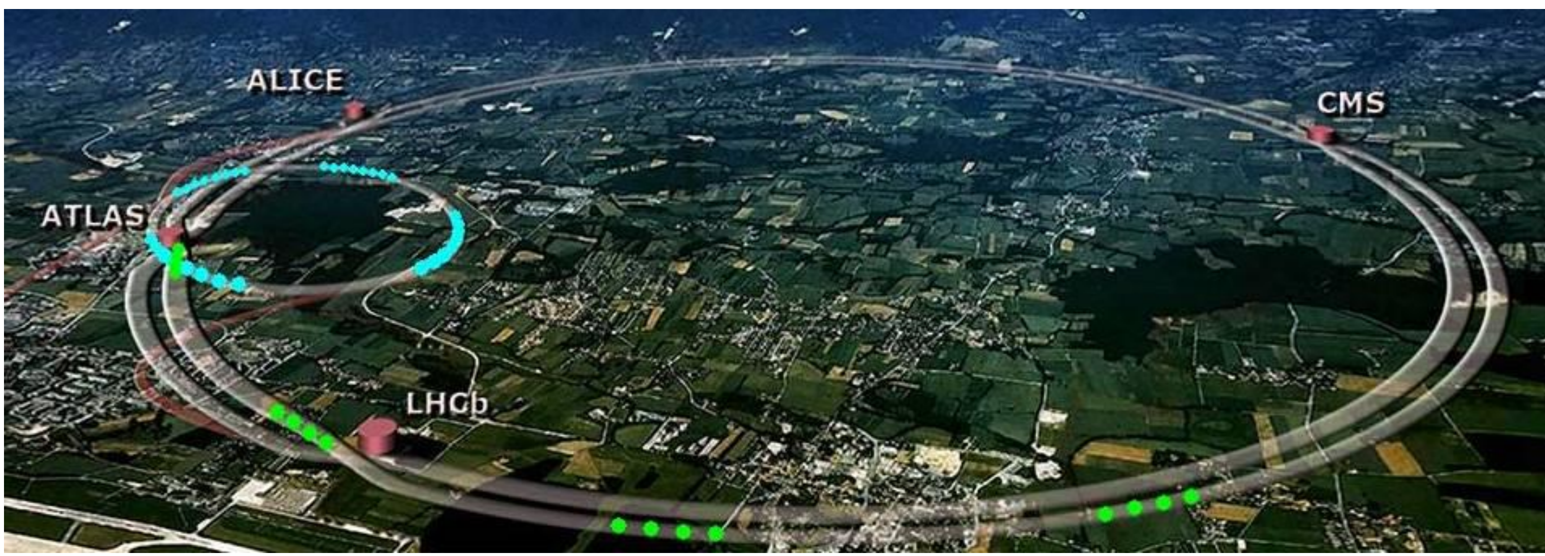
Massively Parallel Computing at the Large Hadron Collider up to the HL-LHC

Paul Lujan
(on behalf of Valerie Halyo)
Princeton University

The Large Hadron Collider

- 27 km circumference ring on France/ Switzerland border
- Design center-of-mass energy of 14 TeV
- Four major experiments: ATLAS & CMS (general purpose), LHCb (b physics), ALICE (heavy ion)



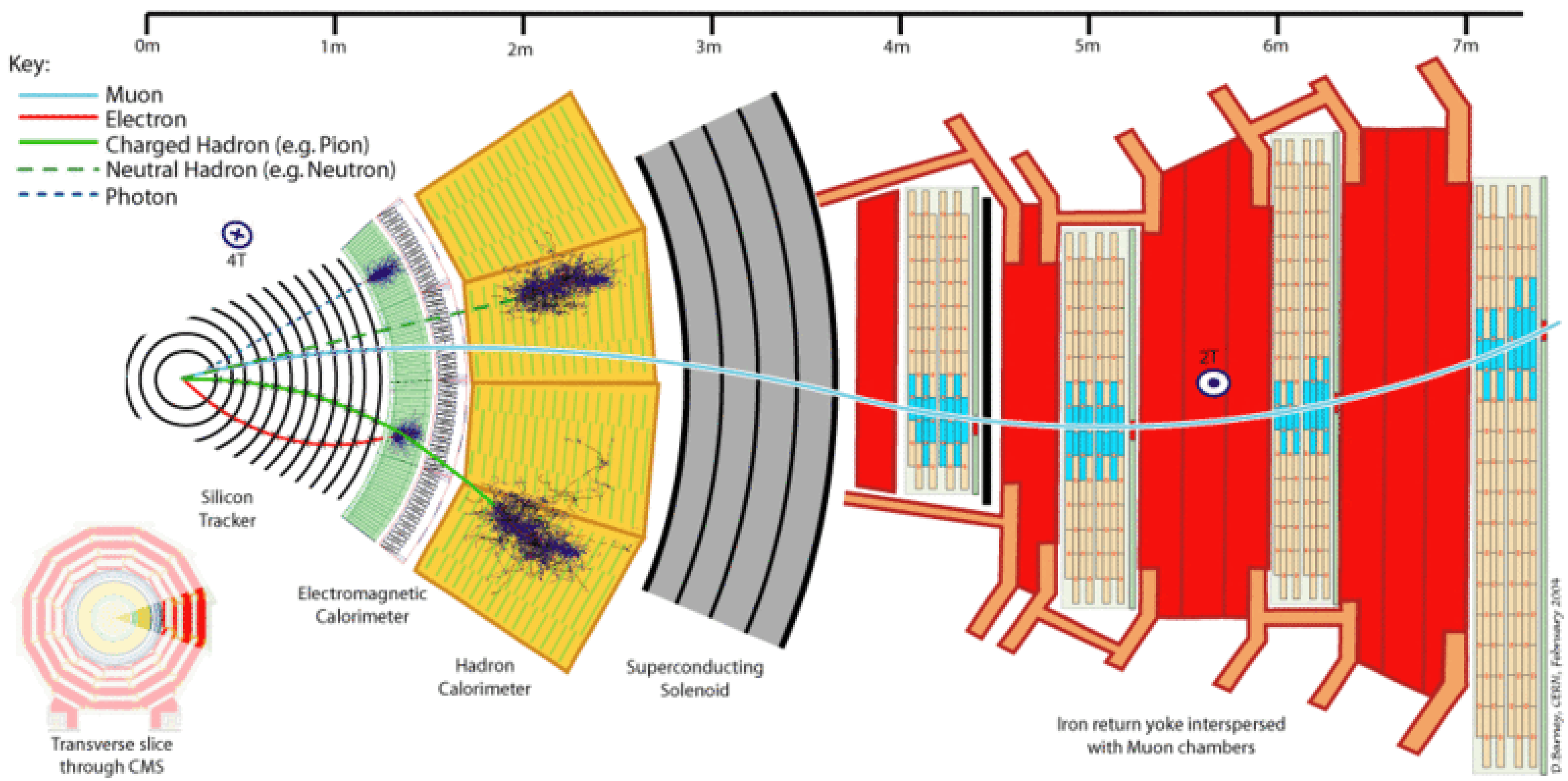
The LHC Detectors

- Generally similar layout (except for LHCb)

	ATLAS	CMS	ALICE	LHCb
Inner tracker	Si pixels & strips	Si pixels	Si pixels & strips	Si pixels (VELO)
Outer tracker	Straw tracker	Si strips	Time projection chamber	Si strips, straws, Cherenkov
Magnet	2T solenoid outside tracker	3.8T solenoid outside calorimeter	0.5T solenoid	Dipole, avg. field ≈ 0.5T
EM calorimeter	LAr in lead/steel	Lead tungstate crystals	$PbWO_4$ (γ) + Pb/scintillator	Scintillators in lead
Hadronic calorimeter	Scintillators in steel	Scintillators in brass/steel	n/a	Scintillators in iron
Muons	DT/CSCs and RPCs in toroids	DT/CSCs and RPCs in solenoid return yoke	CSCs and RPCs in dipole magnet	Multiwire proportional chambers + GEM

