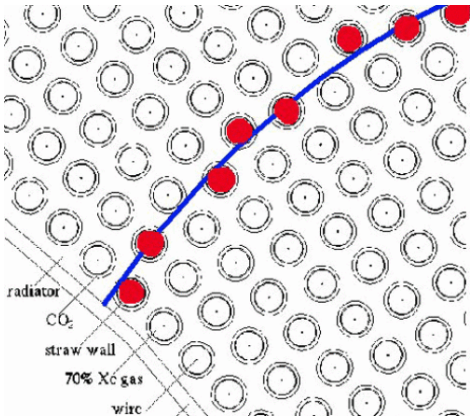
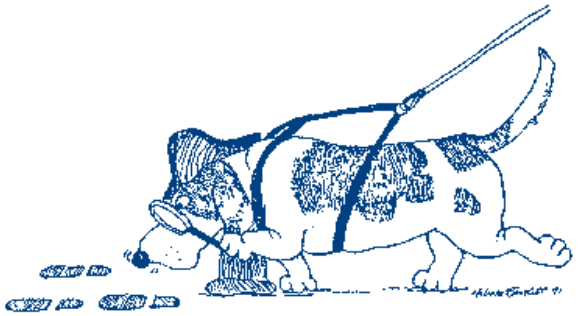


Track finding techniques in experimental particle physics

Dmitry Emeliyanov (RAL)



Introduction

- Track finding is typically the most CPU time consuming operation in HEP data analysis
- There are many methods proposed to date
- The aim of this seminar is to review them
  - keeping in mind that methods deemed unfeasible in the past might become feasible today owing to the fast progress in computing hardware

Track finding

- The goals of track finding
  - data association: identification of measurements belonging to the same track
  - track parameter estimation and trajectory fitting
- In general, there are several groups of track finding methods:
  - the fundamental difference lies in the way how a track is defined

Classification of tracking methods

Track is a

1

sequence of hits in the detector

Sequential, “step-by-step” data association methods, e.g. track following using Kalman filter

2

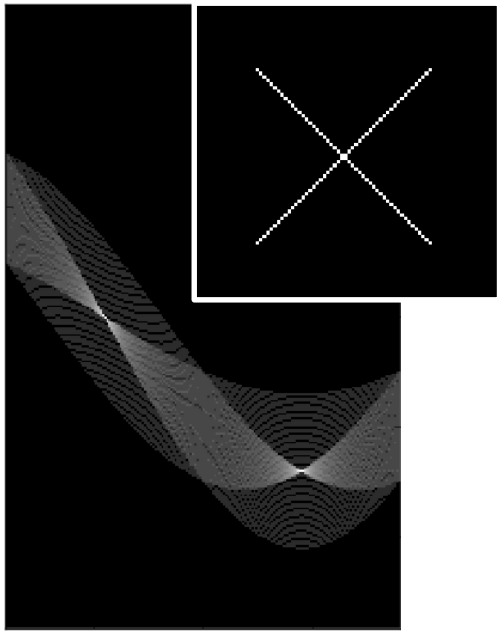
point in the track parameter space or a function space of possible track shapes

Pattern recognition using integral transforms (Radon, Hough), template matching

3

sequence of track fragments = **points** in track parameter space

Track finding using cellular automata with graph-based track models



1. Sequential methods

The track following approach

- A very popular track finding technique – implemented in many particle physics experiments
- The main track finding approach for ATLAS and CMS experiments

Billoir and Qian, NIM **A 294**, 1990

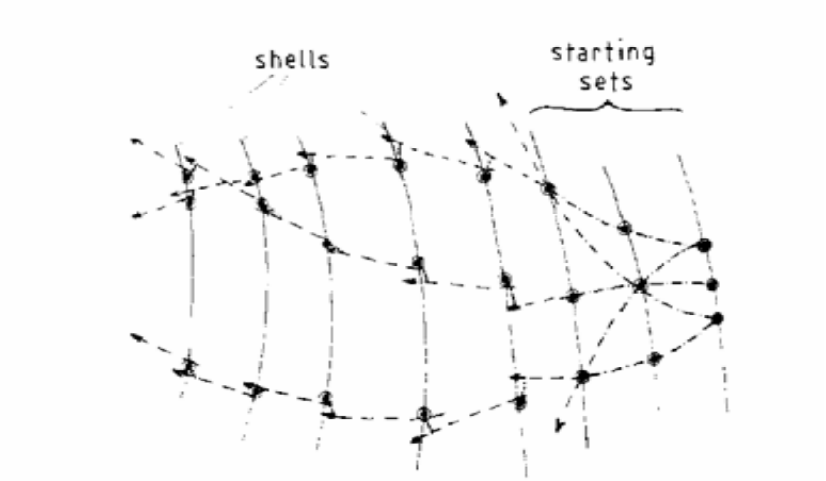
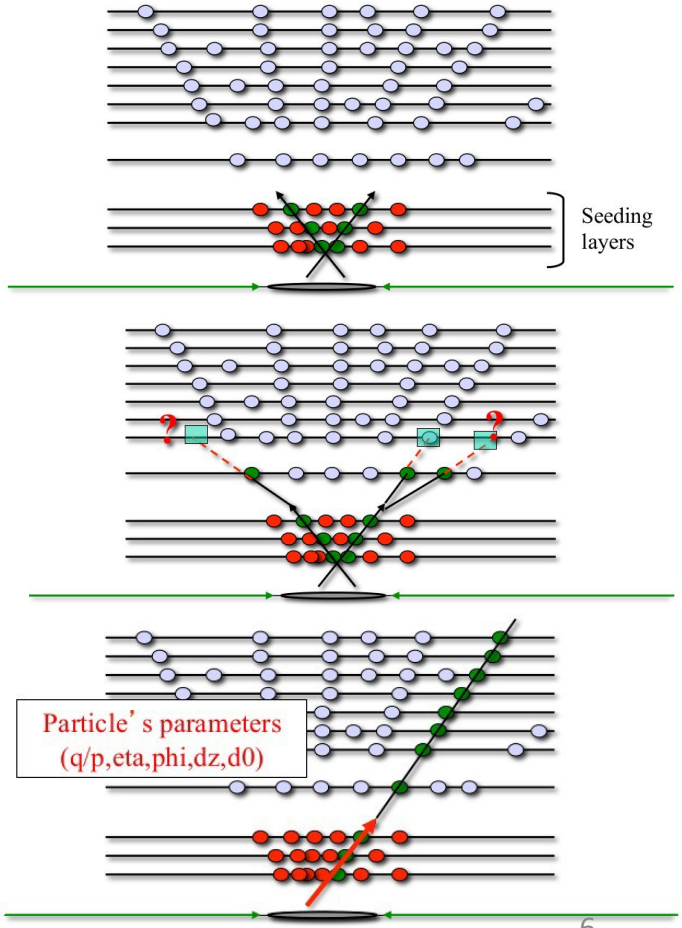
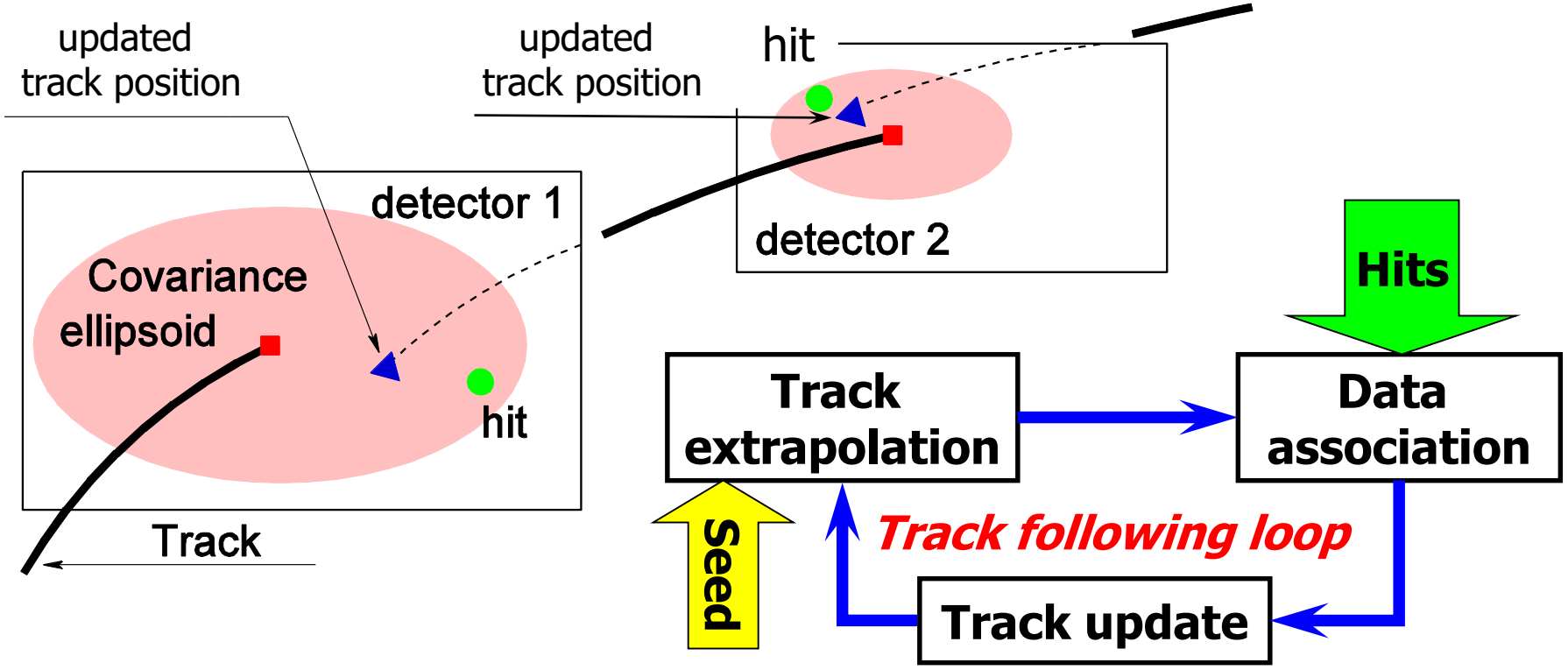


