

# High(er)-coverage 8L bank test

- It looks like 8L architecture is hitting two limits:
  - To get to 90% efficiency, we need billions of 12/768 patterns
  - But even with this SS size, we have too many roads
- I did the following 8L test:
  - Large-statistics 8L banks with ss = 12/768; generated 240M patts/reg
  - Single muon efficiency at 120M/reg: 54% ( $|\eta| < 2.5$ ), 69% ( $|\eta| < 1.0$ )
  - # roads out of 8L AM (per region): ~17,000
  - That's several orders of magnitude away from where we want to be!
    - Even with 32-region arch, we only get a factor 2-4 reduction
    - But to get to 90% efficiency, # roads would completely blow up.
- If 7L shows clear advantage, perhaps we should concentrate on it
  - Could be used as the first stage in default split architecture
  - OR: could be used in new Paola's scheme (no AM at second stage) <sup>1</sup>

# FTKSIM in 32 regions

- From the same email thread from last week: # of hits and # of SS is too large, especially for future inner pix layer
- Paola wants to simulate the 32-region setup before TDR
- This is a lot of work:
  - Need to re-define sectors in terms of groups of superstrips, rather than entire modules (?)
  - Need to create region boundaries away from module boundaries
  - But these are deeply engraved in FTKSIM
- Moreover:
  - Separate TrigFTKLib and TrigFTKSim codes require the work to be done two times
- Is this a priority?

# Summary of patt-from-const study

- Many spurious sectors on each event – slice binning problem?
- **Conclusion: no suggested improvements\*:**
  - Although the existing bin size is large, it still produces good patterns, plus many spurious patterns. But when we cut from the bottom of patt list, we get rid of the bad ones. The end effect is similar to using a smaller bin size
  - But smaller bin size (while giving fewer fakes) suffers from smaller efficiency: we don't have enough training to fully populate narrow-bin sector lookup.
- Reason for many spurious patterns:
  - Consider  $p[5]$  that fall into five parameter bins:  $B[5]$
  - We independently find {sectors} in  $B[0]$ , {sectors} in  $B[1]$ ... Then take logical AND.
  - Consider sector X that was produced by many training evts
    - Event 1 (in sector X):  $\text{bin}[0,1,2,3]=B[0,1,2,3]$ ; but  $\text{bin}[4]\neq B[4]$
    - Event 2 (in sector X):  $\text{bin}[0,1,2,4]=B[0,1,2,4]$ ; but  $\text{bin}[3]\neq B[3]$
  - No training muon in sector X ever went through  $B[5]$ , yet slices still match it to  $B[5]$
- Using direct lookup is impossible: {sectors}[bin0][bin1][bin2][bin3][bin4]
  - Need too much statistics to populate all bins

\* except for 8L case, when using narrow slices allows to run larger production

# Backup slides

- Details on patterns-from-const studies
  - Taking real training muons and passing them through patt-from-const machinery

# Reminder: patt-from-const

- We generate 5 track parameters + constr
- Need sector to get the right constants
- A five-dimensional slice lookup:
  - Bin 5 parameter ranges and train on real trks
    - Default: 100 – 250 bins in each parameter
  - Given 5 pars, look up sectors
- Why are there multiple matches?
  - Overlaps [but there is basic overlap removal]
  - Multiple scattering
  - Coarse binning

# Problem with multiple matches

- Multiple sector matches result in additional patterns in pattern bank, so it's quality is lower than pattgen patterns
  - We explicitly remove overlaps, but can't do anything about other effects
- In 11L / 7L we just make a lot of patt-from-const and then cut from the bottom
- In 8L with small SS, this is problematic since we'd need to produce several billion patterns before we can cut down low-quality patts
  - Not exactly sure why this is particularly bad in 8L case
  - *I suspect this is because SCT-only sectors are less constrained and allow much wider  $p[5]$  parameter spreads within each module. In other words, 11L and 7L sectors are much better constrained in the  $p[5]$  parameter space by the smaller pixel modules.*

