Introduction to Patatrack

Matti Kortelainen

CCE PPS Meeting

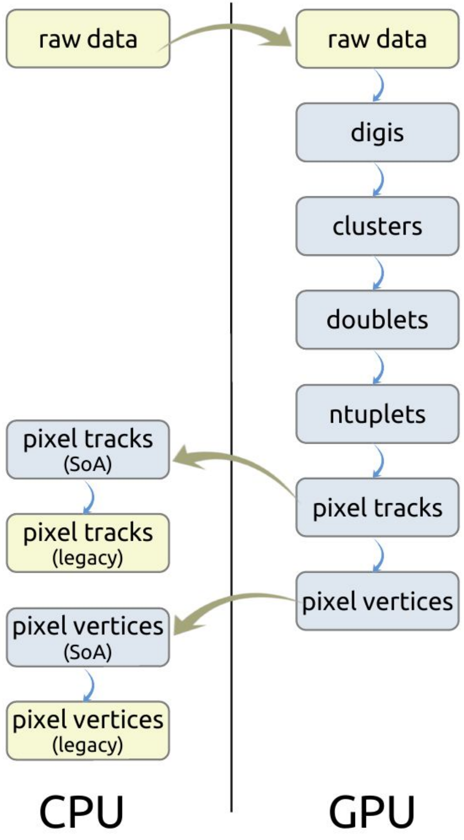
3 February 2020

Introduction

- The overall approach
  - Reconstruct pixel-based tracks and vertices on the GPU
  - Leverage existing support in CMSSW for threads and on-demand reconstruction
    - Also explore adding support for heterogeneous computing into the framework
  - Minimize data transfer
- An earlier step was actually a standalone program for the “hit quadruplets”
  - Fully utilizing CPU and GPU, encouraging performance
  - First developed on GPU, then ported to CPU
    - CPU version became the pixel quadruplet/triplet seeding algorithm since 2017 pixel detector upgrade in both HLT and offline reconstruction
- Results were shown in CHEP 2019
  https://indico.cern.ch/event/773049/timetable/?view=standard#76-heterogeneous-online-recons
- Some material also from Connecting the Dots 2019
  https://indico.cern.ch/event/742793/timetable/?view=standard#93-patatrack-accelerated-pixel

The full workflow

- Copy the raw data to the GPU (~250 kB/event)
- Run multiple kernels (39) to perform the various steps
  - Decode the raw data
  - Cluster the pixel hits
  - Form hit doublets
  - Form hit ntuplets (triplets/quadruplets) with a Cellular Automaton algorithm
  - Clean up duplicates
  - Vertexing
- Copy only the final results back to the host (optimized SoA format)
  - ~4 MB/event for tracks, ~90 kB/event for vertices
  - Convert to legacy format if requested



