Experience using GPUs for ATLAS Z finder algorithm

Second International Workshop for Future Challenges in Tracking and Trigger Concepts

Phil Clark
on behalf of the ATLAS collaboration

University of Edinburgh

7th July 2011

General Purpose GPUs

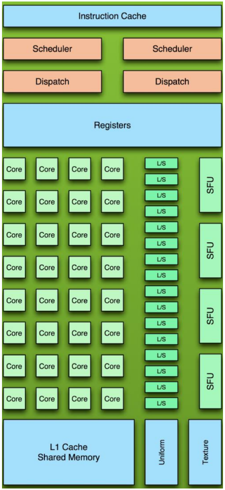
- GPU architectures are designed for running thousands of threads in parallel.
- Little additional overhead from running many threads.
- Suited to problems which can be performed in a data parallel manner.
- APIs allow the *host* to manage the GPU *device*.
- Several APIs and SDKs can be used for GPGPU programming: *Nvidia CUDA, OpenCL, AMD/ATI stream SDK.*

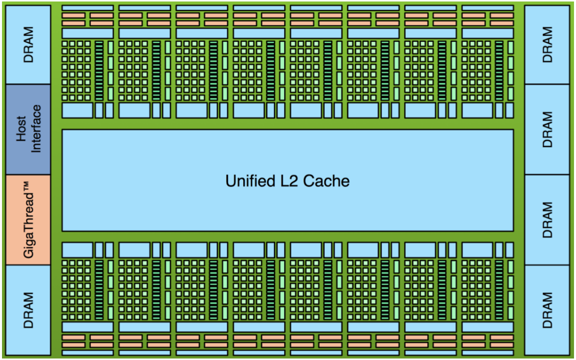
Where to start?

Nvidia CUDA zone

"Fermi" GPU

Images from Gernot Ziegler, Nvidia





CUDA Kernels and Thread Hierarchy

CUDA kernel

```
MyKernel <<< numBlocks, threadsPerBlock >>> (A, B, C);
```

- A CUDA kernel is a function which is executed in parallel by a number of threads on the GPU device.
- A *thread block* is a set of threads which execute together on a single multiprocessor.
- Thread blocks can be arranged into a one or two dimensional grids.

GPU memory

- CUDA devices contain different types of memory, each with their own properties.

Memory Type	**Size**	**Use**
Global	1GB+	Main memory storage on the GPU.
Shared	16/48KB (block)	Allows data to be shared between threads in the same block.
Registers	16/32KB (MP)	Stores kernel variable data (for each thread).
Local	16/512KB (thread)	Overflow for thread variable storage.
Constant	64KB	Automatically cached, read only.
Texture Memory	6-8KB (MP)	Streaming fetches with a constant latency.

GPU Projects at Edinburgh

- Number of GPU related projects at Edinburgh

- **Chris Jones** - *"Porting the Z finder algorithm to GPU"* (MSc in High Performance Computing)
- **Maria Rovatsou** - *"SIMT design of the High Level Trigger Kalman Fitter"* (MSc School of Informatics)
- **James Henderson** - *"An Investigation Into Particles Tracking and Simulation Algorithms using GPUs"*

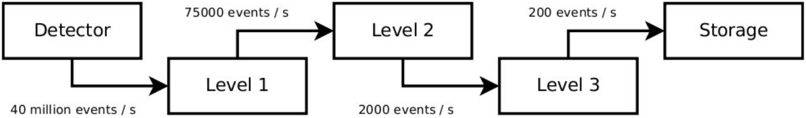
- Project reports and source code available at: ATLAS Edinburgh GPU Computing

Project Resources

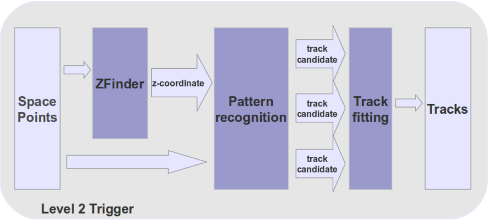
- Access to a number of dedicated GPUs with different architectures (*Tesla* and *Fermi*).
- CUDA code based on CUDA version 1.3

