HL-LHC: Software Prospects

Graeme Stewart
for Trigger, Online, Offline Preparatory Group

Additional thanks to Peter Elmer, Vincenzo Innocente, Rolf Seuster, Marco Cattaneo, Markus Elsing, David Michael Rohr, Ian Bird, Pere Mato, David Rousseau, Sandro Wenzel

University of Glasgow | School of Physics & Astronomy

2014-10-23

Overview

- Software Costs
- Modern CPU Architectures
  - How to get the most from current and future designs
- Frameworks for the Future
  - O2, CMSSW, GaudiHive
- Tracking and Simulation
  - Getting faster without losing performance
- Collaborative Efforts
- Conclusions

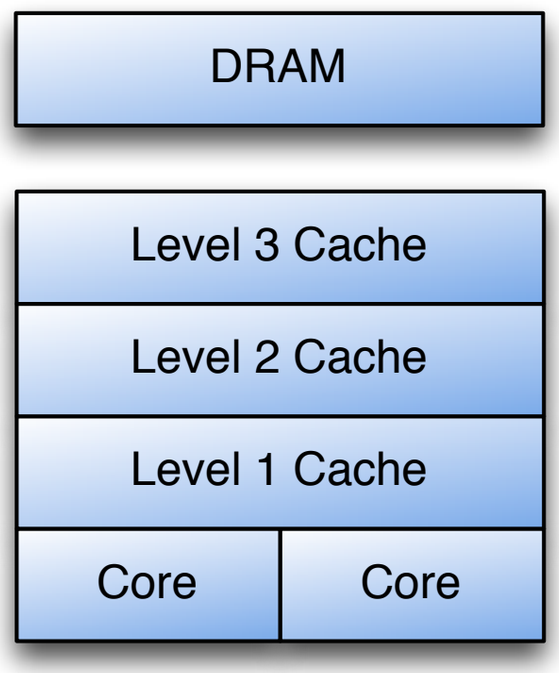
Caveat Emptor

- Computing and software in HEP utilises commodity hardware, which is driven by forces well outside of our control
  - Basically the trends here are to drive the rapid development of low power devices: tablets, phones, wearables
  - Supported by a backend of data centre machines providing compute and data intensive services
- In both cases figures of merit are increasingly based around energy consumption (nJ/instruction)
- In any case, an important consideration for us is architectural flexibility
  - No more x86 mono-culture:
    - ARM64, PowerPC, Nivida CUDA, AMD GPUs, etc.

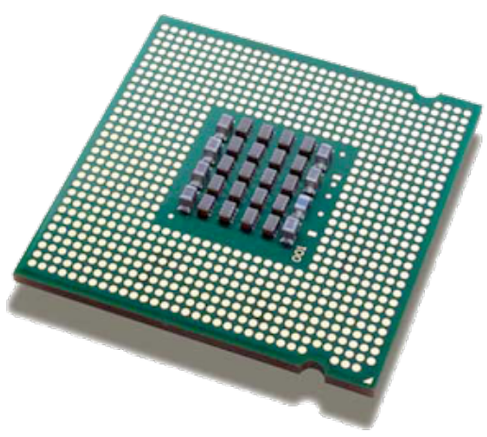
Costs of Software

- Costs:
  - Hardware investment
  - Site running costs (from electricity to people)
  - Software development
  - Software maintenance
- Benefits
  - Value of the physics outcome
- So, like all other subsystems, software has to produce excellent physics, but fit in our budget
- Here I will concentrate on software running on a single box somewhere in one of our data centres (c.f. Mikolaj's talk)

Processor Back Ends

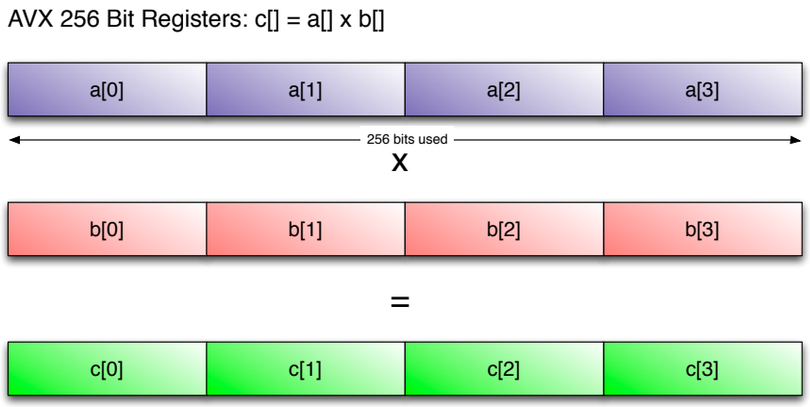


DRAM	Multi-GB Main Memory	200 cycles
Level 3 Cache	8MB Cache (shared)	40-70 cycles
Level 2 Cache	1MB Cache	10 cycles
Level 1 Cache	32kB (Data and Instruction)	4 cycles
...	32 x 64bit registers	1 cycle



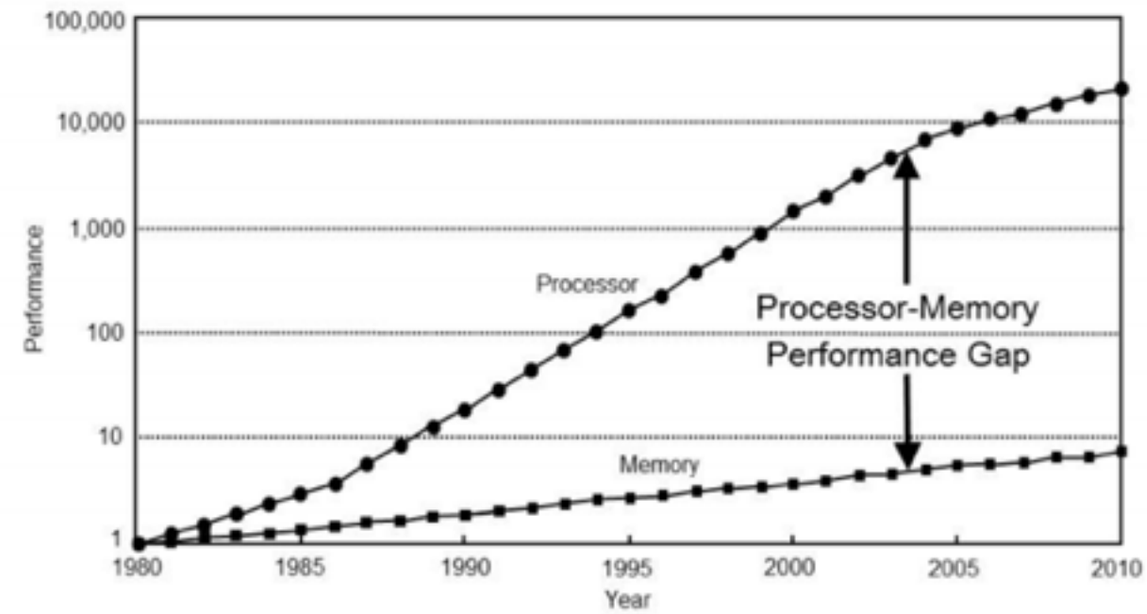
- Costs of waiting for data on modern CPUs is very high
- Need to pay attention to alignment issues as well

SIMD and Accelerators



- Efficient use of SIMD can bring benefits:
  - SIMD fused multiply add **saves 16** cycles over serial operations
- However, now need to load 16 floats and store 8
  - In the worst case this could **cost few 100** cycles
- Similar issues apply to serialisation and de-serialisation of data for accelerators
  - Data needs accessed, rewritten, shipped, recovered and then rewritten again
- Gains of advanced CPU features, like vectorisation, are dwarfed by the loss of efficiency if memory layout is poor

Data Oriented Design



- Trend in high throughput computing to re-orient designs around data rather than ‘objects’
- The point of the program is to transform data from one form into another
  - If you don’t understand the data, you don’t understand the problem
- Have to base reasoning about costs on understanding the data and understanding the hardware
  - There is no abstract solution that works with all data on all hardware
  - Have to maximise the use of all data that is fetched from main memory (think cache lines!)
    - Do not pollute caches with data you know will not be used again

See Mike Acton, Data-Oriented Design and C++, CppCon 2014

Memory Wall

- Continued march of number of transistors leading to increasingly multi/many cores
- Gap between CPU cores an *affordable* memory is increasing
  - Current xeon grid servers have 2-4GB of memory per core
    - Requirement is only 2GB, of course
    - Per hyper-threaded core this is halved: 1-2GB
  - Current Xeon Phi has 60 cores and 16GB of memory
    - ~256MB per core
    - ~64MB per thread to use efficiently
  - Tesla K40 has 2880 cores and 12GB of memory
    - ~4MB per core

To Threading…

- Tricks can be used to try and save memory, even with multi-process models (e.g., ATLAS's AthenaMP using Copy on Write)
- However, these come nowhere near the likely memory savings required for non-Xeon server architectures
  - Thus it is necessary to use a multi-threading framework
  - Memory savings can be huge as all heap memory is shared
  - However, a more difficult programming model as threads can interfere with each other: data races and deadlocks
- Especially difficult to back-port threading into a framework and physics code base which has been run in a serial mode for a decade

Concurrency Toolkits

- Low level toolkits for concurrency (pthreads, C++11 threading) are not easy to use and beyond the level of our typical physicist programmers
  - Used more as a base on which to build a more functional toolkit
- There are many toolkits on the market that are supposed to help with this
  - This indicates that this is a hard problem and mature solutions don't really exist
    - Many solutions target different niches
  - We need to take the best of what's available right now, without mortgaging our future

