Parallelized tracking algorithms

Matti Kortelainen

Joint WLCG & HSF Workshop 2018

Napoli, Italy

March 28, 2018

Outline

- Introduction
- Challenges for massive parallelism
- Tracking R&D projects in CMS
- Summary

- Disclaimer: this talk is heavily biased towards CMS because of my background
  - There are of course many other projects out there

Introduction

- Computing future appears to lie somewhere in many-core and/or heterogenous
- Tracking is a big consumer of event reconstruction time
- Natural focus for parallelization efforts

- In practice experiment frameworks provide the event-level parallelism
- Implies that a simple `tbb::parallel_for` does not gain much, unless
  - Less events in flight than the number of threads/cores
    - Will happen eventually though...
  - Global synchronization points (framework runs out of all other work)
- To really gain performance, one has to go beyond thread/task-level parallelism on CPU
  - SIMD (vectorization)
  - Accelerators like GPU or FPGA

Challenges for massive parallelism

- We have small matrices (5×5 / 6×6)
  - Typical approach for matrix operation vectorization does not scale for us
    - $\star$ = vectorize the calculations of a single matrix operation
- Pattern recognition problem is combinatorial with dynamic output size
  - Need to create new track candidates on the fly
  - In general need dynamic memory allocations
- Traditional algorithms have lots of control logic
  - Harmful for SIMD
- Data transfers from CPU/host memory to accelerator(s)
  - For short computations data transfer time may dominate
- E.g. in Intel Skylake the use of AVX/AVX-512 instructions slows down the frequency of the core
  - So to benefit from AVX, significant portion of computational kernel has to be vectorized

Tracking R&D projects within CMS

- Parallelized Kalman Filter (mkFit)
  - Explore Kalman filter based track finding and track fitting on many-core SIMD and SIMT architectures
  - Goal: Run in CMS HLT for Run3 and beyond
  - Website: http://trackreco.github.io/
- Pixel tracking on GPUs (patatrack)
  - A hybrid CPU-GPU application that takes RAW data coming from the pixel detector and gives pixel tracks (or seeds) as result
  - Goal: Run in CMS HLT for Run3 and beyond
  - Website: https://patatrack.web.cern.ch/patatrack/
- HEP.TrkX
  - Novel deep learning methods for track reconstruction
  - More exploratory than the other two projects above, not covered here
  - Website: https://heptrkx.github.io/

Parallelized Kalman-Filter-Based Reconstruction of Particle Tracks with Accurate Detector Geometry

G. Cerati[4], P. Elmer[3], M. Kortelainen[4], 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. UCSD 2. Cornell 3. Princeton 4. FNAL

University of Oregon just started

Connecting the Dots 2018, University of Washington, Seattle

- Latest presentation in CtD 2018

https://indico.cern.ch/event/658267/contributions/2813732/

Kalman filter

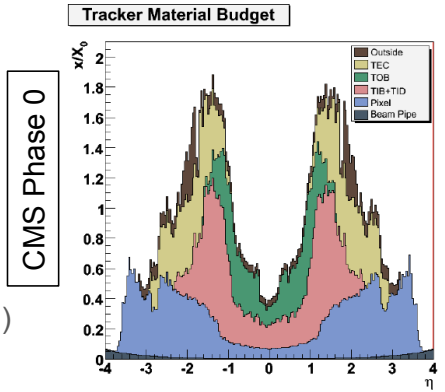
Why use Kalman filter:

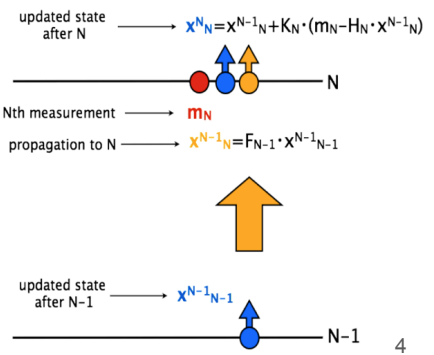
- Widely used & well understood
- Demonstrated physics performance:
  - Can handle multiple scattering and energy loss (badly needed)

Our goals for Kalman filter based track finding:

- Make effective use of parallel and vector architectures
- Maintain physics performance
- Preserve consistent systematics across platforms

Our work is complementary to tracklet-based divide and conquer algorithms.