Fig. 1. Principle of progressive track recognition.

CMS tracking @ 2015
"Connecting the Dots" conf.



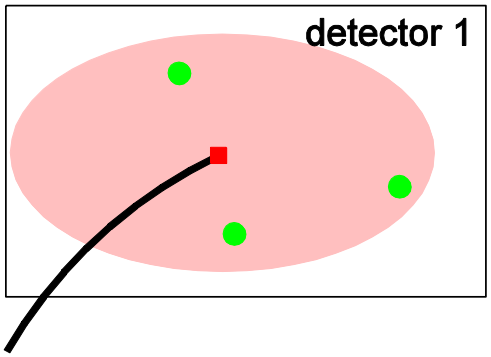
“Step-by-step” data association



- Naturally employs the Kalman filter framework: accounts for the detector material effects during the track finding

The data association problem

Which of the three hits to use for track update ?



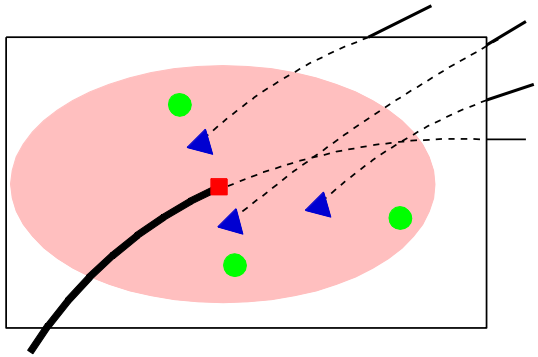
Solution I: (**Nearest Neighbour Filter**):

- use the hit closest to the predicted track position: $\Delta\chi_i^2 \to \min$
- NNF is computationally simple but is **sensitive** to:
- high density of noise hits (**pile-ups**) – the closest hit might be noise
- **detector inefficiencies** – track is undetected so all hits are noise

Solution II: (**Combinatorial Kalman Filter**): used in ATLAS and CMS

- split a track into independent branches
- propagate them to the next detector layer and split if more than one hit found
- discard bad branches using track log-likelihood:

$$L = -\sum \Delta\chi^2 - \text{penalty for missing hits}$$



Solution III: **PDAF**

- Probabilistic Data Association Filter uses *all hits* at once

1. Track parameters update:

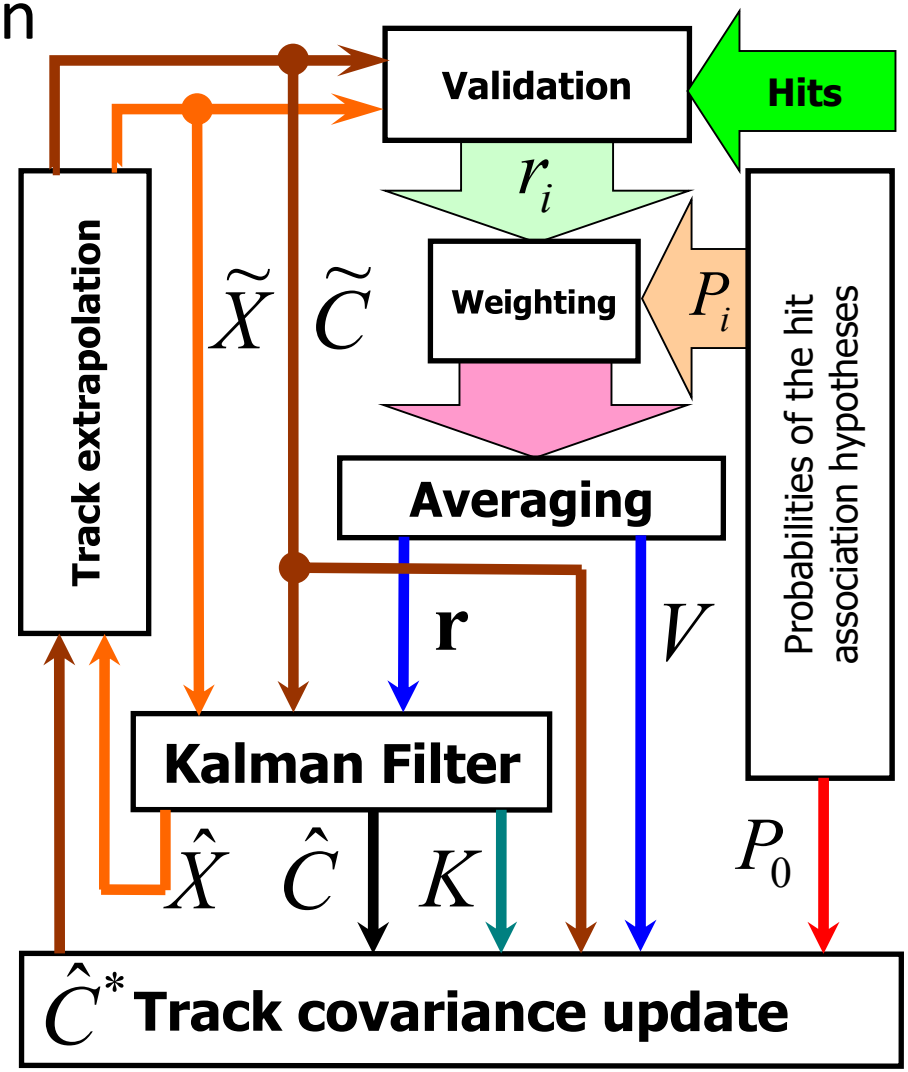
$$\hat{X} = \tilde{X} + K\mathbf{r}$$

combined residual: $\mathbf{r} = \sum P_i \, r_i$

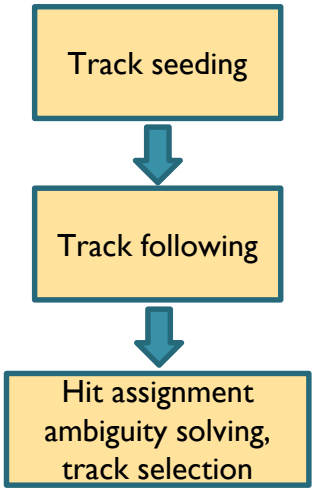
2. Track covariance update:

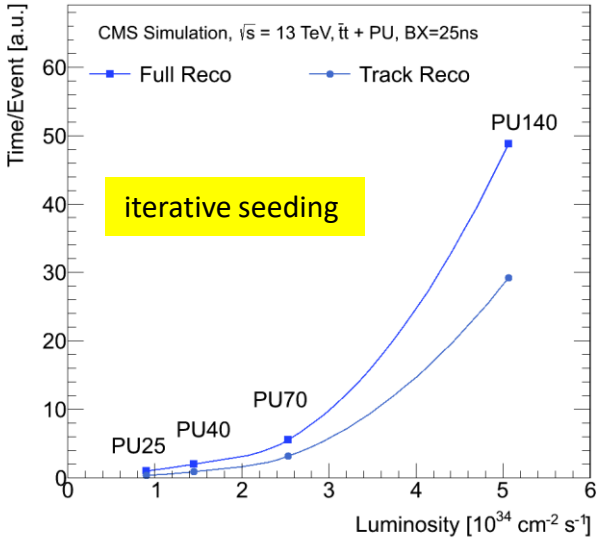
$$\hat{C}^* = P_0\tilde{C} + (1 - P_0)\hat{C} + KVK^T$$

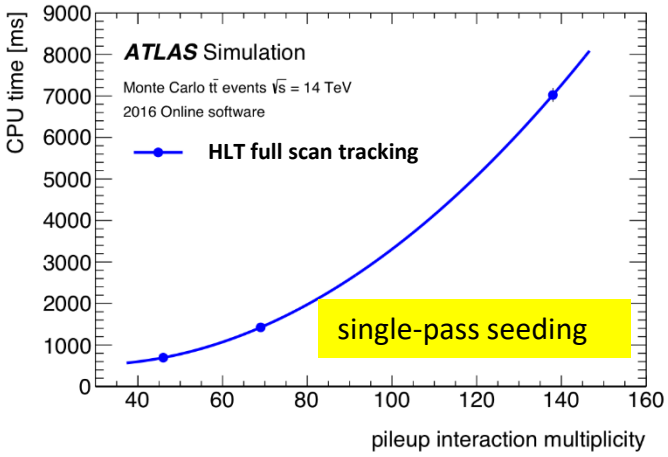
residual ***variance***: $V = \sum P_i \, r_i^2 - \mathbf{r}^2$

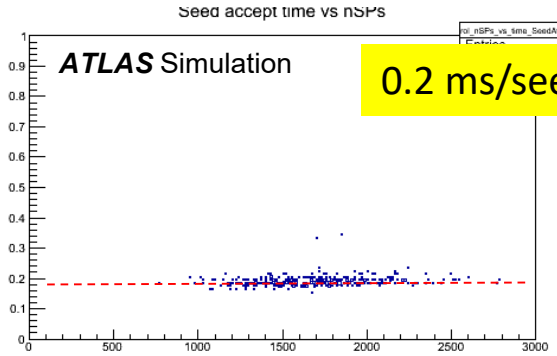


Computational complexity









- Time steeply increases with pileup level
- Processing time / seed stays constant
- CPU time model (ATLAS HLT tracking):

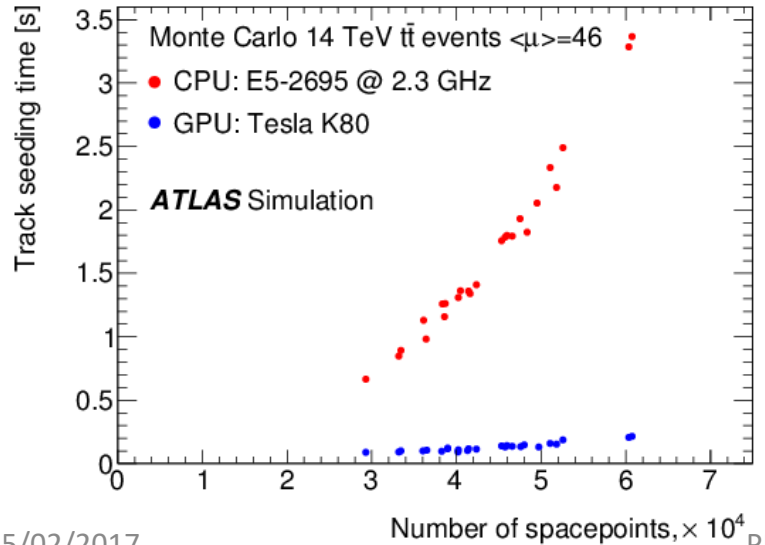
$T_{Total} \sim T_{TS} + N_{seeds} \times T_{TF}$, where TS is Track Seeding, TF is Track Following / seed

$N_{seeds} \sim N_{hits}^{d}$, where $d = 2 \div 3$, T_{TS} has similar scaling w.r.t. the number of hits

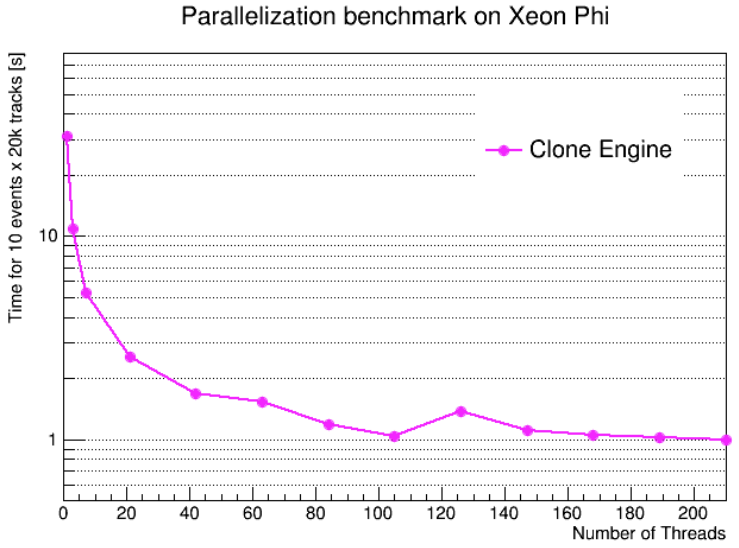
Going parallel

- Both, track seeding and track following could be significantly accelerated via parallelization
  - x20-30 is possible if run on a GPU or Xeon Phi coprocessor
  - typically, the memory bandwidth rather than computing capabilities of a processor is the limiting factor

- Parallelization of track seeding



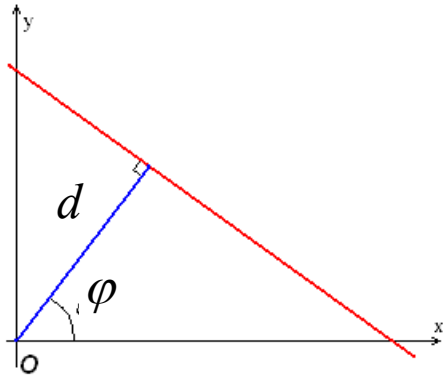
- Parallel track following (CMS)



2. Global pattern recognition

Radon transform

- Classical Radon transform (RT):
  - representation of image $I(x, y)$ in terms of its line integrals



$$\Re\{I\}(d,\varphi) = \int_{\mathbf{R}} I\left(d\cos\varphi - l\sin\varphi, d\sin\varphi + l\cos\varphi\right) dl$$

- Generalized Radon transform (GRT):
  - convolution with parametrized templates $\mathbf{r} = (x, y) = T(\mathbf{p})$

$$\Re\{I\}(\mathbf{p}) = \int_{\mathbf{R}^2} I(\mathbf{r}) K(\mathbf{r};\mathbf{p}) d\mathbf{r}, \text{ where kernel } K(\mathbf{r};\mathbf{p}) = \delta\left(\mathbf{r} - T(\mathbf{p})\right)$$

