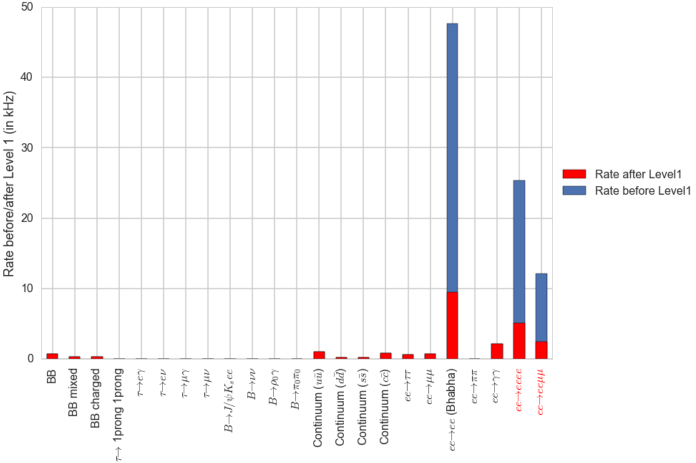
....

Relevant Software Trigger Channels

	Cross Section (nb)	Background
$ee \rightarrow ee$ (Bhabha)	74.4000	(False)
$ee \rightarrow eeee$	39.7000	True
$ee \rightarrow ee\mu\mu$	18.9000	True
$ee \rightarrow \gamma\gamma$	3.3000	False
$ee \rightarrow \mu\mu$	1.0730	False
$ee \rightarrow \pi\pi$		False
$ee \rightarrow \tau\tau$	0.9000	False
$\tau \rightarrow$ 1 prong 1 prong		False
$\tau \rightarrow e\gamma$		False

The numbers are taken from 'Overview of the Belle II Physics Generators' by P. Urquijo and T. Ferber.

Assumed Cross Section after Level 1



The rates of the background channels are scaled to arrive at 20 kHz for the following studies

Once the full TSim and L1 menu is available, the studies will be repeated

SoftwareTrigger Cut Module

A Software Trigger Cut (the main entity used in the Software Trigger Module) is defined by five properties:

The cut condition A string in a format known by the `GeneralCut` with variables defined by the calculator you choose (see below). *Example:* `[visible_energy < 1.8438] and [highest_3_ecl <= 0.3999]`

The cut type A reject or accept cut. See the next slide for the difference.

The prescale An accept cut can be statistically scaled down with this factor (will only lead to a positive result in one of *N* cases).

SoftwareTrigger Cut Module

A Software Trigger Cut (the main entity used in the Software Trigger Module) is defined by five properties:

The base identifier This string defines, which variables are calculated for the event content. Also, you can only choose between cuts with the same base identifier in one run of the software trigger module.
Example: `fast_reco` or `hlt`

The cut identifier This is the identifier making the cut downloadable from the database. Two cuts with the same identifier but different base identifiers are stored separately.
Example: `reject_bkg` or `accept_ee`

The three possible results

Each cut can either be an accept or a reject cut. The overall output of the module depends on this. There are different possibilities:

- `acceptOverridesReject` is set to False (default)
  - **One of the chosen reject cuts is true:** the whole event is tagged with ”rejected” and the module gives -1 as a return value
  - **One of the chosen accept cuts is true and no reject cut is true:** the whole event is tagged as ”accepted” and the module gives +1 as a return value
  - **No cut gives a true result:** the whole event is tagged as ”don’t know” and the module gives 0 as a return value
- `acceptOverridesReject` is set to True
  - **One of the chosen accept cuts is true:** the whole event is tagged with ”accepted” and the module gives +1 as a return value
  - **One of the chosen reject cuts is true and no accept cut is true:** the whole event is tagged as ”rejected” and the module gives -1 as a return value
  - **No cut gives a true result:** the whole event is tagged as ”don’t know” and the module gives 0 as a return value

A cut gives a true result if its cut condition is true and, in cases of accept cuts, the pre scale (drawn from a uniform distribution) also leads to a true result.

