hps-java tracking sw update

PF

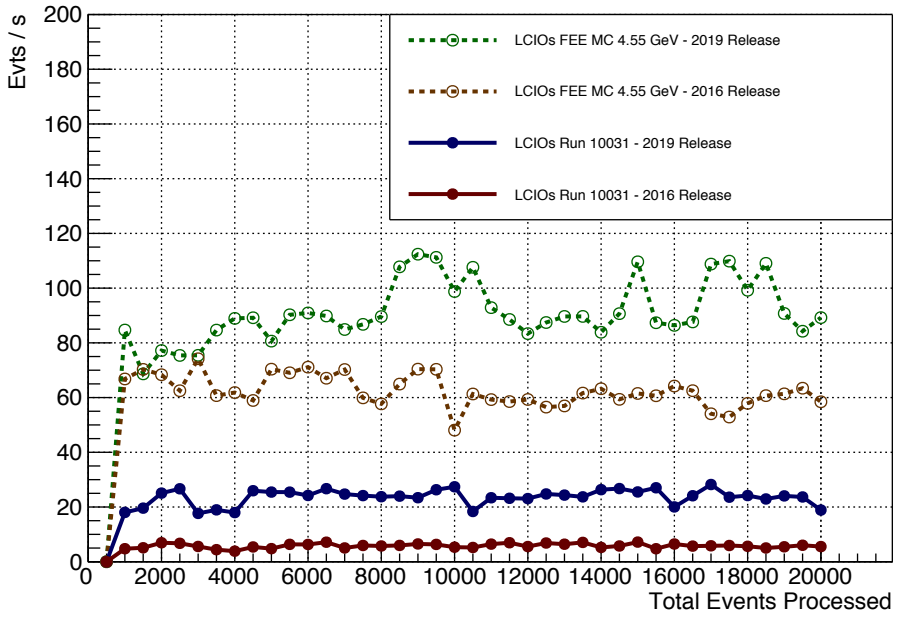
11/19/2020

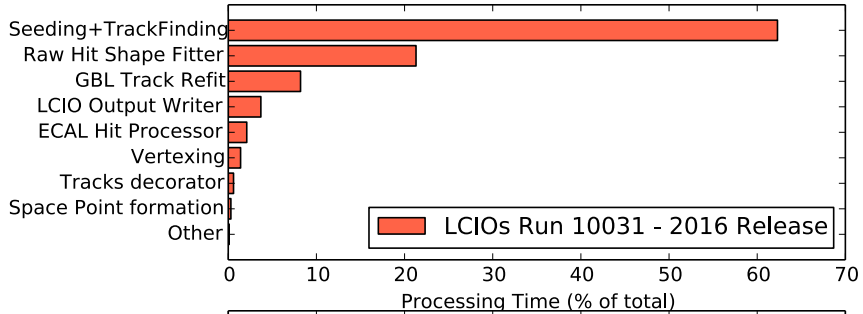
Introduction

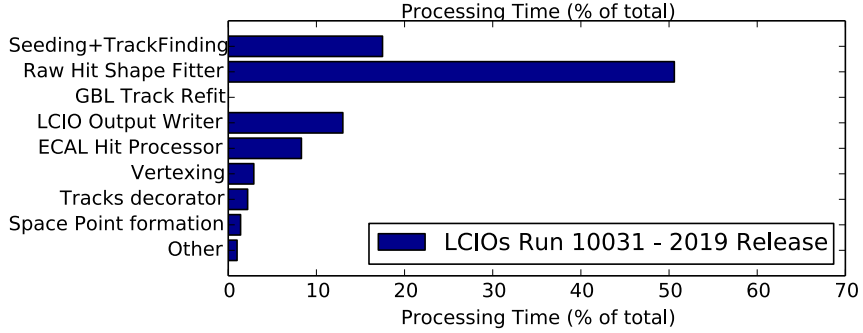
- Lot of work went on tracking side for support of new features
- However:
  - Not all the work was done in separate commits/issues (bad practice and I'm responsible for that)
  - All the changes are in a main branch called pass1-dev_fix which should include all stable changes for 2019 processing.
- Today I'll start discussing some of the changes made and their impact
  - ShapeFitter
  - KalmanFilter
  - Jna support for GBL

Some timing checks

- Comparison in processing time between the master and pass1-dev_fix branch
  - Only Kalman Filter was active in the pass1-dev_fix branch
  - Optimistic scenario where we run only the KF for track finding
  - Check on FEEs and V0 skims
  - From LCIO but re-running the ShapeFits
- In current branch main bottleneck is tracking. Hit shape fitting is the second most expensive processor

