Parallelized and Vectorized Tracking Using Kalman Filter with CMS Detector Geometry and Events

G. Cerati[4], P. Elmer[3], M. Kortelainen[4], S. Krutelyov[1], S. Lantz[2], M. Lefebvre[3], M. Masciovecchio[1], K. McDermott[2], B. Norris[5], D. Riley[2], M.Tadel[1], P.Wittich[2], F.Würthwein[1], A.Yagil[1]

1. UCSD 2. Cornell 3. Princeton 4. FNAL 5. U of Oregon

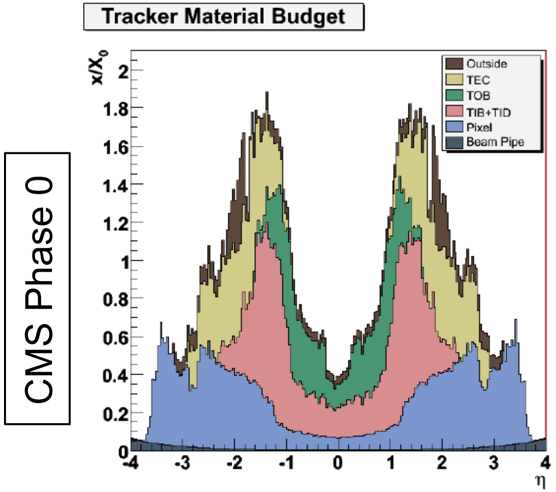
CHEP 2018, Sofia, Bulgaria

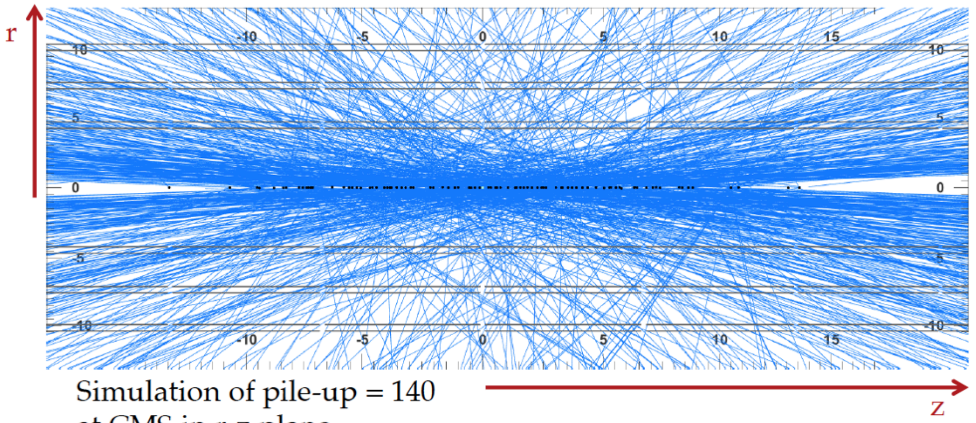
Outline

- Project introduction
  - Motivation for many-core Kalman filter implementation
- Some project details
  - Geometries, event data
  - Vectorization & Multi-threading
  - Architectures & Compilers
- Current focus & Status
  - Physics performance, scaling
- Conclusion

Project overview

- Cornell, Princeton, UC San Diego + Fermilab (all CMS).
  - 3-year NSF grant, now in extension year + CMS R&D project – focus on algorithm development
  - Fermilab and University of Oregon: 3 year DOE SciDAC4 grant (started January 2018) – focus on optimization
- Mission statement: Explore Kalman filter based track finding and track fitting on many-core SIMD and SIMT architectures:
  - KF performance well understood, handles multiple-scattering and energy loss well (badly needed)
  - complementary to tracklet-based divide and conquer algorithms
- Goal: Run in CMS HLT for Run3 and beyond; maybe also parts of offline reconstruction





Simulation of pile-up = 140
at CMS in r-z plane

Project details – What we do and How

Code name: mkFit – Matriplex Kalman Fitter / Finder

One slide status report

- Current focus: Track finding on CMS-2017 geometry, Iteration 0 tracking
  - KNL / Xeon ➛ AVX-512
  - Iteration 0 = Starting from pixel seeds having 4 hits with beam spot constraint
  - Using CMSSW generated events:
    - 10 muon events (for development), ttbar, ttbar + 35 or 70 PU
  - Stand-alone: use a simple event data format, basically a memory dump of our structures.
  - Within CMSSW – in progress, first results already available;
    - mkFit is deployed as external package + CMSSW module ➙ data producer

We can run track finding on full detector, iteration 0, physics performance comparable to CMSSW.