OpenMP

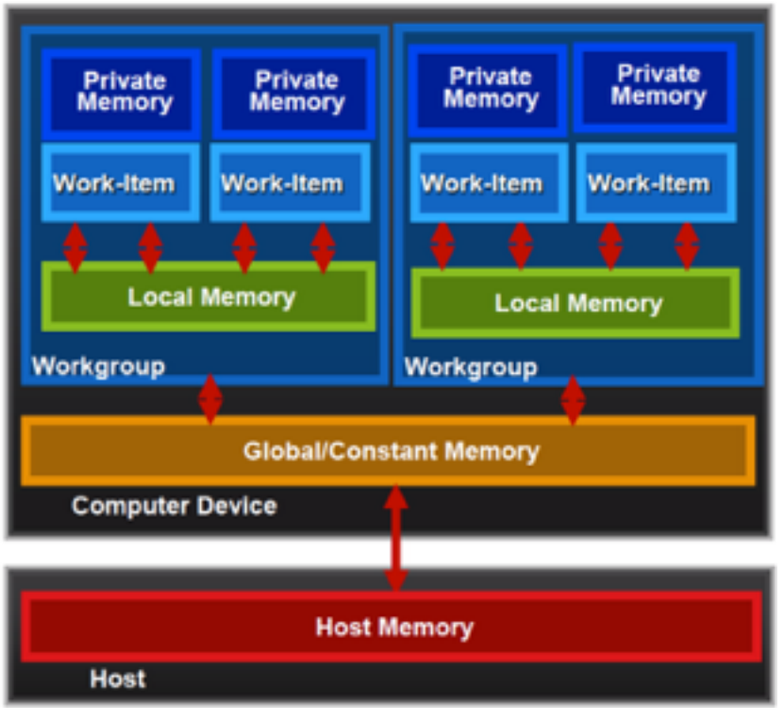
- Long standing extension to C++, C and FORTRAN (v1.0 from 1997)
- Uses a #pragma directive to tell a compatible compiler to parallelise, originally just for multi-threading
- With v4.0 adds support
  - For SIMD parallelism: parallelise function execution down different vector lanes (very GPU-like)
  - For offloaded execution
- Generally very suitable for ‘hotspot’ parallelism, hence it’s popularity in HPC

Threaded Building Blocks

- High level toolkit for managing concurrency in C++ (GPL)
  - Not oriented at threads at all, but at parallel tasks
  - Which is rather more akin to our problem domain: high level pattens supported directly
- Very flexible task scheduler with low overhead
  - Good for framework building
- Support utilities for concurrency: thread safe containers, fast threaded malloc
- Has become something of the *de facto* standard in HEP

CUDA and OpenCL

- OpenCL toolkit designed to abstract the programming of tasks for accelerators
  - Vendor independence
- Has an *explicit memory model*
  - Actually matches the layout of memory on modern devices
    - But don't we need to understand what we're doing here anyway?
- However, generally performance isn't as good as Nvidia CUDA and CPU performance is distinctly lacklustre
- Always playing catchup?

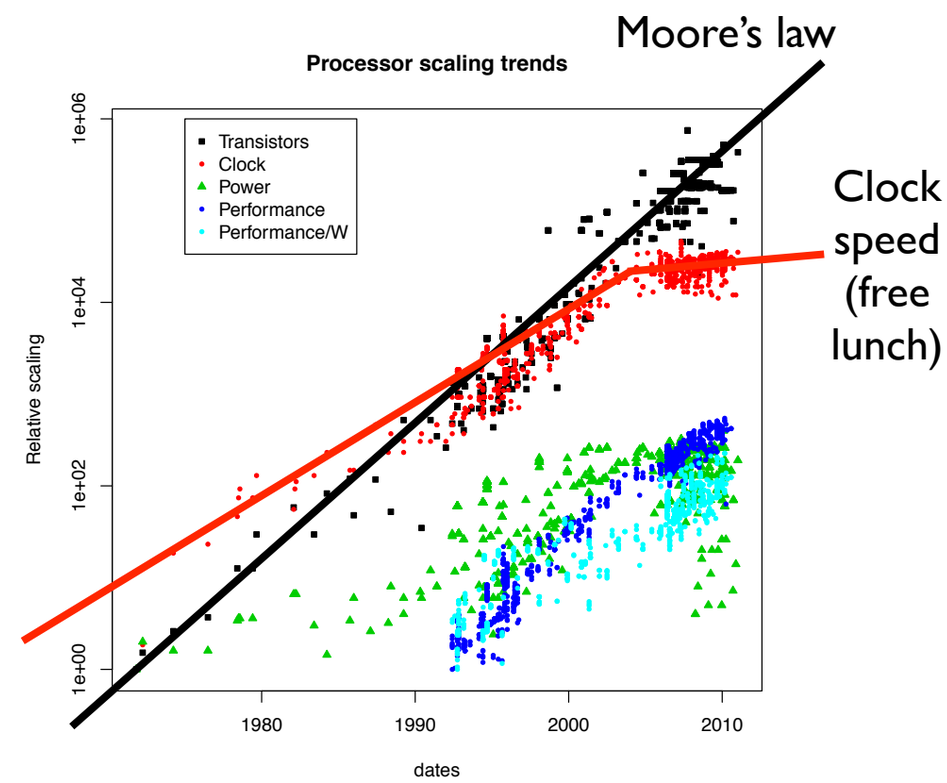


Hooking in to GPUs

- ALICE scheme involves a build time declaration that switches in the correct code
  - Keeps code as common as possible, but switches between CPU and GPU as needed
  - Very good for targeting a cluster known at compile time
- ATLAS have a demonstrator project (APE) that will do a similar thing, but using runtime switching
  - Compile both versions of the code into modules
  - Can switch between modules based on node characteristics
  - Good strategy when the job might fall on clusters with a different accelerators (including none)

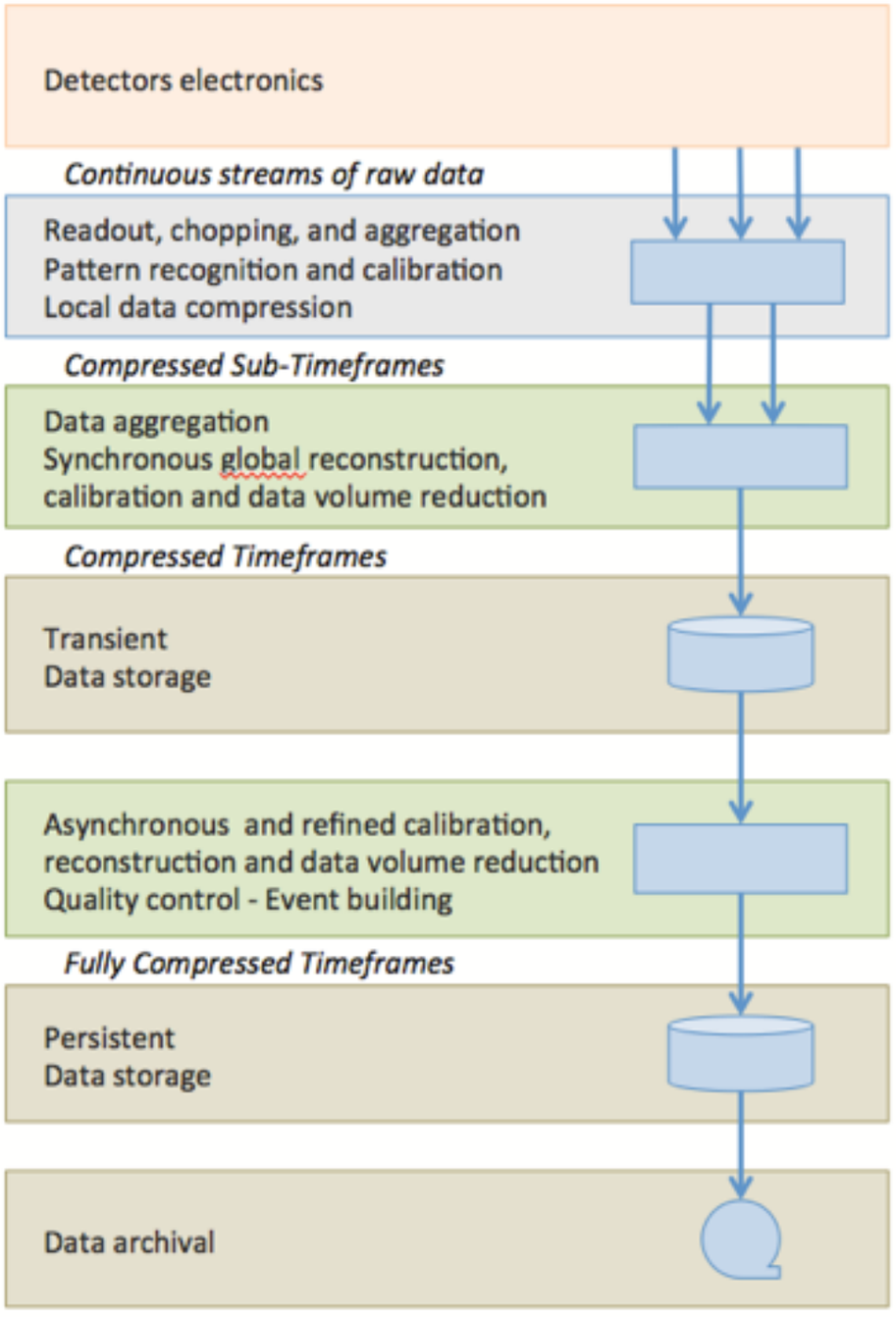
Framework Evolution

- LHC experiments frameworks were designed ~>10 years ago
  - In the era of the 'free lunch' (regular improvements in clock frequency) with one increasingly fast CPU core per chip
  - Concurrency was basically a non-issue
- Landscape has changed greatly now
  - Frameworks need to evolve to provide a parallel environment in which physics code can run
    - Thread safe data access for multiple events
    - Management of thread-unsafe resources
    - Algorithm scheduling
    - Handling thread safe an non-thread safe algorithms and tools



