UC San Diego Cornell University PRINCETON UNIVERSITY Fermilab

Parallelized Kalman-Filter-Based Reconstruction of Particle Tracks on Many-Core Processors and GPUs, Part I

CMS Week: April 4, 2017

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

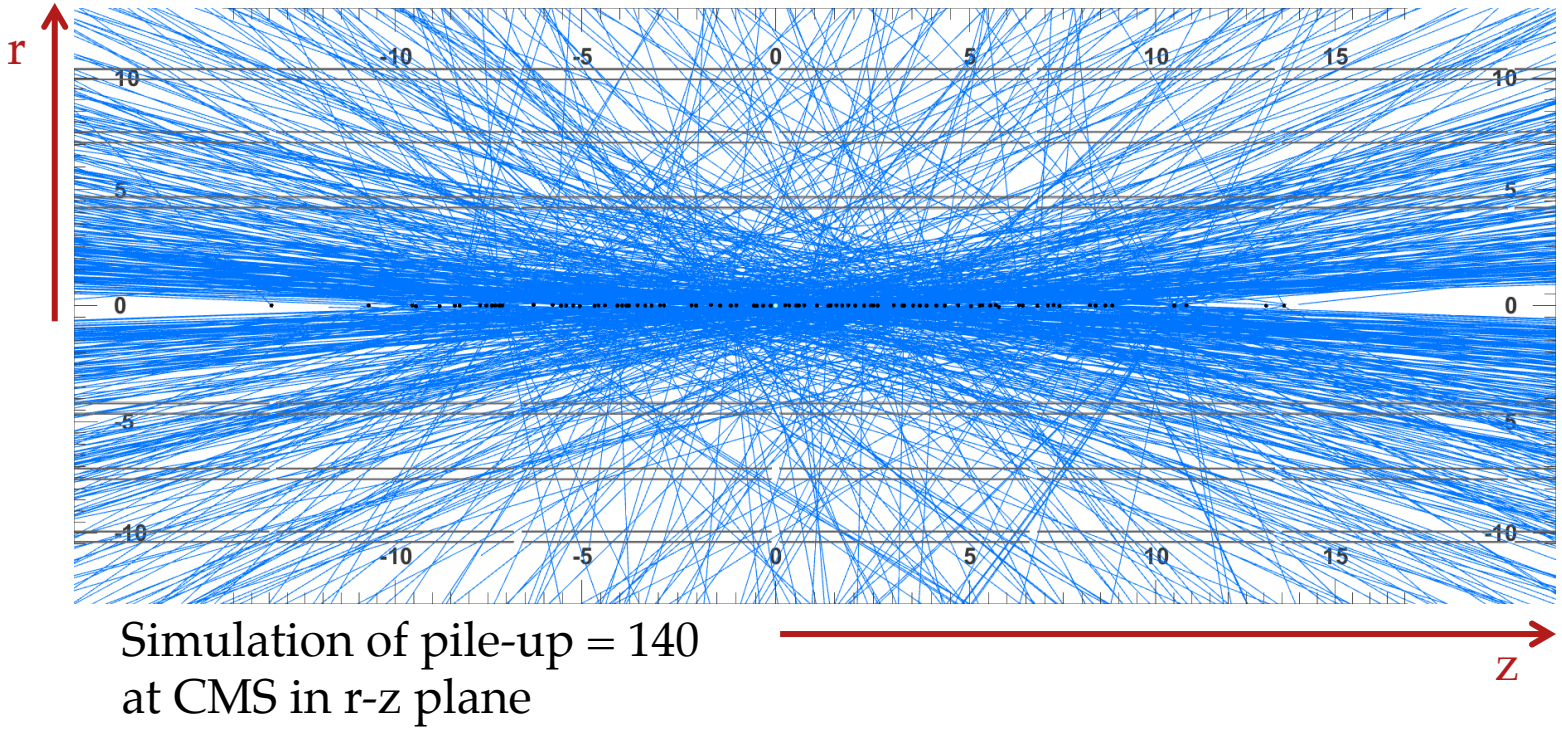
1. University of California – San Diego
2. Cornell University
3. Princeton University
4. Fermi National Accelerator Laboratory

Outline

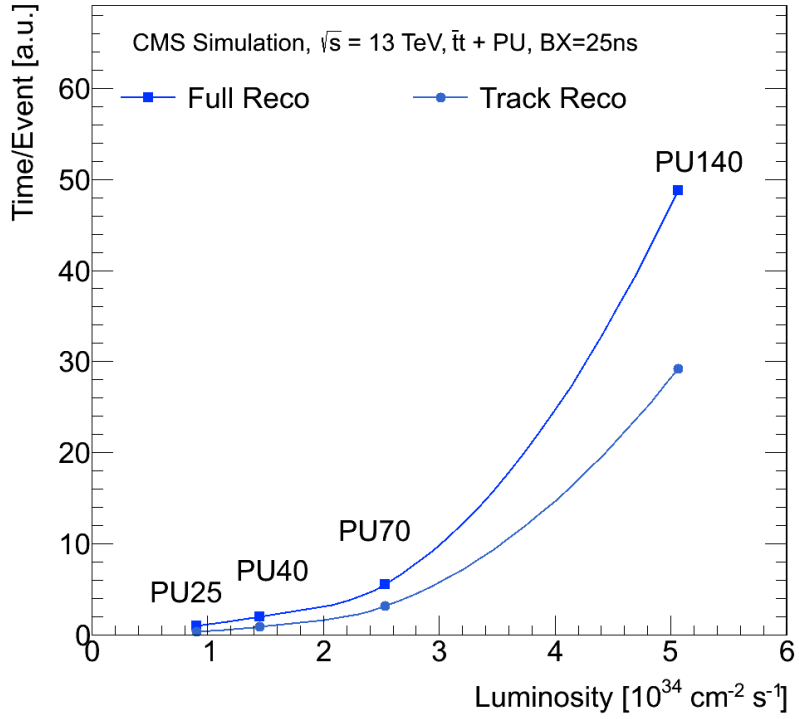
- Project motivation and goals
- Overview of Kalman Filter tracking
- Challenges and experimental setup
- Track fitting
- Track building
- Sneak peak of results so far and things to come

https://trackreco.github.io/

The Challenge of the HL-LHC



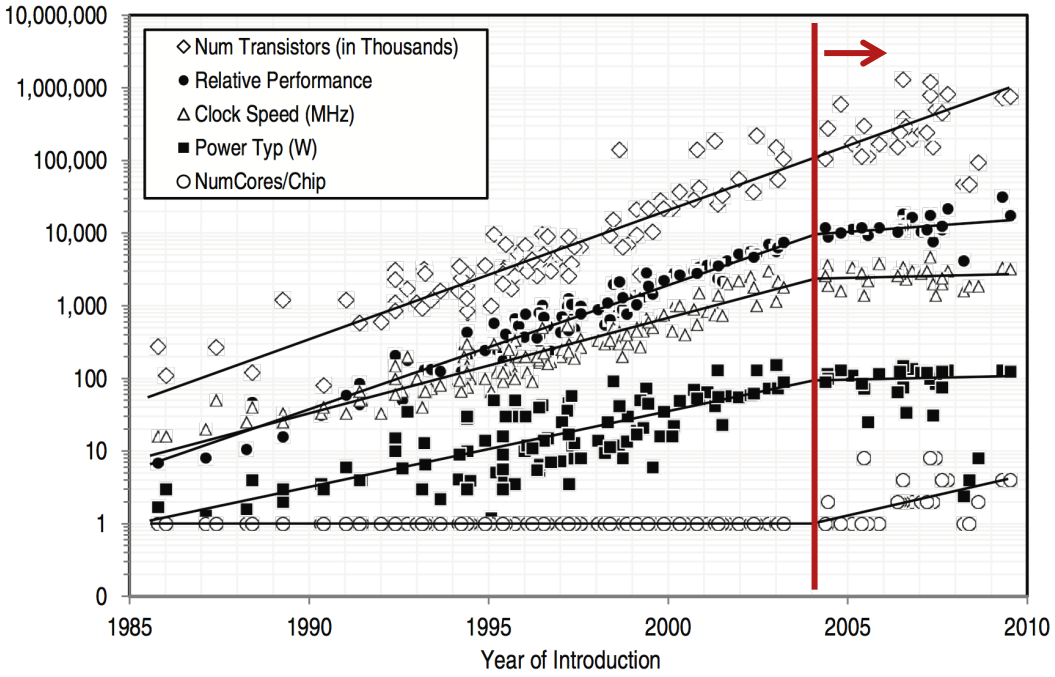
Simulation of pile-up = 140
at CMS in r-z plane