Tasks related to track finding

1. Seed finding: have basic implementation but it is not actively developed
   a) Now we either use CMSSW seeds or MC truth seeding for development
   b) For CMSSW seeds we do cleaning prior to track finding
2. **Track finding**: this is our primary focus. Several algorithms:
   a) Best hit: take the best hit on every layer
   b) Standard: on every layer check all compatible hits, select N best candidates for each seed
   c) Minimize copy: apply cleverness to b. to reduce data copying and unnecessary cloning of Tracks
   d) Full vector: different cleverness in handling & management of candidates belonging to same seed

   a) and b) are reference implementations

   In c) and d) are variations of b) where we are trying to do better …
3. **Track fitting**: secondary focus, is actually much easier
   a) Starting with a vector of found hits and initial track parameters, use Kalman filter on all the hits.
   b) This was the first piece we developed, it gave us great results and encouragement :)
   c) Simple cases saw x8 vectorization speedup on KNC and good multi-thread scaling
4. **Validation**: physics & computational performance

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).

Multi-threading, Architectures & compilers

For multi-threading we use TBB:

- Two parallel_fors over tracking regions and seeds (shown in steering code)
- parallel_for over events - multiple events in flight
  - This is crucial for plugging the gaps arising from unequal load in track finding tasks!

We actually started with OpenMP but it is hard to do dynamic problem partitioning. TBB is already used in CMSSW.

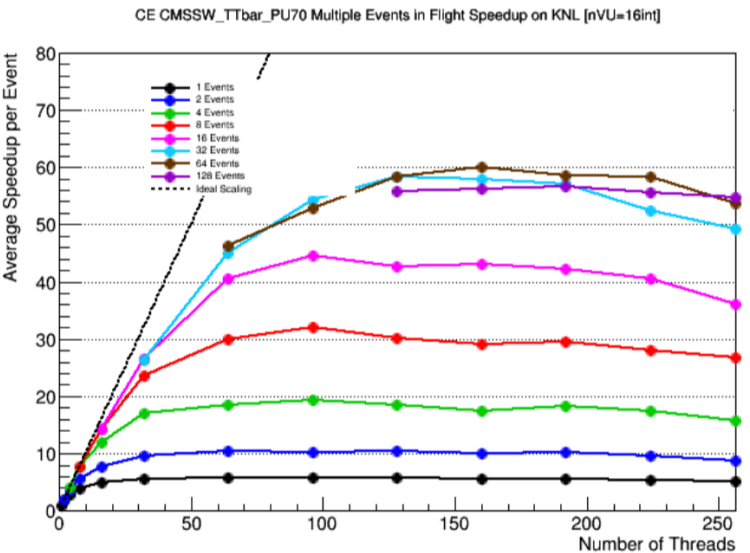
Architectures & compilers:

- x86_64 (AVX, AVX-512), KNC (MIC), KNL (AVX-512)
  - icc, gcc; we use --std-c++11
- Nvidia / CUDA
  - Have implementations of track fitting and track finding (best hit and minimize copy)

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: 80 events / s

KNL



Some lessons learnt

- Minimizing the amount of data and control structures pays off
  - Simplifying detector description reduces memory loads, but increases computations due to approximations
    - ⋆ Fortunately well-vectorized compute cycles are relatively cheap
- Matrixplex approach works!
  - Able to obtain 2–3× speedup with vectorization (AVX)
- Challenging to fully understand performance characteristics even with tools like Intel VTune or NVidia `nvprof`
  - More challenging for GPUs than CPUs
    - ⋆ Arithmetic intensity is a bigger challenge in GPUs
  - Exploring TAU Performance system as a profiling tool
- 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

Use of GPUs at the CMS High-Level Trigger during Phase-1

Felice Pantaleo

CERN, Experimental Physics Department

felice@cern.ch

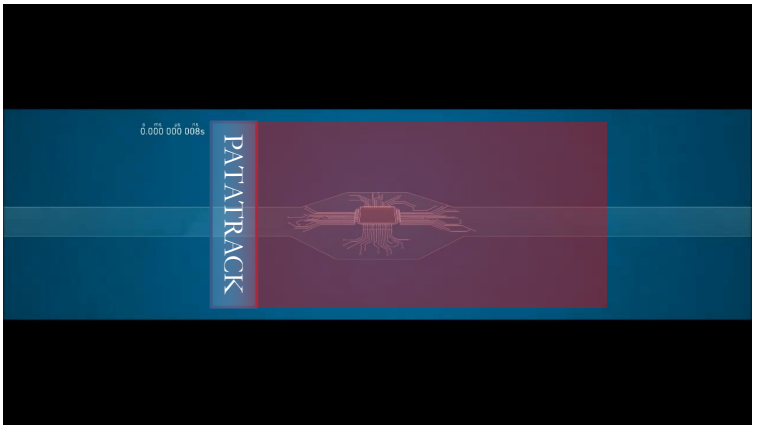
D. Bacciu, A. Bocci, C. Calabria, A. Carta, S. Roy Chowdhury, A. Di Florio, S. Di Guida, S. Dubey, S. Dugad, S. Dutta, R. Fruhwirth, V. Innocente, V. Khristenko, M. Kortelainen, P. Mal, D. Menasce, F. Pantaleo, M. Pierini, M. Rovere, S. Sarkar, V. Volkl