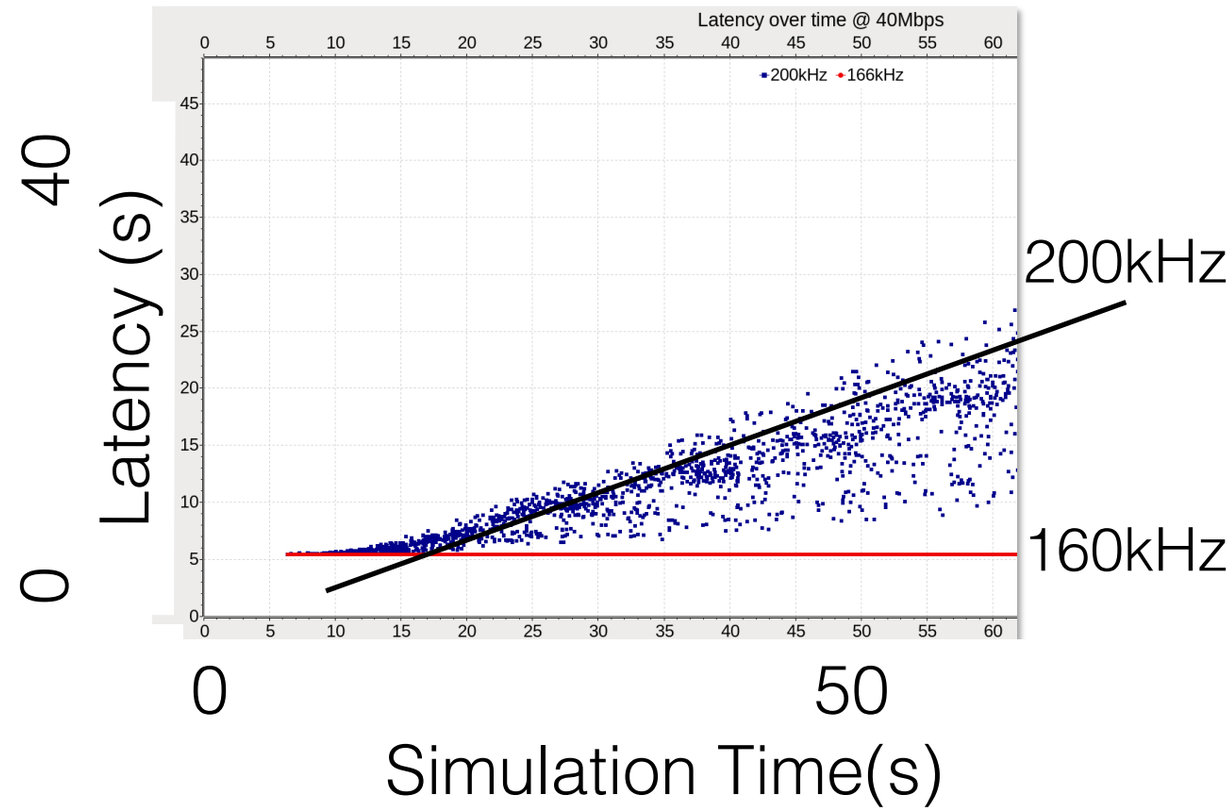
ALICE: O^2

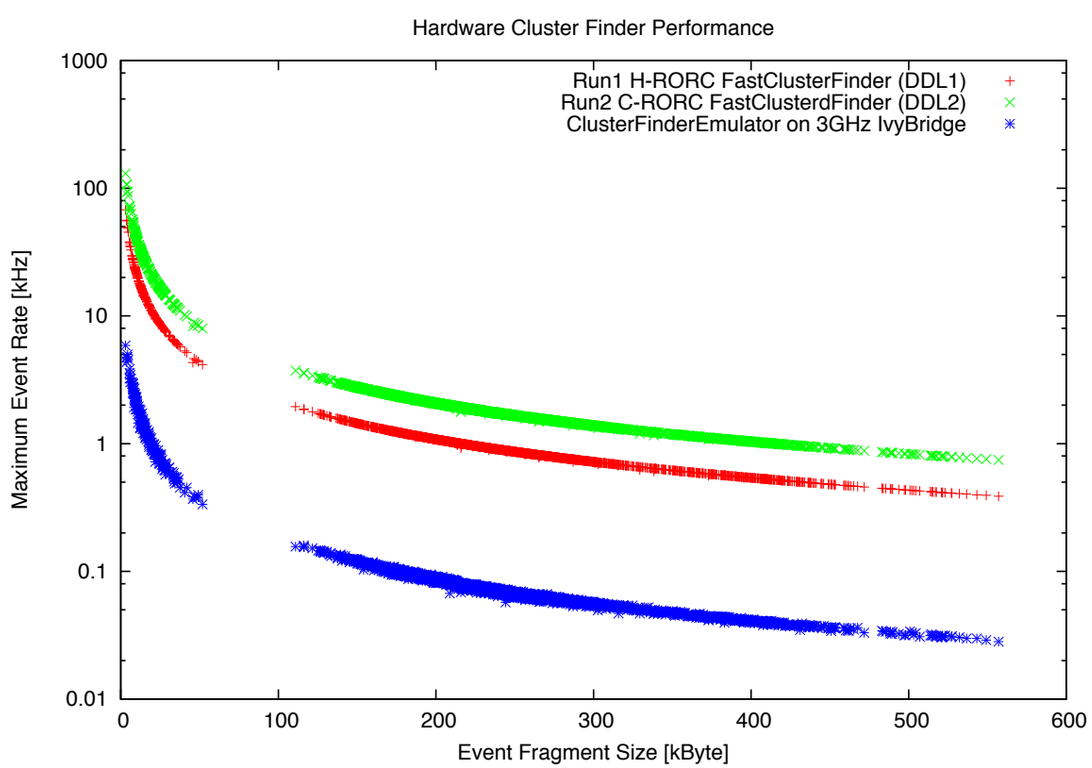
- Design an online and offline software framework for ALICE that supports data flows and processing
  - Reflects a general trend to blur the distinction between online and offline
- Handle >1TB/s detector input and continuous readout from TPC, but Reduce data to acceptable rates for offline storage, 20GB/s average
- Optimal use of compute nodes
- Take full advantage of CPU resources, including hyperthreading, and allow for use of GPUs and FPGAs
  - ZeroMQ allows resources to be connected with very low cost IPC
- Project promotes use of common ALFA development framework
- TDR due sumer next year



O^2 Modelling and Performance Tests

- O^2 scaling test to 160kHz

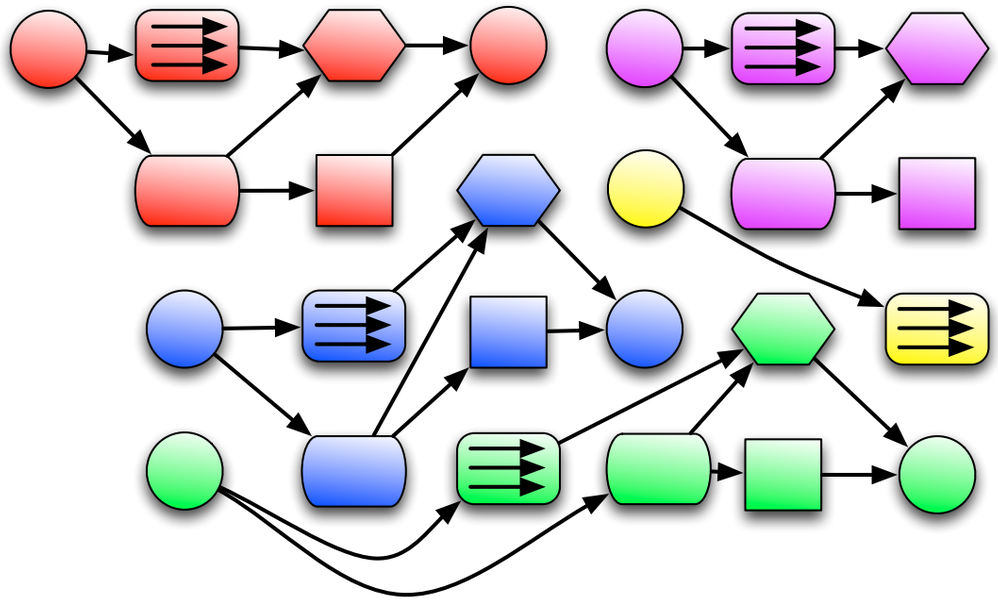




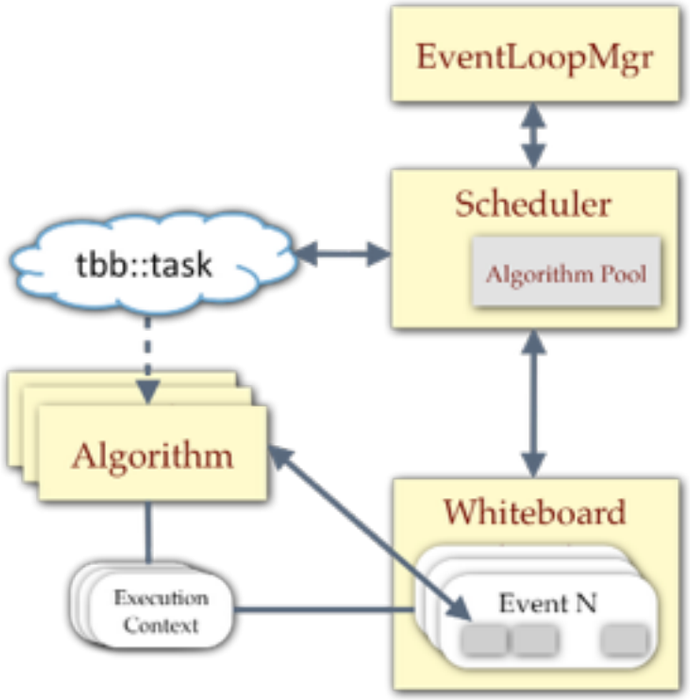
- Comparison of cluster finding rates on FPGA vs CPU (higher is better)