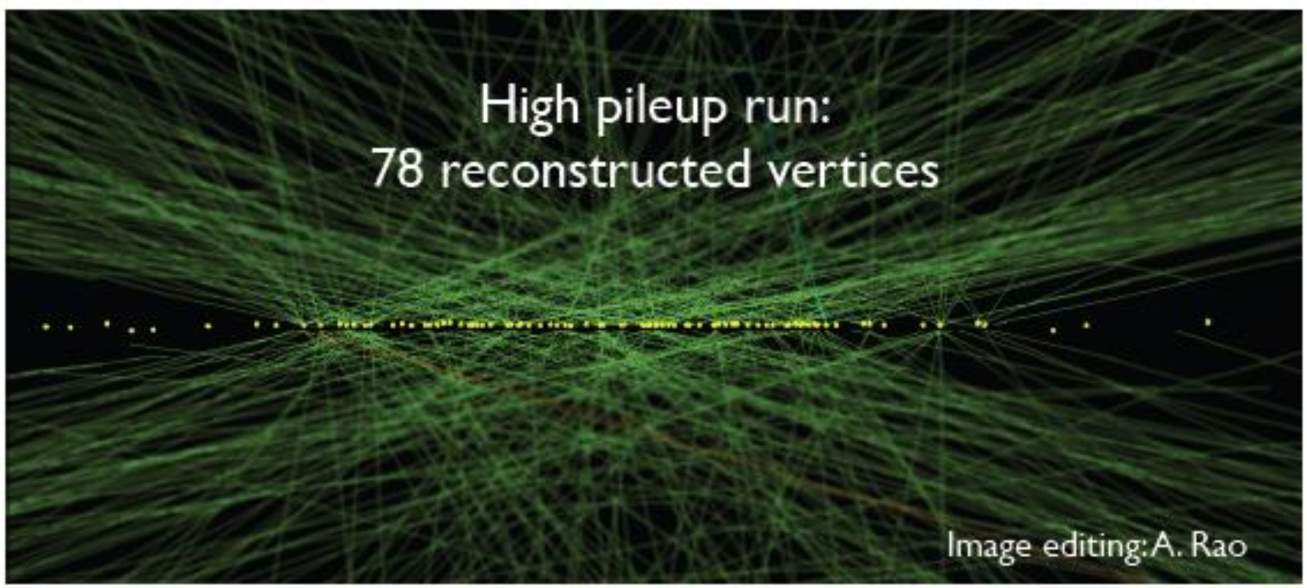
CMS Slice

- Example particle signatures in CMS



Increasing Luminosity at the LHC

- Increased luminosity means increased pileup (number of interactions per bunch crossing)
- Events become more challenging to reconstruct as pileup increases



Run I (through 2012)

- Center-of-mass energy: 7-8 TeV
- Typical luminosity: 5×10^{33} cm^{-2} s^{-1}, up to peak of approx. 7×10^{33}
- Bunch spacing: 50 ns (20 MHz collision rate)
- Typical pileup: ~25, up to peak of ~40
- Total delivered luminosity: ~30 fb^{-1}

Run II (2015-2018)

- Center-of-mass energy: 13-14 TeV
- Typical luminosity: 1.5 x 10^{34} cm^{-2} s^{-1}
- Bunch spacing: 25 ns (40 MHz collision rate)
- Expected pileup: average of ~40
- Total expected delivered luminosity: ~100 fb^{-1}

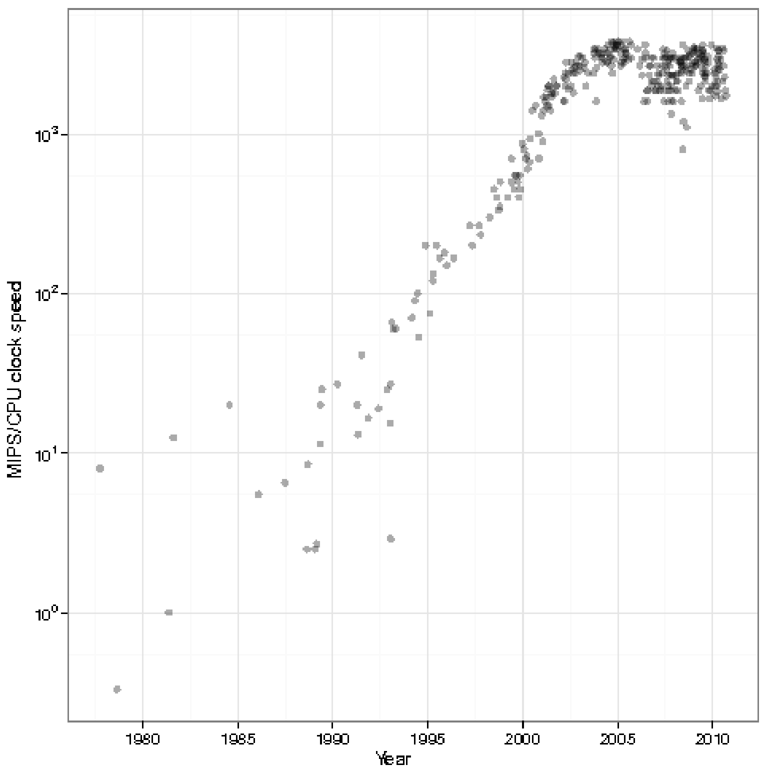
Run III (2020-2022)

- Center-of-mass energy: 14 TeV
- Typical luminosity: 2 x 10^{34} cm^{-2} s^{-1}
- Bunch spacing: 25 ns (40 MHz collision rate)
- Expected pileup: average of ~60
- Total expected delivered luminosity: ~300 fb^{-1}

HL-LHC (2025-)

- Center-of-mass energy: 14 TeV
- Typical luminosity: 5×10^{34} cm^{-2} s^{-1}
- Bunch spacing: 25 ns (40 MHz collision rate)
- Expected pileup: average of **~130**
- Total expected delivered luminosity: ~3000 fb^{-1}
- Major upgrades to all detectors to increase physics capabilities, replace radiation-damaged parts, handle increased luminosity, etc.
- Will require large computing efforts to handle the volume of data, both online and offline

The Need for Parallel Computing



courtesy r-bloggers.com

- Increase in clock speed, which was fairly constant, stalled out around year 2000
- Need other ways to obtain increased performance
- *Parallel* computing: multi-core, GPU computing, many integrated core (MIC)

Massively Parallel Computing

- In contrast to multi-core computing, massively parallel systems can feature 1000s of cores
  - Individual cores are not as powerful, but the parallelism gives an advantage
- GPU computing (e.g. Nvidia Tesla): use the stream processors in a graphics processing unit (GPU) for general-purpose computation
- MIC computing (e.g. Intel Xeon Phi): large number of x86-based processor units in a single chip
- Require different approaches than regular CPU programming
  - Issues such as memory access become very important

GPU Computing: A Brief History

- 2001: first programs using GPU for general purpose computing
  - Required translating the problem into a graphics problem using DirectX/OpenGL
- 2007: release of Nvidia CUDA
  - Allowed GPU programming with nearly-standard C++ code
- 2009-present: development of various standards to simplify GPU programming
  - OpenMP, OpenACC, OpenCL, etc…

Nvidia Tesla

- Tesla K40: Kepler architecture, 2880 thread processors @ 745 MHz, 12 GB GDDR5 RAM @ 288 GB/s

Intel Xeon Phi

- Xeon Phi 7120P: Knights Corner architecture, 61 cores @ 1.24 GHz (up to 1.33 GHz), 16 GB DDR5 RAM @ 352 GB/s
- 4-way hyperthreading
- 512-bit AVX2 vector extensions

Using parallel computing: libraries

- Libraries built to take advantage of accelerators already exist for many common packages
  - e.g. for CUDA: libfftw → cuFFT, libblas → cuBLAS, NPP (Nvidia Performance Primitives), etc.
- Very easy to implement: just drop into an existing project
- Often can provide considerable speedup – sometimes beneficial to use even if not so that a computation can be done entirely on the GPU

Using parallel computing: directives

- Standards such as OpenMP or OpenACC can provide simple but more customizable acceleration.
- It can be as easy as (example in OpenACC):

```
#pragma acc kernels
for (int i=0; i<n; ++i) {
    …
}
```

- Compiler support still incomplete (e.g., OpenACC is not in gcc yet), but making rapid progress

Using parallel computing: CUDA

- Define kernels that run on the GPU, copy memory to GPU as necessary, and call them:

```
//example code from Nvidia -- developer.nvidia.com
__global__
void saxpy(int n, float a, float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int main(void)
{
  int N = 1<<20;
  float *x, *y, *d_x, *d_y;

// put data into x[] and y[]
  cudaMalloc(&d_x, N*sizeof(float));
  cudaMalloc(&d_y, N*sizeof(float));

  cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);

  // Perform SAXPY on 1M elements
  saxpy<<<(N+255)/256, 256>>>(N, 2.0, d_x, d_y);

  cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);
```

Optimizing parallel code

- Not an easy task!
- For instance,

```
saxpy<<<(N+255)/256, 256>>>(N, 2.0, d_x, d_y);
```

- These two arguments are the number of blocks and the threads per block.
- How to choose optimal number of threads per block?
  - The answer depends on the details of the hardware and the problem
  - Often trial and error is the only way
  - Work on automating this optimization is in progress, but still a long way away

Complications with directives

- Naïve use of directives like `#pragma acc kernels` can often result in slower code than the non-parallelized version!
  - For example, the compiler doesn't necessarily know when it needs to copy the data between the host and the device, so it will be conservative.
  - This can result in a lot of additional unnecessary copying.
  - You'll need additional pragmas like `#pragma acc data` to instruct the compiler when it needs to copy data and when it can keep the data on the device.