- HL-LHC poses serious challenges to reconstruction, and tracking will be hit the hardest
- Combinatorics from track finding will be a nightmare for efficiency, fake rate, and time

A way out: Moore's Law

- While relying on CPU clock speed scaling from Moore's is certainly over, Moore's law still applies to the number of transistors in many-core, parallel architectures
- However, the performance of serial applications, like CMS's iterative combinatorial Kalman Filter track finder, will not scale unless redesigned to be run in parallel



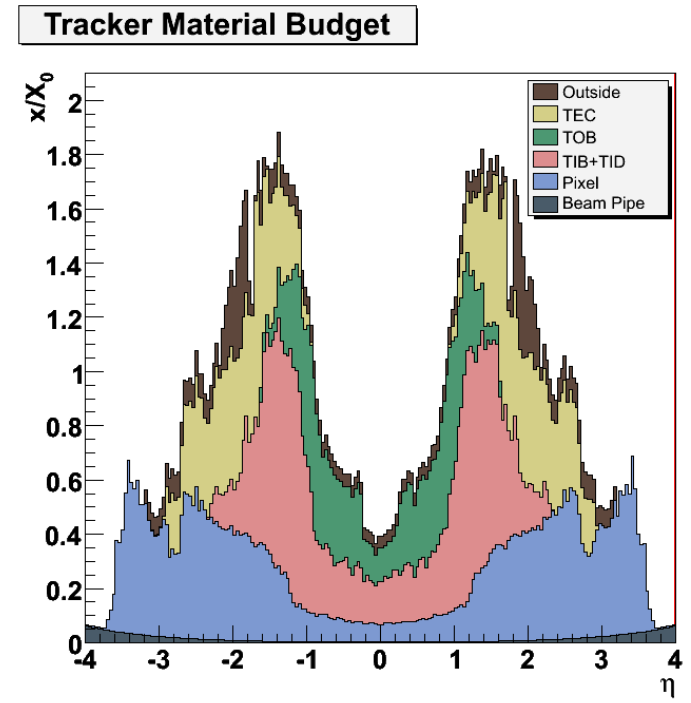
Therefore, we need to adapt the serial algorithms today to the parallel architectures of tomorrow

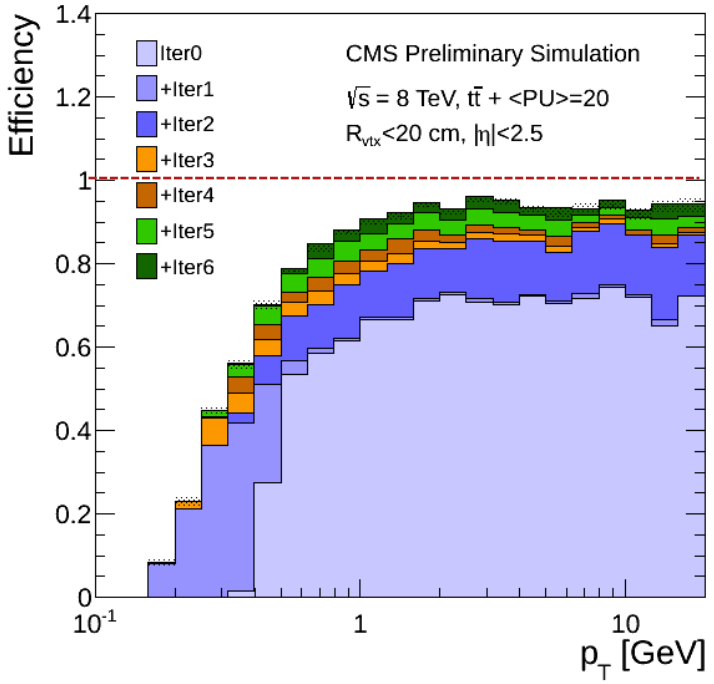
Why the Kalman Filter?

- Naïvely, a charged particle is described by a single helix
  - Only **locally** helical: distortion from multiple scattering, energy loss, etc.
  - KF provides robust treatment of material effects

- Given robustness and expertise already gained on KF tracking, we wish to **extend** knowledge to parallel architectures

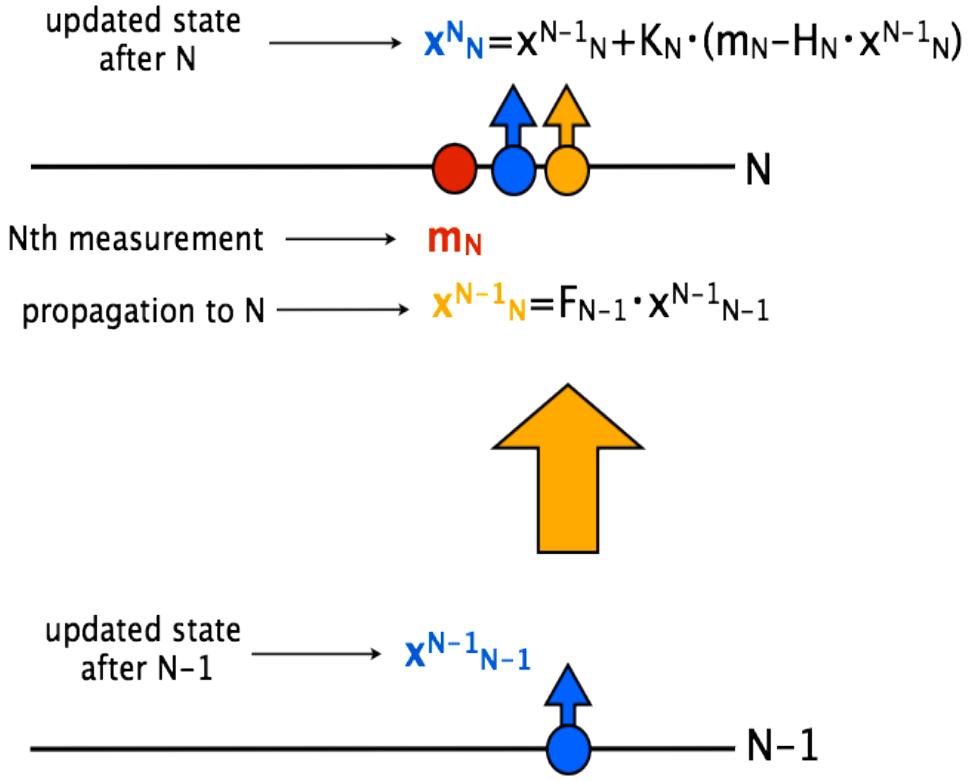
- Project seeks to achieve same high level of physics performance already seen with the LHC experiments while **significantly speeding up** time spent in tracking