GaudiHive

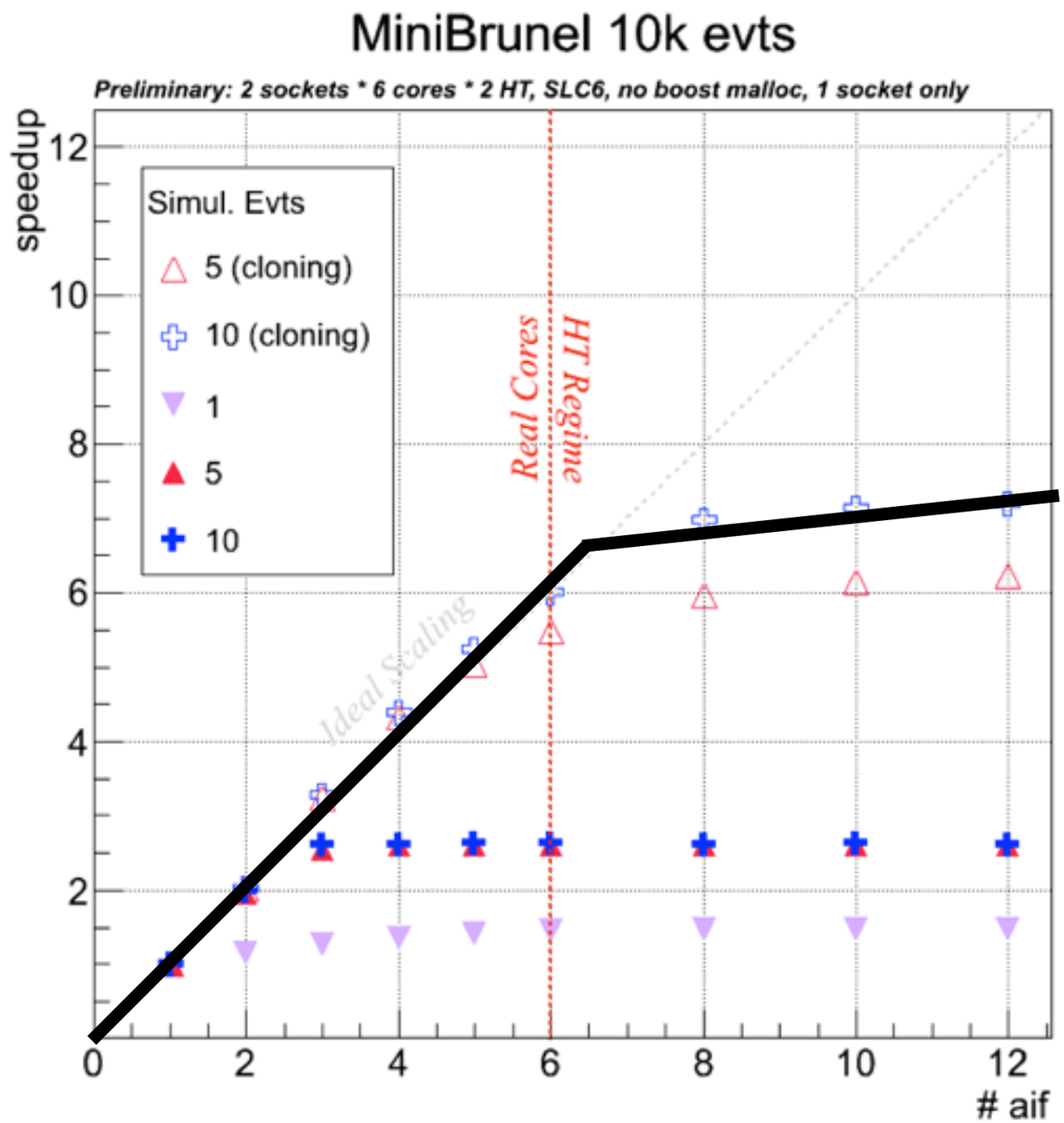
- Development to introduce parallelism into the Gaudi framework used by ATLAS and LHCb
- Take advantage of parallelism between algorithms and across multiple events
- Scheduler is data flow driver, but control flows can also be given (important for online)
- Utilises Threaded Building Blocks
- Considerable experience now with design patterns which are good for threading and anti-patterns which are not
- Encouraging results from mini-Brunel example (LHCb) and CaloHive (ATLAS)



Colours represent different events, shapes different algorothms



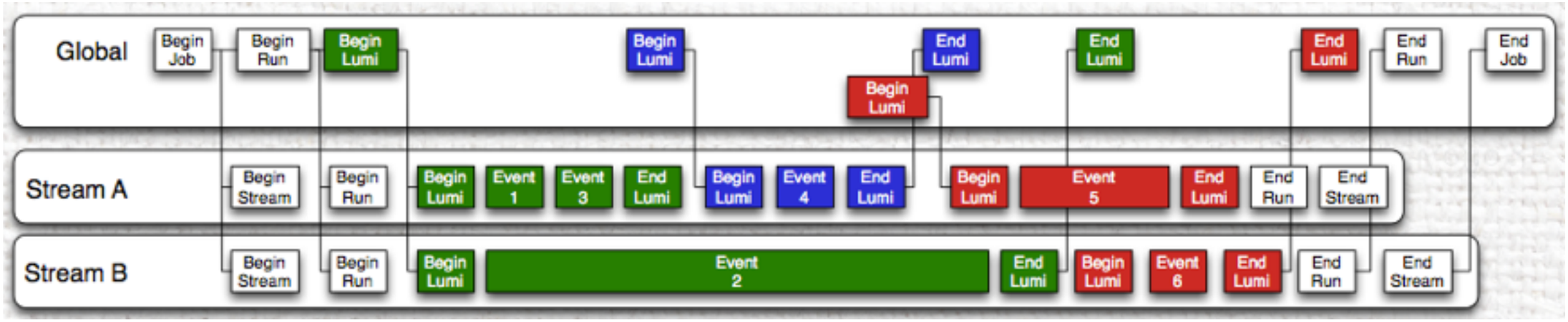
GaudiHive Scaling Tests



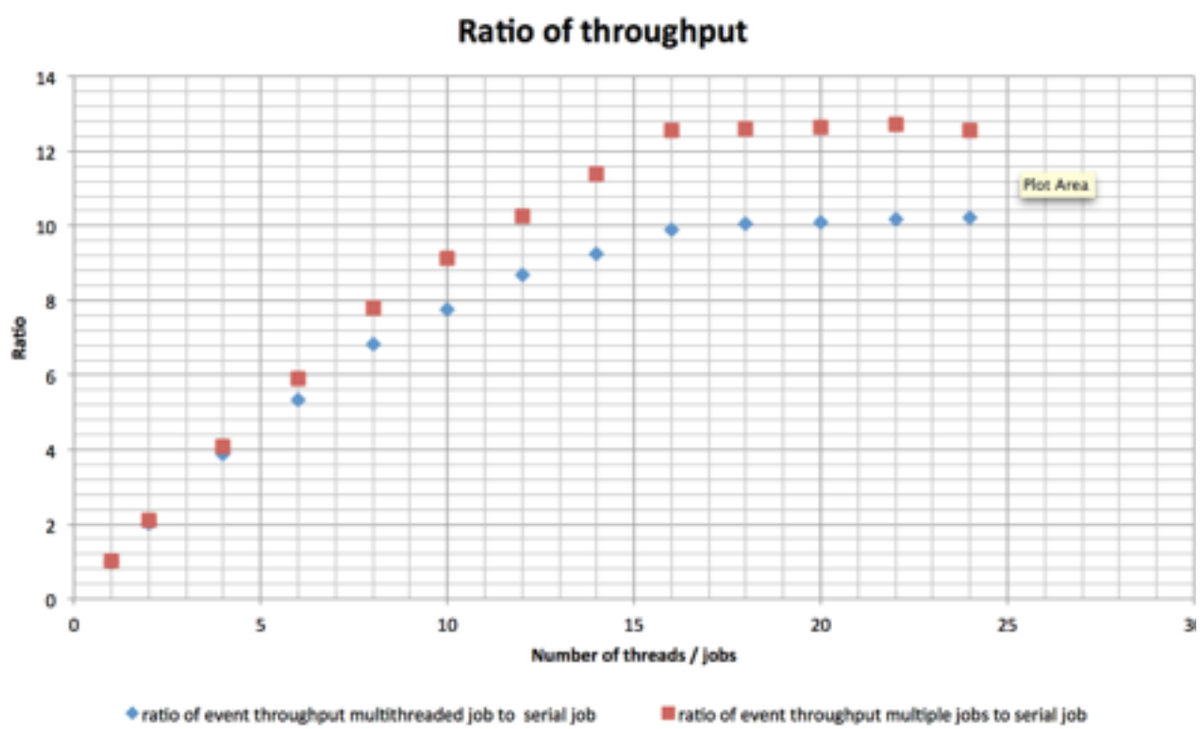
- Scaling of GaudiHive running LHCb mini-Brunel reconstruction
- Linear scaling up to CPU core count
- Expected boost from hyperthreading with only 10 events in flight
- Memory consumption only rises by 7% (limited reconstruction however)