Parallel computing: summary

- Writing massively parallel programs has never been easier.
- Libraries and directives make it very easy to get started.
- However, you still need a good understanding of the problem and the computational issues to get best performance.
- Profiling tools to find which steps are slowest are critical.

Advantages of Accelerators at LHC

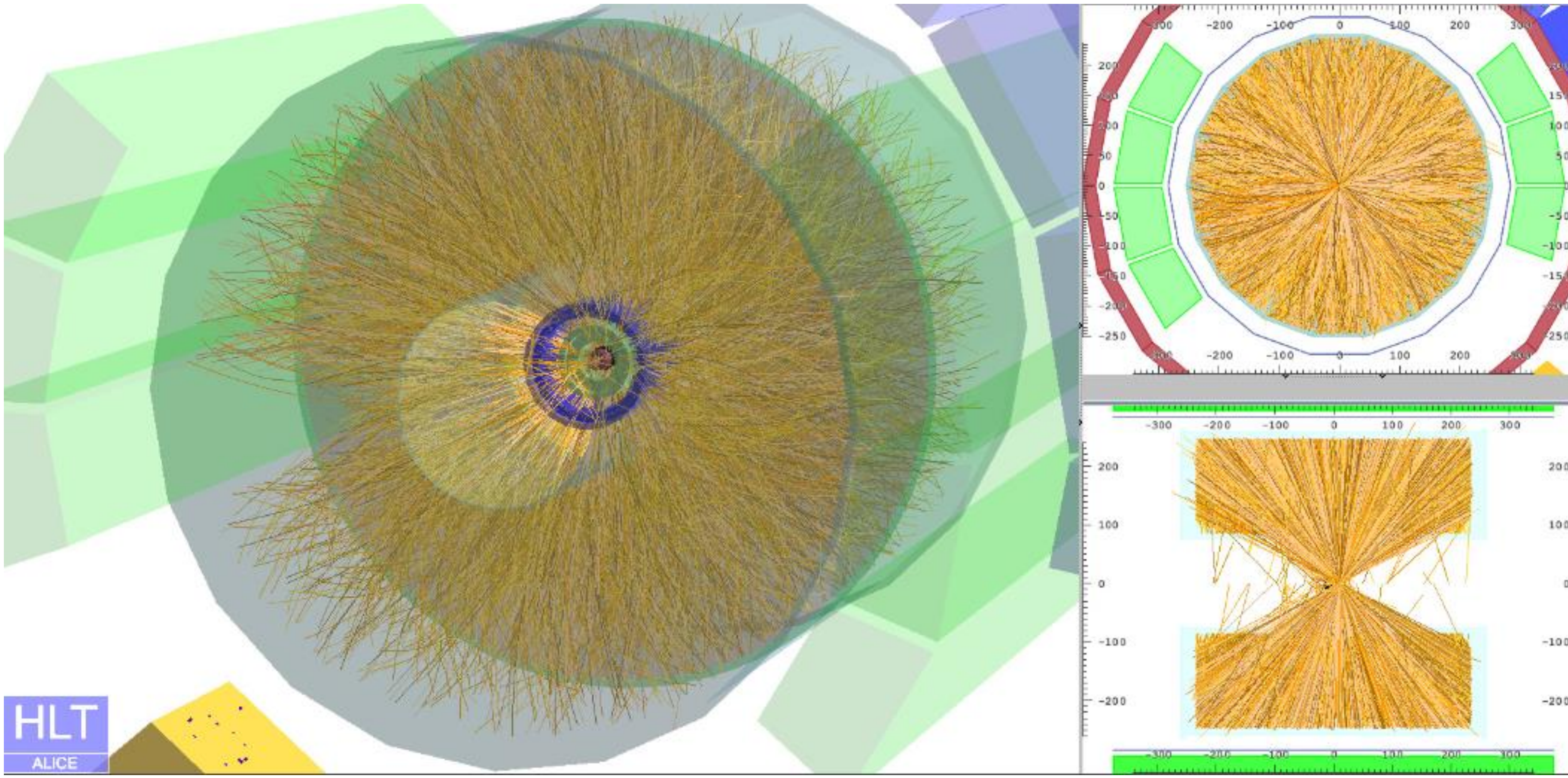
- Easy to integrate hardware into existing computing farms
- Relatively low cost
- Can be gradually integrated as resources are available
- Can write software compatible with variety of hardware setups
- Integrating these capabilities into the current software is not always easy!

Parallel Computing at the LHC

- Examples of current projects
- Future plans
- A case study: tracking & triggering of displaced tracks at CMS

GPU Acceleration of Tracking at ALICE

- Heavy ion collisions have a very large track multiplicity



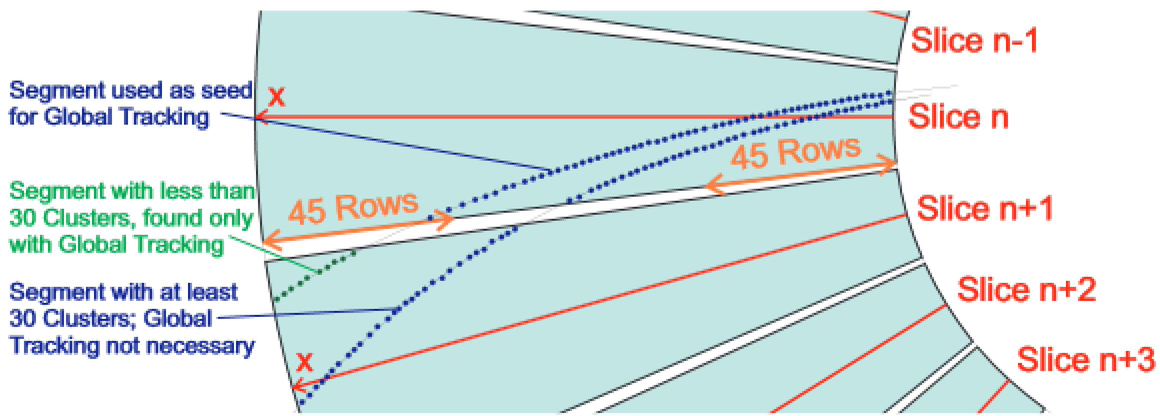
D. Rohr, CHEP12

GPU Tracking at ALICE

- CPU does pre- and post-processing of the tracks
- Actual track finding and fitting is offloaded to the GPU
- Multiple CPU cores used to ensure GPU always remains busy
- Run on relatively simple GPU hardware (originally Nvidia GTX 295, later GTX 480)
- Overall increase of 3x speed

GPU Tracking: additional advantages

- Normally tracking in the TPC is local: the chamber is divided into slices
- If only a small part of a track is in a given slice, the track will not be reconstructed