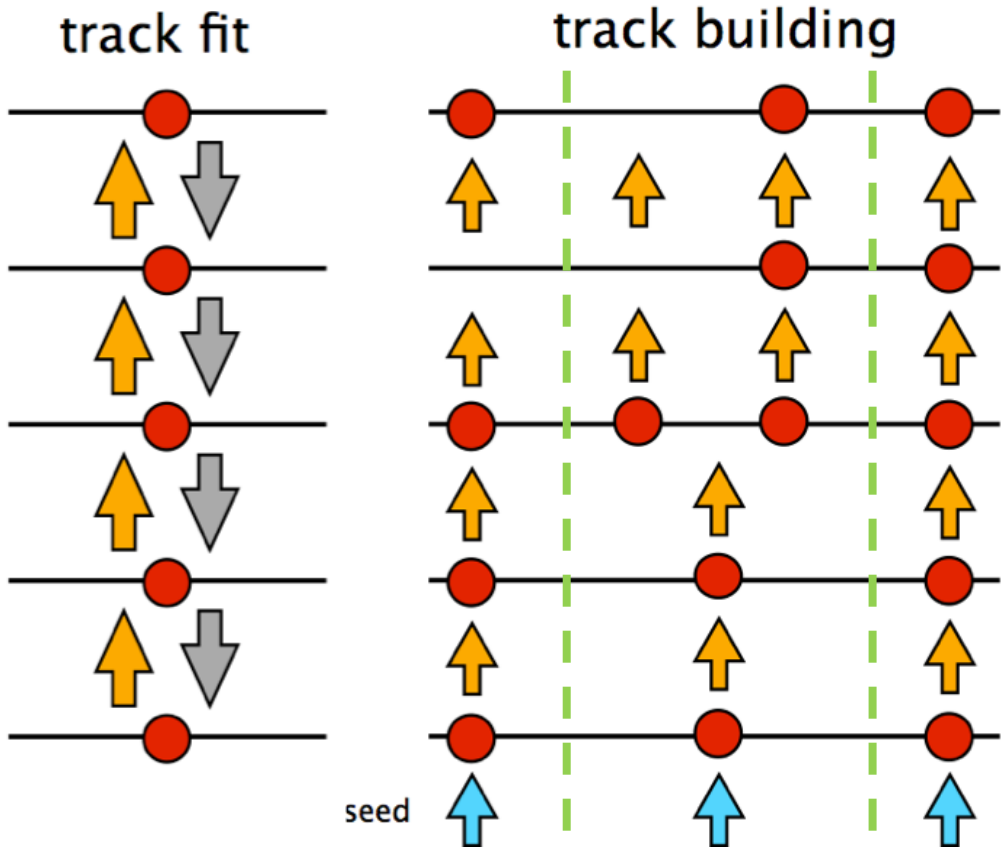
Kalman Filter Basics

- Kalman Filter can be seen as iterative repetition of the same logic unit
- Logic unit is the base of both **track fitting** and **track building**
  - Propagation and update of track state with hit measurements at a given layer
  - Calculations are set of **matrix operations** on small matrices and vectors



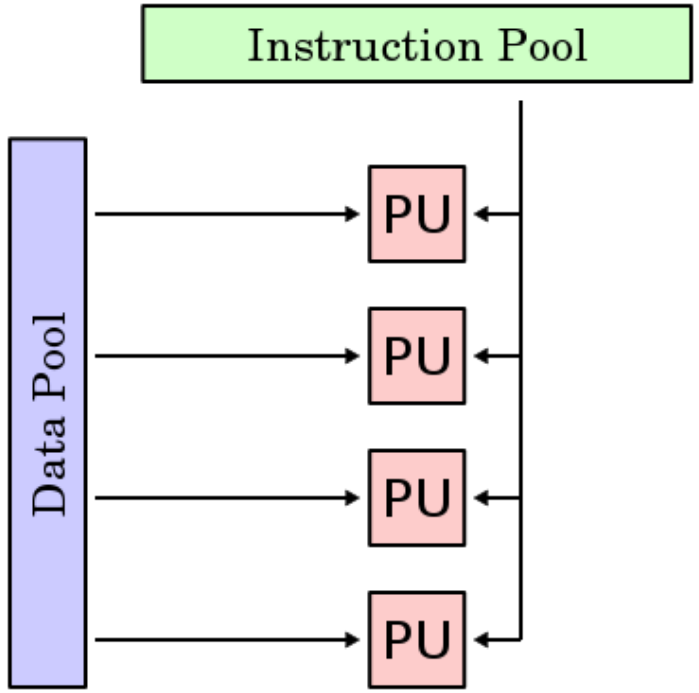
KF Track Reconstruction

- Tracking proceeds in three main steps: seeding, building, and fitting
- In fitting, hit collection is known: repeatedly apply the basic logic unit
- In building, hit collection is unknown and requires **branching** to explore many possible candidate hits after propagation



Challenges to Parallel Processing

- KF tracking cannot be ported in straightforward way to run in parallel
- Need to exploit two types of parallelism with parallel architectures
- **Vectorization**
  - Perform the same operation at the same time in lock-step across different data
  - **Challenge: branching** in track building - exploration of multiple track candidates per seed
- **Parallelization**
  - Perform different tasks at the same time on different pieces of data
  - **Challenge: thread balancing** – splitting the workload evenly is difficult as track occupancy in the detector not uniform on a per event basis

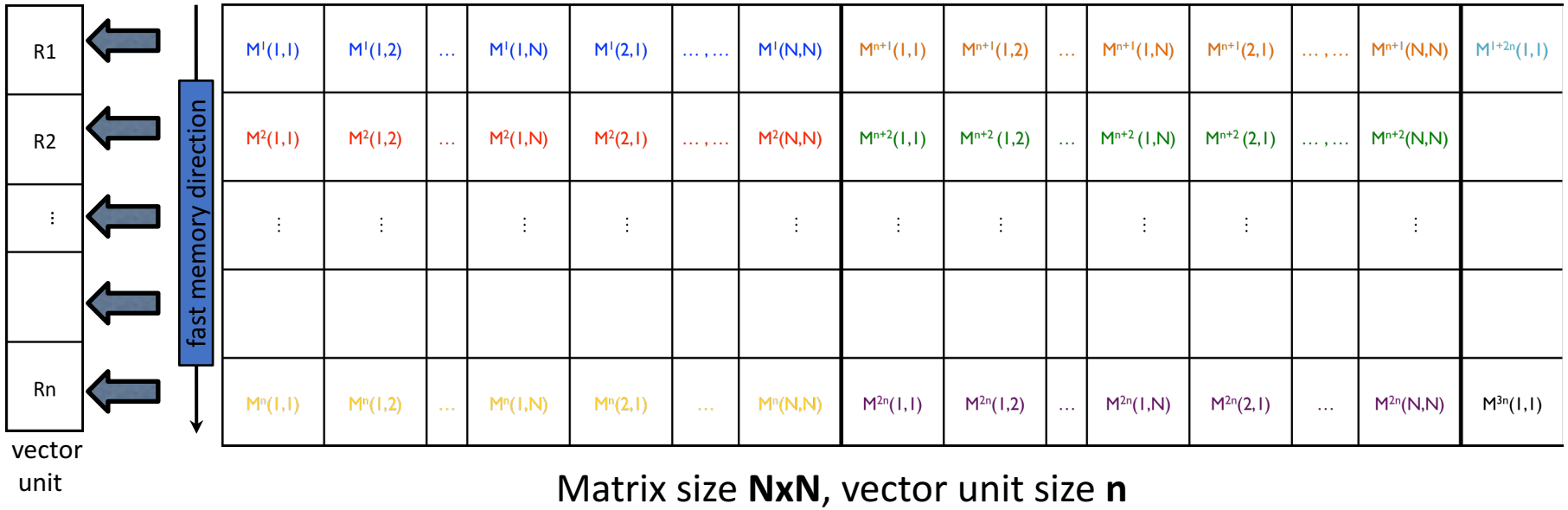


Vectorization

Matriplex

- Matrix operations of KF **ideal for vectorized processing:** however, requires **synchronization** of operations