From where to get the result

There are two possibilities to get the result of a Software Trigger Module event:

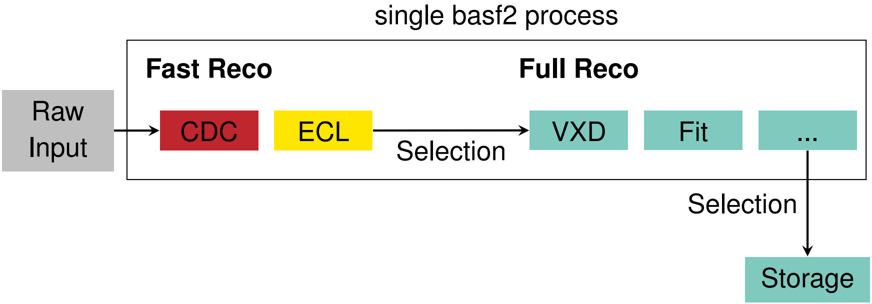
1. The return value of the module is set to -1, 0 or +1, depending on the results of the cuts.
2. The module writes the results of the individual cuts as well as the total result to a `StoreObj` with the type `SoftwareTriggerResult`.
   - If you have more than one Software Trigger Module in your path, the different cut results are all added to the result object.
   - You can get the results of the single cuts with the `getResult` function.

The `SoftwareTriggerResult` replaces the `HLTTag` as an MDST object, as it does not contain hard-coded cut tags.

Where to find more information

- All the code related to the Software Trigger can be found in `hlt/softwaretrigger`.
- An example file of the usage is currently under construction in `hlt/softwaretrigger/scripts/softwaretrigger/__init__.py`.
- The shown information plus some more can be found at confluence `https://confluence.desy.de/display/BI/The+Software+Trigger+Module`.
- In case of questions, please write a mail to `thomas.hauth@kit.edu`.

Software Trigger Processing Chain



FastReco

Idea: Run the the ECL reconstruction and the Legendre-based CDC track finding first

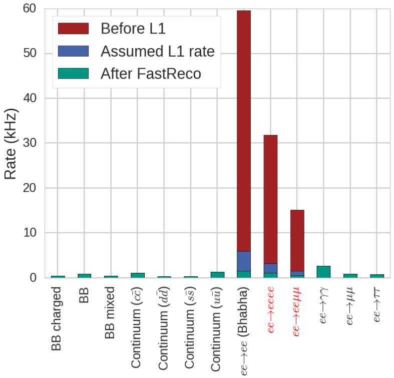
- Only around 10% of the runtime of the full reconstruction chain
- Produces ECL clusters and tracks, which can be used to reject the most copious background sources, esp. Bhabha radiation

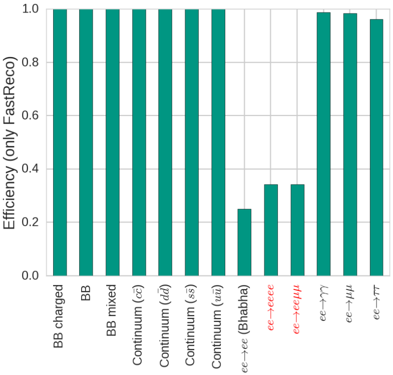
The following variables are used for cuts after the FastReco

- energy sum of high energetic ECL ($> 0.05 GeV$)
- highest 2 ECL cluster energies summend, highest 3 ECL cluster energy summed
- max p_t in event
- mean(abs(z))
- mean(θ)

Please note: All the shown results are preliminary and assume a L1 rate of 20 kHz. This study will be redone once the full L1 simulation is available.