- Things we have also done:
  - Extensive validation suite.
  - Track fitting (forward / backward) – this was initial task and a great success.
    - Will probably return to this to explore also mkFit-based track post-processing.
  - Seed finding – abandoned, we use CMSSW seeds.
  - Development on GPUs (CUDA) is proceeding in parallel. Currently doing in-depth investigation of actually achievable peak performance for fitting and finding (memory/cache bw/ limitations).

Geometry description & approximation

Unlike CMSSW, we **DO NOT** deal with detector modules! We use **layers only**:

- Propagate to the center of a layer and perform hit pre-selection.
- Requires additional propagation step for every compatible hit!
  - But this really vectorizes well. [And we do not have to propagate to a module.]
- Stereo: mono / stereo modules are put into separate layers.
- Can *only* pick up one hit per layer on outward propagation.
  - Could pickup overlap hits during backward fit, or after, for layers where it matters.
- **Simplifies track steering code and minimizes candidate specific code.**

See extras

Geometry is implemented as a plugin! mkFit is **NOT** CMS specific.

Multi-threading, Vectorization, Architectures & compilers

For multi-threading we use TBB:

- Two parallel_fors over tracking regions (5) and seeds (16 or 32 seeds per task)
- parallel_for over events - multiple events in flight
  - This is crucial for plugging the gaps arising from unequal load in track finding tasks!

Vectorization:

- Propagation, simple loops – compiler assisted with pragma simd
- Kalman Filter operations – *Matriplex*, developed as part of the project

See extras

Architectures & compilers:

- x86_64 (AVX, AVX-512), KNC (MIC), KNL (AVX-512)
  - icc, gcc; we use c++14
- Nvidia / CUDA
  - Have implementations of track fitting and track finding (best hit and cache optimized version)

Current focus & Status

What we are working on now

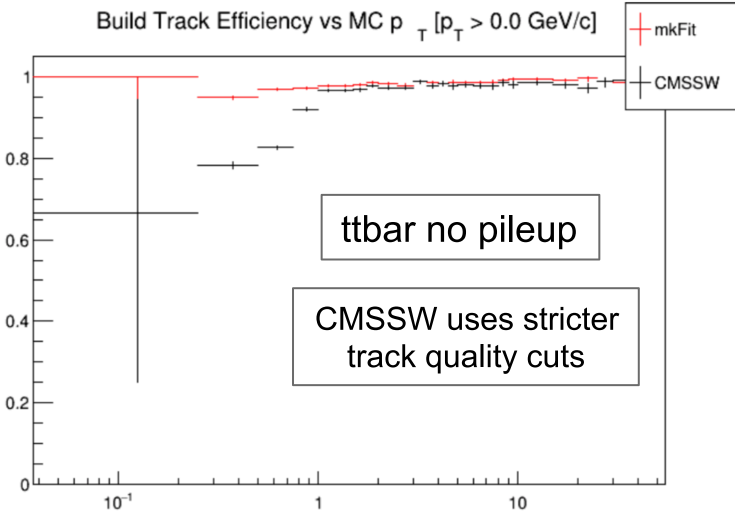
- Meaningful comparison of track finding with CMSSW for Iteration 0
  - Physics performance – almost there:
    - Polishing the edges, tuning of track finding parameters
    - Use cluster charge information to remove hits due to out of time pileup
    - Still need to implement cleaning / merging of resulting tracks
      - While we do seed cleaning, we get duplicates & ghosts, especially in the endcaps where there are a lot of module overlaps within layers.
  - Computational performance, i.e. speed, scaling, and memory footprint
    - x86_64 (Skylake Silver vs. Gold), KNL
- Finalization of CMSSW Integration
  - Consolidation of complete work-chain, including outlier rejection & final fitting
- Still have some ideas to further improve vectorization speedup and overall performance.