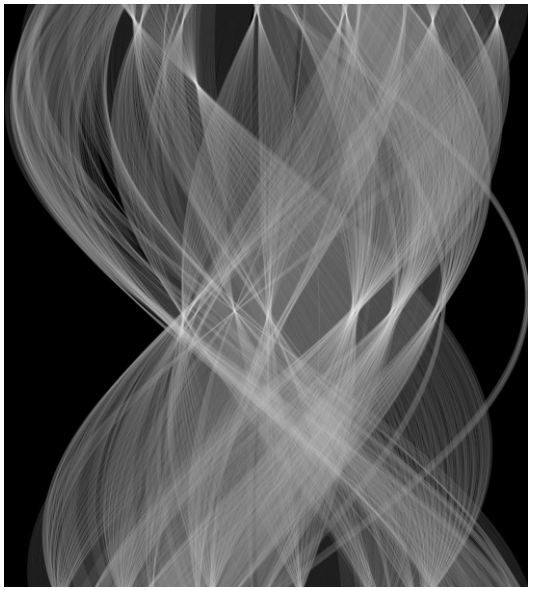
- Probabilistic or “Fuzzy” Radon transform:
  - using Gaussian convolution kernel instead of δ-function

Hough transform

- The HT is a fast computational scheme for the RT:
  - assuming discrete parameter space and integral linearity
- RT vs HT: two different computational paradigms
  - *Reading* paradigm (Radon): for each $\mathbf{p}$ read all the values of $I(\mathbf{r}), \mathbf{r}$ such that $K(\mathbf{r};\mathbf{p}) > 0$ and sum $I(\mathbf{r})K(\mathbf{r};\mathbf{p})$
  - *Writing* paradigm (Hough): initialize the transform $\Re\{I\}(\mathbf{p})$ to zero; for each *non-zero* point in the image determine *all points* $\mathbf{p}$ in the parameter space such that $K(\mathbf{r};\mathbf{p}) > 0$ and update $\Re\{I\}(\mathbf{p})$ with "votes" $V(\mathbf{p};\mathbf{r}) = I(\mathbf{r})K(\mathbf{r};\mathbf{p})$
  - if the input data (image) is sparse the HT offers significant reduction in computation time
    - if the mapping back to parameter space can be easily calculated

From peaks to tracks

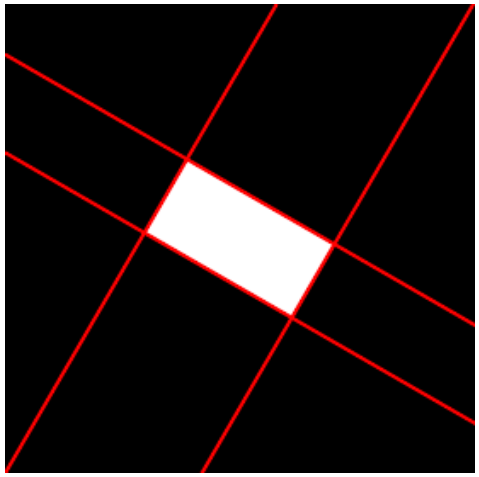
- HT image has a complex structure:
  - because HT does not do any data association – each hit can "vote" for a multitude of possible tracks
  - the data association still needs to be solved after the HT is done



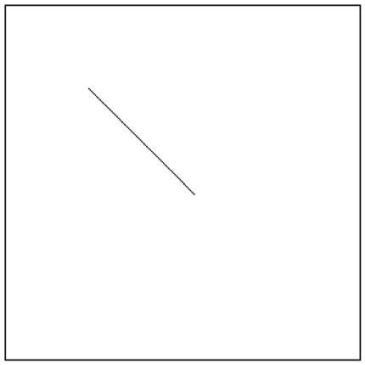
- Possible solutions:
  - "winner-takes-all" – find the highest peak, collect hits, (*remove* theirs votes), repeat until no significant peaks left
  - iterative HT with reweighting: $V(\mathbf{p};\mathbf{r}) = L(I(\mathbf{r})\,|\,\mathbf{p})I(\mathbf{r})K(\mathbf{r};\mathbf{p}),$
    - $L(\cdot\,|\,\cdot)$ is likelihood of image element belonging to track $\mathbf{p}$
    - for this and other techniques see: O.Barinova et.al. "On Detection of Multiple Objects Instances using Hough Transforms", IEEE Conf. on Computer Vision and Pattern Recognition, 2010

Butterfly analysis

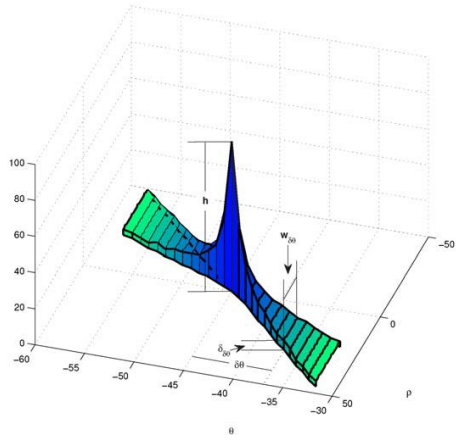
- The HT cannot detect track length
  - and does not care about distribution of hits along track – crucial for track QC/QA
- Analysis of angular distribution in "butterfly" shapes around HT peaks:
  - matched filter tuned to the *expected* butterfly distribution supresses spurious peaks in the HT image



Du S, Tu C, van Wyk BJ, Ochola EO, Chen Z
Measuring Straight Line Segments Using HT Butterflies.
PLoS ONE 7(3): e33790 (2012)
doi:10.1371/journal.pone.0033790