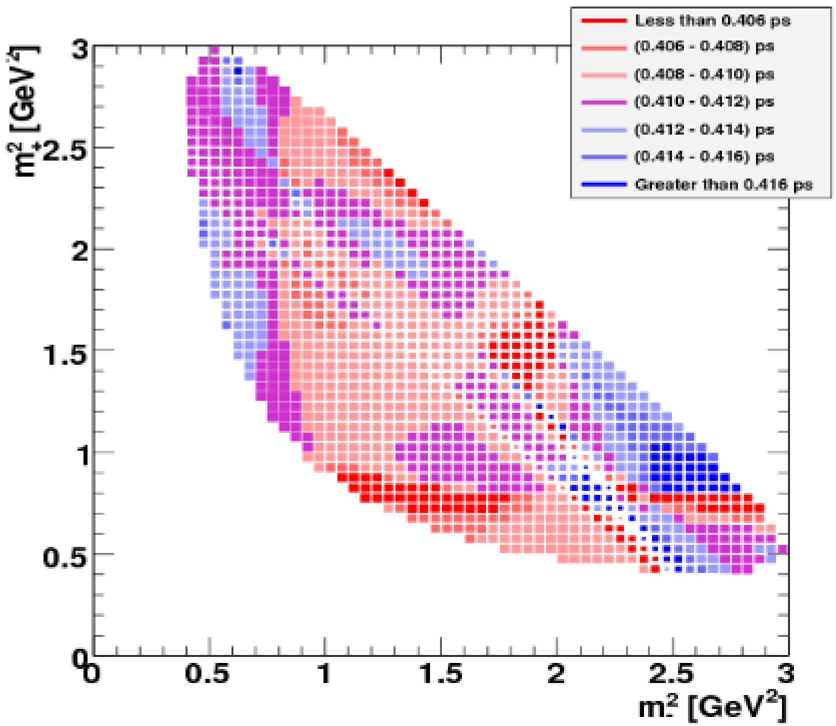
- Global tracking allows the track to be propagated between sectors for more efficient reconstruction

GooFit: RooFit with GPUs

- RooFit – a standard ROOT-based framework for performing a wide variety of fits
- Maximum-likelihood fits can become very lengthy
- GooFit exports this calculation onto GPUs, which can result in large speedups
  - Two backends: CUDA and OpenMP
- Very similar to current RooFit interface

GooFit Example

- Time-dependent amplitude analysis of $D_0 \rightarrow \pi^+\pi^-\pi^0$
  - Unbinned 4-D fit with about 40 signal parameters



R. Andreassen, CHEP2013

GooFit Speedup

Platform	Time [s]	Speedup
Original CPU	19489	1.0
Xeon E5520 OpenMP 1 thread	3056	6.4
Xeon E5520 OpenMP 24 threads	432	45.1
above + Nvidia Tesla C2050	64	**304.5**
Intel i7-3610QM OpenMP 1 thread	2042	9.5
Intel i7-3610QM OpenMP 8 threads	407	47.9
above + Nvidia GeForce 650M	212	**91.9**
Nvidia Tesla C2070	54	**360.1**

Future Tracking Plans

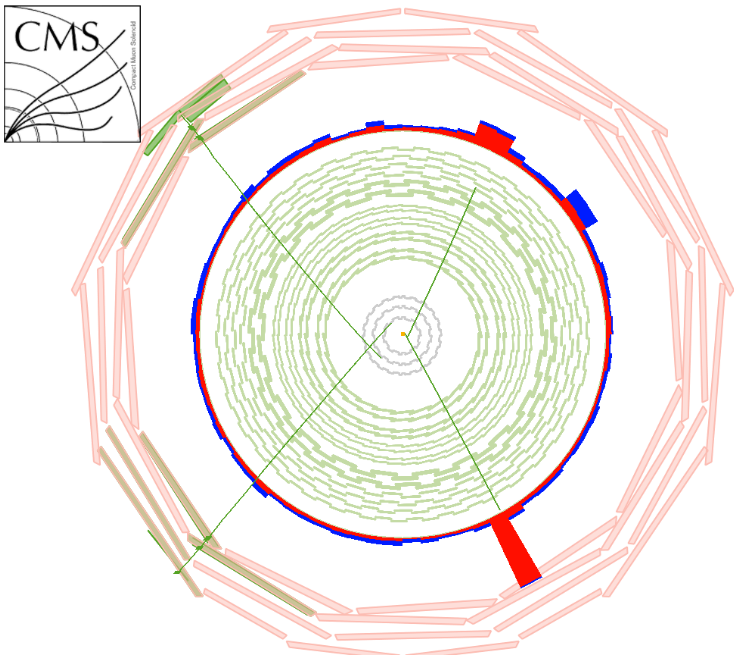
- Lots of research going into improving the tracking at all LHC experiments
  - Algorithms: CTF (Kalman filter), cellular automata, tracklet, retina, pattern recognition, …
  - Technologies: GPU, MIC, FPGA, …
- Improving tracking performance will make a L1 track trigger possible
- Will be a long road ahead, however…

Looking for New Physics

- So far, these examples have shown ways to improve performance of existing algorithms.
- But parallel computing also gives us new algorithms which we can use to look for entirely new physics.
- As an example, let's look at a different kind of signature.

Displaced Tracks

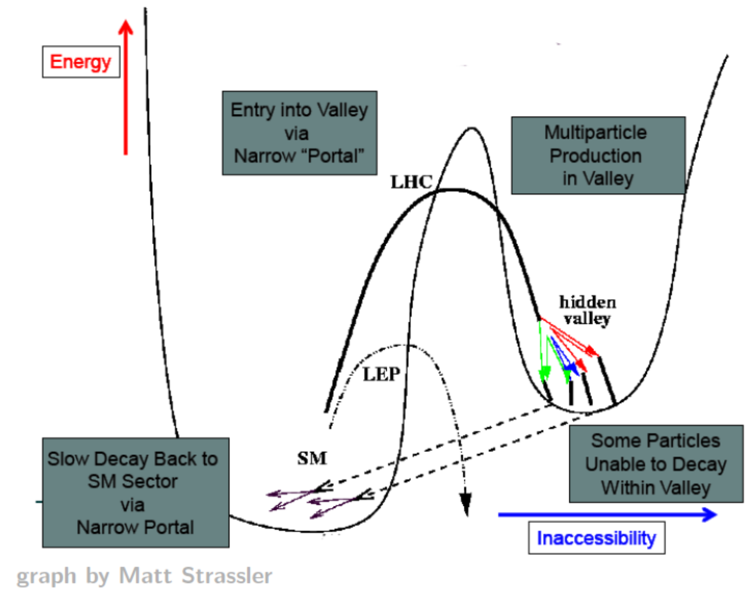
- Consider particles which have a substantial lifetime, so they travel a significant distance in the detector.
- Such an event would be a **clear and unambiguous** signal of new physics!
- Decay could be into leptons, jets, ...
- Possible signatures: displaced tracks, kinked/disappearing tracks, delayed tracks, etc.



simulation of two long-lived neutral particles decaying to muons (left) and electrons (right)

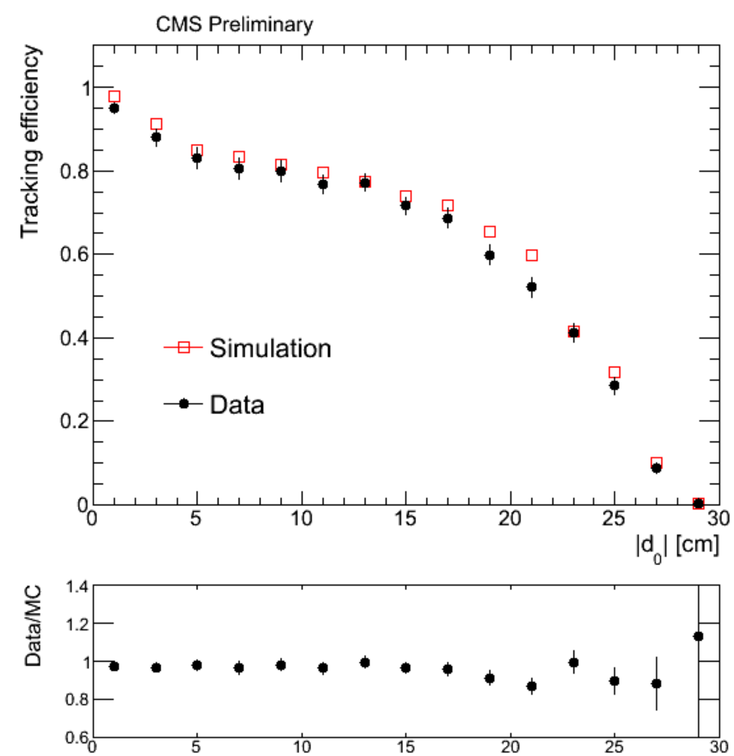
Displaced Tracks: Theory

- A wide variety of theoretical models predict this kind of signature:
  - Hidden valley models
  - Weakly R-parity-violating supersymmetry (SUSY)
  - Split SUSY with long-lived gluinos
  - Z' production and decay
  - "Little Higgs" models
- Even black holes could have such a signature – they could have a significant lifetime in which they can travel away from the primary interaction



Displaced Tracks: The Problem

- However, standard tracking algorithms (especially at the trigger level) are not designed to reconstruct tracks which are significantly displaced like these.
- Efficiency falls off rapidly above about 15 cm and is zero by 30 cm.
- Somewhat driven by tracker size, but also by algorithm limitations.