# Basic test

- 8L produces too many patterns
- Last page argues that 8L sector lookup produces an abnormally high number of spurious sector matches.
  - Given a wrong, but nearby match, it is possible that the final x[8] are each within {0..768}, so a spurious pattern is saved
- To try to understand it, I fed in real training events (with known true sector)
  - Note: same events were used in slice training, so the sector lookup is 100% efficient by definition.
  - Instead, this allows to quantify sector fakes
- What's the effect on # of matches (i.e. fakes) vs slice binning?
  - Consider two cases:
    - NBins= 100 (default)
    - Nbins = 1000

## 8L slice lookup performances

**NBins = 100; sector lookup efficiency = 95%; sector lookup fakes = ~150 / event**

**True pattgen sector: 41**

**Matched slices sectors:** 29 36 **41** 69 93 95 125 133 155 161 170 176 178 189 190 236 239  
276 286 298 302 345 358 364 367 380 388 410 449 462 482 489 504 505 522\  
548 573 584 610 614 639 651 712 730 736 750 764 810 823 835 850 879 883 897 899 908  
931 934 957 963 1046 1059 1066 1090 1091 1100 1129 1143 1192 1284 1345 \  
1377 1405 1489 1506 1507 1528 1588 1641 1940 1957 2057 2119 2135 2234 2254 2271  
2297 2314 2354 2355 2405 2503 2512 2538 2607 2624 2672 2687 2788 2853 2866 2\  
875 2963 3065 3105 3153 3181 3208 3209 3237 3308 3367 3382 3500 3547 3617 3857  
4010 4036 4139 4249 4262 4273 4296 4364 4914 5090 5122 5906 7104

**Nbins = 1000; sector lookup efficiency = 79%; sector lookup fakes = ~80 / event**

**True pattgen sector: 41**

**Matched slices sectors:** 29 36 **41** 69 93 95 125 133 155 161 170 176 178 189 190 236 239  
276 286 298 302 345 358 364 367 380 388 410 462 482 489 504 505 522 548\  
573 584 610 614 639 651 712 736 750 764 810 823 835 850 879 897 899 908 931 934 957  
1100 1129 1143 1345 1528 2354 2503 2607

Smaller bin size results in fewer “fake” sectors, as expected.

Note that we can't go to extremely small bins because then we will have coverage gaps (not enough training to fill up slices lookup for all the bins!).

8

But let us see how this affects the efficiency. Next page shows the efficiency turn-on table (efficiency in different eta regions -vs- number of loaded patterns)

# Efficiency test: 8L\_50x64x144, 20M patt-from-const seeds

## Nbins=100

Produced: **2,268,142** patterns from constants

#patterns [100000, 500000, 1000000, 1500000, 2000000, 3000000]

$|\eta| < 2.5$ :

#evts [3168, 4376, 4553, 4587, 4600, 4603]

Eff ['0.58', '0.80', '0.83', '0.84', '0.84', '0.84']

$0.0 < |\eta| < 1.0$ :

#evts [1082, 1355, 1378, 1383, 1383, 1385]

Eff ['0.71', '0.89', '0.90', '0.90', '0.90', '0.91']

## Nbins=1000

Produced: **1,423,501** patterns from constants

#patterns [100000, 500000, 1000000, 1500000, 2000000, 3000000]

$|\eta| < 2.5$ :

#evts [3183, 4351, 4557, 4579, 4579, 4579]

Eff ['0.58', '0.79', '0.83', '0.84', '0.84', '0.84']

$0.0 < |\eta| < 1.0$ :

#evts [1127, 1357, 1380, 1383, 1383, 1383]

Eff ['0.74', '0.89', '0.90', '0.90', '0.90', '0.90']

*The turn-on curves are nearly identical!*

*Note that CPU time is roughly the same for both production jobs (used 20M seeds).*

*Although narrow slices produced fewer patterns, they are not of better quality. This is because the wide slices still produced the good patterns, PLUS extra spurious patterns (which aren't completely bad, since they are "approximately" correct). The latter are concentrated in the tail of pattern list.*