Top 5 kernels

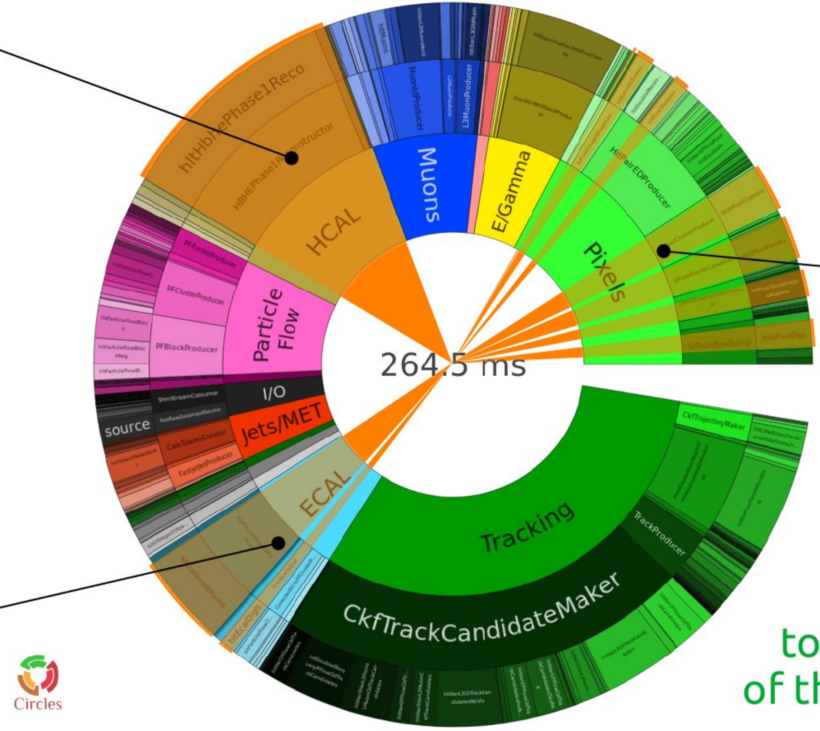
- On a Tesla T4, one CPU thread, one concurrent event
  - 4000 data events with high-pT jets, average time per event, quadruplets only
- 220 µs: `kernel_find_ntuplets()` [ntuplets]
  - Identify ntuplets from the CA connection graph
- 170 µs: `getDoubletsFromHisto()` [doublets]
  - Creates doublets from compatible hits in adjacent layers
- 130 µs: `findClus()` [clusters]
  - Produces clusters on each pixel module
- 120 µs: `kernelBLFit<4>()` [pixel tracks]
  - Fit 4-hit tracks with General Broken Lines algorithm
- 120 µs: `kernel_connect()` [ntuplets]
  - Connect hit doublets (create “CA connection graph”)

In the big picture (HLT)

HCAL: local reconstruction and calibrations

see Monday's talk in Track 9
"High Performance Computing for High Luminosity LHC"