Properties	**Tesla C1060**	**GeForce GTX 470**	**Tesla C2050 (x4)**
CUDA Capability	1.3	2.0	2.0
Global Memory	4.3GB	1.3GB	2.8GB
Multiprocessors	30	14	14
Cores	240	448	448
Threads/block	512	1024	1024

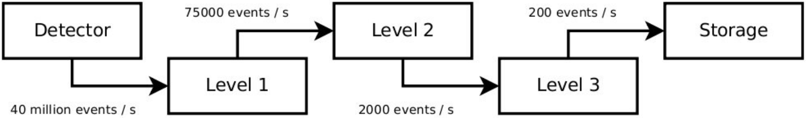
The ATLAS Trigger



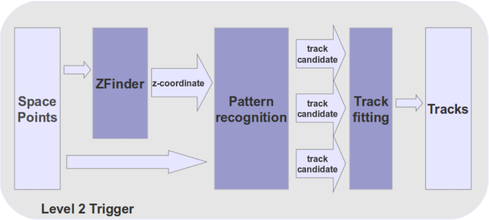
- Level 1: Custom built hardware with special processor units.
- **Level 2**: Software trigger operating independently on detector regions of interest (RoIs).
- Event filter (Level 3): Software trigger analysing whole event signatures.



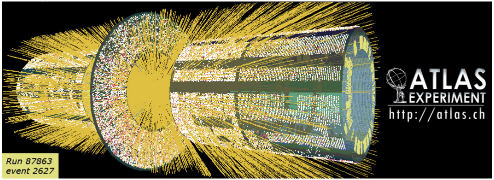
The ATLAS Trigger



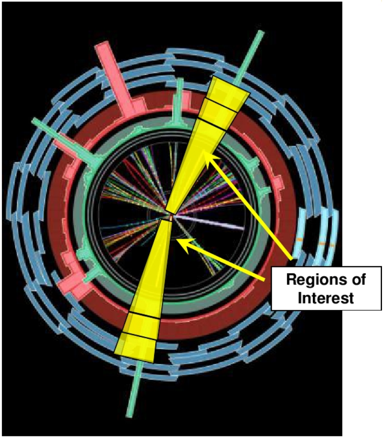
- Level 1: Custom built hardware with special processor units.
- **Level 2**: Software trigger operating independently on detector regions of interest (RoIs). **Ideal for GPGPUs**
- Event filter (Level 3): Software trigger analysing whole event signatures.



Z Finder GPU Motivation

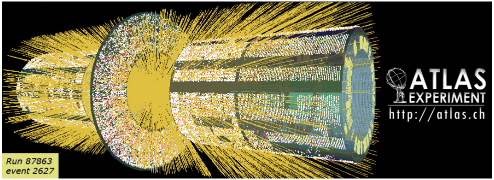


- Already break an event up into regions of interest (ROIs) for distributed processing.
- Break ROIs into slices of ϕ and process independently.

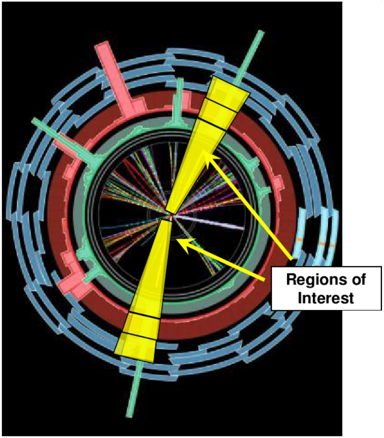


Cross section view of the ATLAS detector

Z Finder GPU Motivation



- Already break an event up into regions of interest (ROIs) for distributed processing.
- Break ROIs into slices of ϕ and process independently.
- Candidate for parallelisation using GPUs.