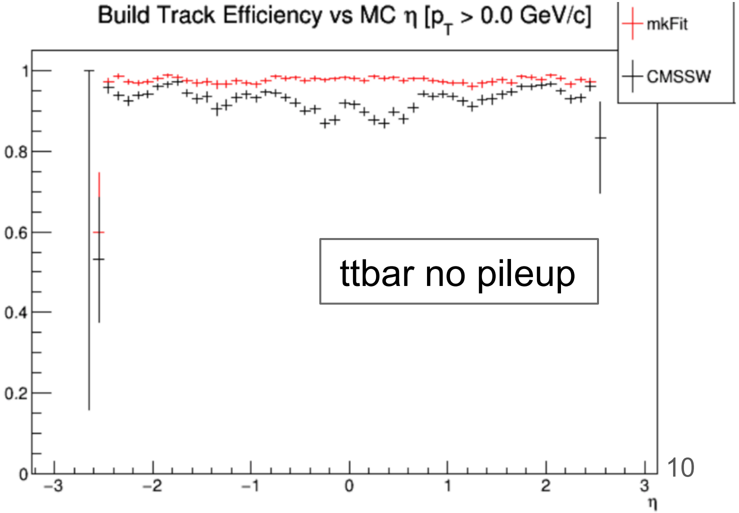
Muon gun & ttbar no pileup

- Efficiency denominator: findable sim-tracks with a matching seed.
  - Remember – this is iteration 0 / initial step using pixel quadruplets as seeds

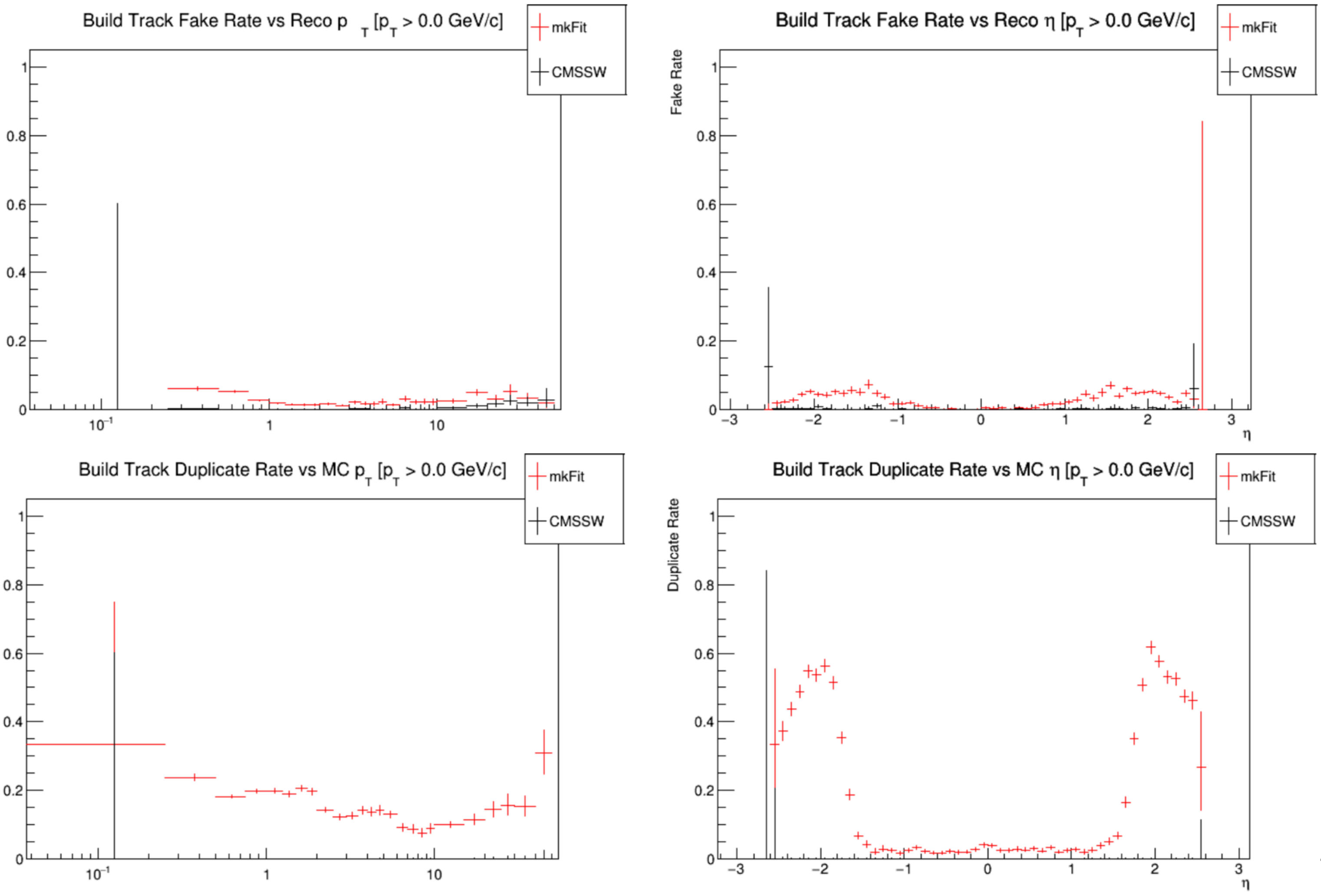
A. 10 mu per event, pt from 0.5 to 10 GeV
   - Practically fully efficient, zero fake rate
   - Duplicate rate spikes to ~50% in endcaps
     - Direct consequence of seed duplicates
     - Should go away once we implement cleaning and merging