ECAL: local reconstruction and calibrations

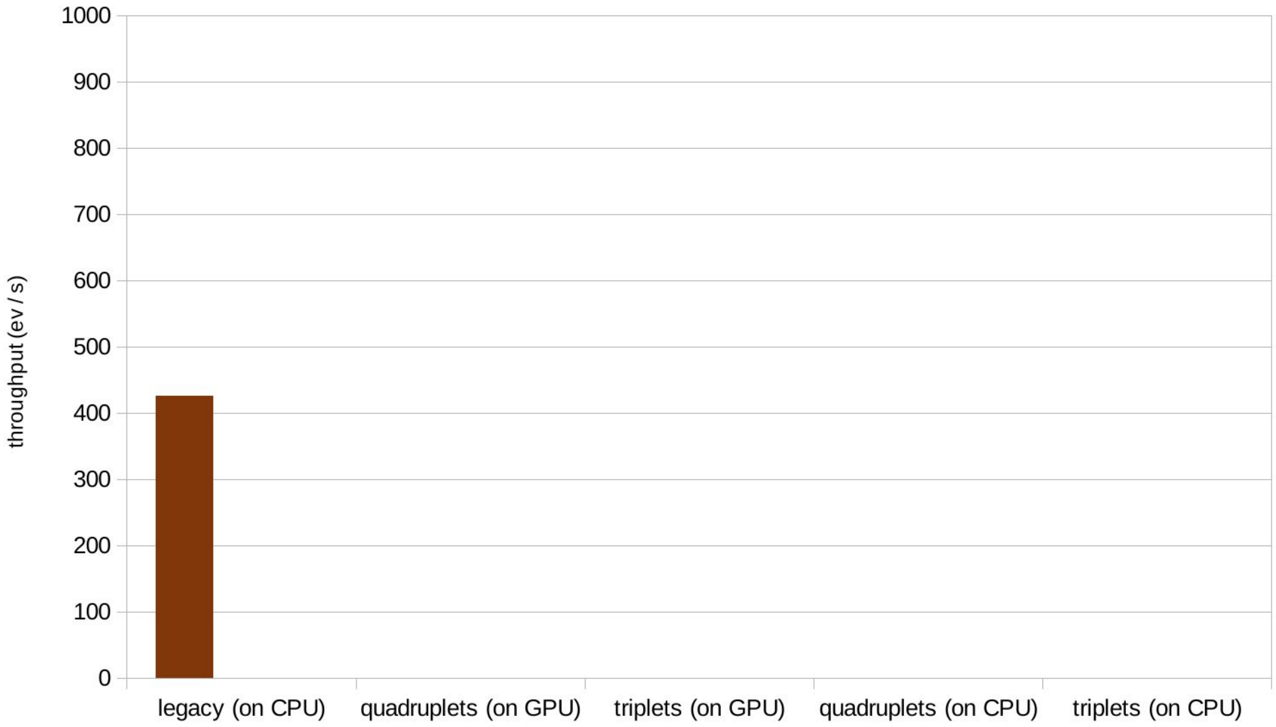


pixel tracking:
global reconstruction
details on the next slides

~10 % of full HLT
~ 5 % of offline reco

today we can offload ~24%
of the online reconstruction !

Performance (legacy)

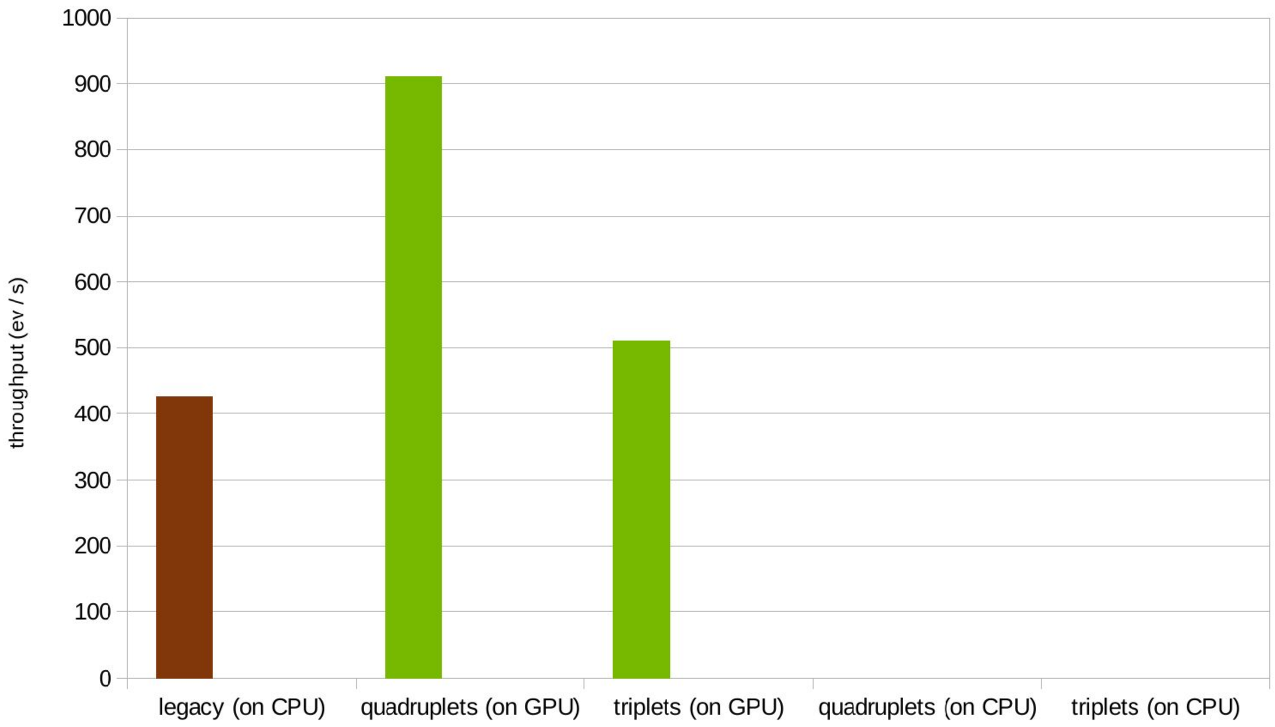


pixel tracks and vertices global reco

CPU

- dual socket Xeon Gold 6130
- 2 × 16 cores (2 x 32 threads)
- throughput measured on a full node
- 4 jobs with 16 threads