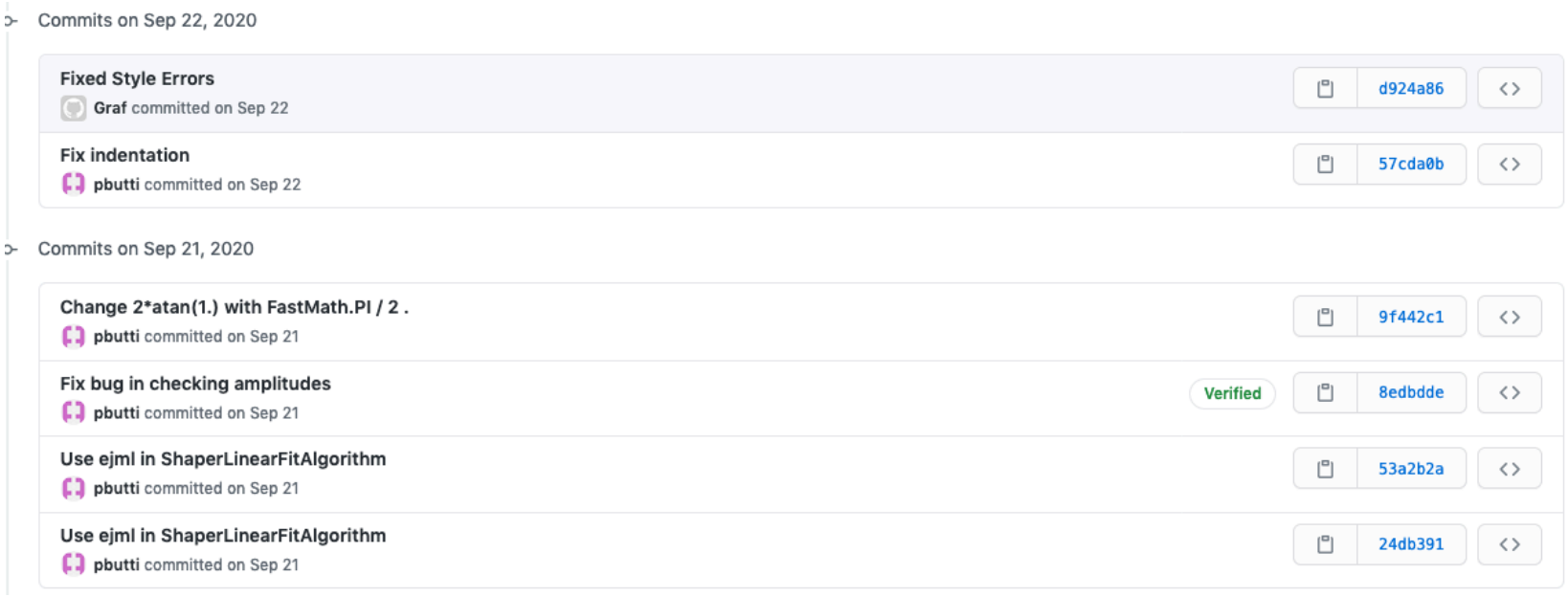






Changes to the ShapeFitter

- Originally in iss730-> merged into pass1-dev_fix
- Only few relevant commit.
- Imported the full jMinuit package into hps-java. Will be located in tracking/src/main/java/org/hps/minuit/
  Looks like lot of extra code, but aim is to fix couple of timing bottlenecks found in computing simple hardcoded quantities (like atan(2)). In the future if we pass to MIGRAD instead of Simplex we have full control of the jMinuit package for speed up modifications if we need to.



Effects of the ShapeFitter change

SLAC

- I've tested the effect of these modifications in terms of timing
  - Shape fitter code improved in speed from 8Hz to 30Hz.
  - Using Migrad push down the processing time back to 6.50 Hz
  - Going back to old processing time, but using better shape fitter can be justified if tracking/physics improvement is substantial => still need to be done
- Checked speed up when using ejml version in the shape fitter for solving the linear algebra on V0 skims evio files. **After improvements still bottleneck**