- Latest presentation in ACAT 2017

https://indico.cern.ch/event/567550/contributions/2627138/

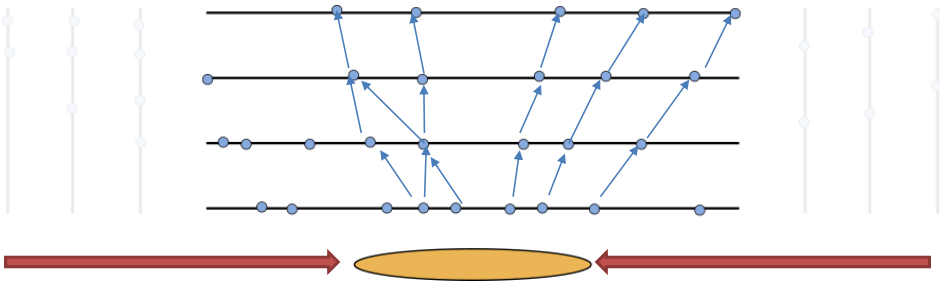
PATATRACK: From RAW data to Tracks

- Objective:
  - A hybrid CPU-GPU application that takes RAW data coming from the pixel detector and gives Tracks as result
- Sustained input event rate of 100kHz from L1
- GPU memory transfers are hidden exploiting instruction level parallelism (function executing while non related transfer happens)
  - increased throughput
  - no impact on total latency
- Ingredients:
  - Massive parallelism within the event
  - Avoid useless data transfers and transformations
  - Simple data formats optimized for parallel memory access
  - Renovation at algorithmic level



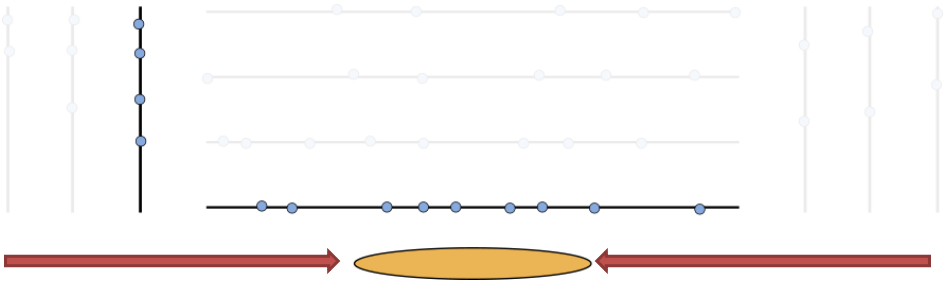
Triplet Propagation Algorithm

- First create doublets from hits of pairs
- Take a third layer and propagate only the generated doublets
- Consider a fourth layer and propagate triplets
- Store found quadruplets and start from another pair of layers



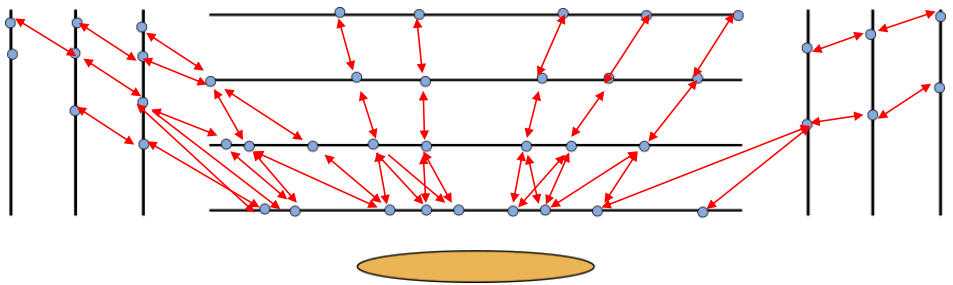
Triplet Propagation Algorithm

- First create doublets from hits of pairs
- Take a third layer and propagate only the generated doublets
- Consider a fourth layer and propagate triplets
- Store found quadruplets and start from another pair of layers
- Repeat until happy…
- Does this fit the idea of massively parallel computation? I don't really think so…