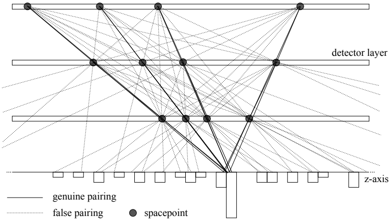
Cross section view of the ATLAS detector

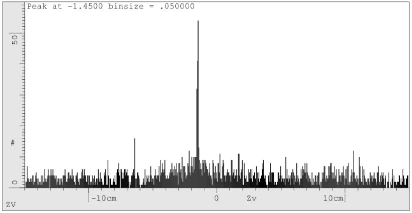
The Z Finder Algorithm

z_V calculation

$$z_V = \frac{z_2 \cdot \rho_1 - z_1 \cdot \rho_2}{\rho_1 - \rho_2}$$

- Process each combination of spacepoints and extrapolate back to the beam line.
- The histogram peak is the chosen interaction point.

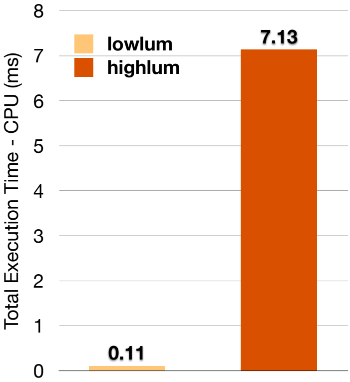




Z Finder Test Case

- Standalone version of Z finder code used for feasibility studies with CUDA.
- Initially optimised for calculating z_V using *pairs* of spacepoints.
- Timing performance measured using two samples of simulated events.

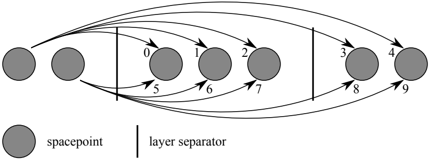
	lowlum	**highlum**
Luminosity ($cm^{-2}s^{-2}$)	$O(10^{32})$	$O(10^{34})$
Number of spacepoints	333	8104

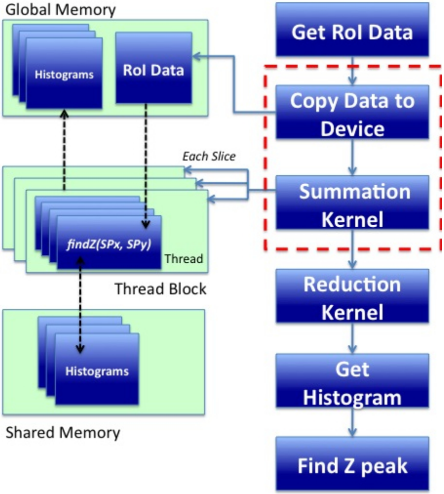


Z Finder Kernel: Histogram Summation

Code Iterations

- Single thread per ϕ slice.
- Thread block per ϕ slice.
- Histogram per thread block in shared memory.
- Improve spacepoint pair allocation method.

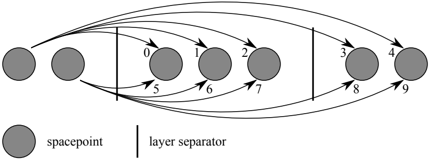


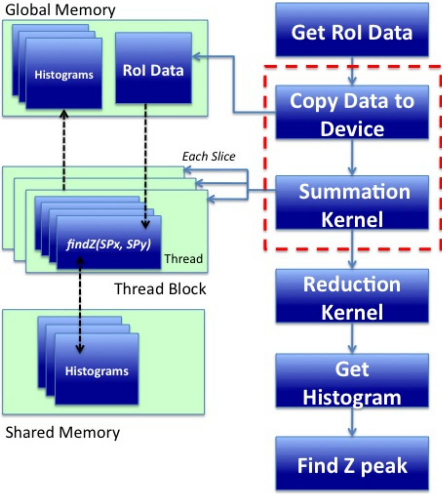


Z Finder Kernel: Histogram Summation

Code Iterations

- Single thread per ϕ slice.
- Thread block per ϕ slice.
- Histogram per thread block in shared memory.
- Improve spacepoint pair allocation method.