So This is the original code

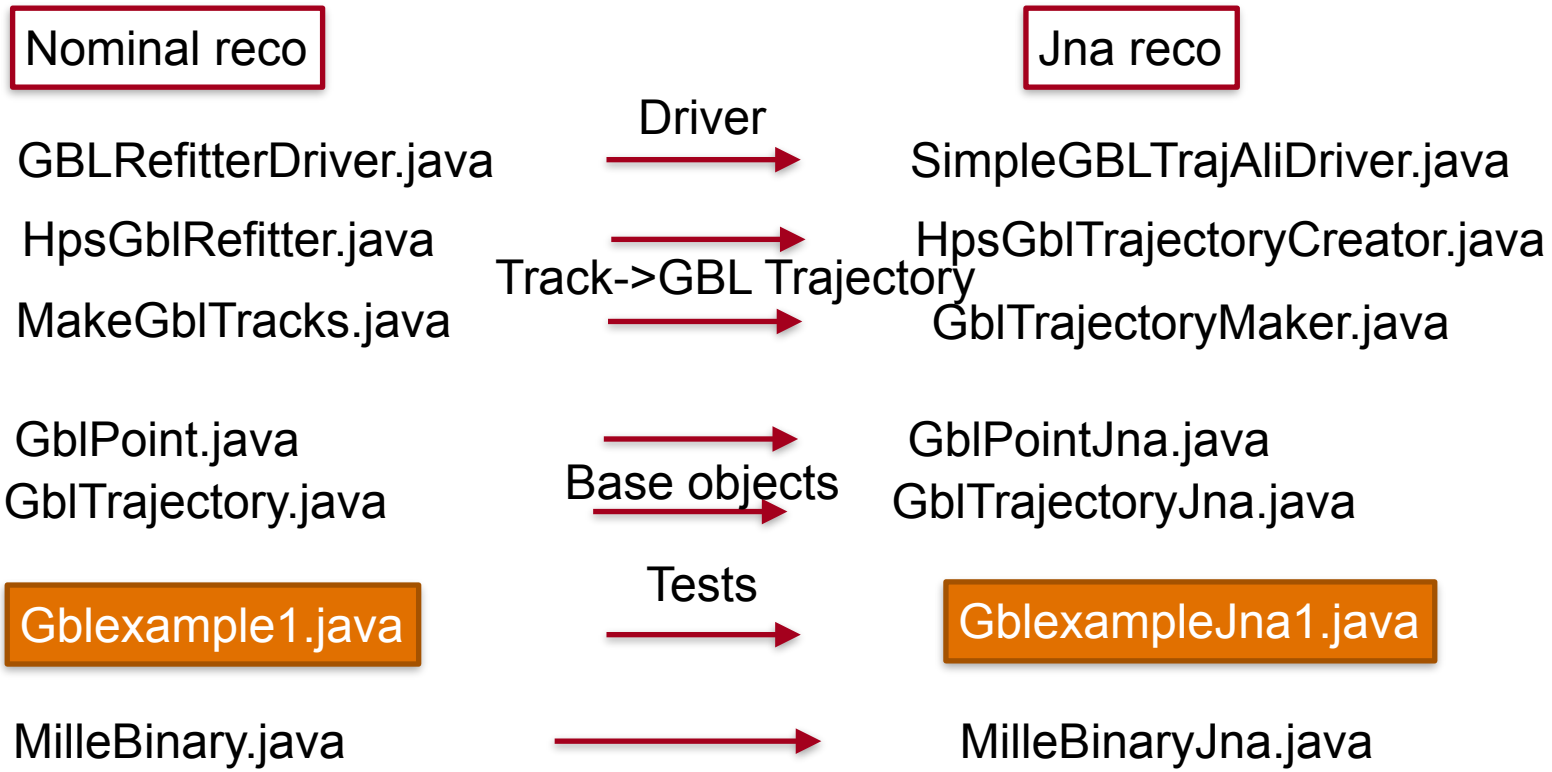
```
Event:   8574118, Run: 10031, Sequence:     500, 0.00 ms/event, 0.00 Hz, Avg: 0.00 Hz
Event:   8574618, Run: 10031, Sequence:    1000, 119.54 ms/event, 8.37 Hz, Avg: 7.85 Hz
Event:   8575118, Run: 10031, Sequence:    1500, 131.52 ms/event, 7.60 Hz, Avg: 7.77 Hz
```

This is my version with ejml

```
Event:   8574118, Run: 10031, Sequence:     500, 0.00 ms/event, 0.00 Hz, Avg: 0.00 Hz
Event:   8574618, Run: 10031, Sequence:    1000, 33.57 ms/event, 29.79 Hz, Avg: 25.98 Hz
Event:   8575118, Run: 10031, Sequence:    1500, 37.24 ms/event, 26.85 Hz, Avg: 26.27 Hz
```