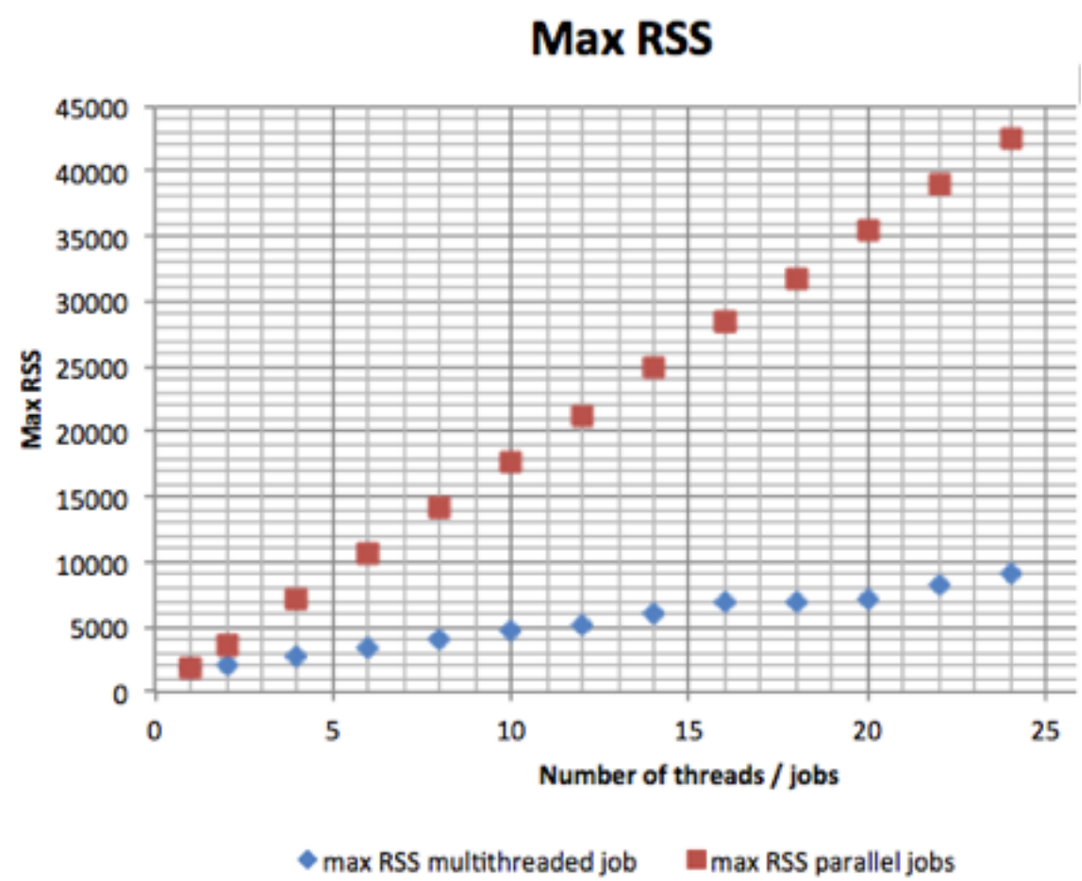
CMSSW Multi-threaded

- Split the concept of event processing into global and stream
  - Global sees the whole event and all transitions
  - Stream sees some events, in a defined sequence
- Underlying toolkit is again TBB
- Thread-safety is vital at the global level, less important at the stream level
  - Allows for a factorisation of the problem for framework transition
  - Good use made of static code checkers
- Good scaling of throughput and memory consumption



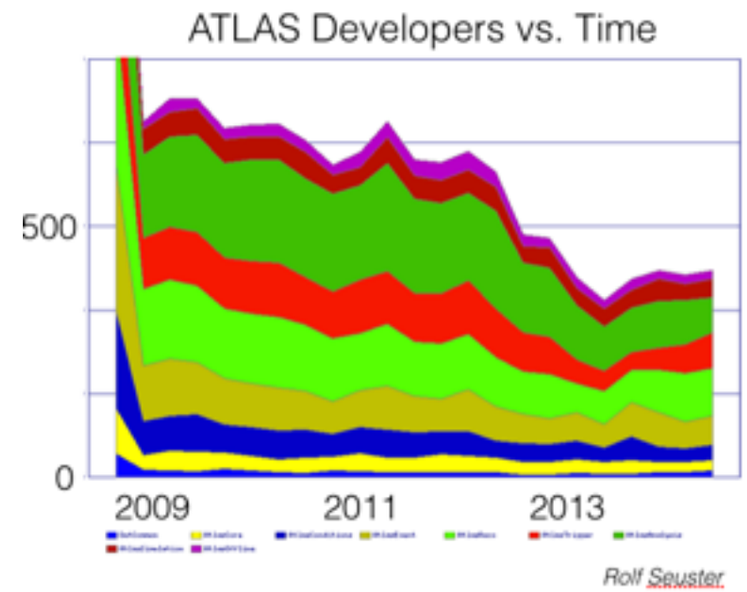
CMSSW Results





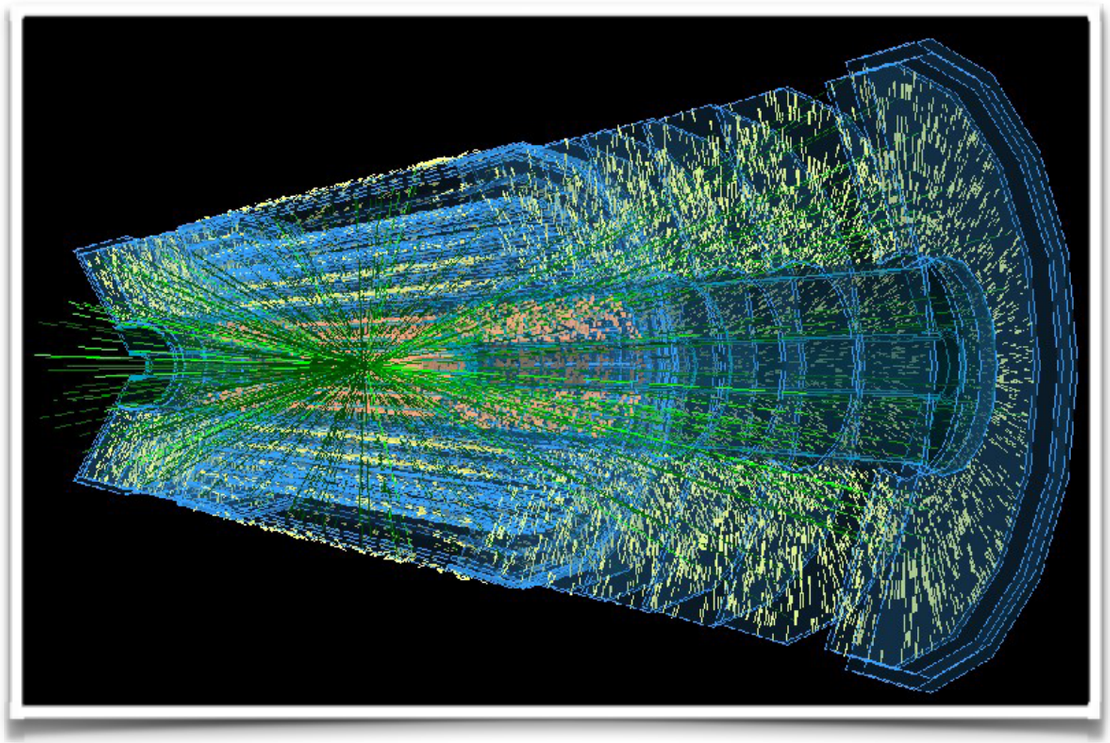
- High throughput maintained by multi-thread job
- Memory consumption hugely reduced