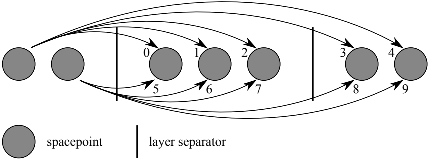


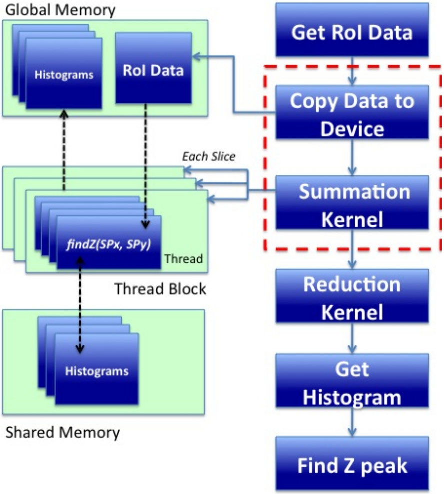


Z Finder Kernel: Histogram Summation

Code Iterations

- Single thread per ϕ slice.
- Thread block per ϕ slice.
- Histogram per thread block in shared memory.
- Improve spacepoint pair allocation method.

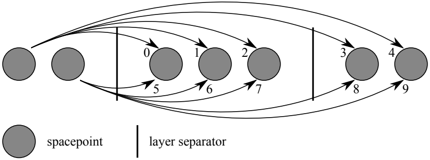


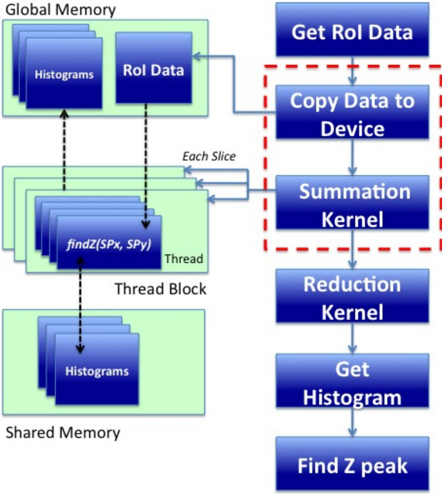


Z Finder Kernel: Histogram Summation

Code Iterations

- Single thread per ϕ slice.
- Thread block per ϕ slice.
- Histogram per thread block in shared memory.
- Improve spacepoint pair allocation method.

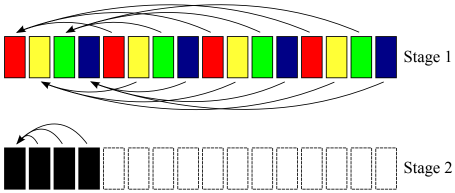


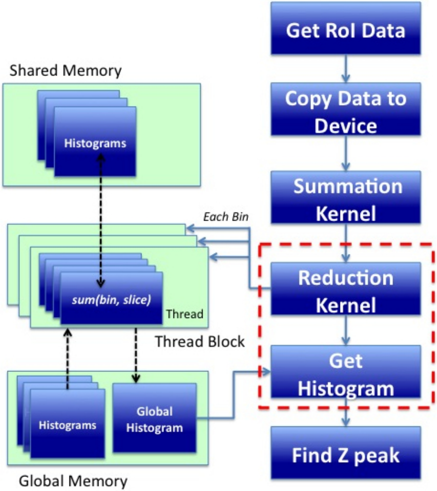


ZFinder Kernel: Histogram Combination

Code Iterations

- Combine histograms on the GPU ⇒ *reduce device to host data transfer by ∼500x.*
- Reduce the data to a single histogram in multiple steps.