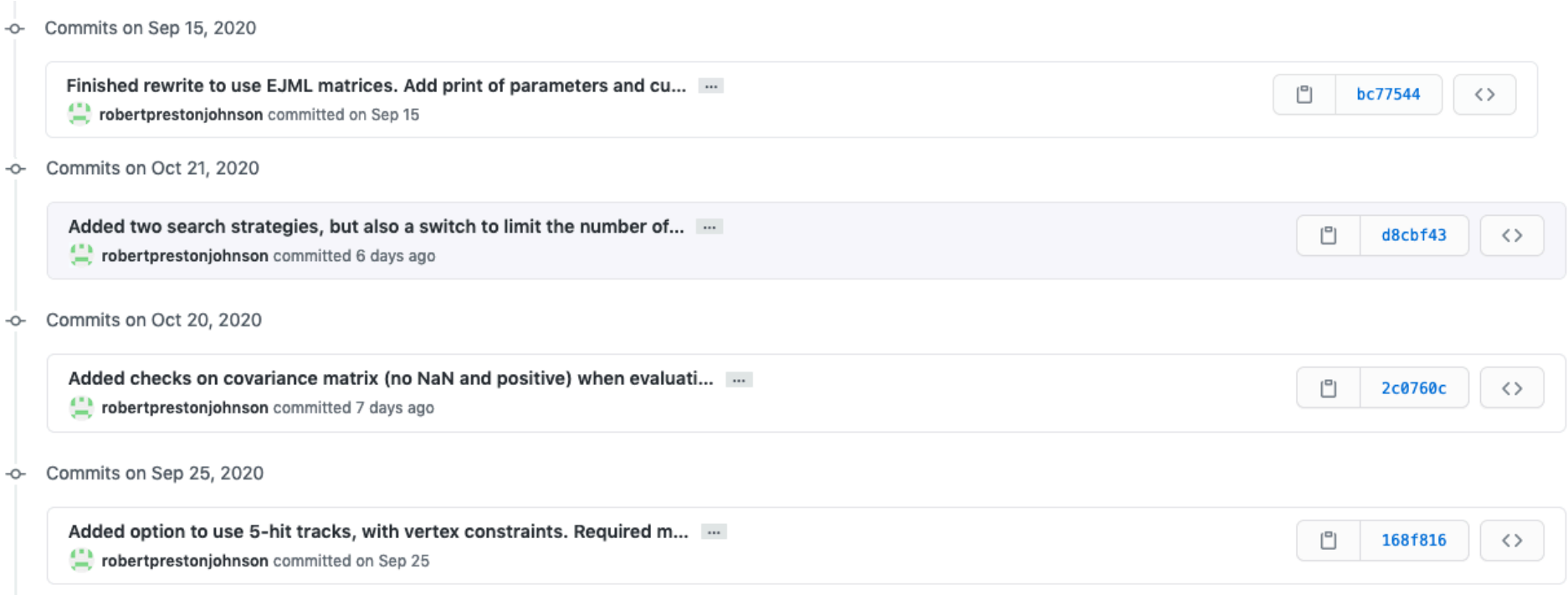
JNA Support for C++ GBL

- The GBL library can be loaded via JNA
- The GBL repository with the correct wrappers is stored in my personal github area - not optimal. Should be centrally deployed and could go into hps-java or external in jefferson lab. Planning to push it to the terascale repository when I finish up the examples and unit-tests.
- The library is only loaded at run-time if one needs to run alignment: no effect on the nominal processing.
- Separate classes and drivers ensure that nominal processing is not touched
- No refit is done in the JNA implementation (but can be done and has been tested and validated). Only the trajectory and the collection of the derivatives into the Millebinary is done



Kalman Tracking

- The Kalman Filter package is organised in a separate package with respect to nominal tracking
- Constant updates by Robert and recent update to ejml linear algebra package.
- The package is separated from nominal reconstruction pipeline and can be brought in without breaking nominal pattern recognition
- Latest commits are quite mature for a MC and Data validation.