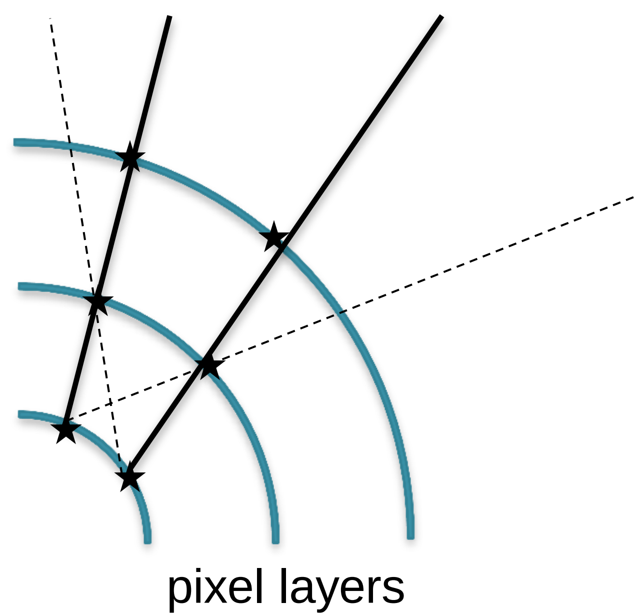
CMS PAS EXO-12-038

Combinatorial Track Finder

- The standard track finding algorithm at CMS and ATLAS is the Combinatorial Track Finder (CTF).
- Well-established, reliable algorithm.
- However, the "combinatorial" in the name suggests the problem…

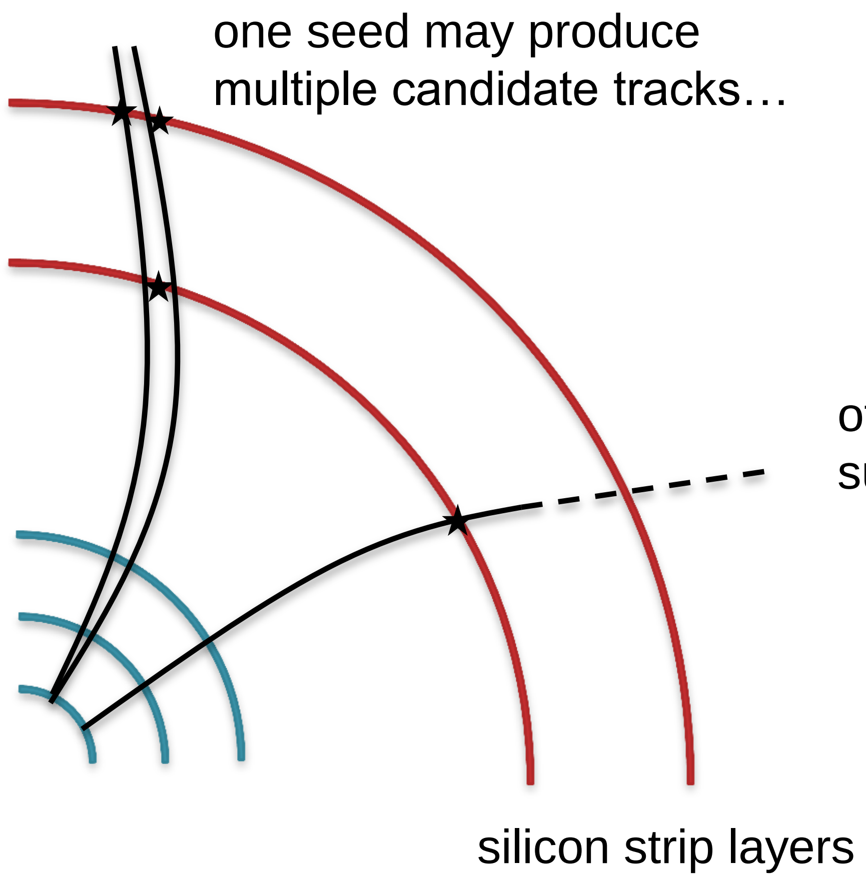
CTF: Seeding

- Form seeds by taking all possible pairs, then looking for a third compatible hit
- If a triplet is found, it is used as the starting trajectory for track finding



CTF: Finding

- The trajectory is then propagated out through the layers and compatible hit(s) are attached



CTF: Fitting and Cleaning

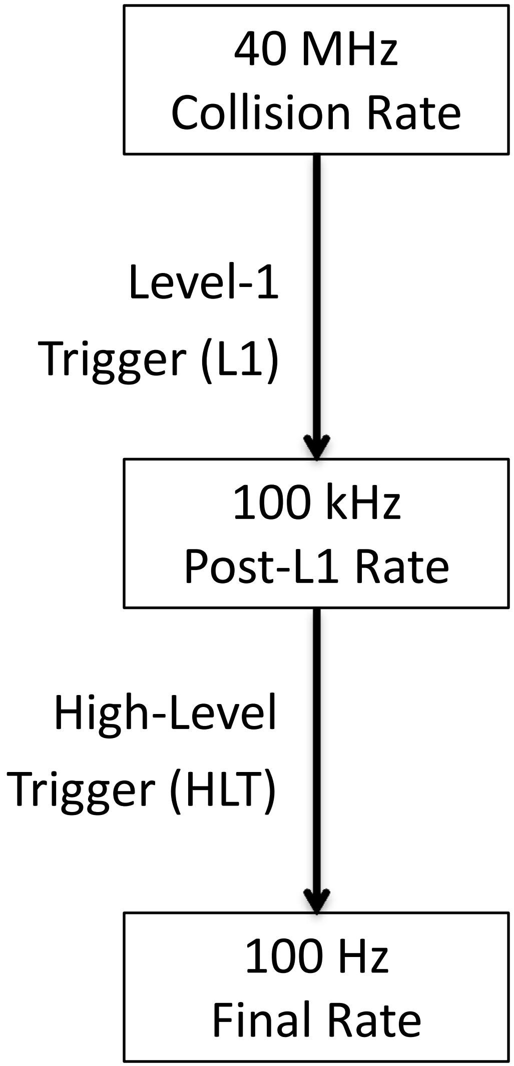
- Once all of the hits have been found and attached to the track, a final Kalman fitter step is performed to get the best fit of the track parameters using all of the hit information.
- Tracks which share a large number of hits are then cleaned by selecting the single best fit track.

CTF: Iterative Tracking

- The CTF also uses "iterative tracking" to reduce the number of combinations.
  - Early iterations look for the easiest tracks to reconstruct: high-momentum tracks with seeds in the pixel layers (better resolution).
  - The hits from these tracks are then removed and the search can proceed to lower-momentum tracks and tracks with seeds in the outer strip layers (including potentially displaced tracks).

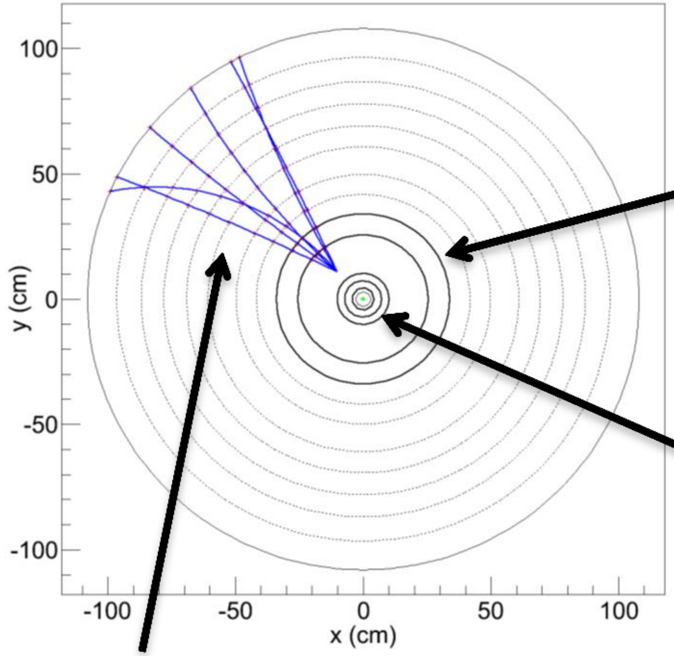
The CMS Trigger

- The HLT performs a nearly full reconstruction of the event and must do so in a very limited time.
- Uses iterative tracking very similar to as previously described, but only in regions of interest around L1 calorimeter/muon hits.