# Efficiency test: 8L\_50x32x144, 20M patt-from-const seeds

## Nbins=100

Produced: **5,751,123** patterns from constants

Let's try with smaller  
SS size

#patterns [100000, 500000, 1000000, 1500000, 2000000, 3000000, 5000000, 7500000]

$|\eta| < 2.5$ :

#evts [666, 2070, 2973, 3421, 3676, 3974, 4249, 4422]

Eff ['0.12', '0.38', '0.54', '0.62', '0.67', '0.73', '0.78', '0.81']

$0.0 < |\eta| < 1.0$ :

#evts [205, 773, 1068, 1198, 1262, 1309, 1343, 1365]

Eff ['0.13', '0.51', '0.70', '0.78', '0.83', '0.86', '0.88', '0.89']

## Nbins=1000

Produced: **5,446,564** patterns from constants

The turn-on curves are nearly identical!

#patterns [100000, 500000, 1000000, 1500000, 2000000, 3000000, 5000000, 7500000]

$|\eta| < 2.5$ :

#evts [822, 2315, 3077, 3480, 3707, 4013, 4306, 4337]

Eff ['0.15', '0.42', '0.56', '0.64', '0.68', '0.73', '0.79', '0.79']

$0.0 < |\eta| < 1.0$ :

#evts [335, 927, 1134, 1226, 1275, 1323, 1349, 1351]

Eff ['0.22', '0.61', '0.74', '0.80', '0.83', '0.87', '0.88', '0.88']

10

*Note that if we actually load all the “spurious” patterns in the wide-bin case, we end up with a 2% higher efficiency than the narrow-bin case. Since both bin sizes take the same amount of CPU time, we can as well generate our patterns with the wider bin, so we have the option to go slightly higher on the efficiency curve using the “spurious” end-of-list patterns.*

# Looking in more detail

- I generated 8L slices with Nbins=10000 (x100 smaller than default)
  - Low lookup efficiency, as expected
  - However, I still see a lot of fakes!
- Example of slice lookup on a real training event:

Pars: pt=2909.18 d0=0.26 phi=-0.59 z0=-25.56 ctheta=-0.13

**True pattgen sector: 6**

**Matched slices sectors: 1 4 6 9 11 22 38 42 61 93 108 115 176 304**

*What are these spurious sectors? Recall the code: 1000\*module\_phi+module\_eta*

```
1 28007 28007 35007 35007 42007 42007 49007 49007 0 85433
4 29007 29007 36007 36007 43007 43007 50007 50007 0 48545
6 29005 29005 36005 36005 43005 43005 50005 50005 0 47082
9 29007 29007 36007 36007 44007 44007 51007 51007 0 44358
11 29005 29005 36005 36005 44005 44005 51005 51005 0 43833
22 28008 28008 35008 35008 42008 42008 49008 49008 0 38317
38 29005 29005 37005 37005 44005 44005 52005 52005 0 31664
42 29007 29007 37007 37007 44007 44007 52007 52007 0 31102
61 29007 29007 36007 36007 43007 43007 51007 51007 0 26902
93 28005 28005 35004 35004 42004 42004 49004 49004 0 23109
```

**Definitely not overlaps!**

True pars:  $pt=2909.18$   $d0=0.26$   $phi=-0.59$   $z0=-25.56$   $ctheta=-0.13$

Sector Slice bins: 16080 13975 4066 3934 4932

Consider fake sector 4:

**4 29007 29007 36007 36007 43007 43007 50007 50007 0 48545 – fake**

**6 29005 29005 36005 36005 43005 43005 50005 50005 0 47082 – real**

- You might ask: how could real athena tracks whose parameters are all within the same narrow bins (bin width =  $2*\pi/10000$  etc) produce sectors that are separated by two modules in eta? Is it a bug?
- No, it's not a bug, but a feature of patt-from-const sector lookup:
  - For curv bin 16080, we save all sectors that landed there
  - For d0 bin 13975, we save all sectors that landed there
  - (etc)
  - In the end, we take logical AND of 5 sets of sectors
    - It is never required that the all 5 bins fired up at the same time (aka on the same training event)!
- I went back to slice training and verified this by hand – next page