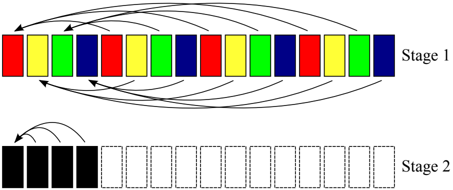


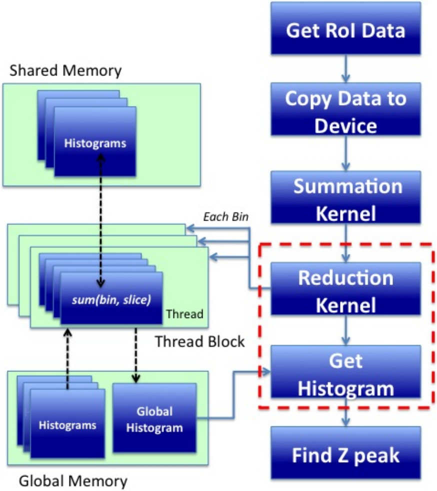


ZFinder Kernel: Histogram Combination

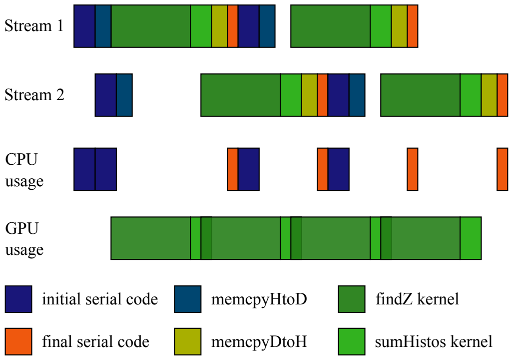
Code Iterations

- Combine histograms on the GPU $\Rightarrow$ *reduce device to host data transfer by $\sim$500x.*
- Reduce the data to a single histogram in multiple steps.



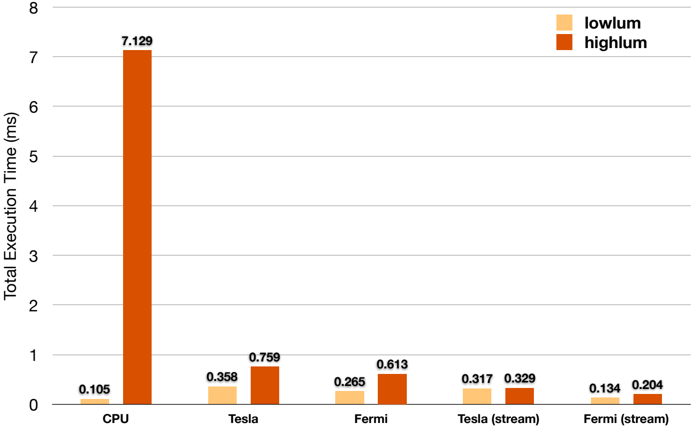


Z Finder Kernel: CUDA Streams



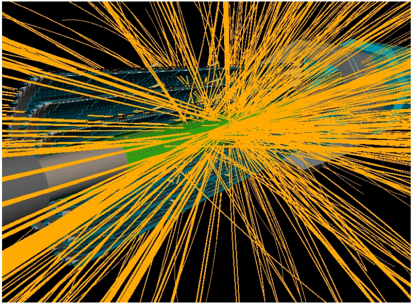
- Each RoI calculation independent ⇒ use CUDA streams.
- Successful in disguising any host to device transfer latency.

Timing Results



- Results for spacepoint pairs show up to *35x* speed-up (Fermi).
- Initial results for spacepoint *triplets* also show speed-up.

Kalman Filter GPU Motivation



- Potentially *thousands* of tracks to reconstruct for every event in the trigger.

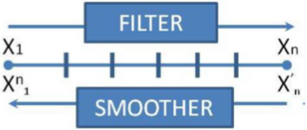
- Significant acceleration possible by reconstructing one track per GPU thread.