- Arrange data in such a way that it can loaded into the vector units of SNB and KNC with ***Matriplex***
  - Fill vector units with the same matrix element from different matrices: **n matrices working in synch on same operation**



Matrix size **NxN**, vector unit size **n**

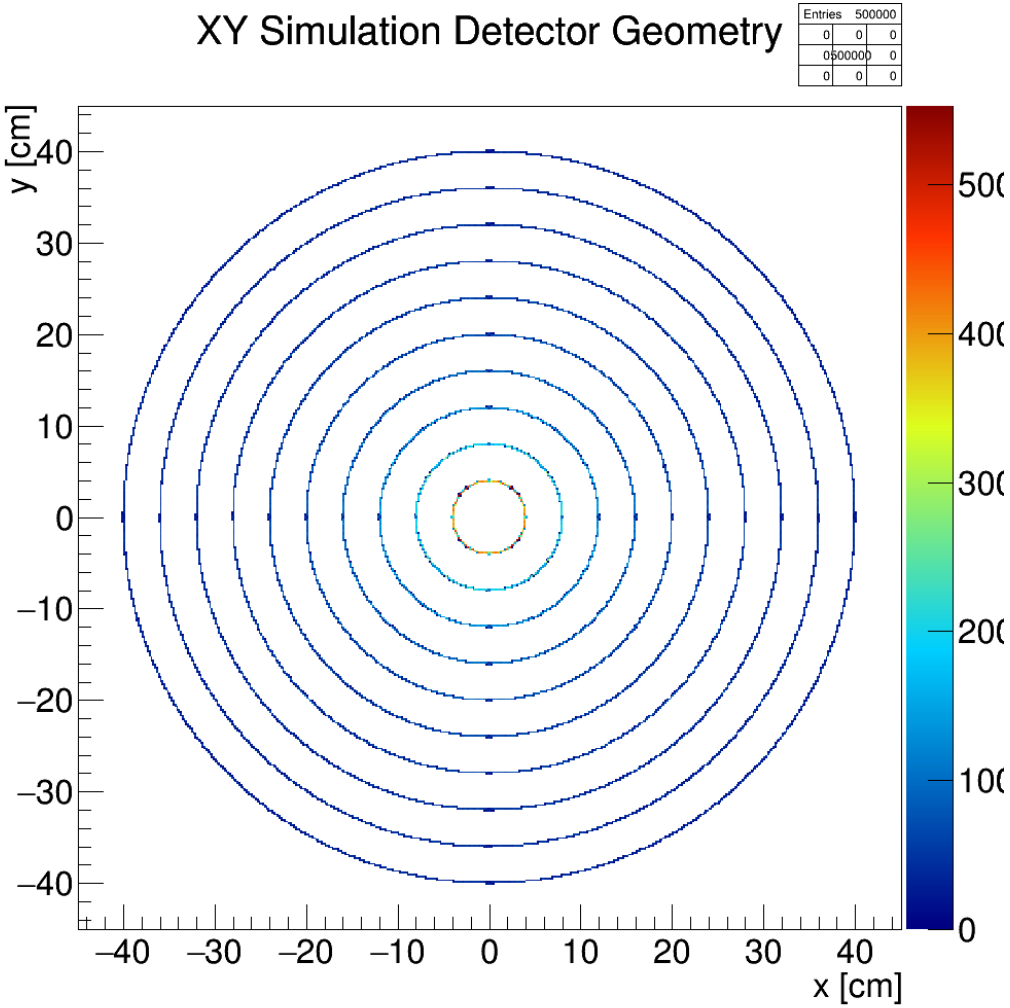
Selected Parallel Architectures

	Xeon E5-2620 Sandy Bridge (SNB)	Xeon Phi 7120P Knights Corner (KNC)	Xeon Phi 7230 Knights Landing (KNL)	Tesla K40
Logical Cores	6x2x2	61x4	64x4	2880 CUDA cores
Clock rate	2.5 GHz	1.24 GHz	1.3 GHz	875 MHz
GFLOPS	120	1208	2660	1430
SIMD width	256 bits	512 bits	2x512 bits	32 thread warp
Memory	~64-384 GB	16 GB	16 & 384 GB	12 GB
Bandwidth	42.6 GB/s	352 GB/s	475 & 90 GB/s	288 GB/s

Experimental Setup

Simplified setup

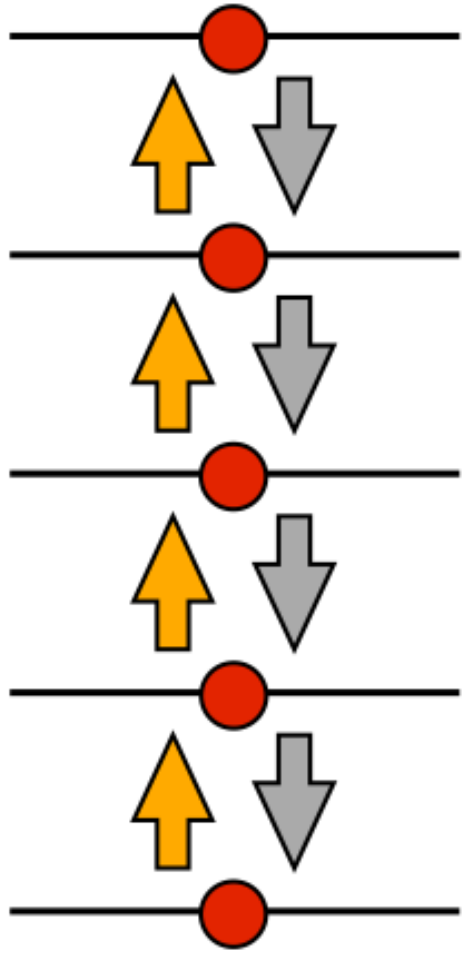
- Detector conditions
  - 10 barrel pixel layers, evenly spaced
  - Hit resolution in r-phi = 100µm, z = 1.0mm
  - Constant B-field of 3.8T
- Tracks uniformly generated (uncorrelated)
- One hit/layer, no misses
- No energy loss or scattering



Track Fitting

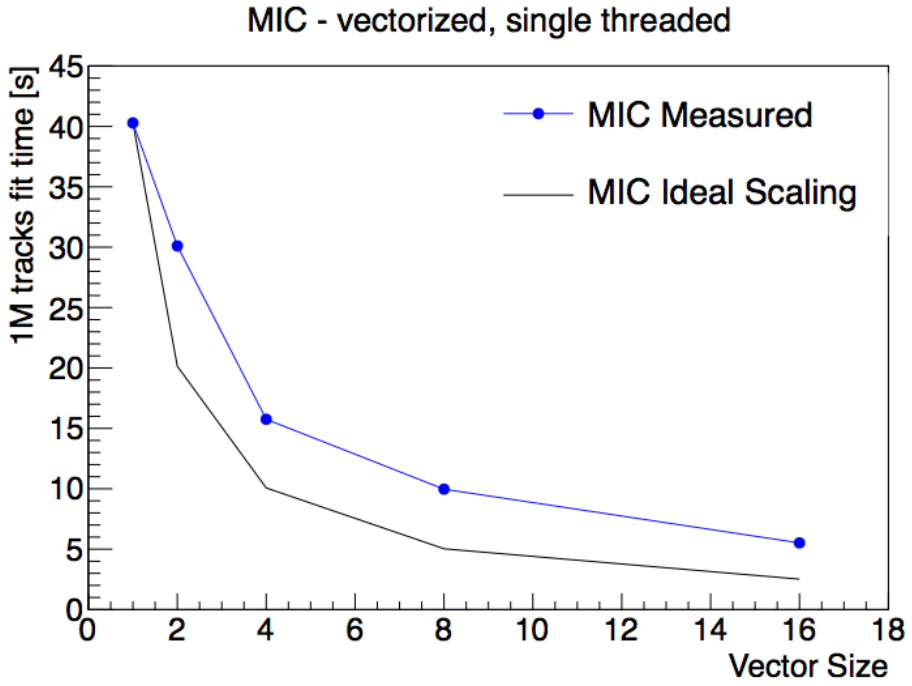
- Track fitting ideal place to start: **repetition** of propagation and update on pre-determined set of hits **without branching** ensures vectorized processing