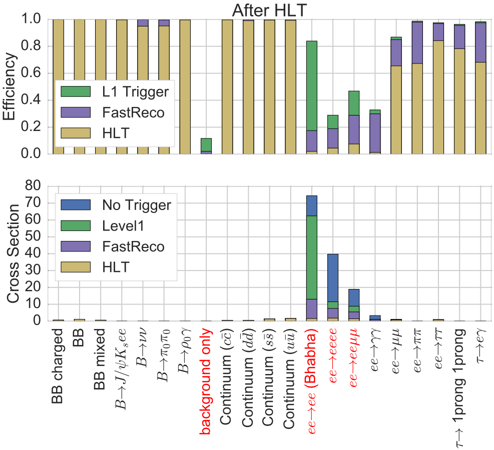
FastReco Efficiency





A selection based on FastReco can reduce the rate from $20\,\mathrm{kHz}$ up to $\approx 12\,\mathrm{kHz}$ - without affecting the signal channels.

Efficiency and Rates after HLT



$$\mathrm{R} = \mathcal{L}\sigma = \mathcal{L} \cdot \left(\sum_{i \in \mathrm{Channels}} \varepsilon_i \sigma_i \right) \approx 8.84\,\mathrm{kHz}$$

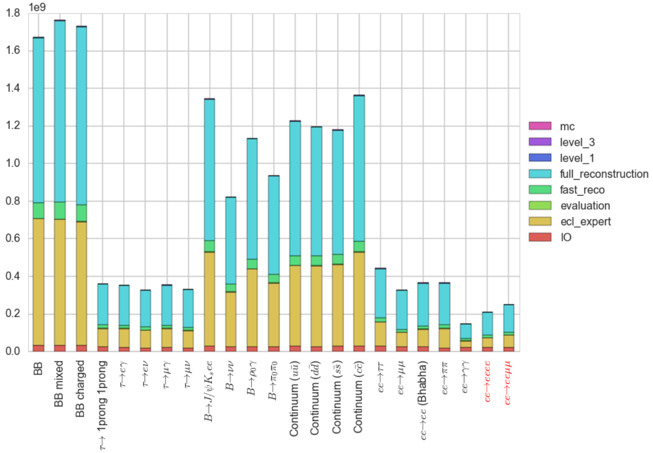
- The shown efficiency is after Level1, FastReco and HLT.
- The HLT and FastReco can reduce the background channels to a very low rate, reaching the requested 10 kHz.
- Note: This study needs to be redone with full TSim and updated HLT trigger menu

Runtime Analysis

Experimental Setup

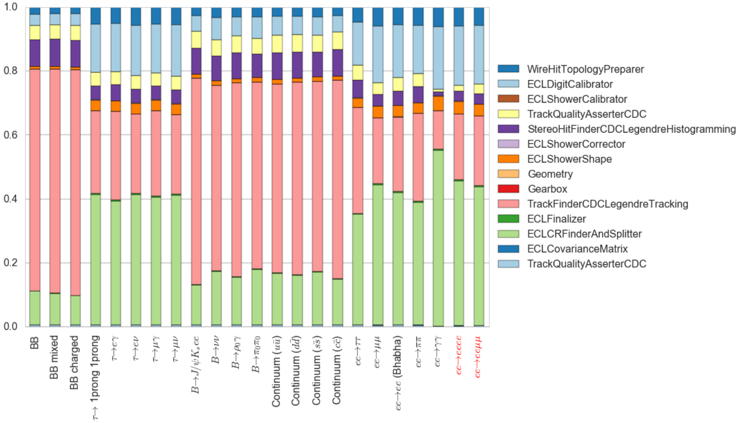
- Measurements on the Runtime of the full reconstruction was performed on the HLT worker nodes.
- All events were processed - cuts on the events were performed in retrospect.

Average Event Processing Time (in ns)



- Processing time depends heavily on the channel.
- ECLExpert will not be used in the following HLT path as it is slow and still under development.

Average Ratio of different *FastReco* Modules



As expected: Tracking (light red) takes up significant more time for topologies with more than one track.