B. ttbar no pileup - basically the same as 10 muon events
   - Some fakes in transition region (~5% eta 1.2 to 1.7)
     - Cleaning / merging can reduce this



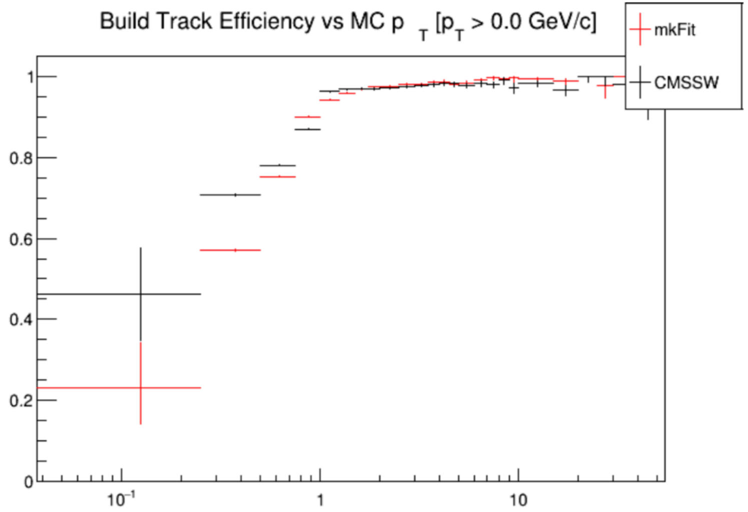


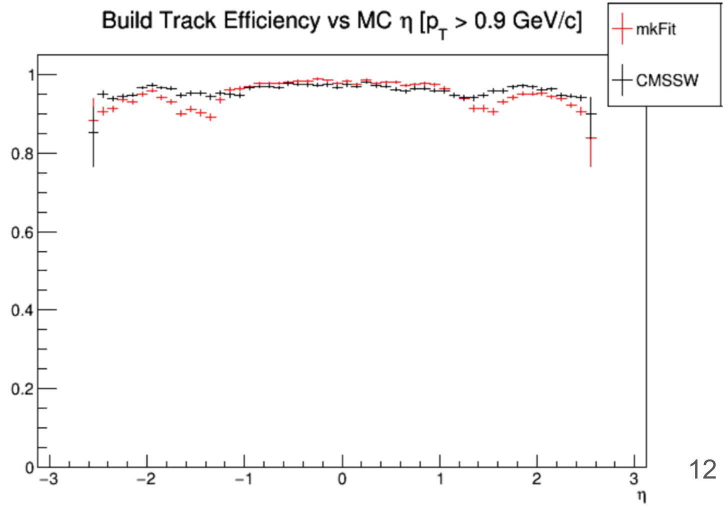
ttbar, no pileup



ttbar + 70 PU

- Efficiency comparable for pt > 0.5 GeV
  - Exploration of endcap inefficiency is ongoing
- Fake rate is more significant
  - Final cleaning should help
  - Investigate quality criteria
- Duplicate rate similar to no pileup / muon case
  - Which means it has the same origin – duplicates in input seed collection.
  - Post-build cleaning / merging will get this down to CMSSW levels