Cellular Automaton (CA)

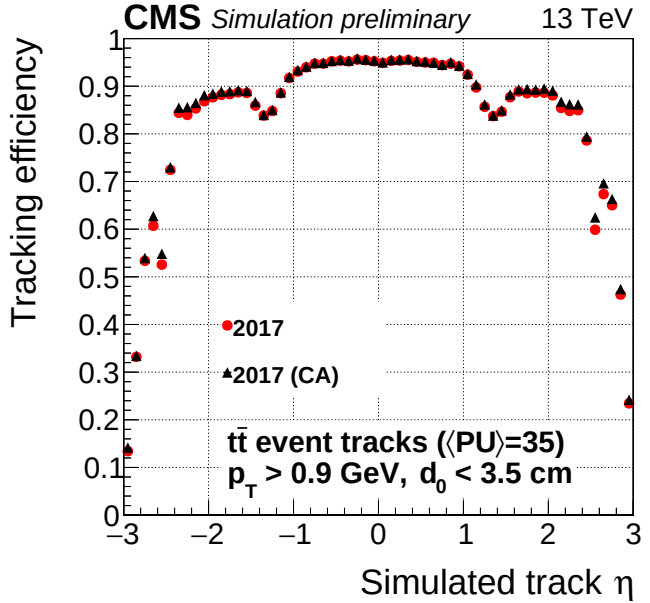
- The CA is a track seeding algorithm designed for parallel architectures
- It requires a list of layers and their pairings
  - A graph of all the possible connections between layers is created
  - Doublets aka Cells are created for each pair of layers (compatible with a region hypothesis)
  - Fast computation of the compatibility between two connected cells
  - No knowledge of the world outside adjacent neighboring cells required, making it easy to parallelize
- However this is not a static problem, not at all…



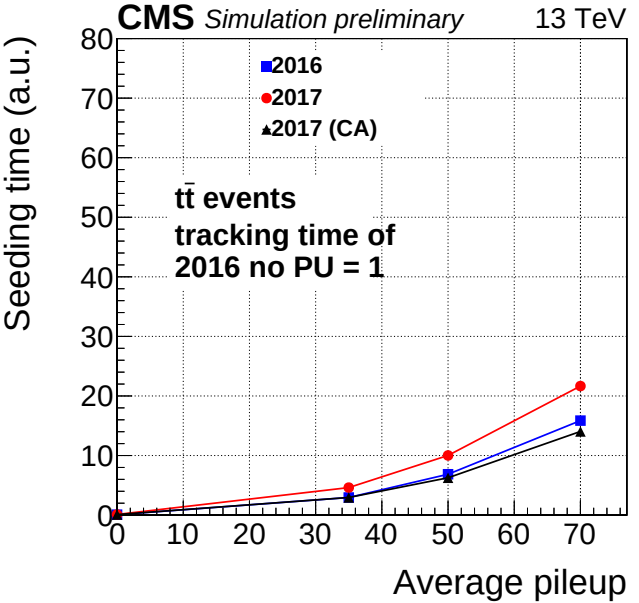
Now a library, v0.1
https://github.com/HEP-SF/TrickTrack

CA seeding in production since 2017

- CPU version of the CA seeding running in production since 2017
  - For both high-level trigger and offline reconstruction



Physics performance as good as competing "traditional" algorithm



Faster, especially at large PU (on CPU)

Some lessons learnt

- Good example of designing an algorithm for GPU and porting the algorithm back to CPU
  - Improves performance on CPU even when running sequentially
- Same codebase running on CPUs and GPUs and producing same results bit-by-bit
  - Eases validation, debugging, deployment
  - Pre-requisite for a fully flexible and dynamic heterogenous system
- In GPU, transferring data and running kernels in parallel increases throughput
- Spin-off: "TrickTrack" seeding package for FCC
  - See presentation in CtD 2018

    https://indico.cern.ch/event/658267/contributions/2813731/

Summary

- Two tracking R&D projects within CMS are in pretty advanced stage
  - Both aiming to be run in production in Run 3
- Algorithms, data structures, data flow have to be designed with the target architecture in mind
- Somewhat generic packages start to emerge
  - Matriplex: vectorization of small matrix operations
  - TrickTrack: geometry-independent CA-based seeding package
- Advanced profiling tools used to understand the current bottlenecks
  - Intel VTune, NVidia `nvprof`
  - Interpretation not always easy
- Most of the code is rather application-specific at the moment
  - E.g. parallel Kalman Filter package is in principle quite general, but is not yet a ready-to-use tracking package