CTF: Online and Offline

- As a consequence of the limited amount of time available at the HLT, the online CTF does not include a step to reconstruct displaced tracks – it's too expensive!



Only bold layers are used for seeding

Only tracks with a transverse impact parameter < 0.5 cm (green circle) are reconstructed

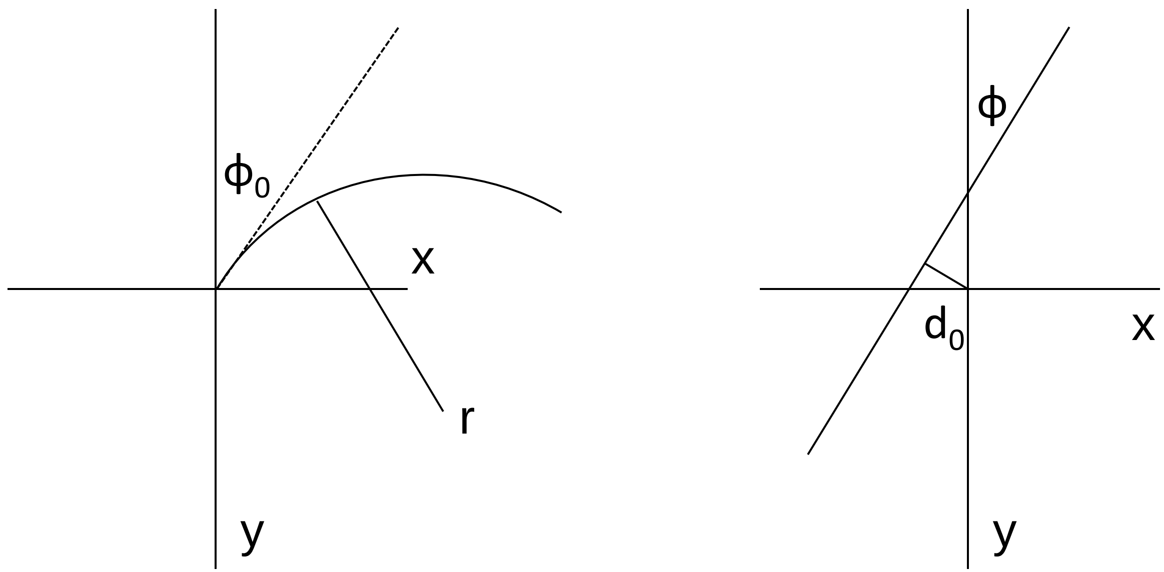
We could miss entirely events like these!

Consequences for Displaced Tracks

- Especially for jets (especially for models where the parent particle has a relatively low mass), it is difficult to construct an effective trigger algorithm.
- This problem will only get worse with increased pileup.
- Potential new physics might be missed!

Hough Transform

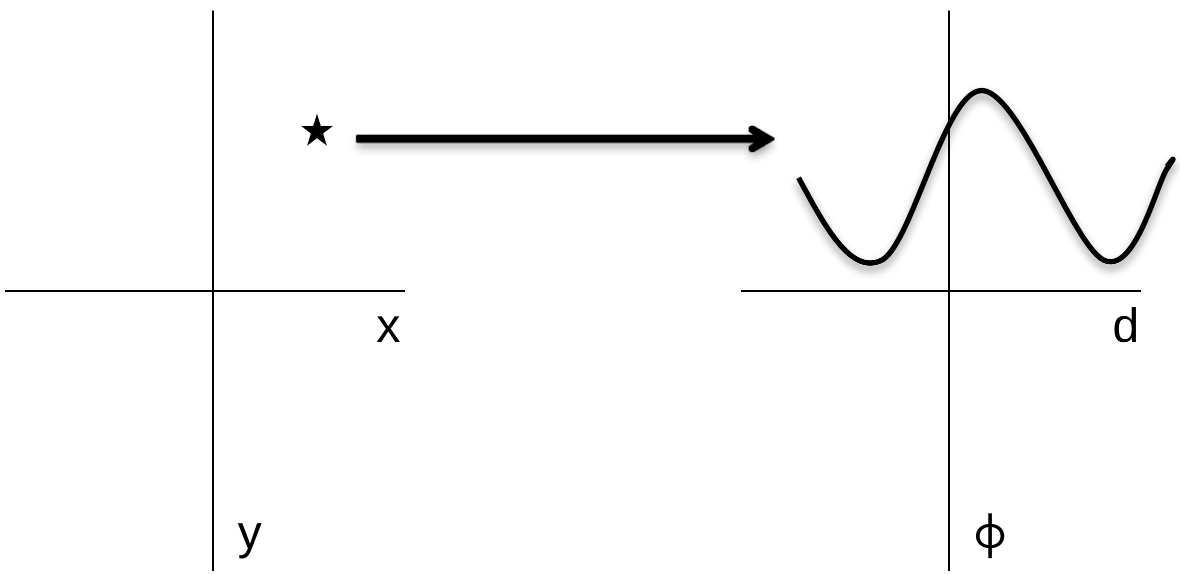
- Try a Hough-transform based tracking approach
- Hough transform: describe track in terms of parameters and define parameter space



- 2 parameters can define a curved track from origin, or a displaced straight track
- 3 parameters necessary for a displaced curved track
- 5 parameters (full set of track parameters) for a 3-D track

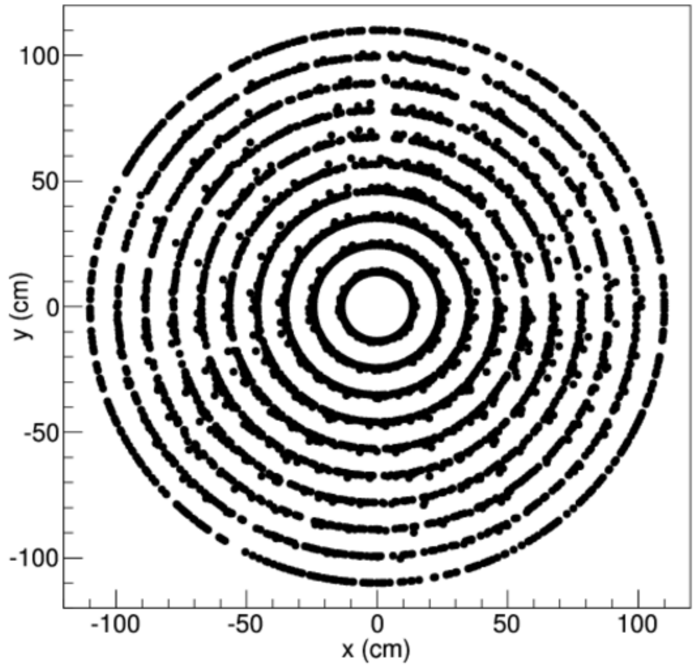
Hough Transform: cont'd

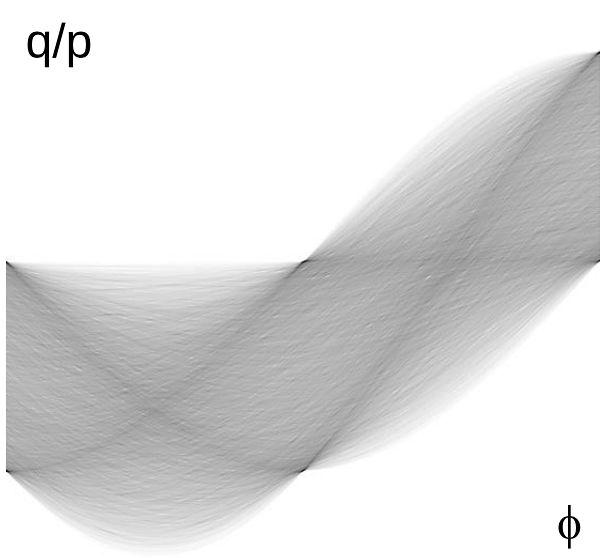
- Each hit in the original space then becomes a curve in parameter space – the family of curves that pass through that hit