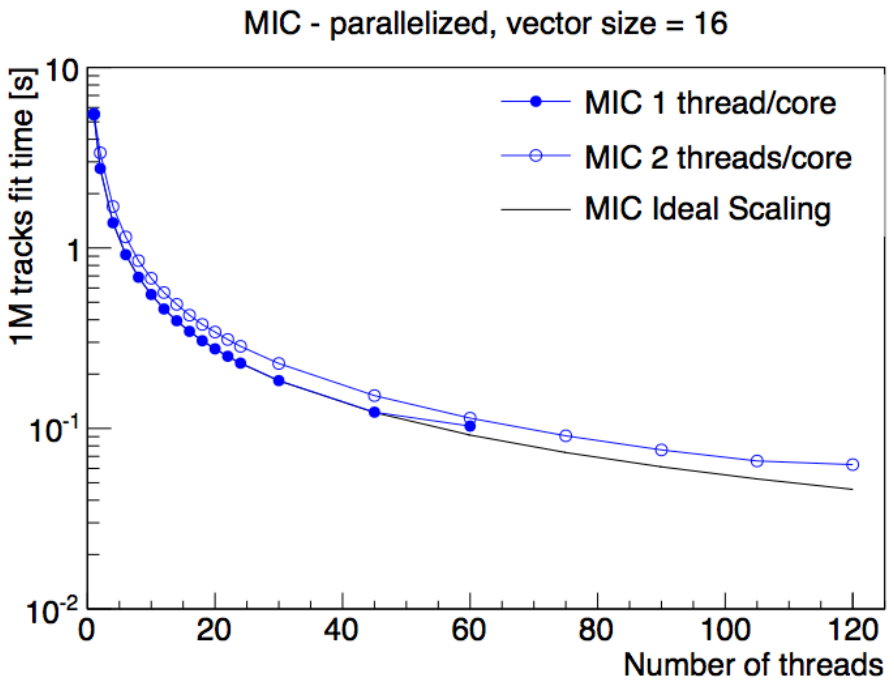
- Parallelization achieved by simply dividing tracks to be fit evenly across the number of threads



Track Fitting Results: KNC

arXiv:1409.8213





- Significant **speed-up** is observed for both **vectorization** and **parallelization**
  - Similar features on both SNB and KNC
  - **Vector utilization is roughly 50%**
  - Parallelization near ideal for 1 thread/core, overhead observed in 2 threads/core

Demonstration of feasibility on fitting, move to track building

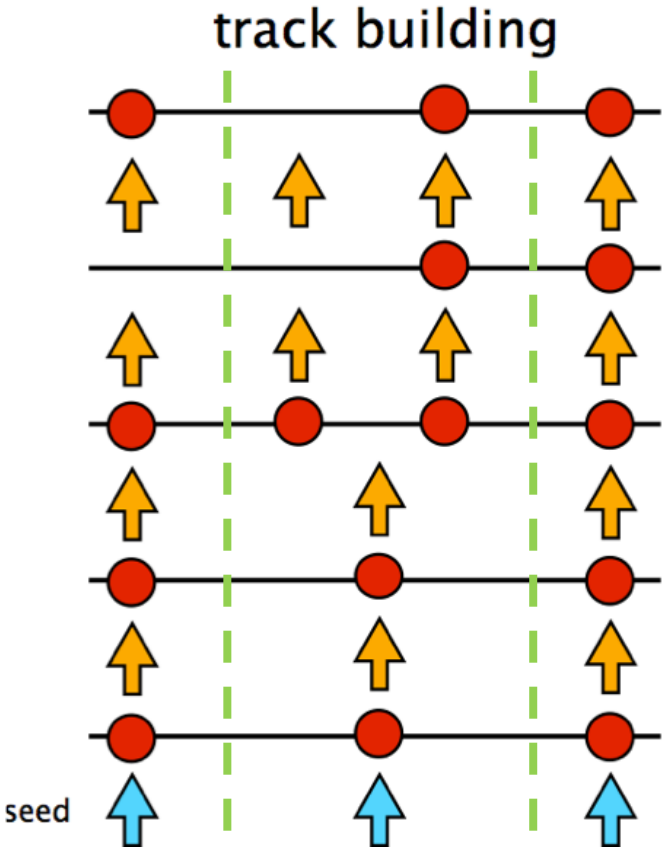
Caveats to previous slide

- Results are over **2.5 years out-of-date**
- 1M tracks/event is a bit unrealistic…
  - Wanted high enough stats to yield “steady-state” to balance out any hiccups in processing
  - Moving to smaller number of tracks does impact performance: more susceptible to unlucky tracks, thread imbalance
- OpenMP to **TBB**
- Switched coordinate system

Track Building

- Track building uses same core KF calculations as track fitting, with two major complications for **vectorizable operations**
  - **nHits problem**: Hit set is **undefined** in update of KF and have to choose between O(10^4) hits per layer.
  - For candidates with more than 1 compatible hit defined by a χ^2 test, must **branch** and **copy** track candidates

- Problems can be **factorized** for impact on physics, vectorization, and parallelization
  - First deal with nHits problem by just choosing **best hit** from χ^2 test without copying more than one candidate
  - Then deal with **combinatorial** approach by capping number of candidates to limit branching and mitigate the effects of serial work of copying candidates through the use of dedicated methods