Scaling on Parallel Hardware

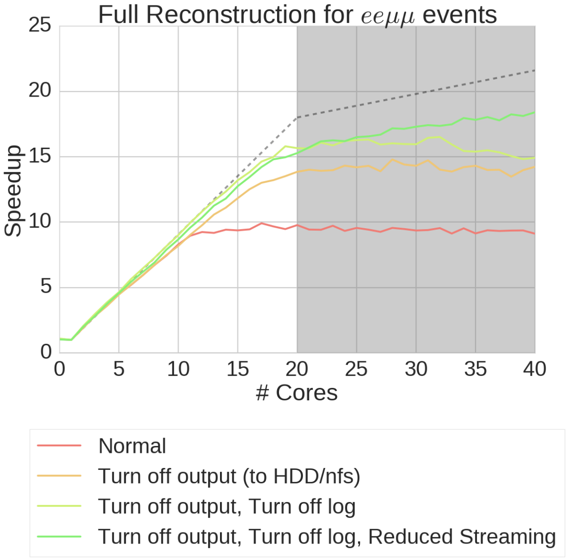
Introduction

- A good multiprocessing is a crucial feature for the HLT setup, because only when using the capacity of all 20 cores (x2 with Hyperthreading), the events can be reconstructed fast enough
- The most challenging problem are short-running events, where the streaming and work-distribution is a quite large percentage of the whole execution time.
- As an example, we look into $ee\mu\mu$-events only doing a part or the full reconstruction (which is the setup for HLT, approx. 70 % of the HLT events will be ”small” events).
- One important measure for multiprocessing is the speedup

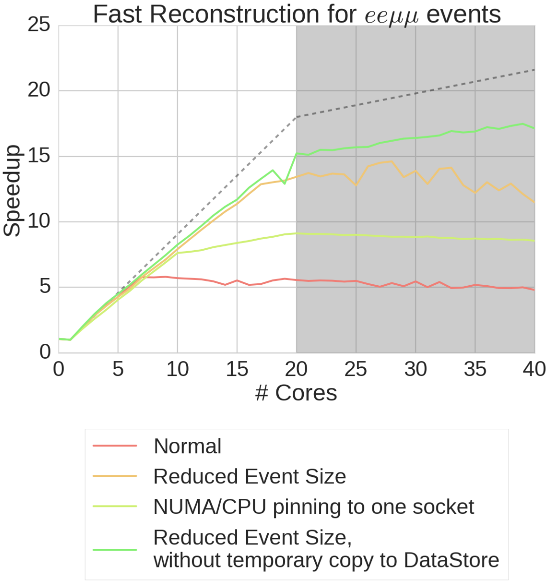
  $$S(n) = \frac{T(n) - T_{\text{init}}(n)}{T(1) - T_{\text{init}}(1)}$$

  where n is the number of started processes.
- Only raw objects are used as input: smaller object size and less complex streaming procedure

Scaling FullReco for small events



Scaling FastReco for small events



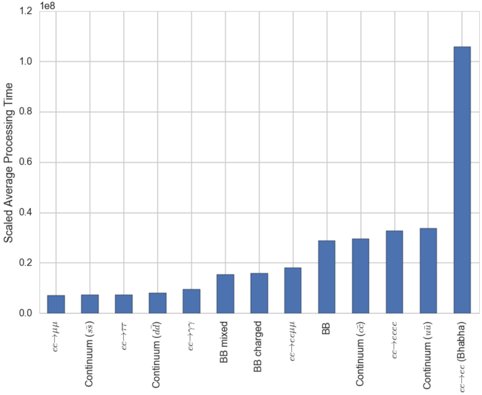
Summary on Scaling Studies

The following items are important for keeping a good multiprocessing performance:

- Reduce log output
- Make sure the input and output process can keep up with the data rate:
  - DataStore-Streaming is expensive
  - HDD disks can not cope with our data rate
  - With many worker processes, the input and output process take away one core each
- NUMA seems not to be a bottleneck (at least not on the HLT nodes)