Hough transform: simple example

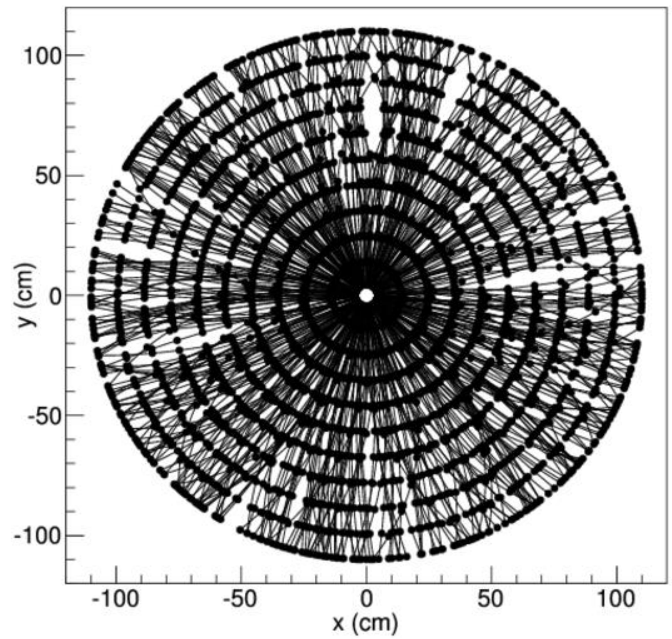
- Begin with hits from 500 simulated curved tracks (left).
- Apply the Hough transform to get the parameter space (right).





Hough transform: simple example (2)

- Finding the maxima in the parameter space reconstructs the original tracks.
- In this example, efficiency is about 85%.
- Highly parallelizable – the conversion of each hit into parameter space can be done separately
- Inherently accounts for differing resolution of different hits



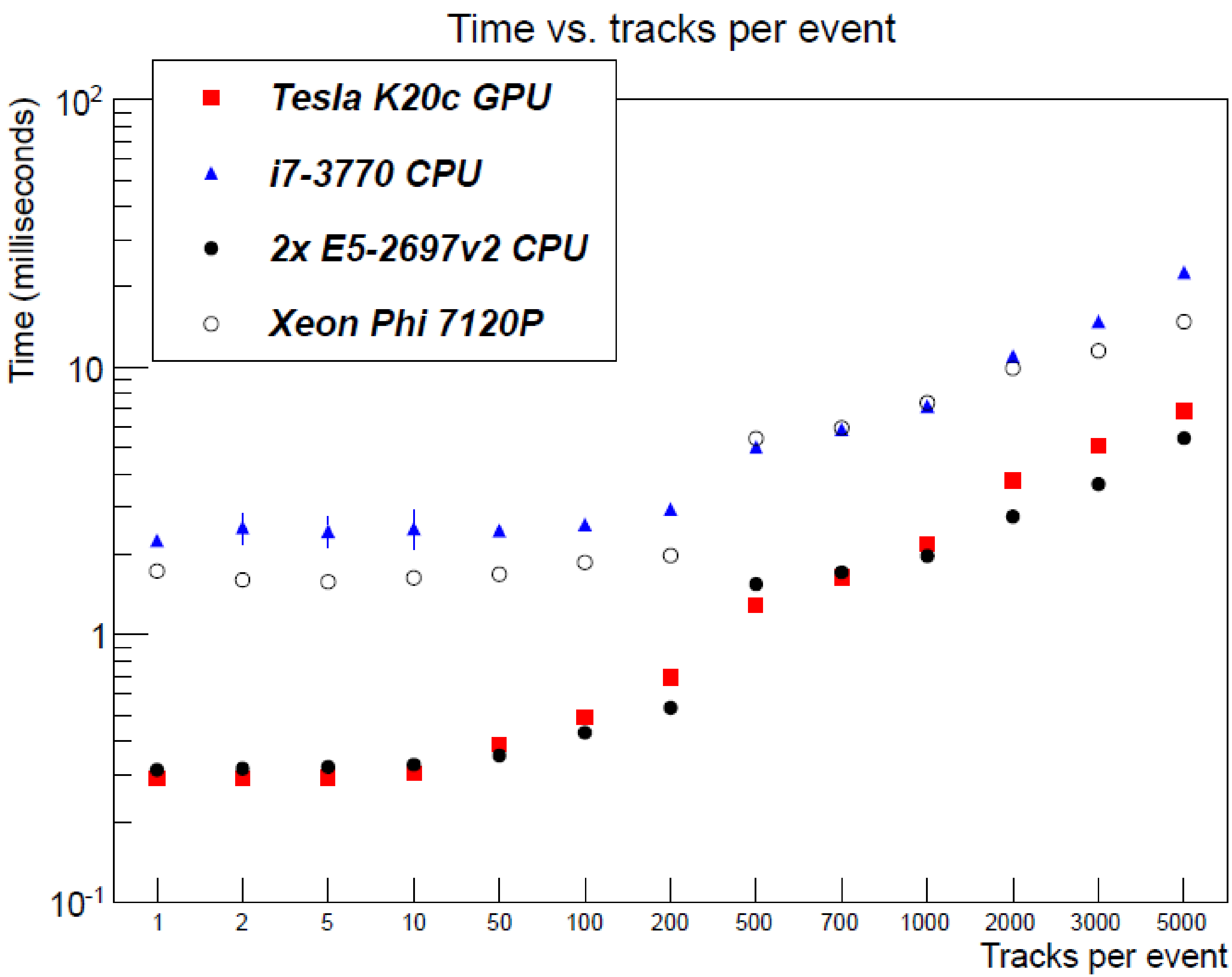
Performance Testing

- How can we determine which architecture is the best?

Implementation of Hough Transform

- CPU version (included for comparison) on Intel i7 using OpenMP for parallelism
- GPU version implemented with CUDA and tested on Tesla K20c
- Xeon version tested on dual-socket Intel Xeon CPU (E5-2697v2) and Xeon Phi coprocessor
  - Code written to allow compiler to perform automatic vectorization
  - In terms of power consumption, the dual-socket Xeon CPU is roughly equivalent to Xeon Phi
- Size of event is small, so no problems with data transfer

Results



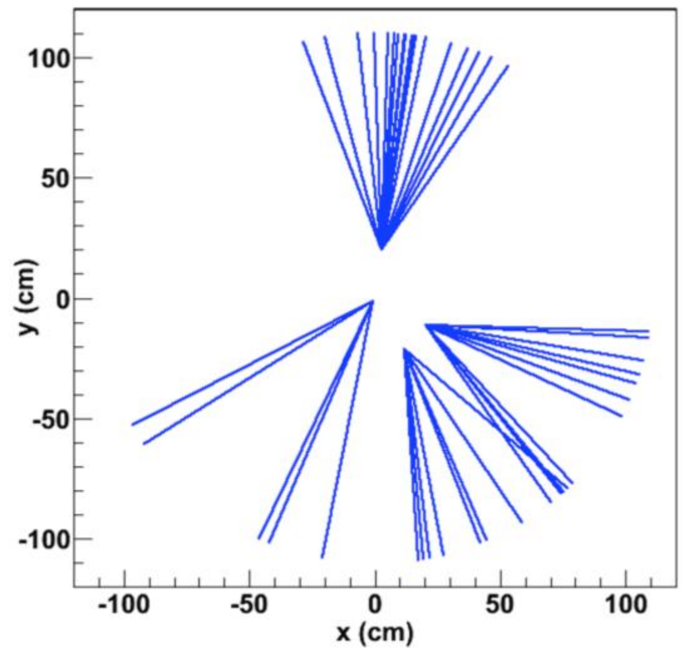
NIM A **744**, 54 (2014)

Discussion

- Adding parallel computing can provide a significant speedup
- Not clear which architecture is best – can vary depending on problem
- Difficult to optimally parallelize this problem – memory accesses are not efficient for these architectures
- Not an easy task to implement in current CMS software!