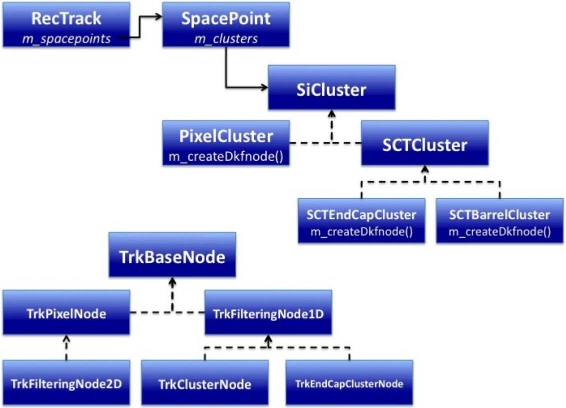
GPU benefits at other experiments

- Kalman Filter port to CUDA (GSI Scientific Report 2008, FAIR-EXPERIMENTS-38)
- ALICE TPC HLT code GPU based / future PANDA TPC code
- GPUs to be used for STS (Silicon Tracking System) within CBM (Compressed Baryonic Matter) experiment at FAIR/GSI.

Track Reconstruction in ATLAS

- Tracks reconstructed using the Kalman filter method.
- The trajectory of a track is predicted using detector hits as input.
- Backward smoothing filter applied after final Kalman Filter estimation.





C++ Class Hierarchy of Track Objects

Kalman Filter for CUDA

Initial Complications

- Class inheritance structure captures filter specialism for each sub-detector.
- Dynamic creation of objects in the main routine.
- Track state retention at each filtering step.
- Break down main routine for a smaller kernel.

Feasibility Studies (Maria Rovatsou)

- Standalone version successfully ported to C.
- Pre-allocated memory needed for track objects.
- Promising results ⇒ memory footprint per track needs to be reduced.

Kalman Filter for CUDA

Initial Complications

- Class inheritance structure captures filter specialism for each sub-detector.
- Dynamic creation of objects in the main routine.
- Track state retention at each filtering step.
- Break down main routine for a smaller kernel.

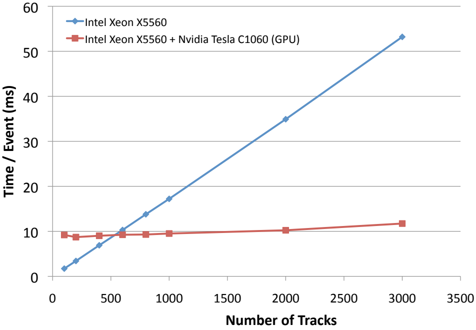
Feasibility Studies (Maria Rovatsou)

- Standalone version successfully ported to C.
- Pre-allocated memory needed for track objects.
- Promising results ⇒ memory footprint per track needs to be reduced.

Kalman Filter for CUDA

D. Emeliyanov (first results)

- Standalone version successfully ported to C.
- Structs of arrays used to store track data.
- Vector data types (e.g. *float4*) for compact representation of data.
- One GPU thread per track.
- Modification of smoothing algorithm required for single precision arithmetic.



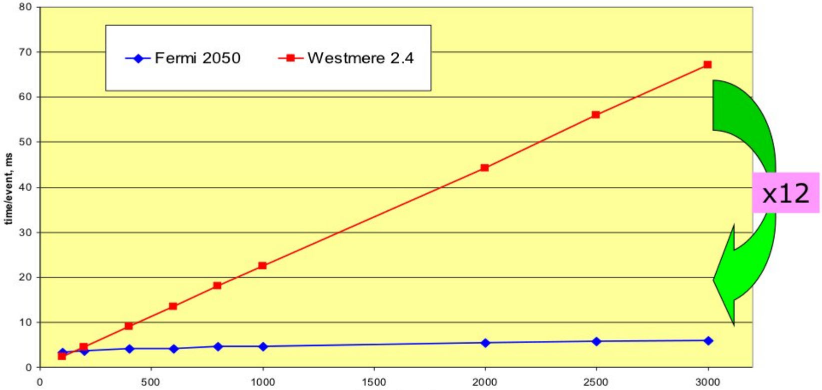
Muon tracks, p_T=10GeV, full MC simulation

- Over *5x* speed-up seen at 3000 tracks.