Performance (GPU, no output on CPU)



pixel tracks and vertices global reco

CPU

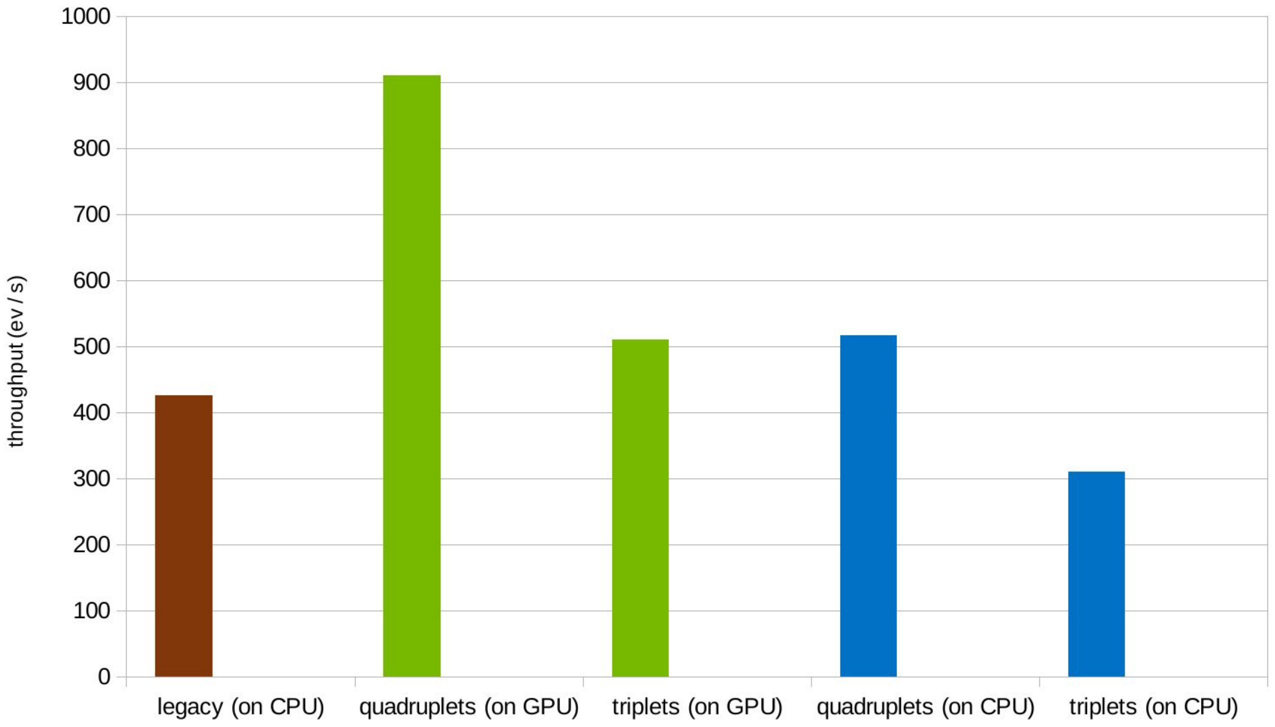
- dual socket Xeon Gold 6130
- 2 × 16 cores (2 x 32 threads)
- throughput measured on a full node
- 4 jobs with 16 threads

GPU

- single NVIDIA Tesla T4
- 2560 CUDA cores
- single job with 10-16 concurrent events

Fermilab

Performance (backported algorithms)