Memory issues and workload balancing

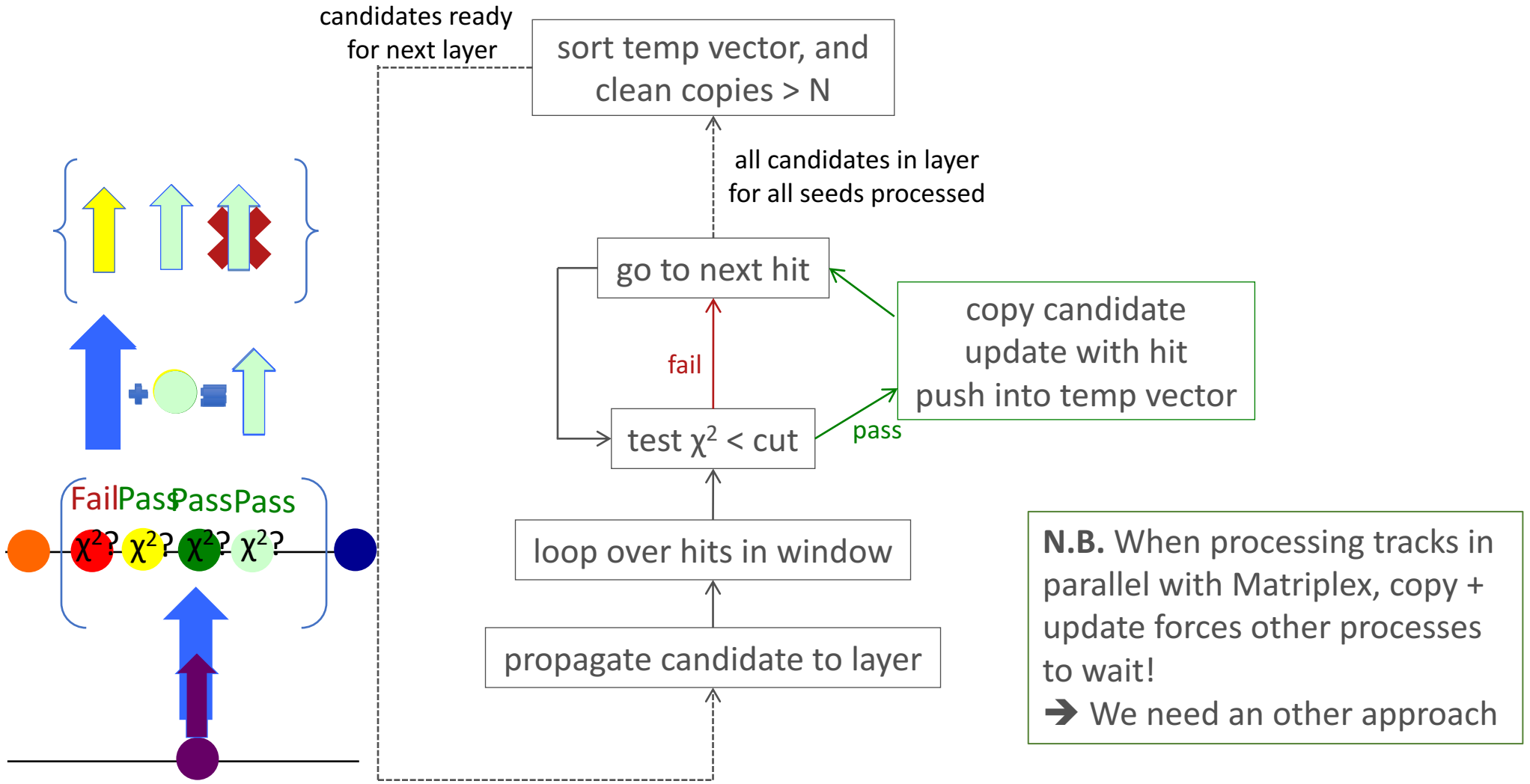
- First results on track building presented at CHEP 2015 (**arXiv: 1505.04540**), with subsequent presentations at CTD 2016 (**arXiv:1605.05508**) , CHEP 2016 (**arXiv:1702.06359**), and CTD 2017

- Regular profiling with **VTune Hotspots** revealed bottlenecks mainly from **memory operations**

- Fixes led to independent, multiplicative speedups
  - Vectorize copying-in tracks into Matriplex with systematic scatter/gather **intrinsics**
  - Avoid **resizing** of hit indices vector inside track object
  - Unnecessary **instantiation** of objects used every event could be *reset* and *recycled*
  - Reduced the *size* of **track** and **hit** objects, by 20% and 40%, respectively
  - Limit variable **scope** as much as possible, as well as decorating everything with **"const"**
  - Convert all unintentional **double** precision operations to **single** precision throughout entire code

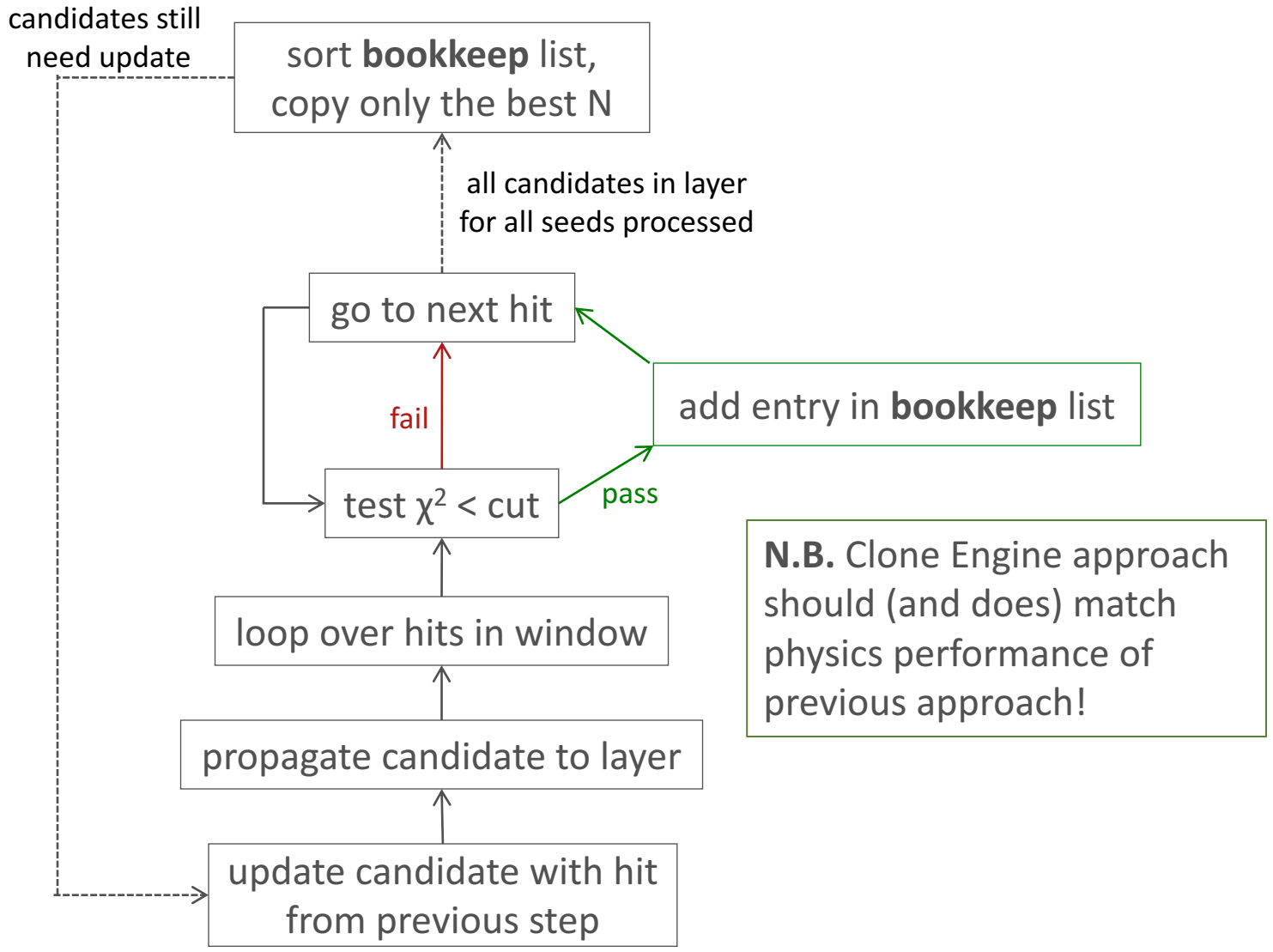
- Mitigate impact from serial work of copying track candidates in **combinatorial approach** by moving all copying **outside** of vectorizable operations: "**Clone Engine**"

Handling Multiple Track Candidates: First Approach



N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!
➔ We need an other approach

Optimized handling of multiple candidates: “Clone Engine”



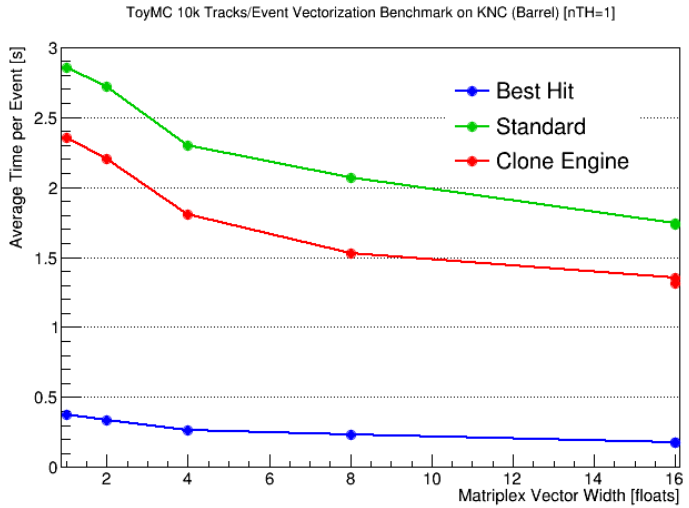
N.B. Clone Engine approach should (and does) match physics performance of previous approach!