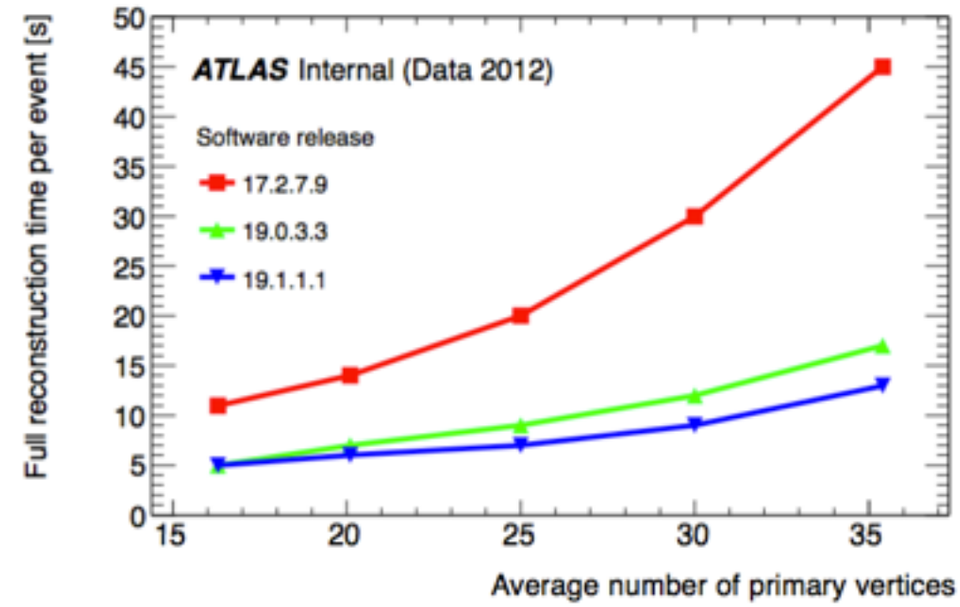
Algorithmic Code Evolution

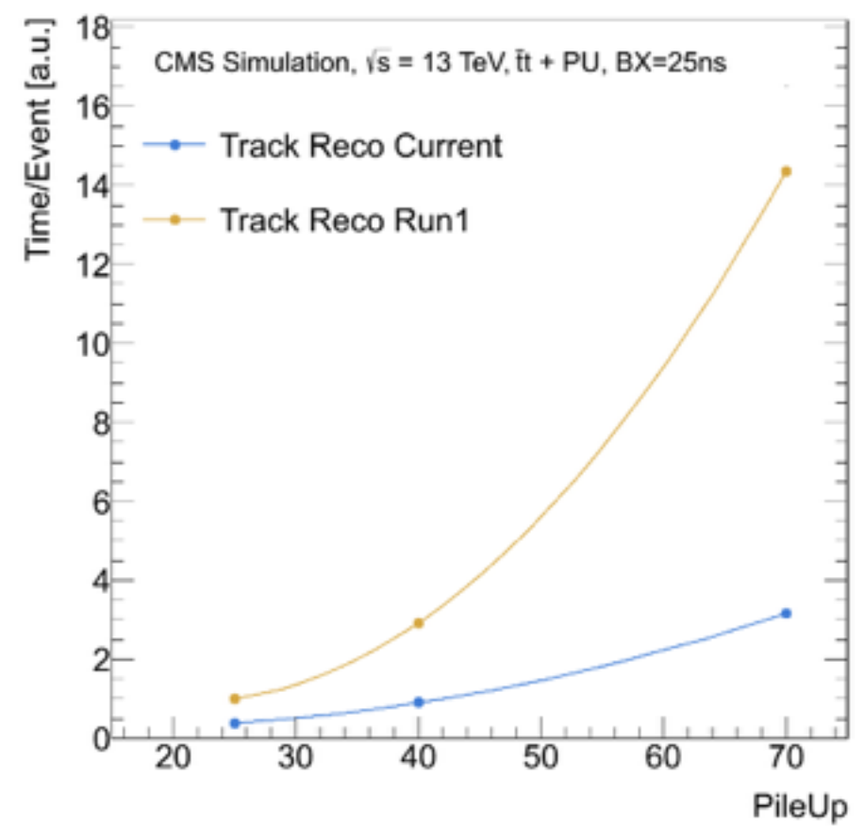


- Highest investment in algorithmic code
  - Vast majority of offline packages
- Migration of this code to new concurrent frameworks is not trivial
  - And is independent of the framework migration (in fact it is the lion's share of the work)
- Essential to provide good models for programming
  - Given that most of our coders are physicists, supported by a very few coding experts
  - Capturing software effort has definitely proved difficult when real LHC data exists
- In algorithm parallelism is essential for the most expensive CPU cycle consumers
  - However, machine resources need to be controlled at a global level
  - Multiple threading toolkits are not aware of one another and will overcommit resources
    - Loss of efficiency from unnecessary context switching
  - Frameworks have to provide concurrency services to single algorithms

Tracking



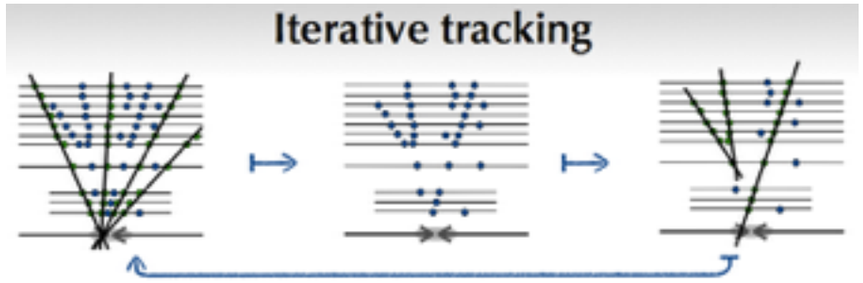




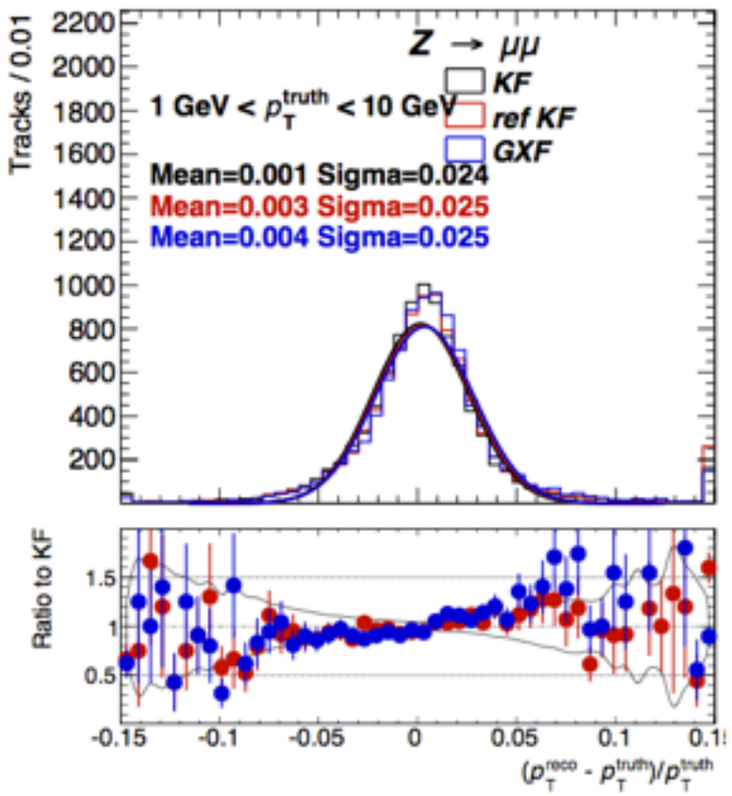
- High luminosity means high pileup
  - Combinatorics of charged particle tracking become extremely challenging for GPDs
  - Generally sub-linear scaling for track reconstruction time with μ
- Impressive improvements for Run 2, but we need to go much further

Current and Future Strategies

- Current strategies are based on early rejection
  - Serial chain of algorithms designed to beat down combinatorics
- However, serial dependencies between algorithms limit concurrency opportunities
  - Can try throwing more events at the problem
    - More events consume more memory and we may run out of memory before filling all cores
  - Alternatively, seek algorithms that cost more CPU, but allow for a higher number of cores to be brought to bear on the problem
    - More CPU cycles, but improve throughput
- Clearly a lot more studies are needed, but physics performance is paramount
  - Fast algorithms with poor results is not what we want
- Details of each experiment are very important in maximising the physics performance, but definite scope to share core strategies



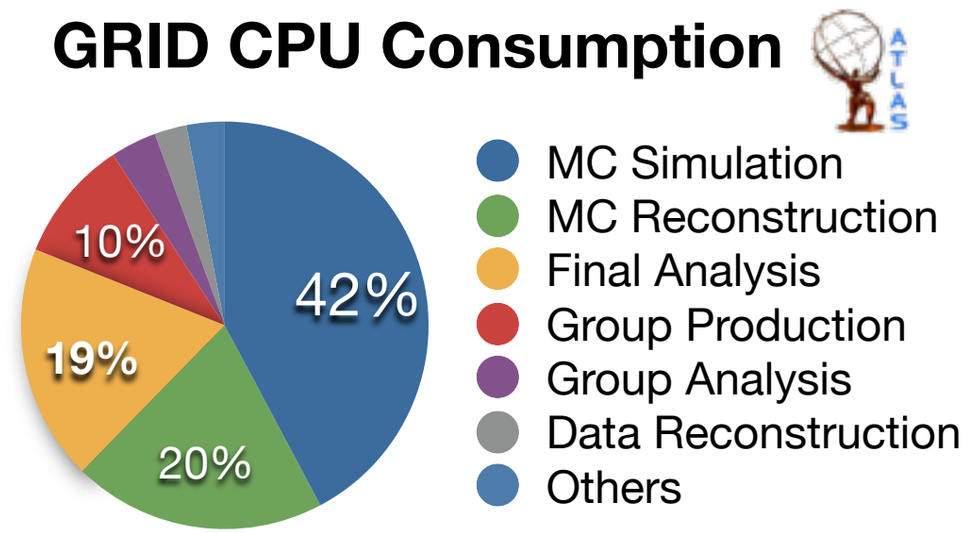
Iterative track finders are inherently more paralleisable