(a) A straight line segment

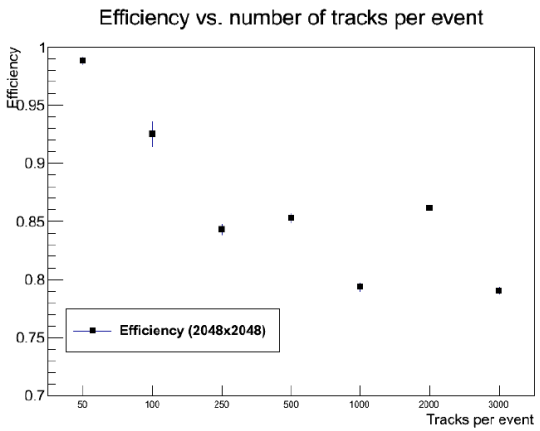


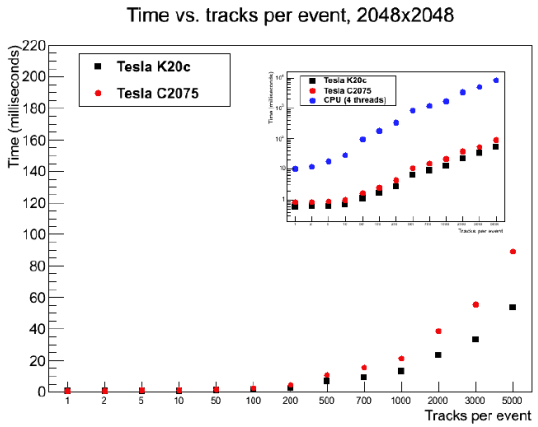
(b) The HT butterfly of the segment

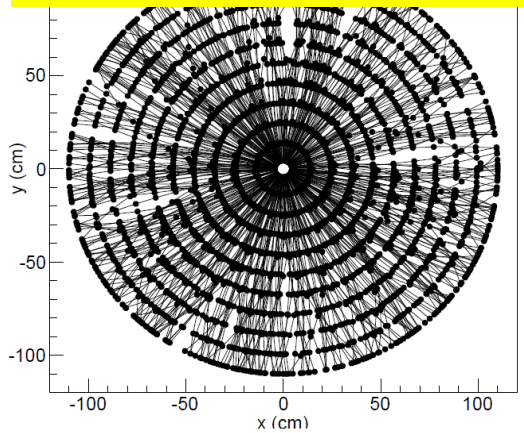
Parallel implementations

- 2D $(\varphi_0, 1/p_T)$ HT using a GPU

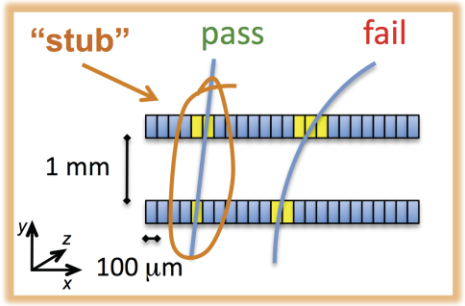
V.Halyo et.al. JINST 8, 2013

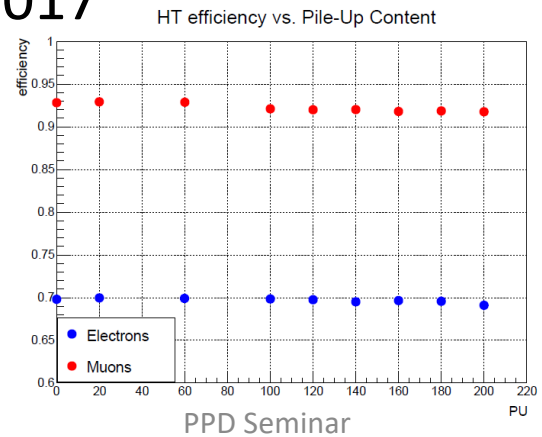


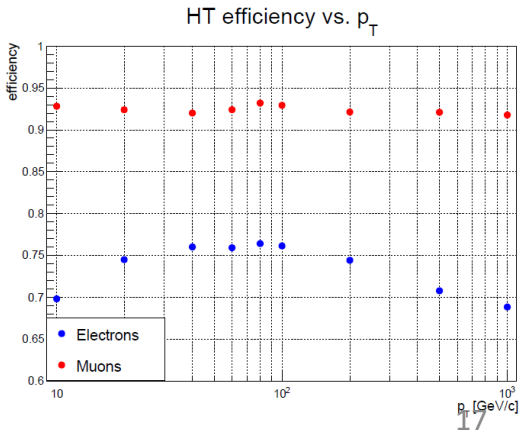




- FPGA-based 2D HT for CMS Trigger upgrade
  - PPD sem. 25/01/2017







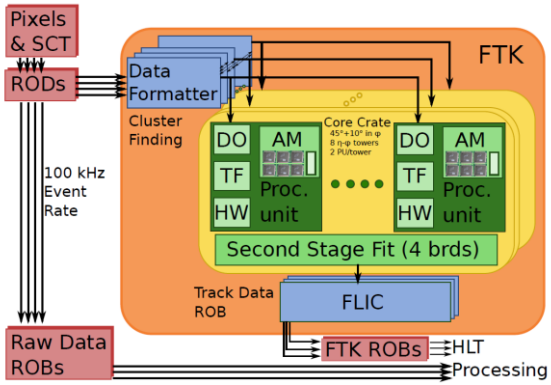
Pattern-based discrete RT

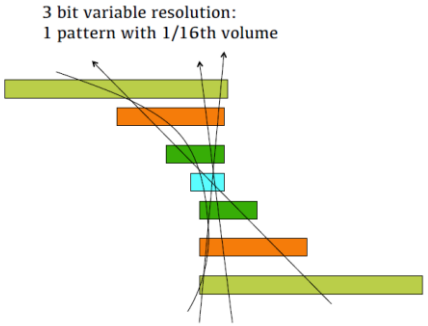
- Discrete RT can be formulated as matching hit-patterns (i.e. coarse tracks) to hits in an event
- Space of possible detector hits $\mathbf{H} = \{h_k\}$
- Space of hit patterns $\mathbf{P} = \{P_i\}, P_i = \{h_1^i, \ldots, h_n^i\} \subset \mathbf{H}$
- Event $\mathbf{E}: \{h_j\} \subset \mathbf{H}$
- The transform $\Re(\mathbf{P}, \mathbf{E}): \forall P_i \in \mathbf{P} \Rightarrow F(F_i, \mathbf{E})$, where
- $F(\cdot,\cdot)$ is the matching function of hit patterns,
  - for example, a number of pattern hits in the event

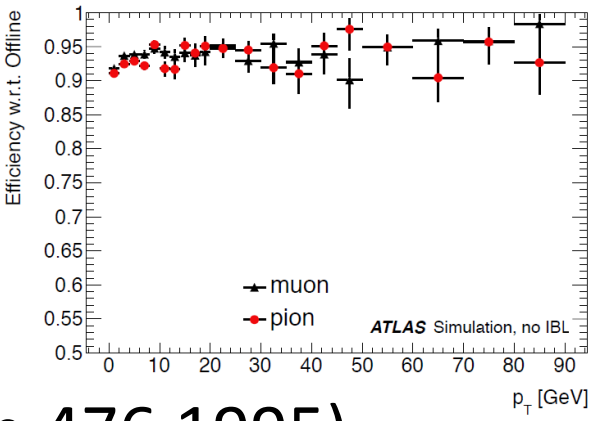
$$F = |P_i \cup \mathbf{E}|$$

Parallel pattern matching

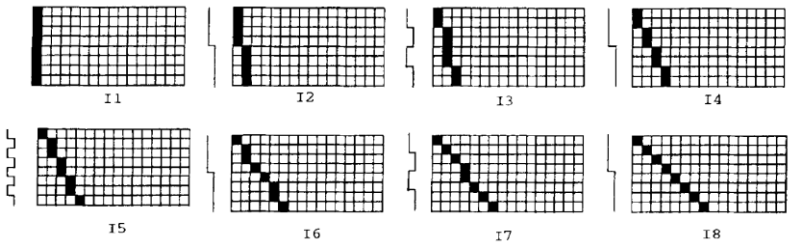
- ATLAS Fast TracKer (FTK)
  - 10^9 pre-stored patterns matched using Associative Memory

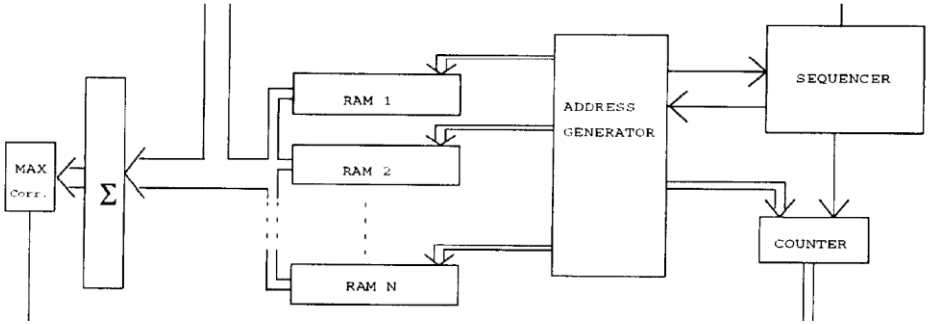






- Hierarchical correlator (NIM A 356 p.476 1995)
  - on-the-fly pattern (=groups of RAM addresses) calculation





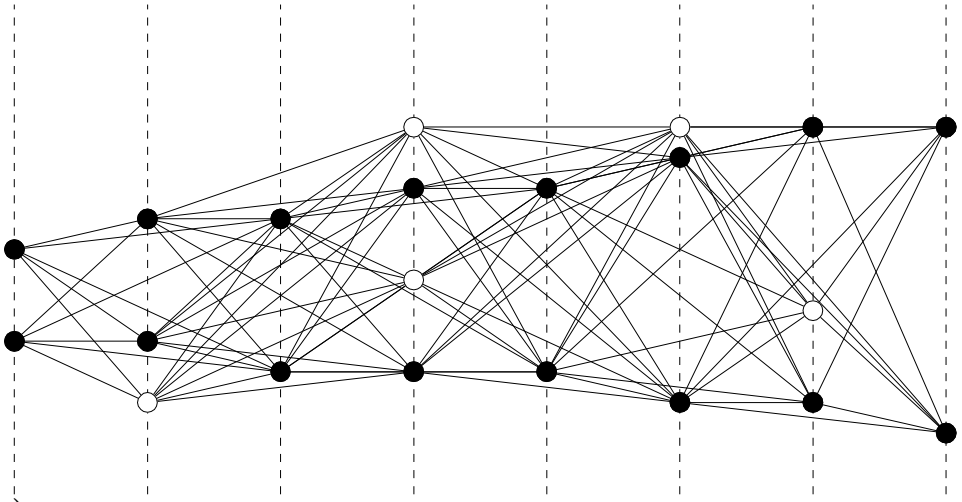
Global methods: conclusion

- Some observations for HT
  - dimensionality of track parameter space limited by 2-3
  - HT track extraction needs to be accompanied by a robust track refit capable of resolving data association ambiguities
- Hardware-supported pattern-based RT
  - optimization of the pattern set is crucial for performance and overall feasibility of the approach
    - basically, the problem is to find the best approximation for “hit image” created by reference tracks – it might be a good application for machine learning techniques
  - “fuzzy” matching can be supported by some hardware:
    - for instance, texture memory in GPUs