True pars:  $pt=2909.18$   $d0=0.26$   $phi=-0.59$   $z0=-25.56$   $ctheta=-0.13$

Sector Slice bins: 16080 13975 4066 3934 4932

Training events in sector 4 that landed in these parameter bins:

**curv:**

AK 4 16080 0.000172 6101 -1.126138 4156 -0.530237 7834 68.016777 4984 -0.030445

**d0:**

AK 4 17068 0.000219 13975 0.259691 4168 -0.522520 7364 56.754395 5057 0.114613

**phi:**

AK 4 14180 0.000081 9162 -0.587362 4066 -0.586750 5689 16.539371 5026 0.052650

AK 4 17200 0.000226 10961 -0.270757 4066 -0.586731 1898 -74.437172 5172 0.345223

<...>

**z0:**

AK 4 23234 0.000515 20978 1.492146 4263 -0.463070 3934 -25.576973 5044 0.088904

AK 4 21112 0.000413 5379 -1.253162 4134 -0.543508 3934 -25.568138 5132 0.264644

AK 4 21783 0.000446 23037 1.854604 4233 -0.481503 3934 -25.574043 5082 0.165412

AK 4 19862 0.000353 9699 -0.492948 4117 -0.554715 3934 -25.570253 5044 0.088219

<...>

**eta:**

AK 4 15602 0.000149 17250 0.836108 4099 -0.565888 9999 138.455750 4932 -0.135915

AK 4 24999 0.000601 14563 0.363164 4268 -0.459871 9999 159.474625 4932 -0.135058

AK 4 21434 0.000429 168 -2.170334 4197 -0.504039 8373 80.972160 4932 -0.135832

AK 4 24527 0.000577 17701 0.915514 4250 -0.470831 9999 159.717316 4932 -0.134601

<...>

All five bins never come from the same training event! Yes the combination of events make sector 4 “compatible” with this particular set of parameter bins!

# Ah, but instead you could...

- What if we restructure the slice lookup, so that all 5 bins had to fire up on the same training event?
- I.e., the lookup becomes a large array:
  - <list of sectors>[bin1][bin2][bin3][bin4][bin5]
- I quickly rewrote the slice infrastructure (thanks to STL multimaps!)
- With this, I repeated the above test of feeding pattgen training events. As expected, patt-from-const guesses the right sector, and nothing else in ~90% of the cases. In the remaining ~10%, it picks up only 1 extra fake sector.
- But this was done on the same pattgen events used in slice generation. With independent single muons, it finds zero sectors in ~90% of cases!
  - **To cover the entire phase space of 10000 slice bins in 5 parameters, we'd need a lot more training muons:-)**
- To reduce this phase space, I had to dramatically increase the bin size. But then it again starts finding a lot of garbage patterns.
- Thus this method of sector lookup looks hopeless given our statistics.

# Conclusion on patt-from-const

- For 7L/11L, we can continue using the old method:
  - Make a lot of patts **using default slices**, and cut from bottom
  - For large banks, we can use 16 subregions
- Indep muon test of slices: percentage of time it guesses the right 8L sector
  - Nbins=100 (default): 95%
  - Nbins=1000 (new) : 79% → can be improved with more training
- For 8L, it is possible to produce pattern banks using new slices with reduced bin size
  - But my test shows that even with ss=12, we have >10k roads
  - If no new ideas come up, we can abandon this configuration
- The slice binning can be optimized further, but it doesn't really make a difference in terms of CPU run time needed to produce a bank of a given quality.
- Additional improvements:
  - Overlap removal code does not handle forward region overlaps
    - I can try to fix this, but forward-region patterns are already as good as the central patterns (e.g, looking at eff-vs-eta plots)
  - More statistics (extra ~330M events) → better slice coverage → can use smaller slice binning with no loss in efficiency