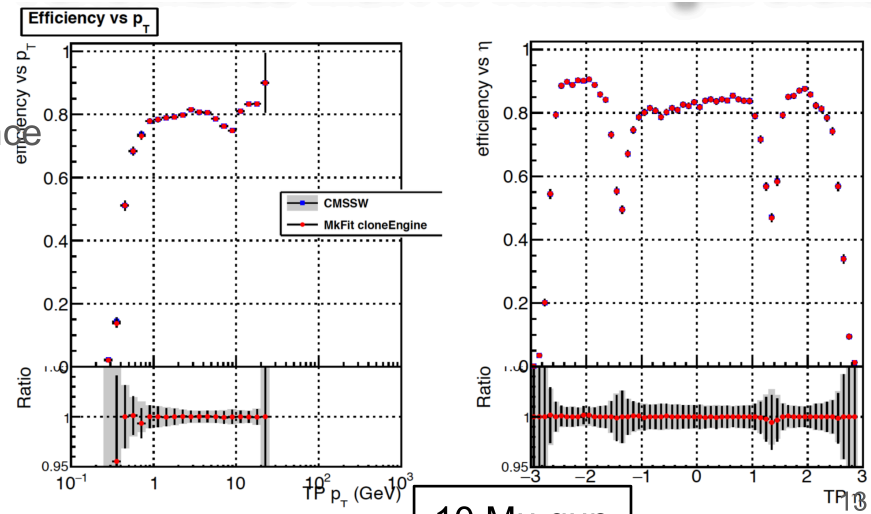




CMSSW Integration – Preliminary Results

- mkFit is wrapped in a standard CMS module / data producer:
  - compiled as an external library
  - tracker hits and seeds as input – convert them to format expected by mkFit
  - produces standard Track collection as input
- Running in CMSSW gives us access to standard CMS validation tools.

  Denominator: simulated tracks
  (physics efficiency
  - inefficiencies dominated by tracker acceptance (Iter 0 tracking requires 4 out of 4 pixel lyrs)
  - 10 Mu gun – perfect match
- Some small issues still to be resolved.
- Ready for detailed validation & performance optimizations.

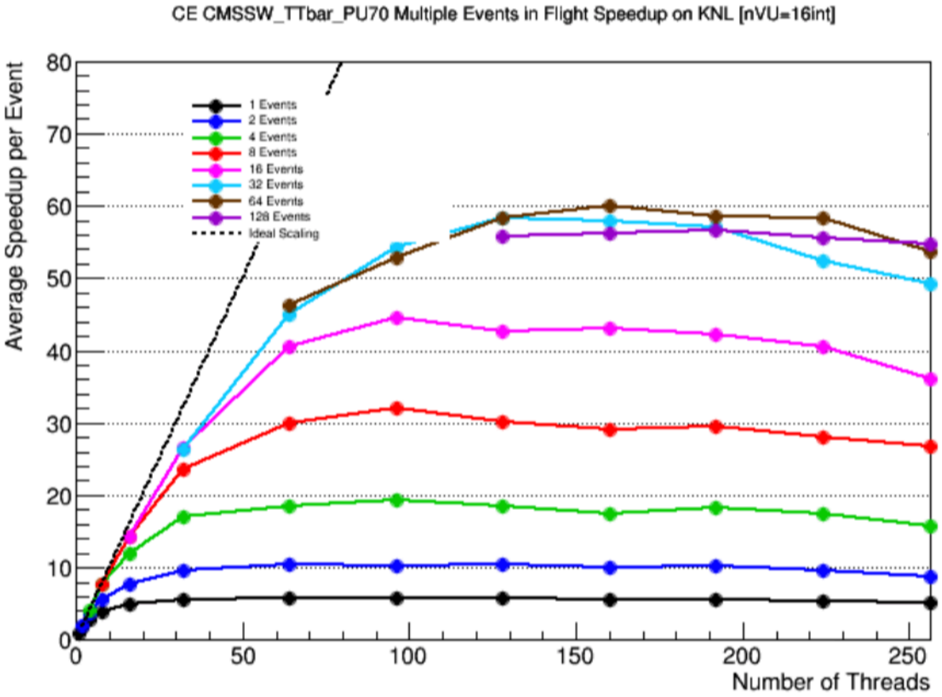


10 Mu gun

Computational performance

- Vectorization (building only) gives about 2 to 3x speedup (AVX, AVX-512)
- For multi-threading, having multiple events in flight is crucial!
  - Currently cleaning up “administrative” tasks we didn’t care much about before, e.g., loading of hits, seed cleaning.
- Compared to CMSSW, mkFit is about 10x faster (both single-thread).
  - Intentionally vague as this is work in progress.
  - icc significantly boosts mkFit performance
- ttbar + 70 PU   @ KNL: 115 events / s
  @ Skylake Au (32 core): 250 events / s