Displaced Vertices

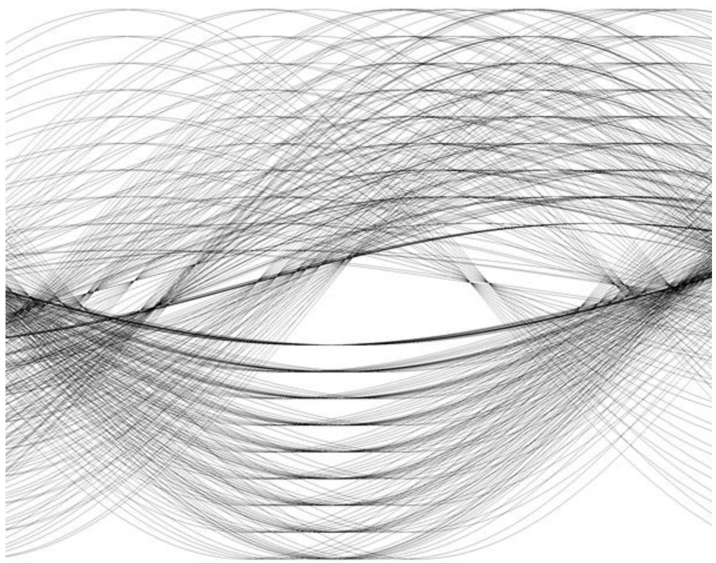
- Many of these theoretical models predict not only displaced tracks, but displaced **vertices** – an even clearer signal of new physics
- Displaced vertices could arise from jets, or black holes, or other new phenomena



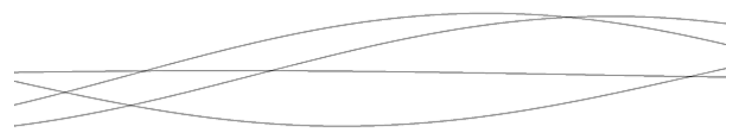
sample simulated event with four displaced jets

Identifying Displaced Vertices

- We can quickly and easily identify these displaced vertices by employing the Hough transform a second time!



The first Hough transform identifies the tracks…



the second finds locations that correspond to intersections of the tracks – the displaced vertices!

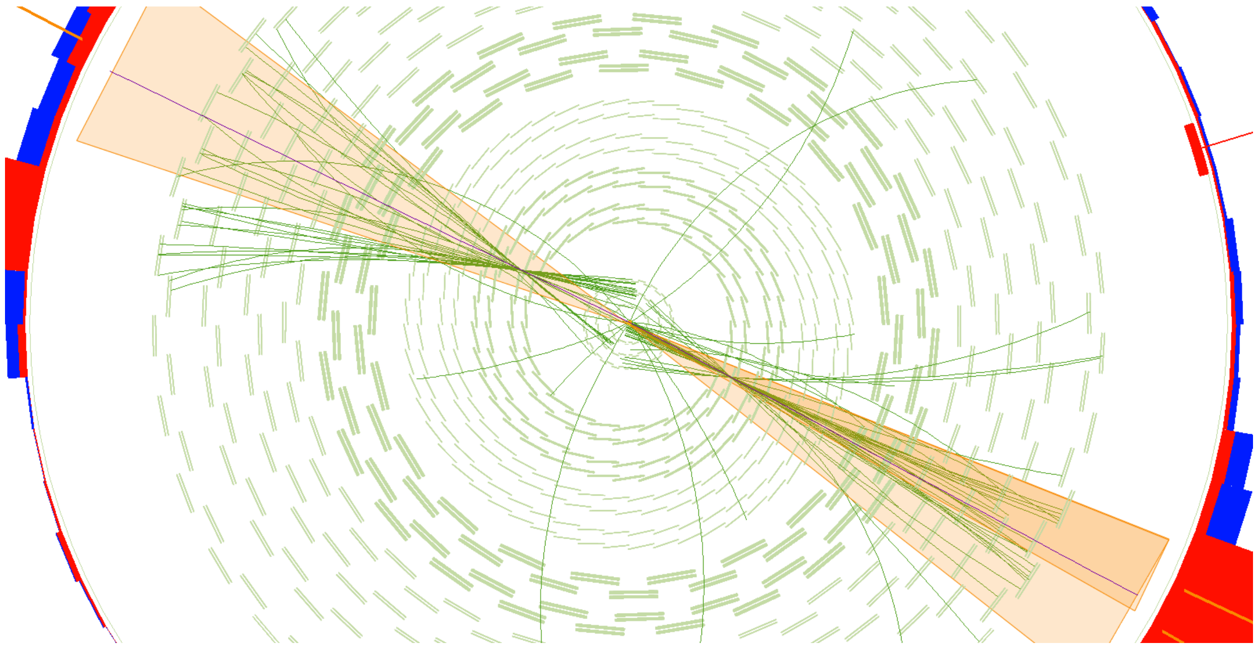
JINST 8 P10005 (2013)

Discovery of New Physics

- Let’s look at an example that we can’t find currently
- The key is to be able to trigger on events with these displaced topologies

Missing the Displaced Higgs?

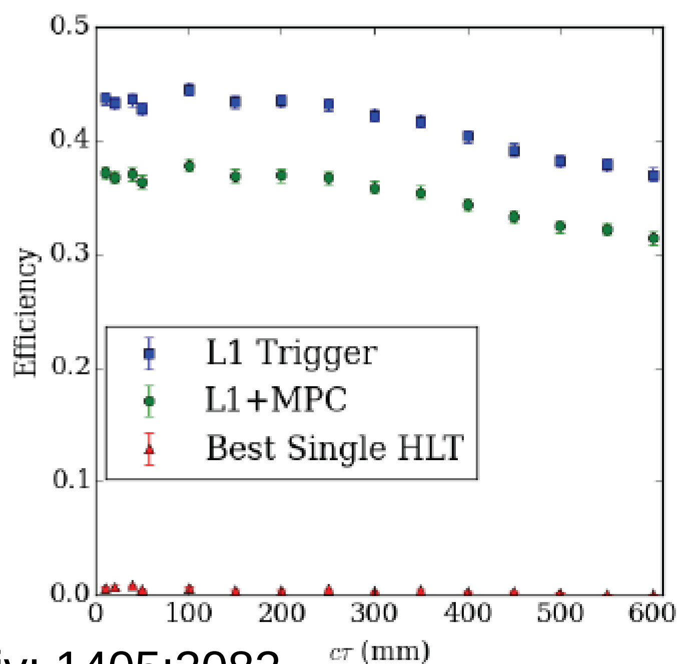
- Consider a specific model:
  $H \rightarrow XX \rightarrow bbbb$

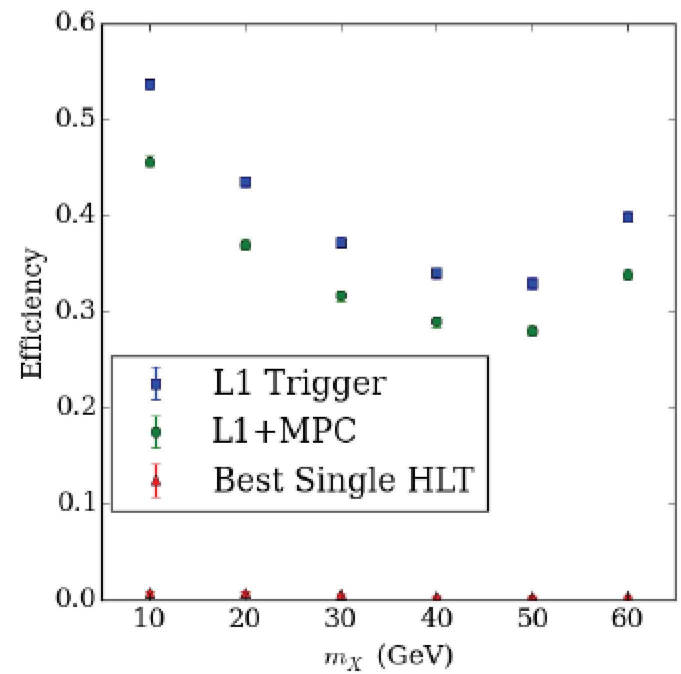


- Very difficult to detect with current triggers for smaller m_H (~125 GeV/c^2) – **losing the potential for discovery!**

Discovery Potential of New Triggers

- Currently, such events are very difficult to trigger on
- With a parallel-based trigger, could increase efficiency from <1% to ~30% at m_H = 125 GeV – bringing discovery within reach!





arXiv: 1405:2082

Conclusions

- New tracking algorithms allow us not only to improve time performance, but search for entirely new models of physics not currently accessible.
- Parallel computing make these algorithms possible, but it is not an easy task to tell which architecture is best or to implement them in the software environment at LHC.
- Much work lies ahead, but the potential is great.