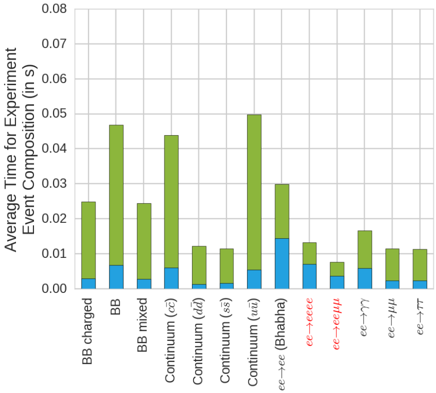
- Understood (and remedied) the major limitations of parallel processing on HLT
- Non-optimal scaling can result from second-order effects (shared CPU caches, memory bandwidth)
- Positive Outcome: Good performance also for small events and sufficient for B2 HLT online processing

Processing Time Scaled with Cross Section



- With these ration, the average processing time (without cuts) for one average event is: **0.30 s**
- Assuming 6400 cores (without degregation because of hypertreading, IO, etc.) with 20 kHz (30 kHz), the limit is **0.32 s** (0.21 s).

Average Processing Time with *FastReco*



FastReco time is blue, FullReco Time is green

Especially the Bhabha processing time greatly reduced, the average processing time is **0.198 s** (was 0.30 s) → Fits well within the limits of the HLT farm

Preparations for Phase 2

Preparation of Online Reconstruction

Fast Reconstruction

- As CDC and ECL algorithms are reused from the offline code, no special adaptations are required
- The CDC track finding code uses MVA methods for background rejection which should be retrained with first measured background events

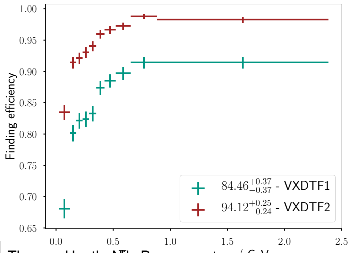
Full Reconstruction

- Alignment and calibration constants are loaded from the database
- Ensure the correct global tag is used and each sub-detector reconstruction will load the correct content
- Prepare a fixed software version (monthly build or specific release) used throughout data taking in phase 2 to ensure reproducibility of the trigger decision

Both reconstruction stages need to be tested and validated on the Phase II geometry.

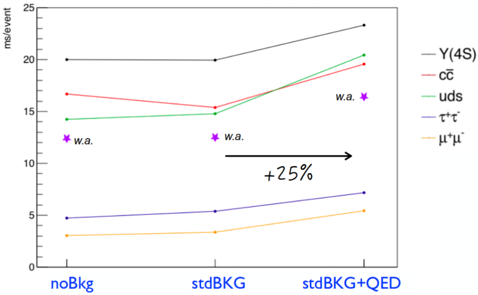
VXDTF 2 Integration

- The rewritten track finder for the VXD (named VXDTF 2) is now available in the release
- Online ROI for PXD relies on the SVD tracks found via the VXDTF 2 on the HLT machines
- It has a superior performance and will replace the old VXDTF 1 in the near future
- VXDTF 2 needs to be integrated into the online reconstruction chain
    - First tests show runtime btw. VXDTF 1 and 2 close, but might be different for the relevant channels in the online use-case
    - Memory consumption of VXDTF 2 challenging (being optimized)



Region-of-Interest Finder

- Important software component to decide (based on SVD tracks) which part of the PXD sensor is read out
- Implemented by the `PXDDataReductionModule` and extensively tested at DESY testbeams in the past
- Runtime of this module was optimized by Giulia Casarosa and is now runtime below 25ms



https://kds.kek.jp/indico/event/24276/session/2/contribution/56/material/slides/0.pdf

Summary

- SofwareTrigger modules have been developed and can be used in a flexible manner for the FastReco and HLT cuts
- Studies on efficiencies on a range of signal and background channels have been performed and show a good performance
  - This studies need to be repeated with the actual input form the L1 trigger stage
- FastReco method is a way to reduce the processing time of HLT by $\frac{1}{3}$ without losing any efficiency in the signal channels
- The basf2 software can scale up to 20 cores
- Extrapolations to the full size of the HLT farm show that the current processing time fits well within the budget of the size of the final farm