KNL



Conclusion

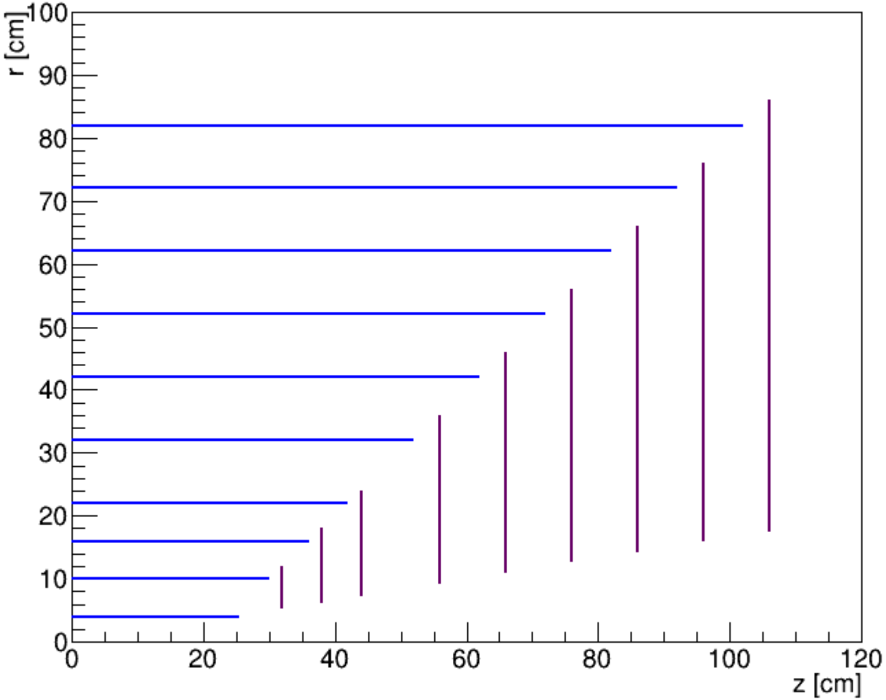
Conclusion

- mkFit is basically ready to be used in testing environment of CMS HLT
  - investigate efficiency discrepancies for low-pT / endcap tracks in high pileup data
  - implement post-build cleaning to reduce duplicate rate
  - improve scaling – optimization of code that was considered “out of scope” until now
- mkFit is approaching its first production release.
  - Opportunity to do some deep cleaning of the code.
- Code is in principle quite general … but mkFit is not a ready to use tracking package
  - We will continue to make efforts in that direction.

