Generation and performance of pattern banks for FTK track reconstruction

Lilly Luongo

28 Maggio 2012

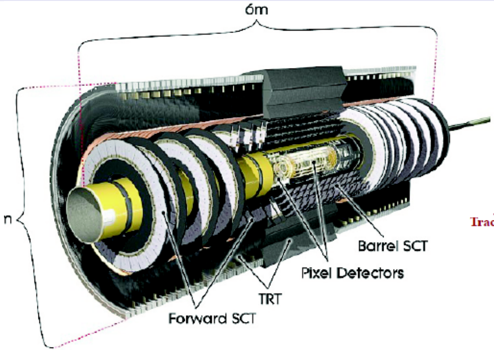
Motivations

- High LHC luminosity ⇒ enormous background
- The most interesting processes are rare and hidden under this extremely large background
- Immense real-time data reduction is needed
- Online track reconstruction is very important to separate the interesting events from the background

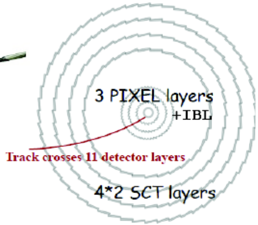
⇒ Solution: multi-level trigger system

⇒ In ATLAS: FTK

System description and segmentation



Example:
R-phi view of Barrel region:



Segmentation

- 4 η-rings: 2 rings in the barrel, 2 rings in the endcap ($[1, 2.5]$ and $[-2.5, -1]$)
- Each ring is divided into 16 ϕ-slices

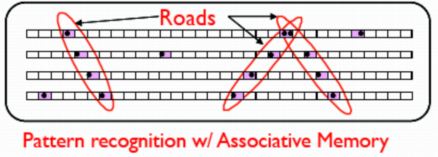
$\Rightarrow$ 64 towers

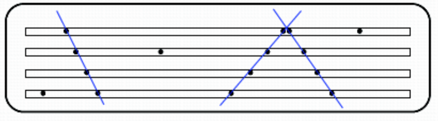
- Endcap: [0..15] and [48..63]
- Barrel: [15..47]

FTK algorithm

Two sequential steps:

1. Pattern recognition, carried out by a dedicated device called Associative Memory (AM). Find coarse-resolution track candidates called "roads".
2. Fit the full-resolution hits inside the road to determine the track parameters. Only the tracks passing the χ^2 cut are kept.





Parameters to define the pattern-bank performance

Pattern bank

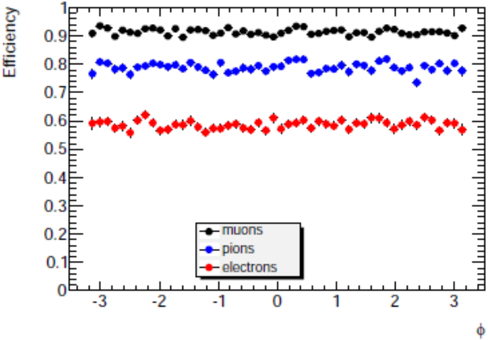
Each track generates a hit pattern. The collection of all these patterns defines both the space of the tracks we are looking for and how they appear in the detector: this collection is the pattern bank

Bank efficiency

$$Eff = \frac{NT(F)}{NT} \qquad (1)$$

where:

- NT = number of truth tracks
- $NT(F)$ = number of truth tracks matching FTK roads



Parameters to define the pattern-bank performance

- Critical parameter: road width
  - Too narrow $\Rightarrow$ too large needed size of the AM $\Rightarrow$ too large cost
  - Too wide $\Rightarrow$ big number of fake roads $\Rightarrow$ excessive work for the track fitter
- Solution: "Don't Care" (DC) bits
  - Variable resolution patterns

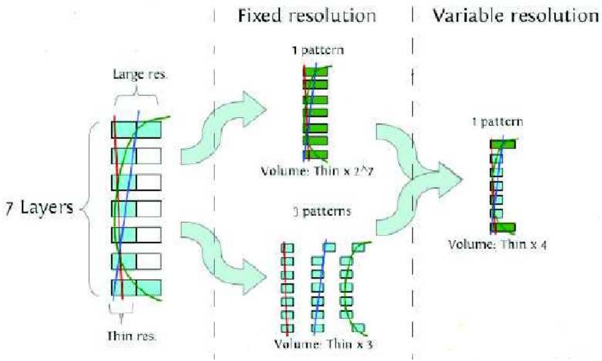
  $\rightarrow$ Each pattern for any layer can have an optimal width

  $\rightarrow$ Define the granularity layer by layer and pattern by pattern

$\Longrightarrow$ Limits the number of patterns within the hardware limits

$\Longrightarrow$ High rejection of fake roads

“Don’t care” bits



- High compression factor in case of similar patterns
- Only one pattern location

Goal

1. Performance study
   - High efficiency with limited number of patterns
   - Limited data bandwidth to obtain high fake suppression

   $\Rightarrow$ Optimal use of DC bits
2. Improving pattern bank generation
   - Reducing time of pattern generation
   - Improving the quality of pattern generation sw

Multiple DC bits study

1. Generate a bank with maximum resolution patterns: the TSP bank (the violet SSs are those shared by the red and blue TSP patterns)
2. Cluster the TSP patterns in the widest patterns for the AM chip
3. Apply the DC-bits to change the shape of the AM patterns

HW constraints per pile-up events

Constraints for each board:

- 8M patterns
- 8K roads
- 40K fits

1. 46 pile-up events: 16 boards working on 32 towers
   $\Rightarrow$ constraints for each tower:
   - $\#AM < 4M(4 * MPattern)$
   - $Roads < 4 * 10^3$
   - $Fits < 20 * 10^3$
2. 72 pile-up events: 128 boards working on 64 towers
   $\Rightarrow$ constraints for each tower:
   - $\#AM < 16M(16 * MPattern)$
   - $Roads < 16 * 10^3$
   - $Fits < 80 * 10^3$

Configurations

- TSP bank configuration: 15x16x36
  - → 15x36 = number of pixels clustered in the same Super Strip
  - → 16 = number of strips clustered in the same Super Strip
- Dataset with 72 pile-up events
- Constraints:
  - $\#AM < 16M(16 * MPattern)$
  - $Roads < 16 * 10^3$
  - $Fits < 80 * 10^3$
- AM bank configurations: ($N_{DC}(pixel_x)$,$N_{DC}(pixel_y)$)-N_{DC}(SCT)

1. (1,0)-1 → 30x32x36
2. (1,1)-1 → 30x32x72
3. (1,2)-1 → 30x32x144
4. (1,1)-2 → 30x64x72

⇒ Grouping with DC bit makes the SS granularity decreases

Endcap

0-7 towers:

DC bit	#TSP $\cdot 10^6$	#AM $\cdot 10^6$	Efficiency(%) R=64	Roads/evt $\cdot 10^3$	Fits/evt $\cdot 10^3$	Tracks/evt
(1,0)-1	120	34	87	5.8	33	109
(1,1)-1	120	18	91.2	7.1	56	106
(1,1)-1	112	16.8	91.2	6.9	55	...
(1,1)-1	100	15	91	6.2	50	...
(1,2)-1	...	8	92	5	90	...
(1,1)-2	...	8	93	9	154	...

Table: Results in 0-7 towers - endcap. #AM patterns, #Roads, #Fits and #Tracks are evaluated in tower 0.

- The TSP bank size is defined on the cut of the less frequent patterns
- The efficiency is evaluated on the single muon dataset
- #AM patterns, #Roads and #Fits are evaluated on the dataset with 72 pile-up events
- The #Roads provides a measure of the fake roads

Endcap

10-15 towers:

DC bit	#TSP $\cdot 10^6$	#AM $\cdot 10^6$	Efficiency(%) R=64	Roads/evt $\cdot 10^3$	Fits/evt $\cdot 10^3$	Tracks/evt
(1,1)-1	120	...	...	7259	58	102
(1,1)-1	100	15	91	6256	50	93
(1,1)-1	112	16.8	91.22	6871	55	99

Table: Results in 10-15 towers - endcap. #AM (#TSP) patterns, #Roads, #Fits and #Tracks are evaluated in tower 10.

Barrel

32-39 towers:

DC bit	#TSP $\cdot 10^6$	#AM $\cdot 10^6$	Efficiency(%) R=64	Roads/evt $\cdot 10^3$	Fits/evt $\cdot 10^3$	Tracks/evt
(1,0)-1	120	34	...	3836	23	51
(1,1)-1	120	21	94.71	4916	42	52

Table: Results in 32-39 towers - barrel. #AM patterns, #Roads, #Fits and #Tracks are evaluated in tower 32.

Barrel

26-31 towers:

DC bit	#TSP $\cdot 10^6$	#AM $\cdot 10^6$	Efficiency(%) R=64	Roads/evt $\cdot 10^3$	Fits/evt $\cdot 10^3$	Tracks/evt
(1,0)-1	120	34	90	3.8	23	...
(1,1)-1	120	21	94.75	3.9	33	42
(1,1)-1	100	18	94.07	3.4	28	38
(1,1)-1	90	16.8	93.35	3.2	26	36
(1,2)-1	...	8	95	4	60	...
(1,1)-2	...	8	96	6	98	...

Table: Results in 26-31 towers - barrel. #AM (#TSP) patterns, #Roads, #Fits and #Tracks are evaluated in tower 26.

Work in progress

- Exploring new TSP bank configurations
- TSP bank configuration: 11x12x18
  - $\longrightarrow$ 11x18 = number of pixels
  - $\longrightarrow$ 12 = number of strips
- We have generated the TSP bank
- We are trying some DC-bits bank configurations:
  - (1,2)-1 $\rightarrow$ 22x24x72
  - (2,2)-2 $\rightarrow$ 44x48x72
- We will have the efficiency numbers soon

Improvement

"State of the art"

Generating the TSP bank:

- C-written sw 5 years old
  - Still large SVT legacy
- The old sw is efficient but it's almost impossible to maintain or to improve it
  - It has many functionalities (sectors and patterns from real, pattern from constants)
  - A specific target should improve its performance
- The old sw can't be integrated with ATLAS production

Improvement

- GOAL: Generating the TSP pattern bank as quickly and efficiently as possible

⇒ Collaboration with the Japanese group to update the existing tools in order to make the existing generation system:

  - modern
  - efficient
  - quick
  - maintainable

Steps

1. Porting the old source code from C to C++
   - Formats optimization
   - ASCII $\rightarrow$ ROOT
   - Performance check
   - Final tests
2. Revisiting the existing algorithm and its optimization
   - Don't change the original algorithm and check the performance
3. Searching for alternative methods for pattern generation in order to
   - Improve the patterns' efficiency
   - Improve the patterns' quality

Results

Step 1: almost done!

- Pattern production algorithm ported

Performance (time for each track): old MakeBank vs new PattBankGen

- MakeBank
  - cpu: $0.035 ms/track$
  - real: $0.036 ms/track$
- PattBankGen
  - cpu: $0.037 +/- 0.001 ms/track$
  - real: $0.123 +/- 0.018 ms/track$