3. Graph-based methods

Tracking on graphs

- General idea:
  - find short track segments, e.g. pairs or triplets of hits
  - create the graph of track segments by setting *admissible* connection between segments
  - a track is an optimal (in some sense) path on the graph
- Example:
  - the segments connect spacepoints (SPs)
    - in the adjacent layers
    - across one layer
  - admissible connections:
    - segments have common SP
    - small breaking angle $\Delta\varphi(s_i, s_{i+1})$



Recursive track search

- Let $S_k = \{s_1, \ldots, s_k\}$ be a sequence of track segments
- Consider the following "best-path" criterion
  - basically, this is the number of connected segments with imposed cut on the breaking angles

$$J(S_k) = \sum_{i=1}^{k-1} I\left(\left|\Delta\varphi\left(s_i, s_{i+1}\right)\right| \le \Phi\right), \quad I(x) = \begin{cases} 1, \text{ if } x = \text{true} \\ 0, \text{ if } x = \text{false} \end{cases}, \quad \Phi \text{ is a cut}$$

- $J^* = \max_S J(S)$ satisfies Bellman recursion property:

$$J^*\left(S_{k+1}\right) = J^*\left(S_k\right) + \max_{s_{k+1}} I\left(\left|\Delta\varphi\left(s_k, s_{k+1}\right)\right| \le \Phi\right)$$

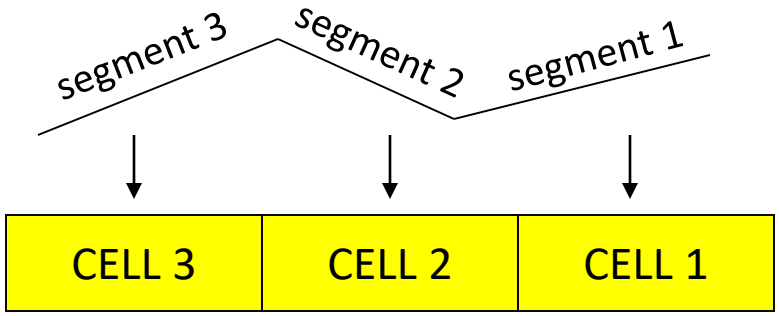
  - so that optimal segment sequences (i.e. track candidates) can be constructed recursively

The cellular automaton approach

Track segments are called ***cells***, cell can have ***neighbours*** – left neighbouring segments

Each cell has a ***state*** – positive integer number

The algorithm works on the cell set and has two passes:



Iter 0	s=1	s=1	s=1
Iter 1	s=1	s=2	s=2
Iter 2	s=1	s=2	s=3

• Forward recursion:

if cell ***i*** with the state ***s_i*** has a neighbour ***j*** with the equal state ***s_j=s_i,*** then, at the next iteration, the state is incremented:

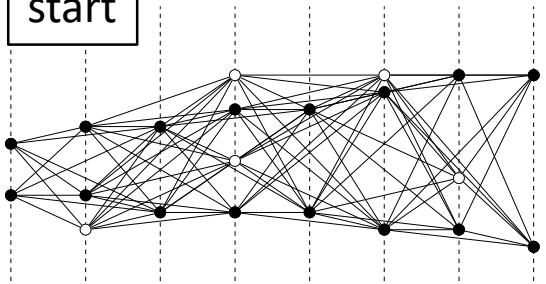
$$s\_i = s\_i + 1$$

The iterations stop when there is no neighbouring cells with equal states. The final cell state is equal to the length of the segment sequence which can be traced to the left starting from this segment

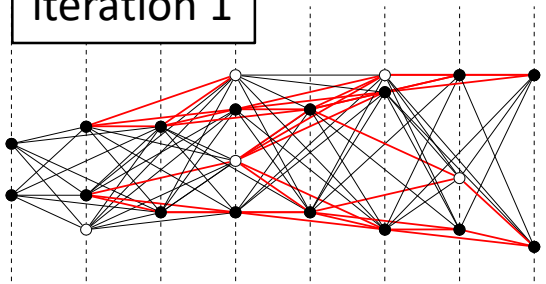
• Backward pass: Find the cell with the highest state ***S***, then find its neighbour with ***S-1*** and so on until state=1. If there are a few neighbours with the same state, take the segment with the smallest breaking angle. All cells assigned to the track candidate are removed from the cell set and the backward pass is repeated until all the track candidates are collected

Cellular automaton in action

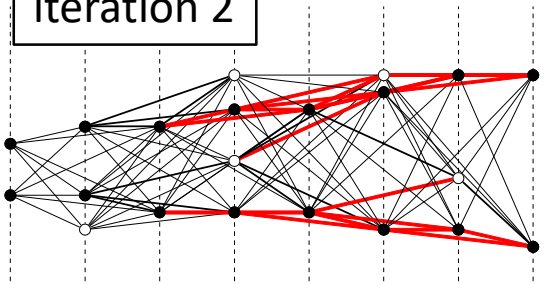
start



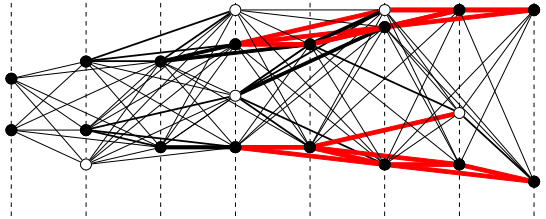
iteration 1



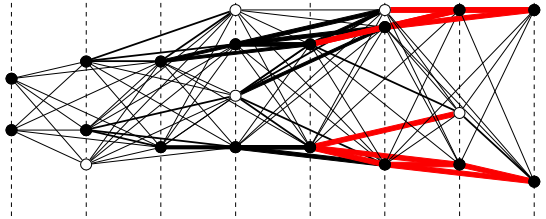
iteration 2



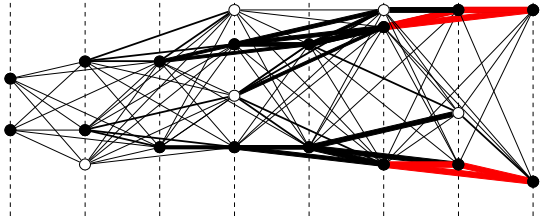
iteration 3



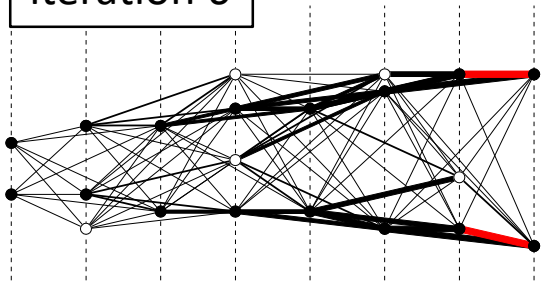
iteration 4



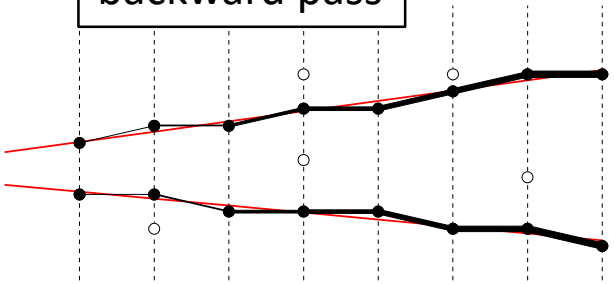
iteration 5



iteration 6



backward pass



Line width = cell state
Red: state updated at the iteration
Black: no state update