Comparison of standard Kalman Fitter vs. Kalman Fitter with Reference Trajectory

Sebastian Fleischmann

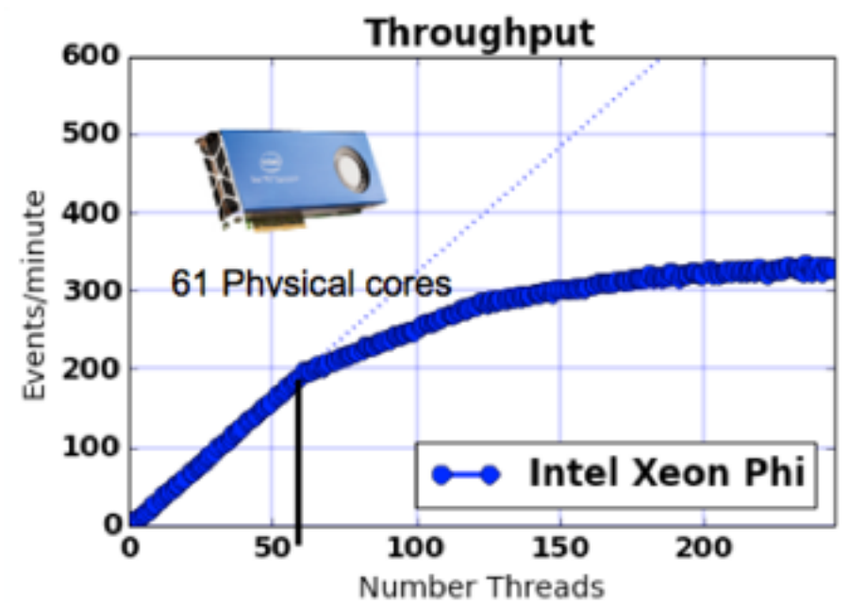
Simulation

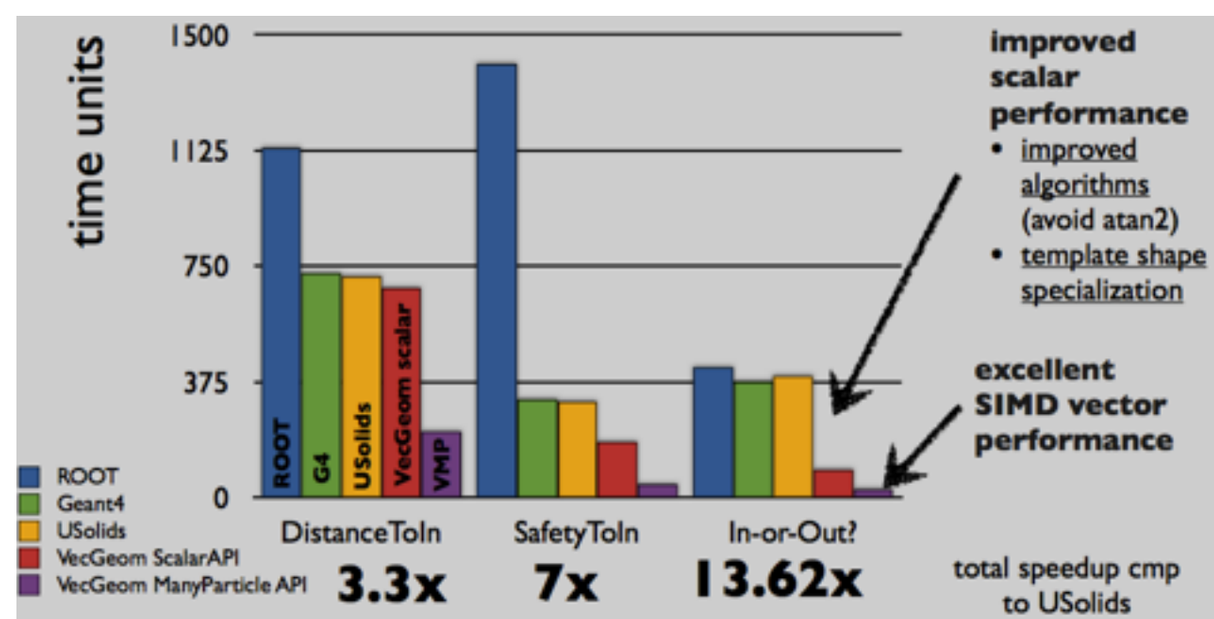


- Simulation is a very large consumer of offline computing resources
  - GEANT 4 full simulation of GPDs is expensive: up to ~1000 seconds per event
  - Simulation subject to the same general software issues as any other piece of software:
    - GEANT 4.10 adds multi-threading support
    - In the further future GEANT V explores vectorisation potential

GEANT Progress

- GEANT 4.10 split data into static and dynamic parts
  - Static heap shared, dynamic in thread local storage
  - Spawn individual events on each thread
  - Massive memory savings and thus impressive scaling
- GEANT V vector prototypes use new VecGeom classes
  - Vectorise for specific targets using templates
  - i.e., plugins for different backends
  - Aim is to have large performance gains with few interface changes for experiments

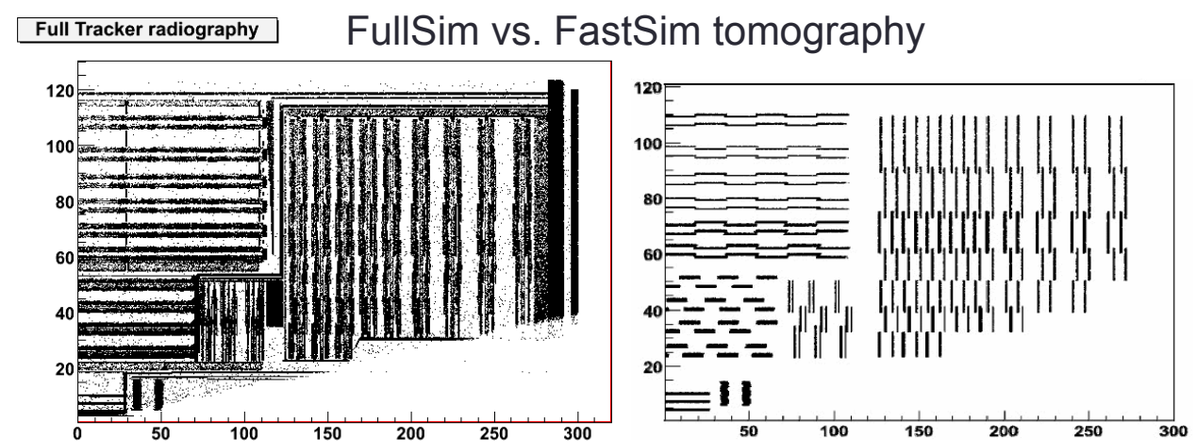




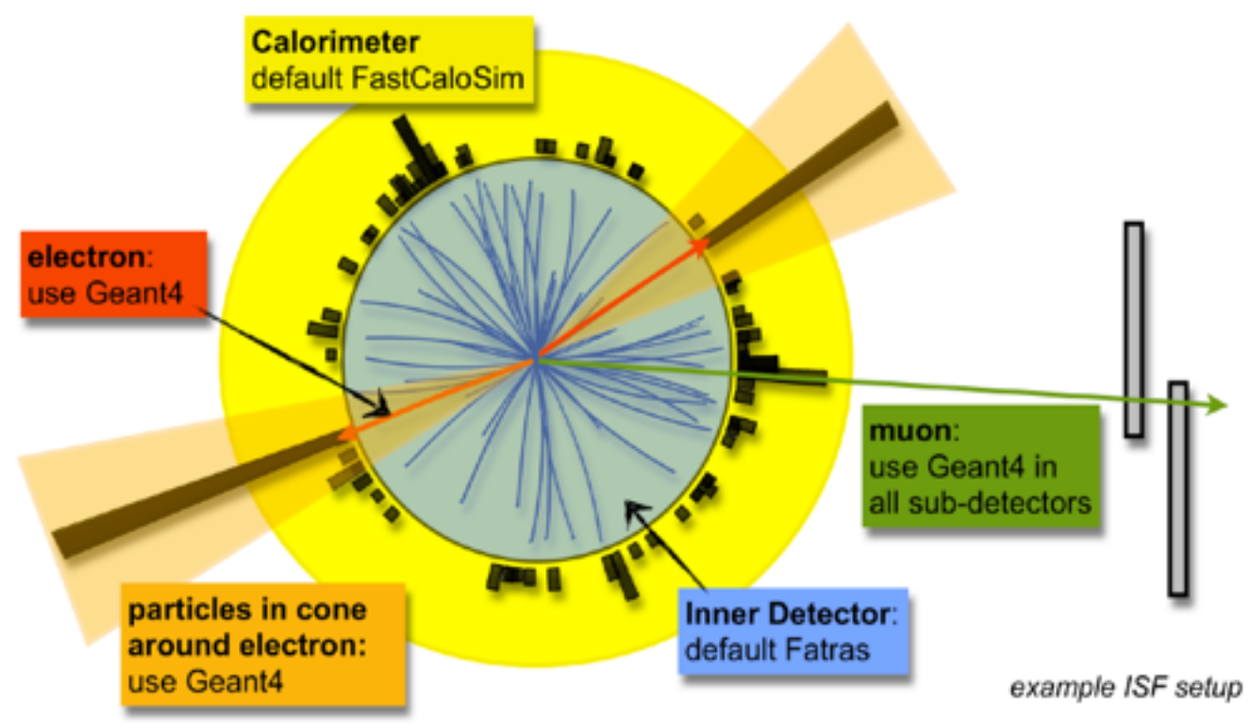
Fast Simulation

See Fast Detector Simulation and the Geant V Project, ACAT 2014 Andrei Gheata

- Need to take advantage of fast simulation where appropriate
  - Tradeoff accuracy for speed
    - Smearing
    - Frozen Showers
    - Parametric techniques
- ATLAS's Integrated Simulation Framework allows clever mixing of fast and full simulation within the same event
  - Keep high precision for some particles and regions
  - Use fast simulation in areas that are not so important
  - x100 speed ups possible, with much better results than normal fast simulation
- Feedback ATLAS ISF experience into GEANT4