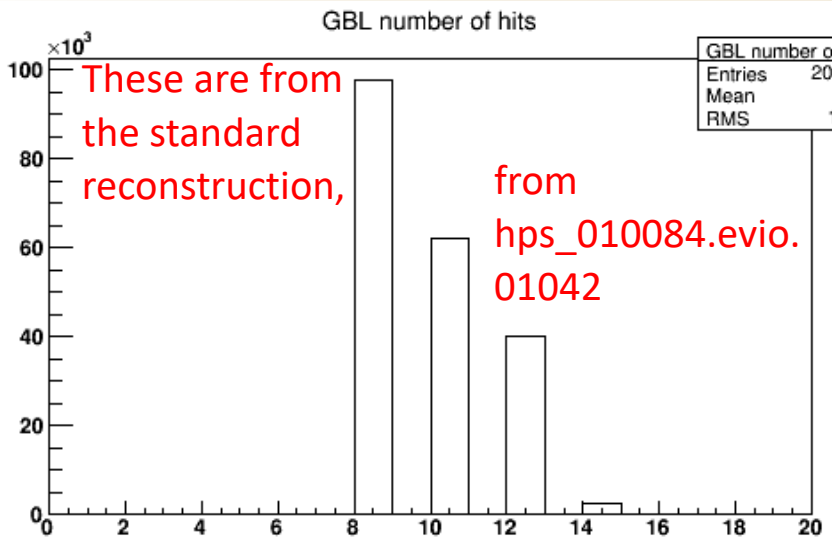


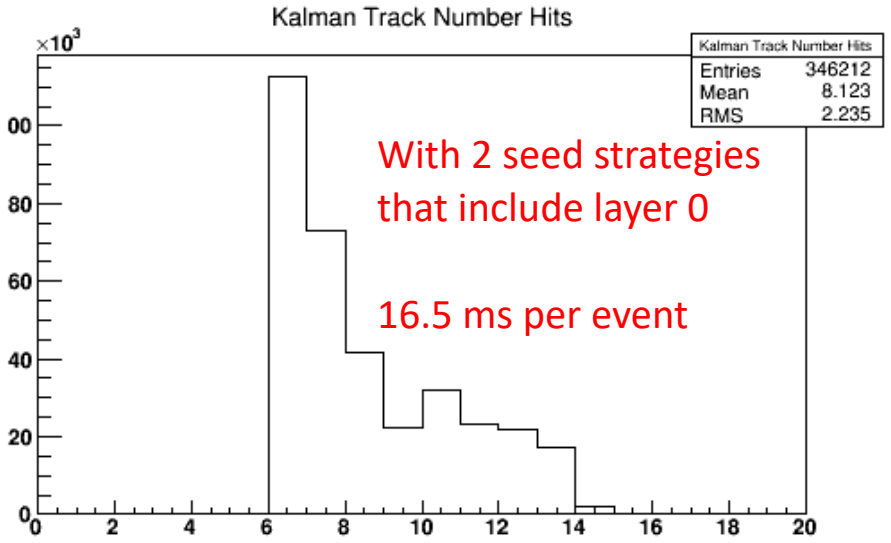
Kalman Pattern Recognition Status - ROBERT

- A lot of recoding was done to speed up some matrix operations, gaining nearly a factor of 2 in speed:
  - use the EJML (Efficient Java Matrix Library) package for matrix algebra
  - Using only the EJML low-level procedural interface. The former object-oriented style wasted too much time creating new memory space and copying numbers at each calculation step.
- More work was done looking for and correcting some pattern recognition mistakes.

- The most recent code is in branch pass1-dev_fix in Github.