“Legacy” produces only quadruplets, and has lower efficiency

pixel tracks and vertices global reco

CPU

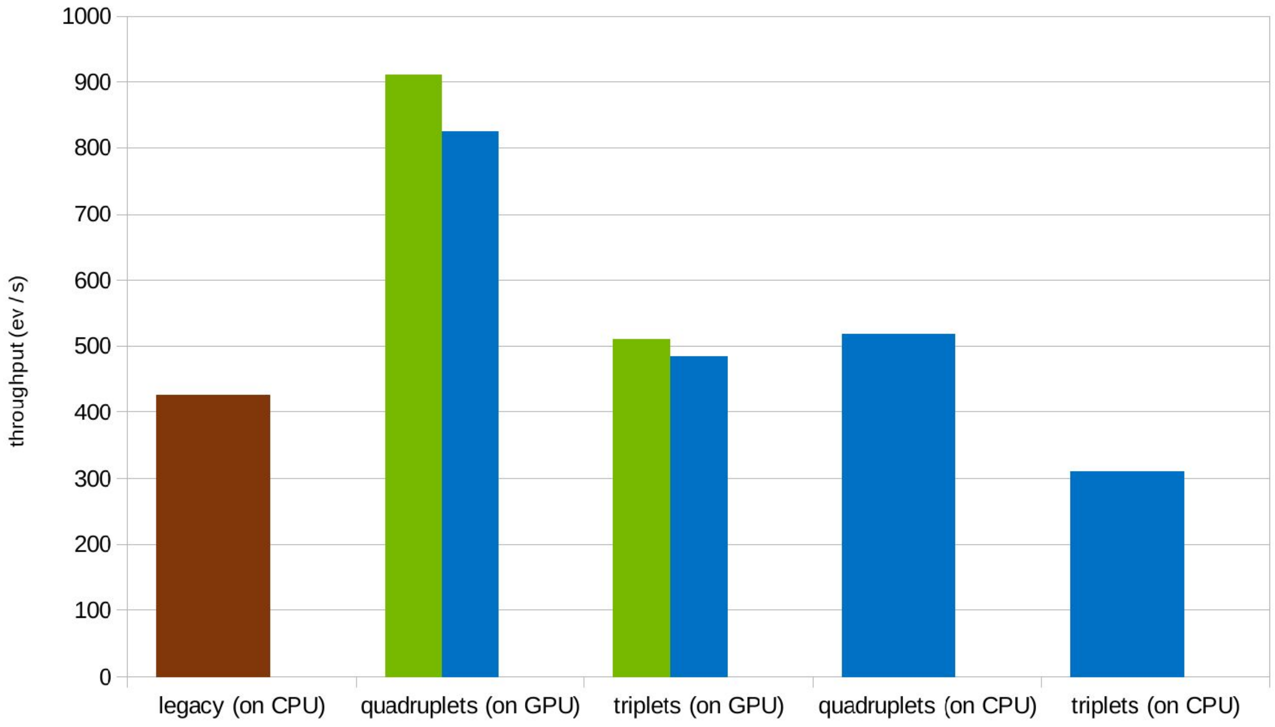
- dual socket Xeon Gold 6130
- 2 × 16 cores (2 x 32 threads)
- throughput measured on a full node
- 4 jobs with 16 threads

GPU

- single NVIDIA Tesla T4
- 2560 CUDA cores
- single job with 10-16 concurrent events

Part of the GPU algorithms were backported to CPU with an ad-hoc “CUDA compatibility” layer.

Performance (transfer output to CPU)



pixel tracks and vertices global reco

CPU

- dual socket Xeon Gold 6130
- 2 × 16 cores (2 x 32 threads)
- throughput measured on a full node
- 4 jobs with 16 threads