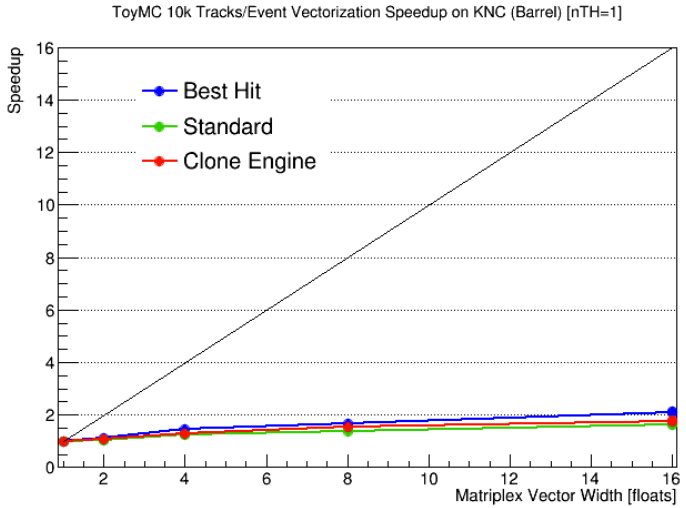
Track Building Results: KNC

- Simplified barrel-only ToyMC geometry with 10k tks/evt
- Scaling tests with 3 building algorithms
  - Best Hit - less work, recovers fewer tracks (only one hit *saved* per layer, for each seed)
  - Standard & Clone Engine - combinatorial, penalized by branching & copying
- Vectorization - speedup is limited in all methods
  - Faster by **40-50%** on both platforms
- Multithreading with Intel TBB - speedup is good
  - Clone Engine gives best overall results
  - With 120 KNC threads, CE speedup is **65x**

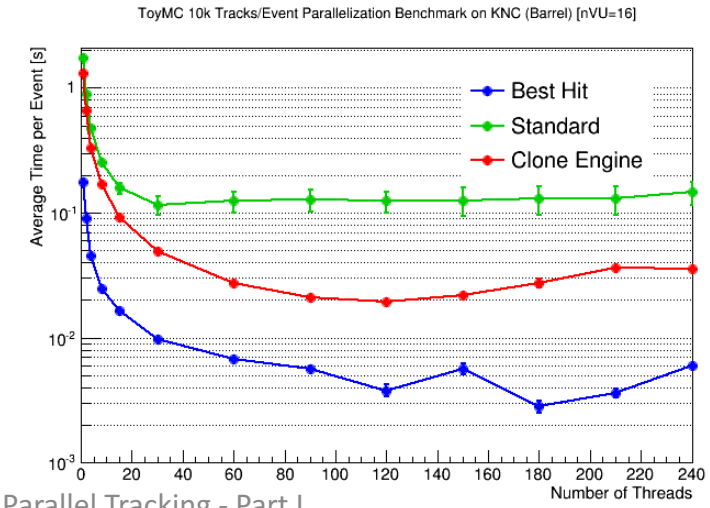
Absolute Time | **Relative Speedup**

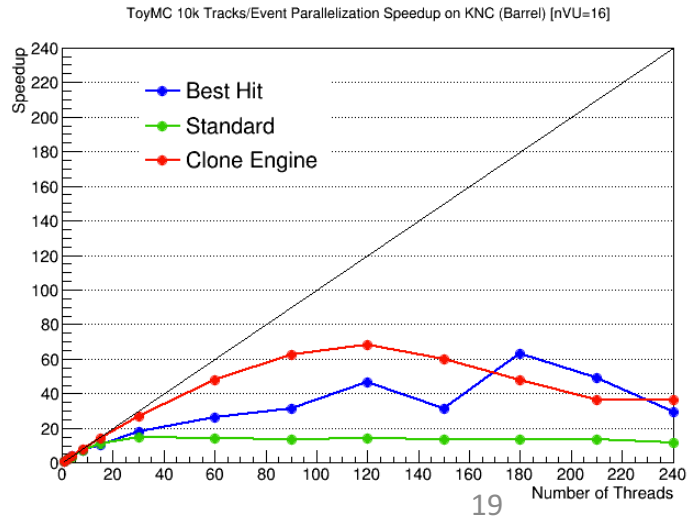
Vectorization





Parallelization





Conclusions and Outlook

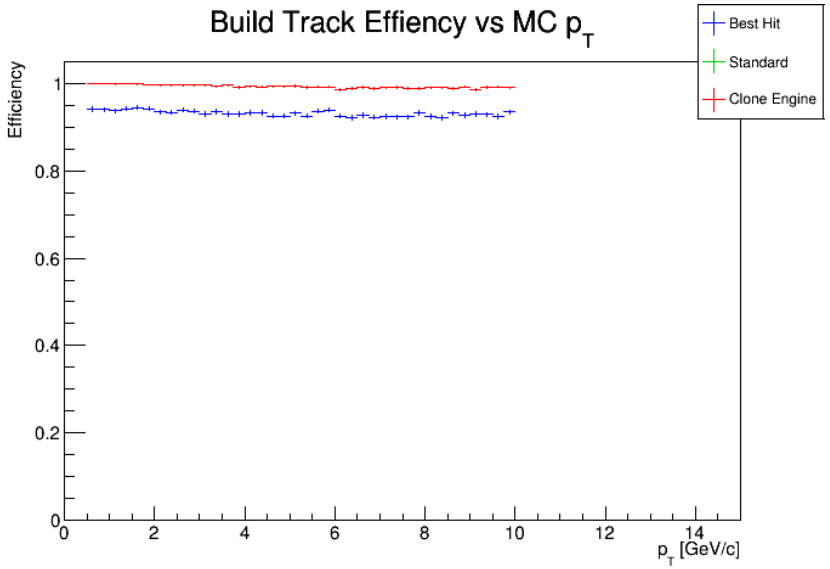
- Initial tests for track building show good performance for both **vectorization** and **parallelization** on KNC