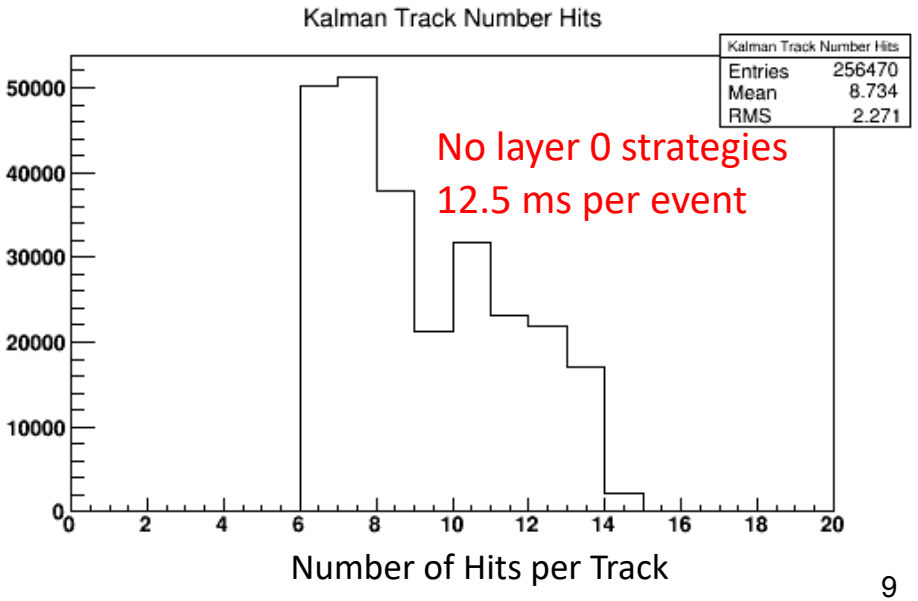
2019 Data Track Finding - ROBERT



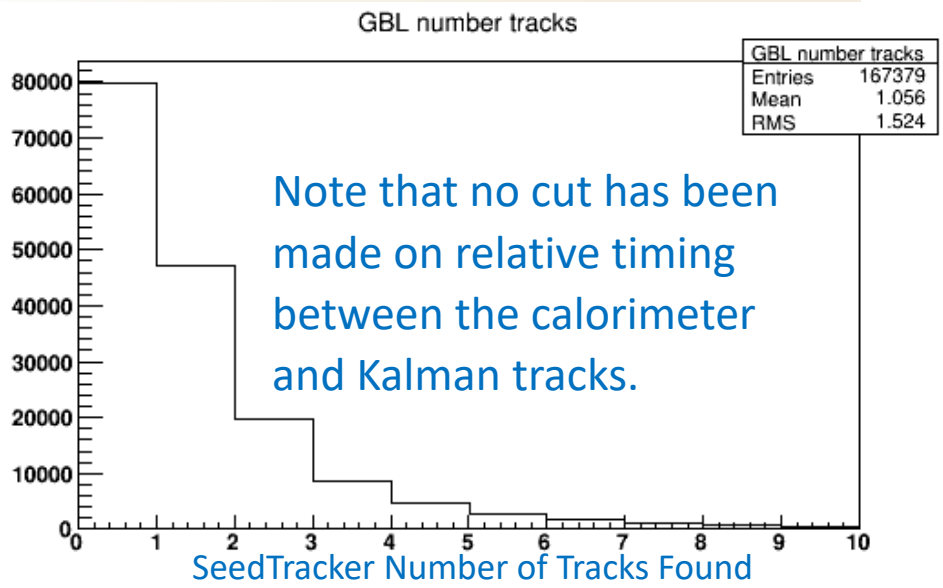


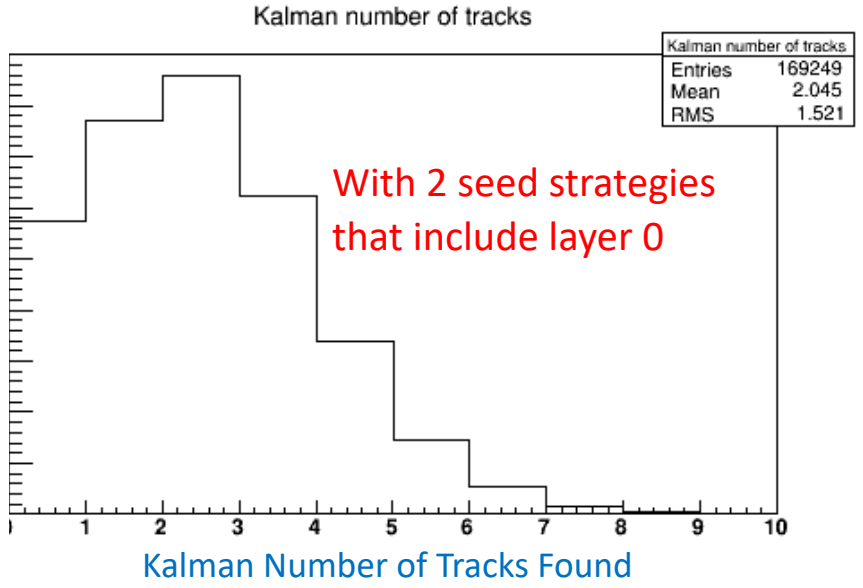
The Kalman-based pattern recognition can find a lot of short tracks, but we pay some CPU price when including the first two noisy layers in the combinatoric finding and testing of 5-hit seeds.

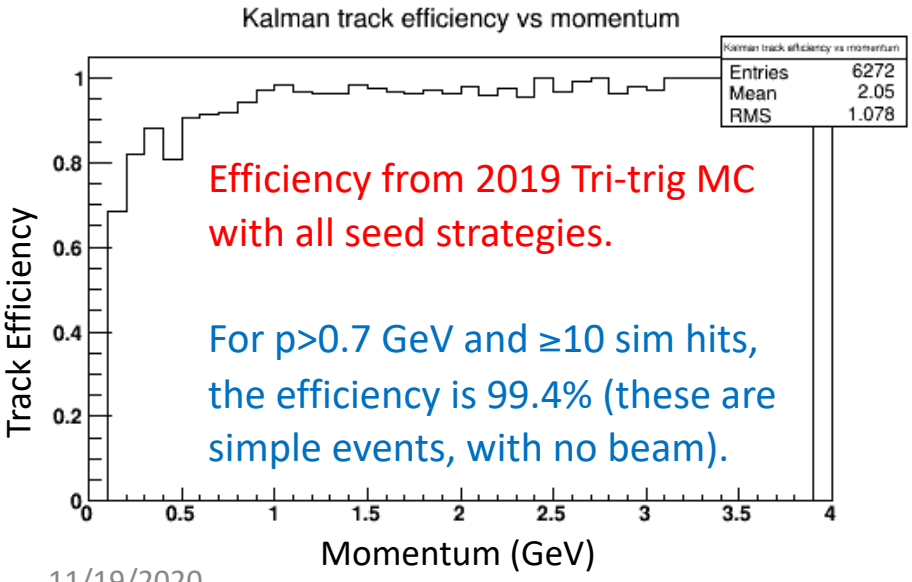
These are CPU times for the pattern recognition and fitting on a Core-I7 notebook computer.