CA-based tracking

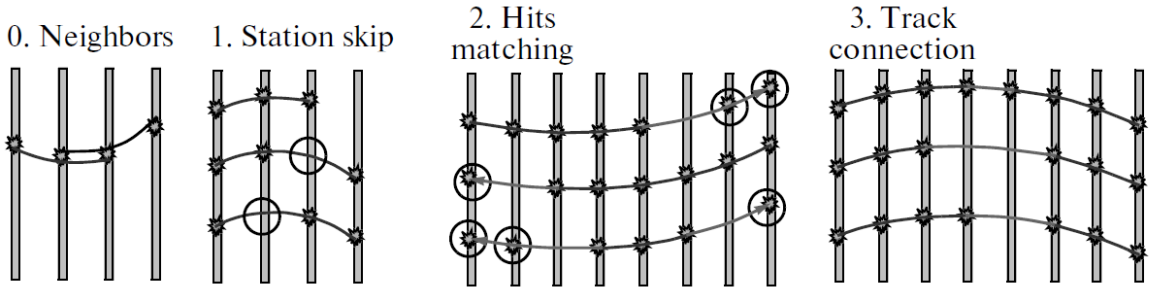
- Very fast, highly parallel algorithm
  - good data locality – ideal for GPUs
- Track finding efficiency
  - depends on track segment reconstruction efficiency
  - typically, requires exhaustive combinatorial search for track segments, e.g. triplets
    - defines the method “time vs. multiplicity” behaviour
    - as we discussed already, the triplet search can be significantly accelerated via parallelisation
- Fast track fit is required in order to
  - resolve remaining data association ambiguities
  - determine track parameters and test track quality

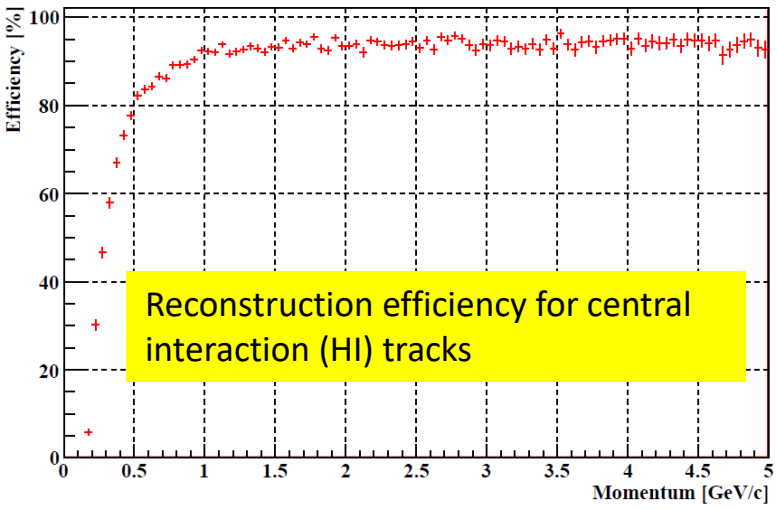
Implementation example

- CBM @ GSI FAIR: Fixed-target heavy-ion experiment

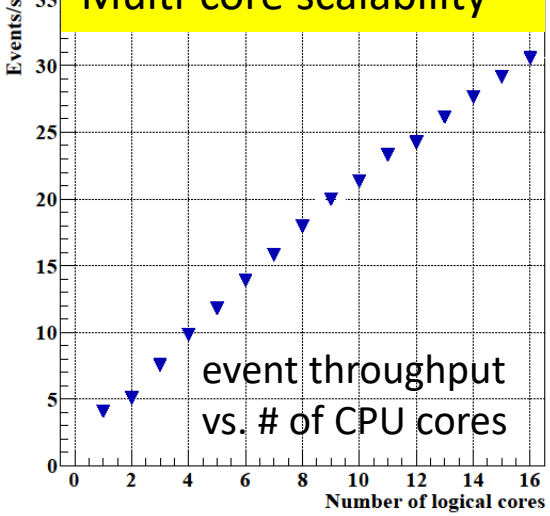
Triplets of hits are neighbours if they share **two** hits and have similar momentum estimates

$$|\Delta p| \leq \varepsilon_p$$





Multi-core scalability



Split CPU timing

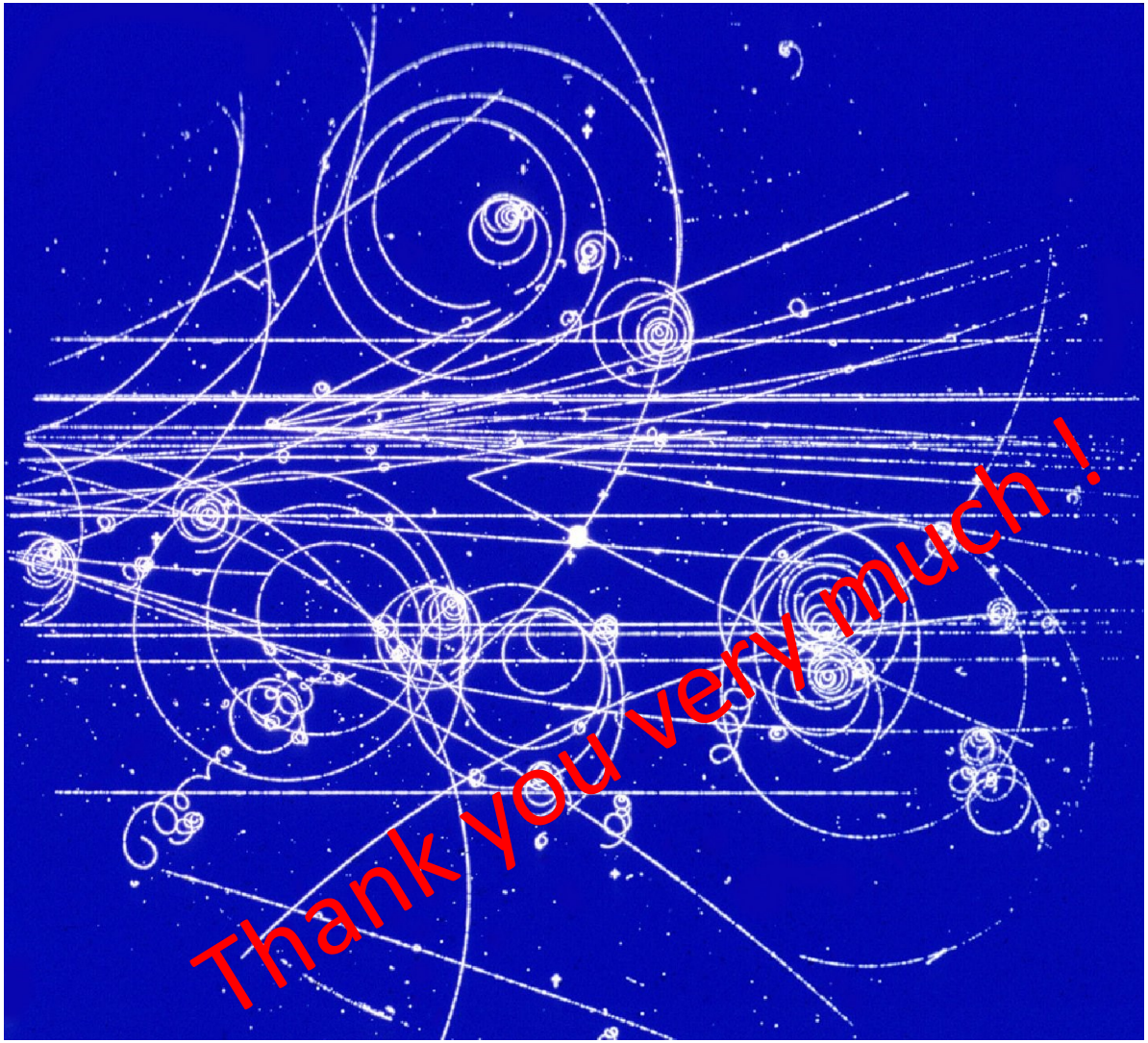
Stage of the algorithm	% of total execution time
Initialisation	8
Triplets construction	64
Tracks construction	15
Final cleaning	13

$$T_{Total} \sim N_{hits}^2$$

Conclusion and outlook

- We discussed 3 approaches for track finding
  - feasibility of each depends on: tracking scenario, implementation, and target hardware
- Outlook:
  - sequential methods:
    - intelligent track seeding and parallelization are essential for the tracking in LHC Phase II upgrade
  - global pattern recognition:
    - non-combinatorial nature is its advantage
    - could be the method-of-choice with the right balance between optimized pattern matching and robust track fitting
  - cellular automata:
    - could be competitive against track following as a fast track building algorithm; require fast parallel track segment finder