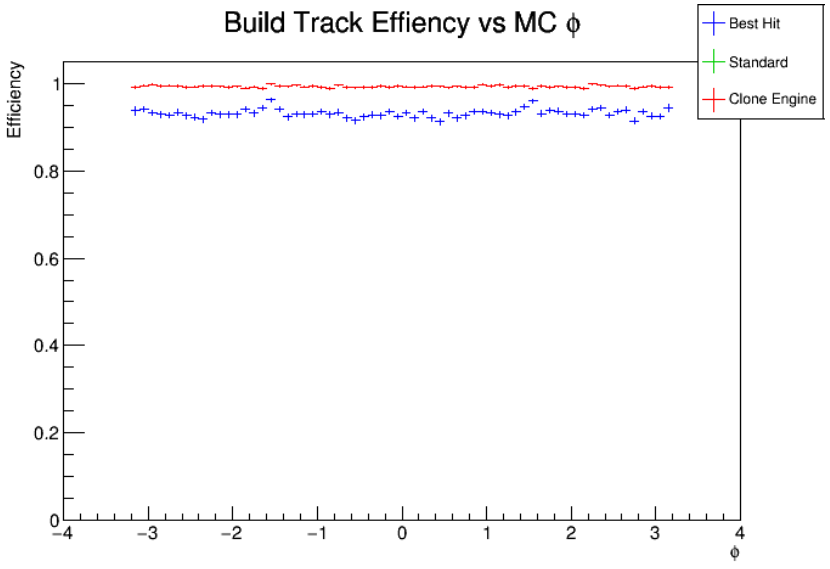
- Teasers for next round of presentations:
  - Approaching CMSSW-like track building
  - SNB vs. KNC vs. KNL track building performance
  - GPU fitting and building with a Matriplex-like approach

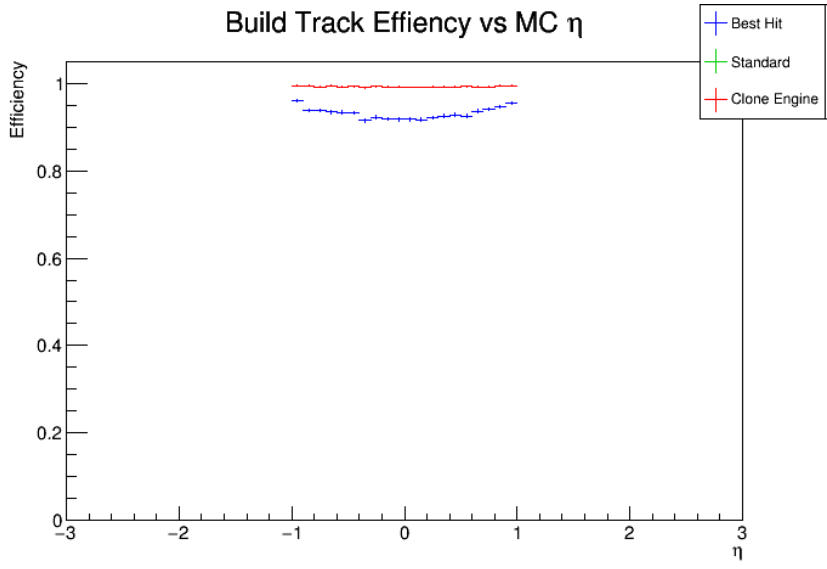
- Major developments currently in the pipeline:
  - Unified barrel and endcap geometry to perform building across the full detector
  - Multiple-events-in-flight to recover performance for small numbers of tracks/event
  - Iteration0-like seeding to be used for testing end-to-end tracking

Backup

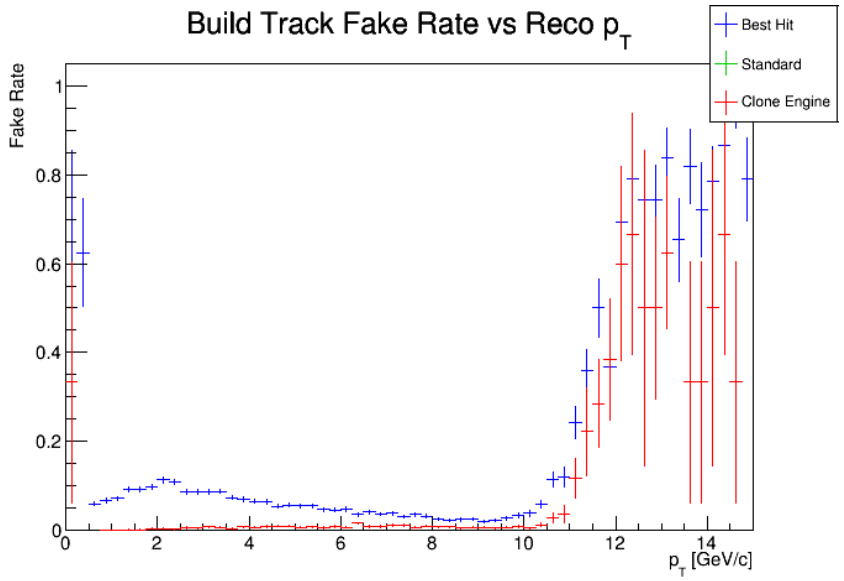
Track Building: Physics Performance – Efficiency

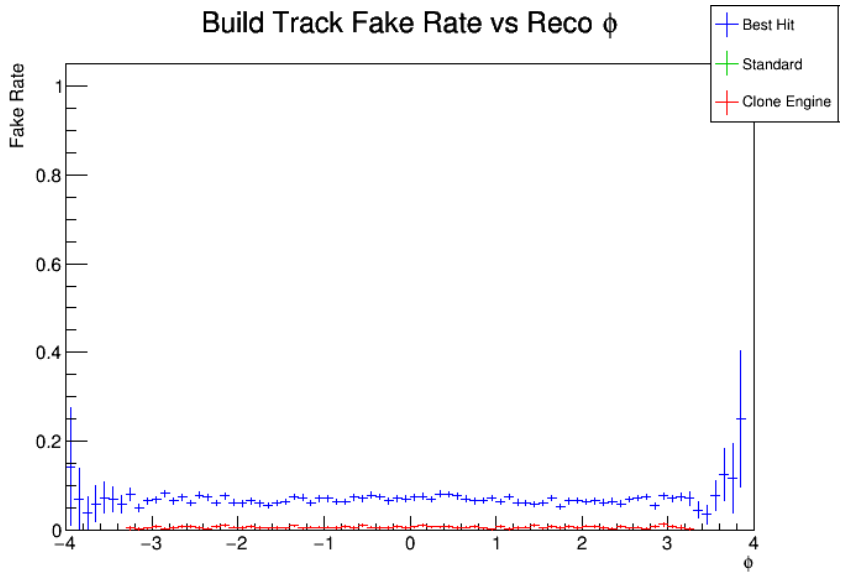


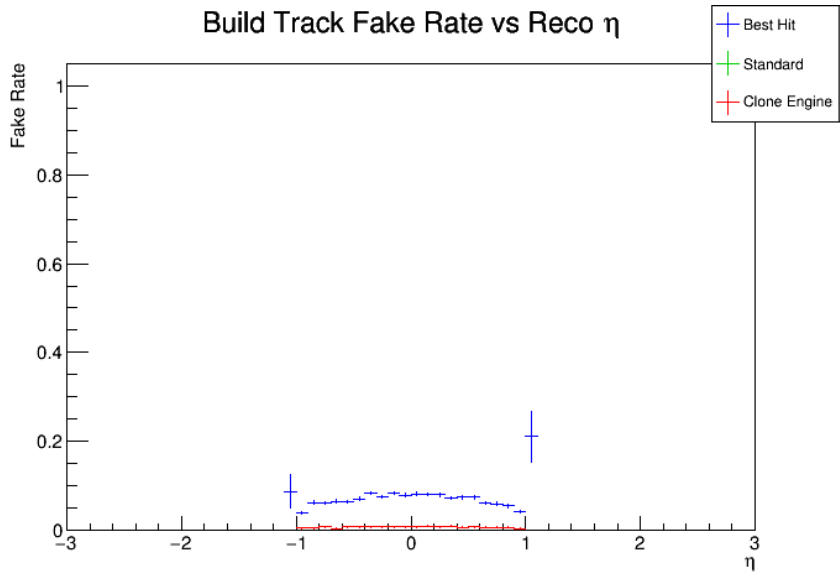




Track Building: Physics Performance – Fake Rate







Track Building: Physics Performance
Number of Hits per Track