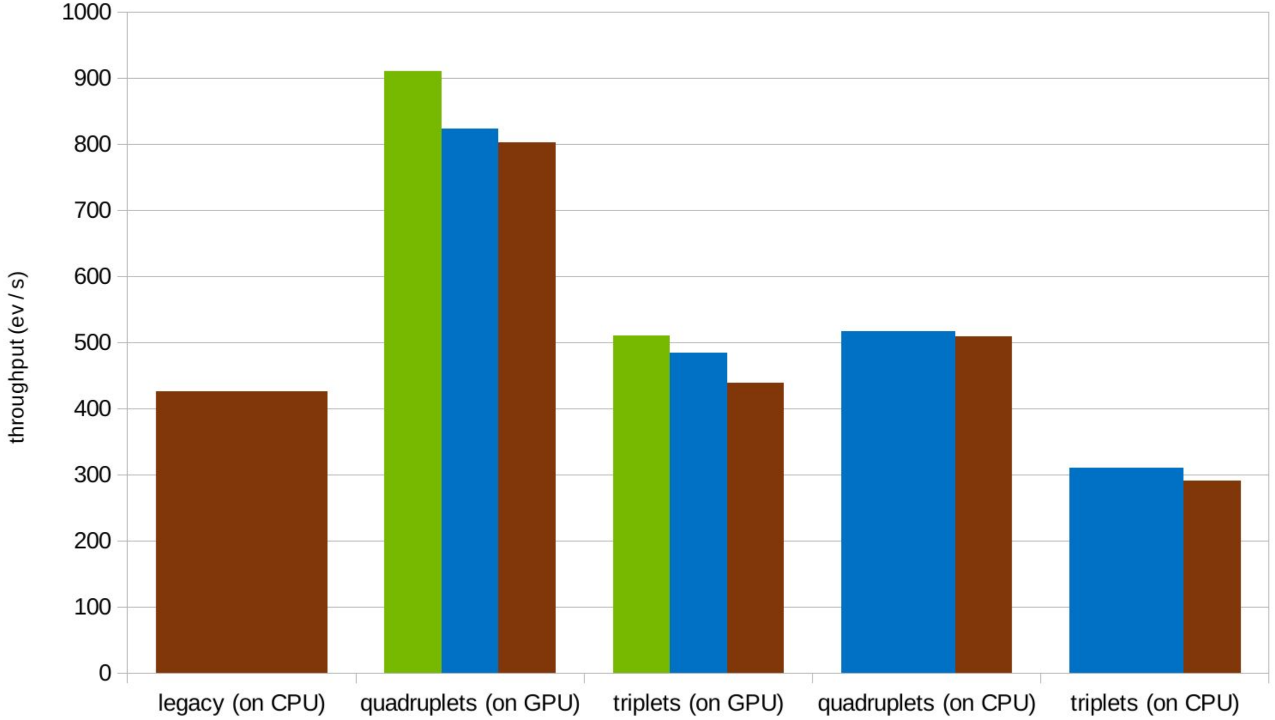
GPU

- single NVIDIA Tesla T4
- 2560 CUDA cores
- single job with 10-16 concurrent events

transfer from GPU to CPU

- on demand
- small impact on event throughput

Performance (convert SoA to legacy)



pixel tracks and vertices global reco

CPU

- dual socket Xeon Gold 6130
- 2 × 16 cores (2 x 32 threads)
- throughput measured on a full node
- 4 jobs with 16 threads

GPU

- single NVIDIA Tesla T4
- 2560 CUDA cores
- single job with 10-16 concurrent events

transfer from GPU to CPU

- on demand
- small impact on event throughput

conversion to legacy data formats

- on demand, to be minimised
- small impact on event throughput
- high cost in CPU usage

Technical implementation

- CUDA streams are used to process multiple events concurrently
  - Data from one event not enough to saturate a GPU
  - One CUDA stream / branch in the module data dependence DAG / concurrent event
- Aim is to utilize both CPU and GPU
  - There are no calls to cuda*Synchronize()
    - Except one which is mostly a sanity check, we are thinking ways to remove it
  - Instead use callback functions to notify the CMSSW framework when CPU work that waits for GPU work to finish can proceed
    - Allows CPU threads to do other work in the meantime
- Device and pinned-host memory allocations made through a memory pool
  - Currently based on cub::CachingDeviceAllocator
  - Most flexible towards minimizing device memory usage compared to alternatives
- Supports multiple GPUs