The end



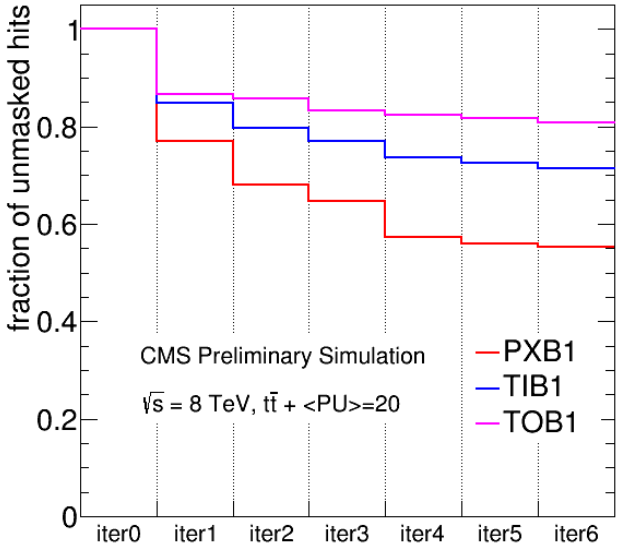
Backup slides

Tracking scenario characterization

- Simulated ATLAS events $t\bar{t} + 2\times10^{34}\ \mathrm{cm}^{-2}\mathrm{s}^{-1}$
- HLT full scan tracking $p_T \geq 1\,\mathrm{GeV}, d_0 \leq 5\,\mathrm{mm}$

<N seeds>	<N tracks>	<Nhits>	<nHits> used in HLT tracks
16000	405	40000	3940

- Less than 10% of hits are used in the ATLAS HLT tracks, the rest of data is, effectively, “noise” in the pattern recognition
- In CMS, after several iterations of seeding / track following ~50% of hits are not assigned to any tracks



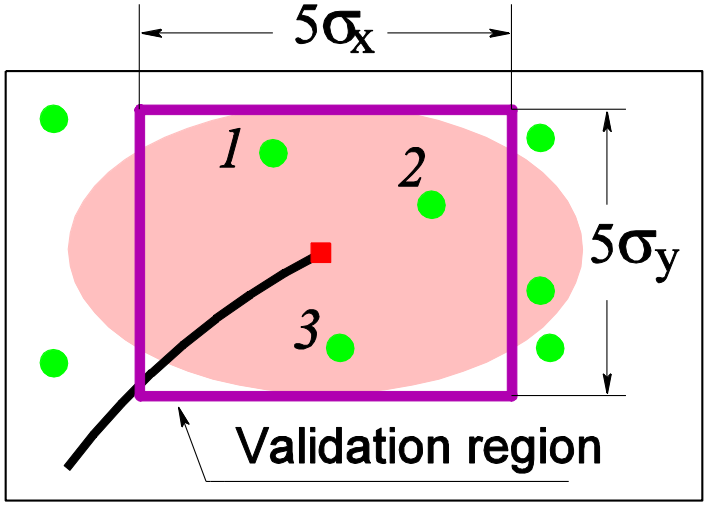
The probabilistic approach

- Basic assumptions of this method:
  - We accept only hits close to the track, for instance, falling in 5σ box (validation region) – *validated* hits
  - there is a single track **T** with:

    extrapolated parameters $\tilde{X}$

    extrapolated covariance $\tilde{C}$

    $\Big\} \tilde{I}$
  - ***At most*** one of the hits belongs to the track **T** and the other hits are produced by background (**B**)
  - The track can pass undetected
- Then the origin of the validated hits can be *explained* by one of the *hit association hypotheses*



hypo \ hit	hit u_1	hit u_2	hit u_3
Γ_0	***B***	***B***	***B***
Γ_1	***T***	***B***	***B***
Γ_2	***B***	***T***	***B***
Γ_3	***B***	***B***	***T***

Optimal track update using all hits

- By definition, the optimal track parameter update is a mean conditioned on all hits and extrapolated track information (parameters + covariance)

$$\hat{X} = \mathrm{E}(X \mid \{u\}, \tilde{I})$$

- The association hypotheses are mutually exclusive and cover all possible hit assignments – applying the total probability theorem gives:

$$\hat{X} = P(\Gamma_0)\tilde{X} + \sum P(\Gamma_i)\mathrm{E}(X \mid u_i, \tilde{I})$$

- $P(\Gamma_i) = P_i$ – probability of the i-th hypothesis being true
- $\mathrm{E}(X \mid u_i, \tilde{I}) = \hat{X}_i$ – track parameter estimate obtained using the i-th hit = output of the Kalman filter

Probabilities of association hypotheses

- From Bayes' theorem, an association hypothesis probability is

$$P_i = P\left(\Gamma_i \mid \{u\}\right) \propto p\left(\{u\} \mid A_i\right) P\left(A_i \mid \tilde{I}\right)$$

$\{u\}$ – validated hits, A_i is hit assignment postulated by hypothesis Γ_i and $P\left(A_i \mid \tilde{I}\right) = \pi_i$ is a prior probability of A_i

- Assuming hits to be statistically independent, the joint p.d.f. conditioned on the hit assignment A_i is the product of their p.d.f.s:

$$p\left(\{u\} \mid A_i\right) = \prod_{j=1} \begin{cases} p_T(u_j), & j = i \\ p_B(u_j), & j \neq i \end{cases}$$

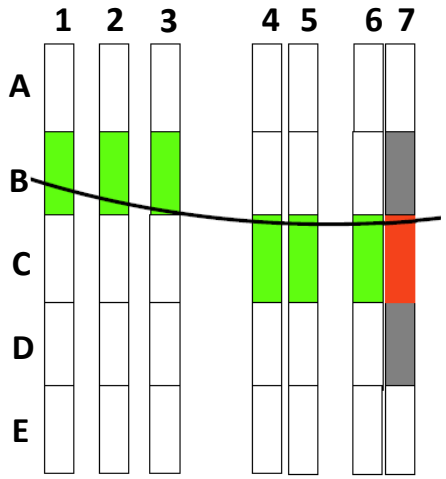
- p_T, p_B are p.d.f.s of track- and background-induced hits
- These p.d.f.s as well as the prior probabilities $\{\pi_i\}$ of hit assignments depend on tracking scenario and detector model – they must be chosen and tuned according to a concrete application

RT in higher dimensions

- parameter space becomes sparser
  - HT with a multidimensional accumulator array is less feasible
- voting techniques:
  - “peer-to-peer”: vector or tensor voting by communication between sparse measurements
  - “white-board” : for each bin in the RT space propagate votes to a common surface, find the bin (for each measurement) which maximises the total vote
    - iterative implementation is essentially the Expectation Maximization (EM) algorithm

Sparse Clustered NN for track finding

- A hit pattern is a fully connected graph (a.k.a. *clique*) of nodes representing coarse measurements
  - hit pattern



7 detector layers = 7 clusters
each layer/cluster has 5 possible hits

  - a *clique* – fully connected graph