Extras – CMS geometry in mkFit

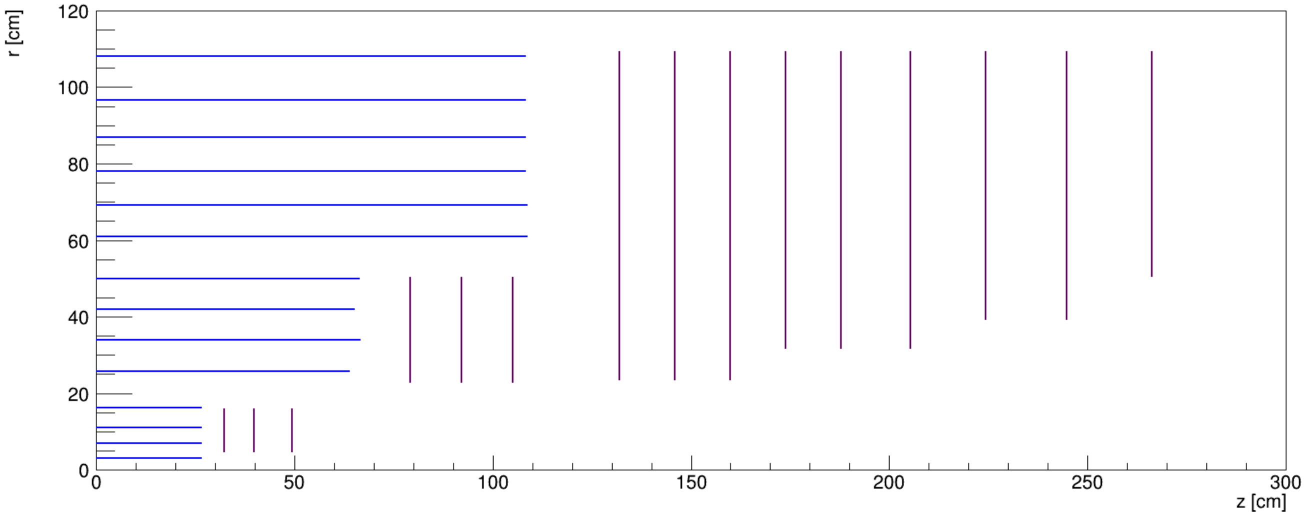
Cylindrical Cow with Lids

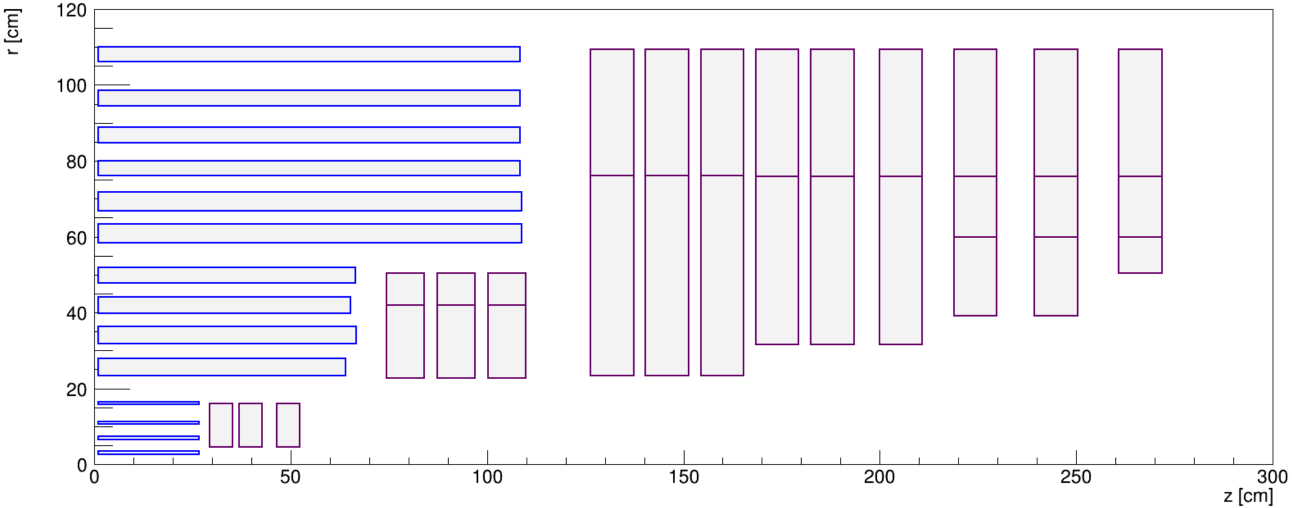
- Simple basic geometry
  - transition region |eta| 1 to 1.3
  - “long” pixels on all layers
- Supporting several geometries keeps tracking algorithms independent of actual geometry!
  - And points to required generalizations
- Geometries are implemented as a plugin / code that runs during program initialization and sets up geometry and algorithm steering structures.