Backup

Choosing the correct database

It is very important to use the correct database tags for downloading the cuts. We will first use the local database, then use the central database with the software trigger cuts and in the end use the production database for the other entries.

```
import basf2

basf2.reset_database()
basf2.use_database_chain()
basf2.use_local_database()
basf2.use_central_database("software_trigger_test")
basf2.use_central_database("production")
```

Using the module

Make sure to include the database code snippet also in your steering file!

```python
import basf2
path = basf2.create_path()
# Import the simulated events (or something analogous)
path.add_module("RootInput")
# Reconstruct the information
# needed for the fast reco decision
add_reconstruction(path, trigger_mode="fast_reco")
# Add the cut module to the path
cut_module = path.add_module("SoftwareTrigger",
  baseIdentifier="fast_reco",
  cutIdentifiers=["reject_ee", "accept_ee", "reject_bkg"])
# Create three new paths (filling is not shown)
events_accepted_path = basf2.create_path()
events_rejected_path = basf2.create_path()
events_dont_know_path = basf2.create_path()
# Do something with the result of the module
cut_module.if_value("==-1", events_rejected_path)
cut_module.if_value("==1", events_accepted_path)
path.add_path(events_dont_know_path)
```

Debug Output

The Software Trigger Module can output all calculated variables for each event into a ROOT TNTuple file

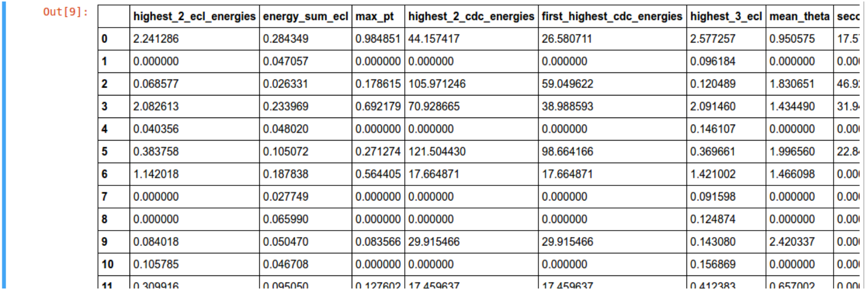
```
import basf2
path = basf2.create_path()

# Add the SoftwareTrigger
path.add_module("SoftwareTrigger",
                baseIdentifier="fast_reco",
                cut_identifiers=[],
                storeDebugOutput=True,
                debugOutputFileName="variables.root")
```

It is important to choose the correct base indentifier, because this defines which variables are calculated and stored!

Debug Output - continue

After processing the path, a new ROOT file ”variables.root” are created with a single TTree containing the calculated variables with as many rows as there were events in the process.

```
from root_pandas import read_root
df = read_root("variables.root")
```



Out[9]:

	highest_2_ecl_energies	energy_sum_ecl	max_pt	highest_2_cdc_energies	first_highest_cdc_energies	highest_3_ecl	mean_theta	secc
0	2.241286	0.284349	0.984851	44.157417	26.580711	2.577257	0.950575	17.5
1	0.000000	0.047057	0.000000	0.000000	0.000000	0.096184	0.000000	0.00
2	0.068577	0.026331	0.178615	105.971246	59.049622	0.120489	1.830651	46.9
3	2.082613	0.233969	0.692179	70.928665	38.988593	2.091460	1.434490	31.9
4	0.040356	0.048020	0.000000	0.000000	0.000000	0.146107	0.000000	0.00
5	0.383758	0.105072	0.271274	121.504430	98.664166	0.369661	1.996560	22.8
6	1.142018	0.187838	0.564405	17.664871	17.664871	1.421002	1.466098	0.00
7	0.000000	0.027749	0.000000	0.000000	0.000000	0.091598	0.000000	0.00
8	0.000000	0.065990	0.000000	0.000000	0.000000	0.124874	0.000000	0.00
9	0.084018	0.050470	0.083566	29.915466	29.915466	0.143080	2.420337	0.00
10	0.105785	0.046708	0.000000	0.000000	0.000000	0.156869	0.000000	0.00
11	0.309916	0.095050	0.127602	17.459637	17.459637	0.412383	0.657002	0.00