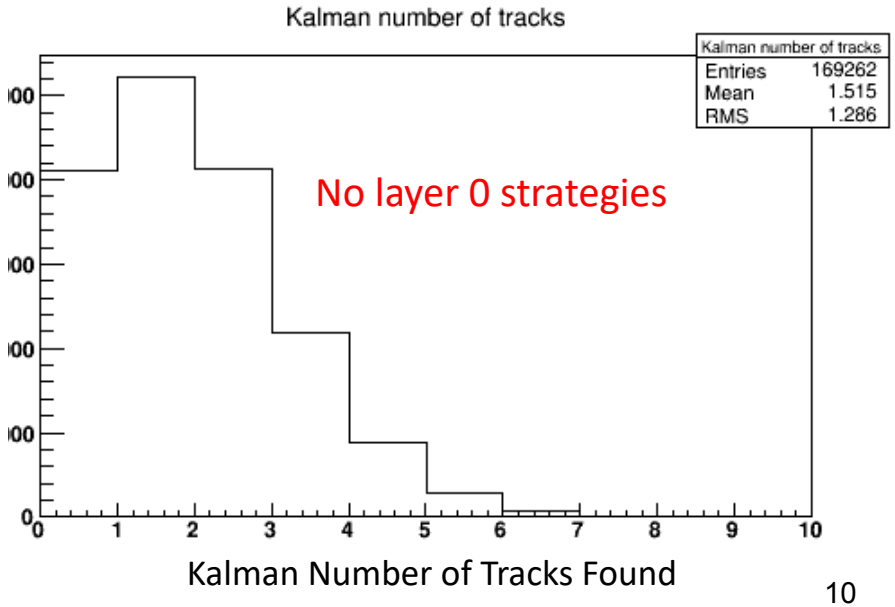


2019 Data Track Finding - ROBERT



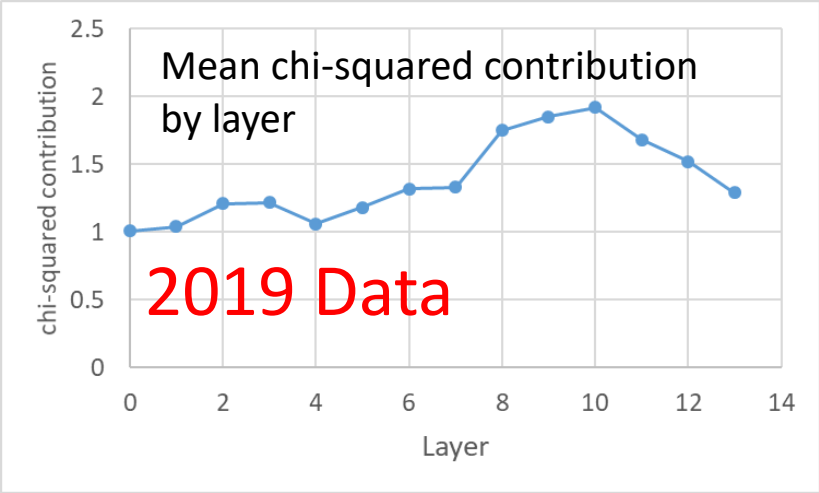


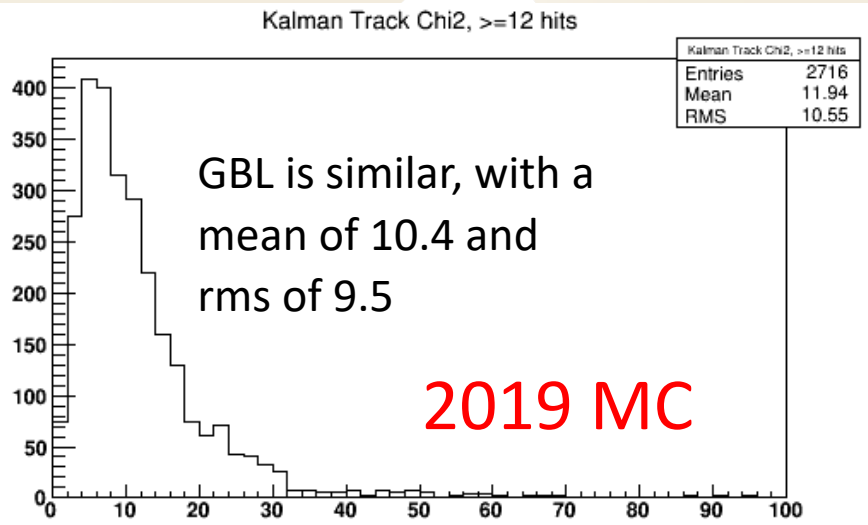


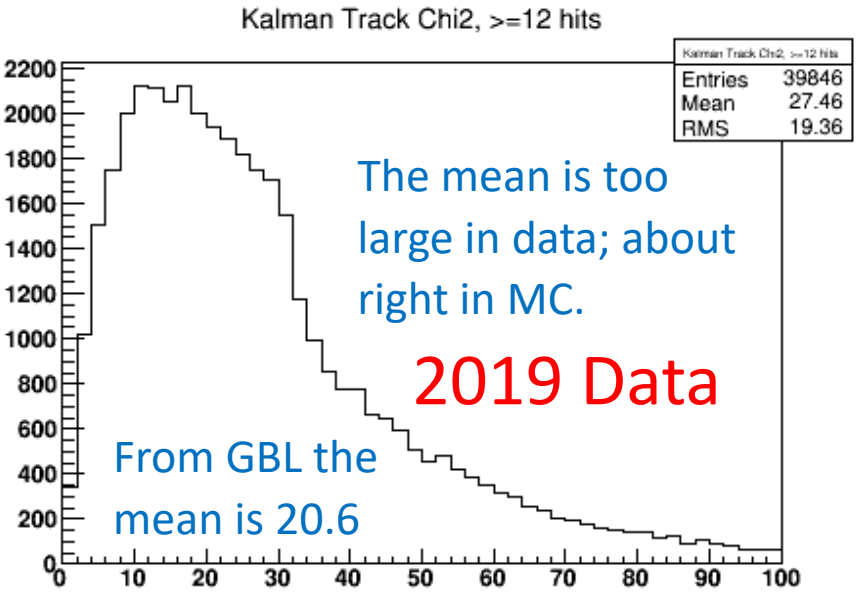


2019 Kalman Data / MC Performance - ROBERT

	Kalman	GBL
Wrong hits per track	0.055	0.22
Number of tracks per event	1.83	1.61
Number of hits per track	10.9	10.7
Execution time per event	2.1 ms	?
ρ0 pull rms	1.12	
φ0 pull rms	1.36	
1/pt pull rms	1.09	
tanλ pull rms	1.07	



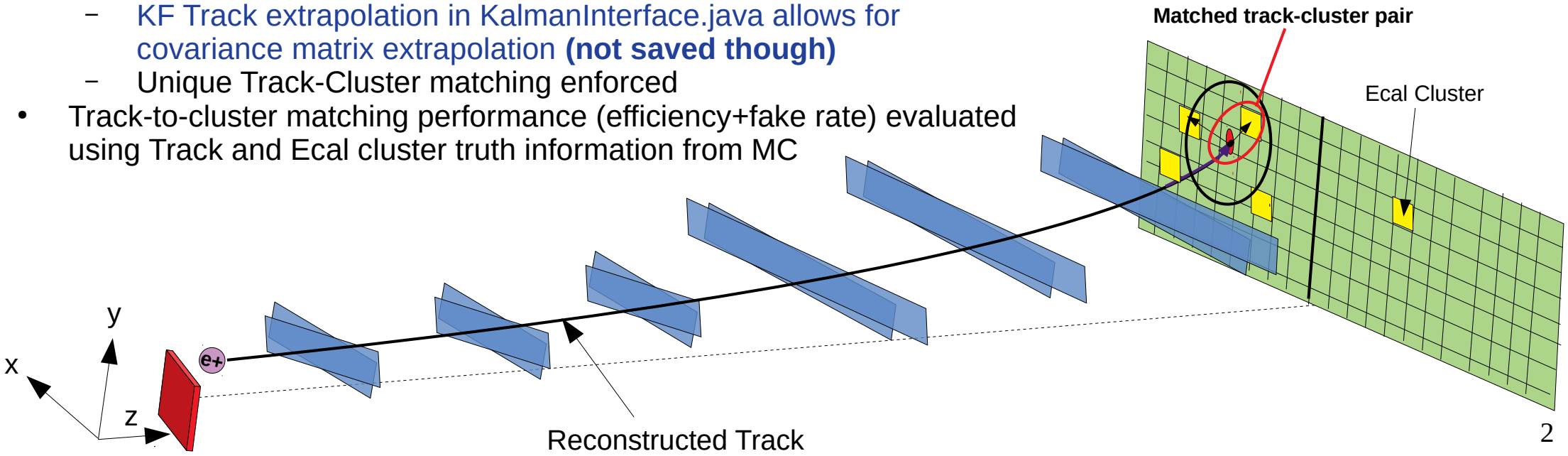




Track-to-Cluster Matching

- Hps-java track reconstruction missing track-to-cluster matching for Kalman Tracks
- **Currently updating track-to-cluster matching in hps-java to work for both GBL and KF Tracks**
- Using timing coincidence and simple **"closest-distance matching algorithm"** to match RK4 extrapolated Tracks to Ecal face with Ecal Clusters, based on minimum position residual dr
  - KF Track extrapolation in KalmanInterface.java allows for covariance matrix extrapolation **(not saved though)**
  - Unique Track-Cluster matching enforced
- Track-to-cluster matching performance (efficiency+fake rate) evaluated using Track and Ecal cluster truth information from MC

by ALIC



Track-to-Cluster Matching

- Truth matched track-cluster dx residuals for GBL (top right) show charge dependent bias
- KF (bottom right) show dx mean closer to 0, potentially indicating better KF track extrapolation to Ecal
- **GBL:** Matching algorithm matches an ele track to cluster 80% of the time, 98% for positrons
  - 17% of ele matches / 9% of pos matches are incorrect based on truth information
- **KF:** Matching algorithm matches an ele track to cluster ~65% of the time, 100% for positrons
  - ~17% of ele matches / 9% of pos matches are incorrect based on truth information
- KF track-to-cluster matching efficiency seems strangely low compared to GBL tracks
  - Potential bug in truth validation code...
  - **Currently validating truth-matching + cleaning code**
  - Also **investigating KF vs GBL efficiency discrepancy**