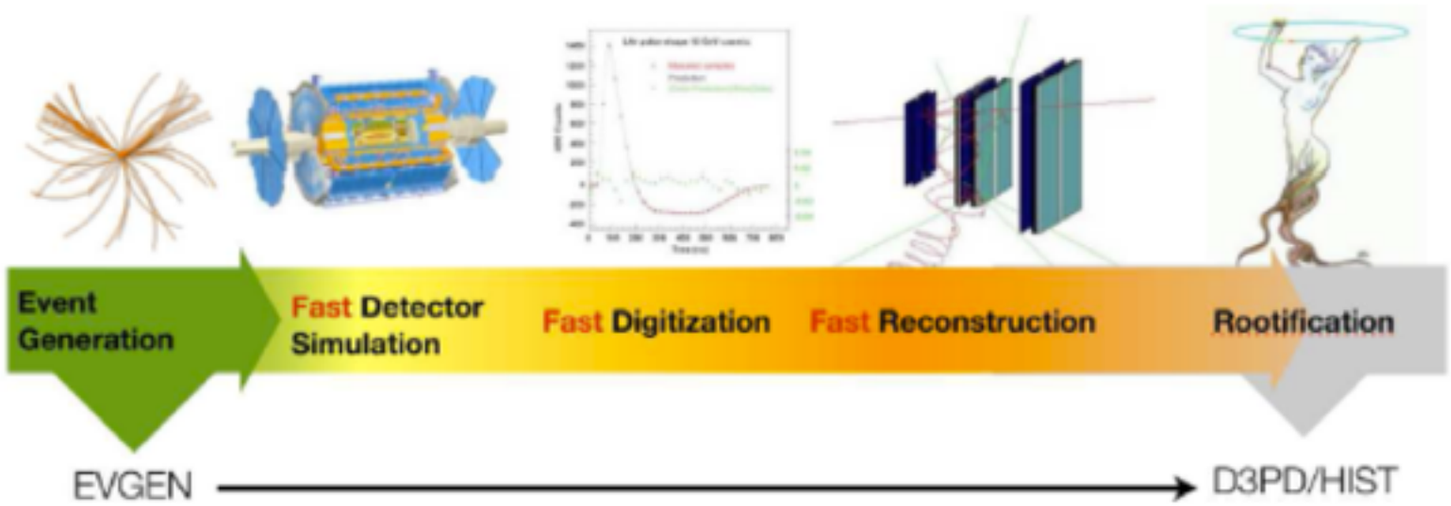


CMS FastSim Geometry

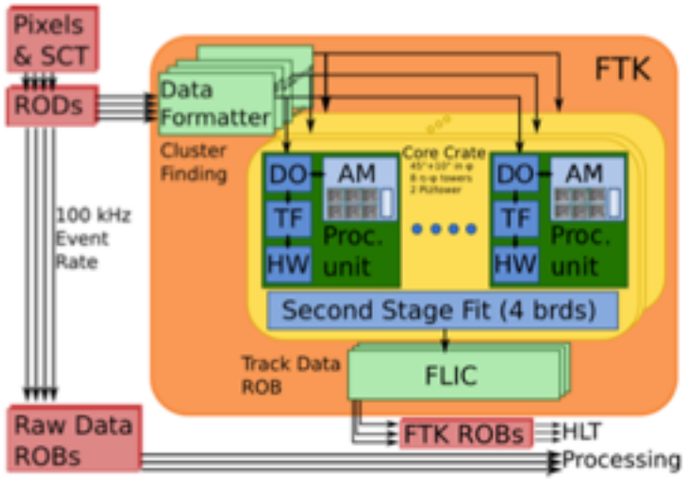


Fast Everything!

- As simulation speeds up other steps in the MC chain become bottlenecks
  - Digitisation
  - Reconstruction
- So there is a need to support other faster techniques to improve the MC chain from event generation to analysis outputs



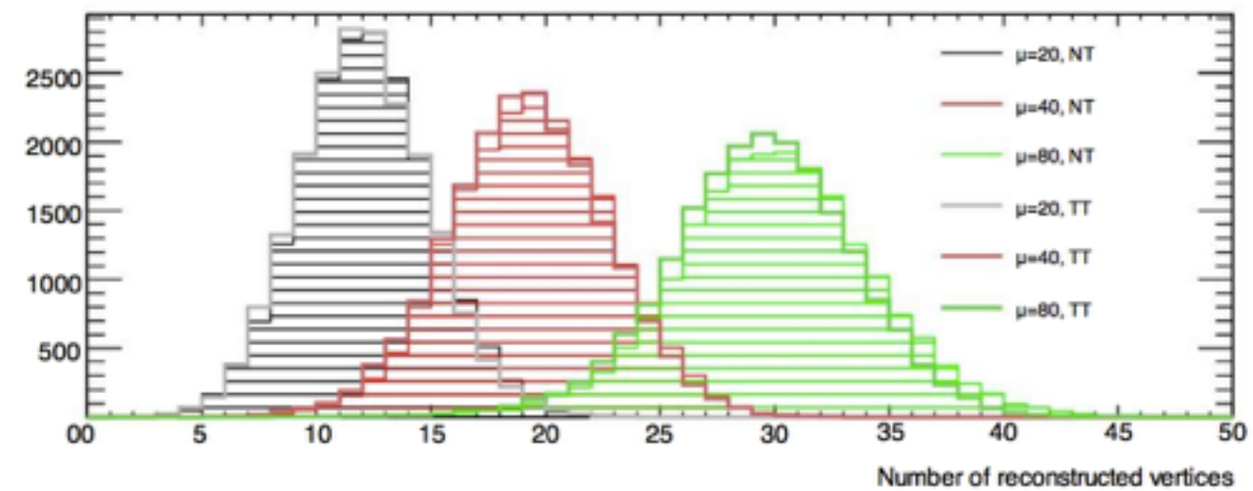
Track Trigger Simulation

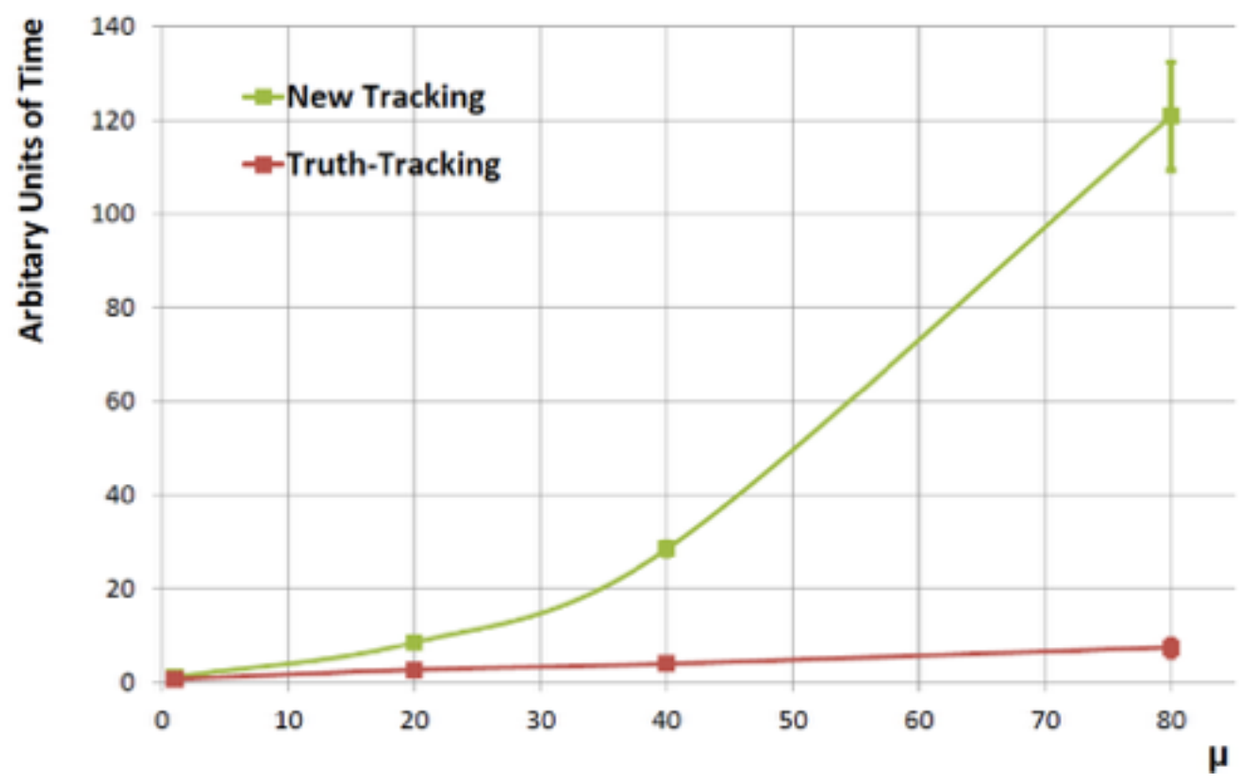


- A particular problem for ATLAS and CMS will be the simulation of the track trigger
  - This must be kept efficient in order to produce sufficient MC to keep systematics under control
- ATLAS FTK simulation already faces the problem of how to simulate associative memory devices
  - These are very expensive to simulate on CPUs as what an AM can do in one cycle takes CPU+DRAM millions of cycles
  - ATLAS simulation time for FTK currently ~300s per event for μ=80
  - This is too expensive for massive simulation
- Options:
  - Use a second set of FTK hardware for FTK simulation farm
  - Attempt to use GPUs on HPC sites to help
  - Use a ‘fast’ simulation that uses MC truth to determine if FTK would have a matching track
    - Useful as long as fakes do not dominate FTK performance

Fast Tracking

- Use truth tracks to generate track seeds instead of normal seed finding
  - Can restrict to only pileup events
- Allows for fast reconstruction of tracks from simulation
- Can we do a similar thing from *data* with utilising FTK and track trigger information to help offline tracking?





Community Efforts

- HEP Concurrency Forum has been a valuable place to exchange ideas across experiments
  - Concentration on technical problems
  - Annual meeting particularly useful for surveying progress
- New HEP Software Foundation could also play a positive role
  - Community needs to define how the foundation can help projects and experiment software development
  - January workshop at SLAC

Summary

- Software problems of HL-LHC are significant
- Evolution of computing hardware make realising ‘Moore’s Law’ type growth ever more difficult
    - We need significant R&D in the best strategies to use going forward
- Data oriented software design needed to tackle memory hierarchy problem
- Experiment framework evolution has already started and has made significant progress
- Fast simulation techniques can help close the gap between MC and high data rates
- Community initiatives help share knowledge and workload
- However, making up performance gaps of x5 or x10 is not at all easy
    - We will require skilled people, clever ideas and lots of hard work