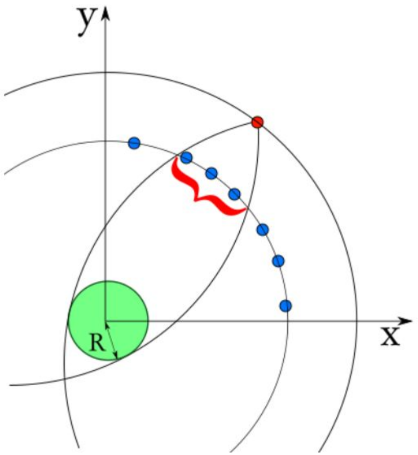
Standalone program

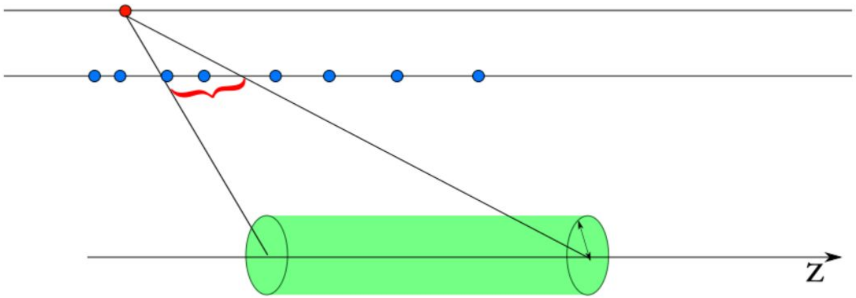
- The standalone program in https://github.com/makortel/pixel-standalone is essentially a mini-app of the first kernel in digis step intended to explore portability technologies
  - Contains data from one event
- Currently has implementations for
  - CPU (naive)
  - CUDA
  - Kokkos
  - Alpaka (both directly and through CUPLA that provides more CUDA-like interface)
  - Data Parallel C++ (oneAPI)

Backup

Doublets

- The local reconstruction produces hits
- Doublets are created opening a window depending on the tracking region/beamspot and layer-pair
  - The cluster size along the beamline can be required to exceed a minimum value for barrel hits connecting to an endcap layer
- Hits within the bins are connected to form doublets if they pass further "alignment cuts" based on their actual position
- In the barrel the compatibility of the cluster size along the beamline between the two hits can be required
- The cuts above reduce the number of doublets by an order of magnitude and the combinatorics by a factor 50



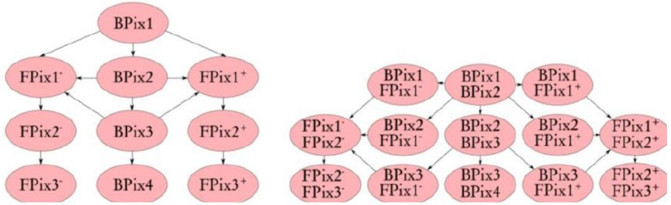


12

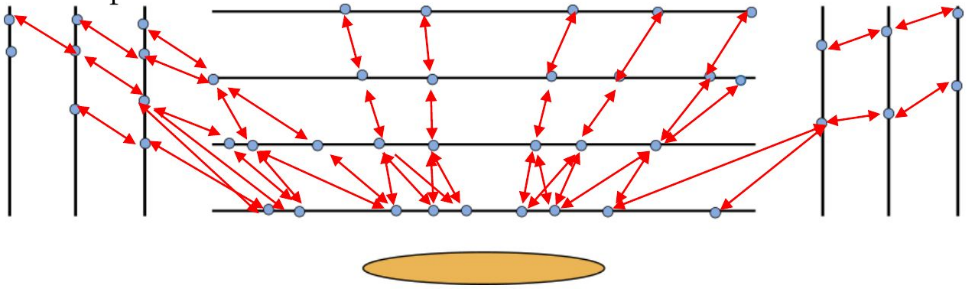
Cellular Automaton -based Hit Chain Maker

The CA is a track seeding algorithm designed for parallel architectures

It requires a list of layers and their pairings



- A graph of all the possible connections between layers is created
- Doublets aka Cells are created for each pair of layers, in parallel at the same time
- Fast computation of the compatibility between two connected cells, in parallel
- No knowledge of the world outside adjacent neighboring cells required, making it easy to parallelize



- Better efficiency and fake rejection wrt previous algo
- Since 2017 data-taking has become the default track seeding algorithm for all the pixel-seeded online and offline iterations

- In the following, at least four hits are required, but triplets can be kept to recover efficiency where geometric acceptance lacks one hit

13

Fermilab

CA compatibility cuts

- The compatibility between two cells is checked only if they share one hit
  - AB and BC share hit B
- In the R-z plane a requirement is alignment of the two cells
- In the cross plane the compatibility with the beamspot region