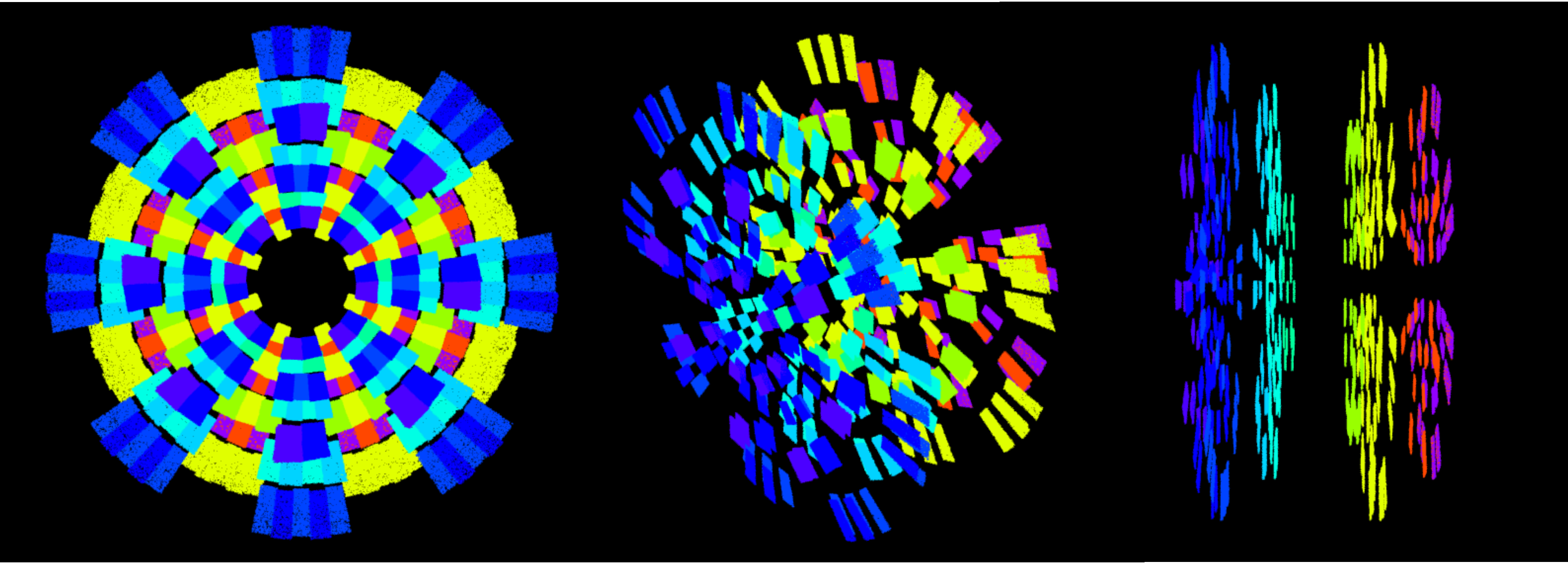
CMS-2017

- Top – what is usually shown.
  - Lines at layer centroids
- Bottom – actual size of layers accounted for.
  - Actual geometry used by mkFit.
  - Extracted automatically from CMS sim hit data.
  - Note: stripes on endcap disks are results of partial stereo layer coverage





CMS, example of an endcap disk



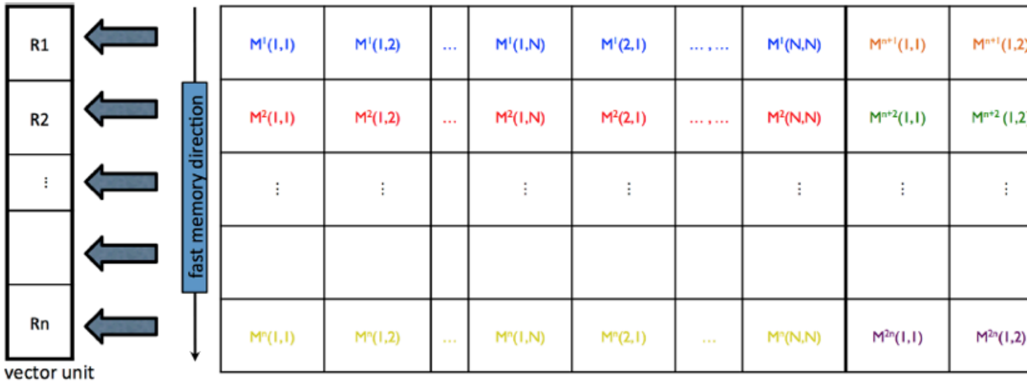
Extras – Matriplex

Matriplex - Vectorization of small matrix operations

"Matrix-major" matrix representation designed to fill a vector unit with **n** small matrices operated on in synch

Use vector-unit width on Xeons

- With or without intrinsics
- Shorter vector sizes w/o intrinsics
- For GPUs, use the same layout with very large vector width



Interface template common to Xeon and GPU versions

Matriplex - GenMul code generator

GenMul.pm - Generate matrix Multiplication code for given matrix dimensions
Features:

- Generate C++ code or Intrinsics (AVX, MIC, AVX-512)
    - Output is then included into a function.
    - For intrinsics it takes into account instruction latencies
- Can be told about known 0 and 1 elements in input and output matrices:
    - This reduces number of operations by more than 40%!
- Can do on-the-fly transpose of input matrices
    - Avoids transposition for similarity transformation.

We use this for vectorizing all Kalman filter related operations.

For propagation we rely on compiler vectorization (#pragma simd for the outer propagation loop over track candidates).