Cut Creation and Uploading

Create a reject cut (to reject background events after *FastReco*) and upload it to the local database:

```
from ROOT import Belle2
from Belle2.SoftwareTrigger import SoftwareTriggerCut
from softwaretrigger import db_access

bkg_cut_st = SoftwareTriggerCut.compile(
  "[[visible_energy < 1.8438] and
    [highest_3_ecl <= 0.3999] and [max_pt <= 0.3152]]",
  1, True)
db_access.upload_cut_to_db(bkg_cut_st,
                          "fast_reco", "reject_bkg")
```

After that, the cut is stored in the same folder as you called the python code from in a folder called ”localdb”. If you run your steering file accessing this cut in this folder, you can use this new cut.

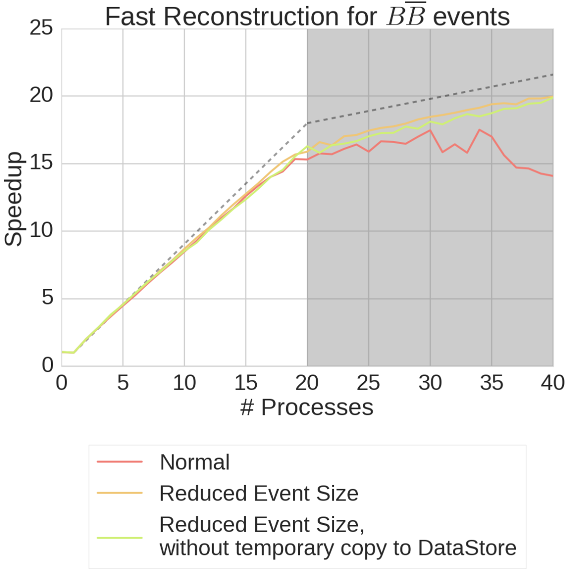
Cut Creation and Uploading - continue

If you are happy with the cut, you can go and upload it to the central database. In the moment, I do not push to the production cut but rather have created my own global tag (software_trigger_test). If you also want to push to this tag, call

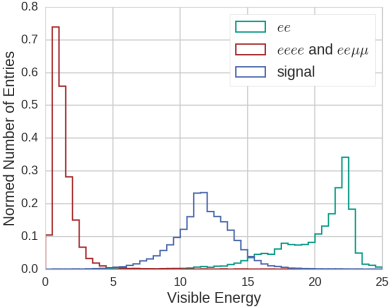
```
upload_localdatabase --tag software_trigger_test  \
                     localdb/database.txt \
                     --final-exp 0
```

Of course, you are free to create your own global tag and push to this (or push to another already present global tag). You just have to remember to change to this cut also in your other steering files.

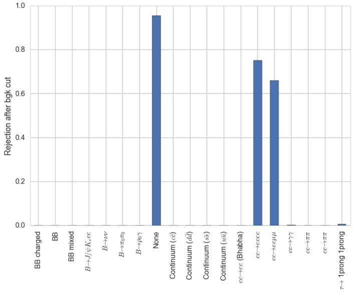
Scaling FastReco of $B\overline{B}$

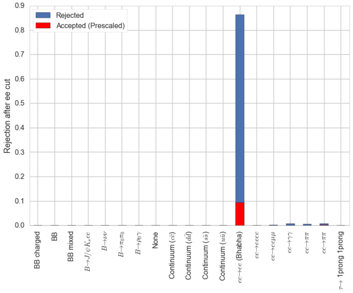


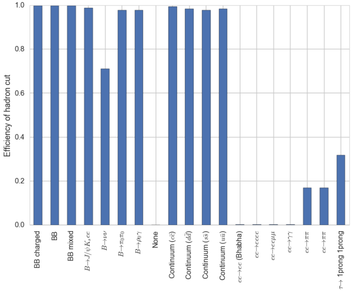
FastReco example variable: Visible Energy



Cuts applied after *FastReco*







The shown cuts are:

- Background Cut
- EE Cut
- Hadron Cut