Kalman Filter for CUDA

D. Emeliyanov (latest results)

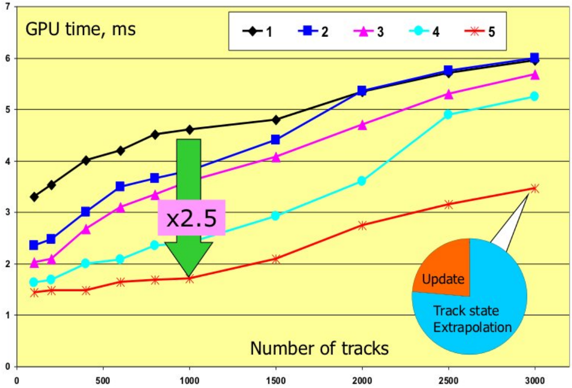
- CPU: Intel Westmere 2.4 GHz, GPU: NVIDIA Tesla C2050 (Fermi arch.)
- Data: full ATLAS Monte Carlo simulation
  - muon tracks, $p_T = 10\ GeV$, arranged into "events" with N tracks up to 3000



Kalman Filter Optimisations

D. Emeliyanov (latest results)

- A set of optimizations has been applied
- Optimised code gives ∼20x speed-up w.r.t. the CPU



1. Original code
2. 32 threads/block
3. Reduced memory footprint (fewer local variables, upper-triangular covariance matrix
4. Track state (cov. + parameters) stored in fast ("shared") memory
5. Jacobian in "shared" memory to speed-up JCJ^T calculation

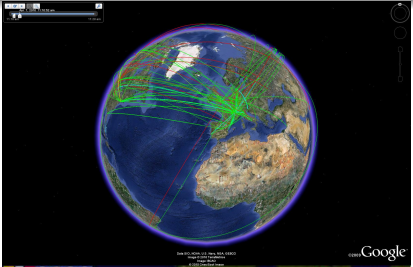
Where is most HEP CPU consumed?

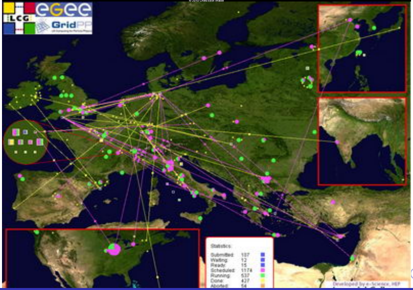
After triggering the LHC experiments still produce vast amounts of data!

Where is most HEP CPU consumed?

After triggering the LHC experiments still produce vast amounts of data!
We developed worldwide LHC computing grid infrastructure

- Approximately 15 PB of data recorded per annum
- Currently $>$100,000 processors across Grid
- 130 sites in 34 countries



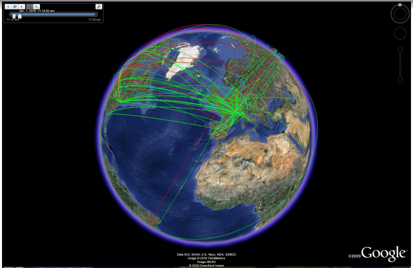


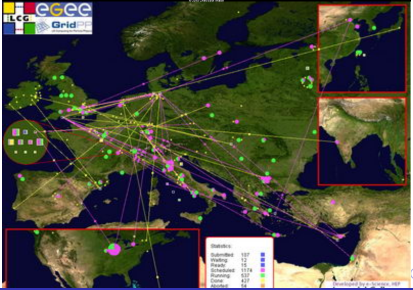
Where is most HEP CPU consumed?

After triggering the LHC experiments still produce vast amounts of data!
We developed worldwide LHC computing grid infrastructure

- Approximately 15 PB of data recorded per annum
- Currently >100,000 processors across Grid
- 130 sites in 34 countries

Geant4 simulation of detector response ($\sim$ 1000 cpu seconds per event)





Where is most HEP CPU consumed?

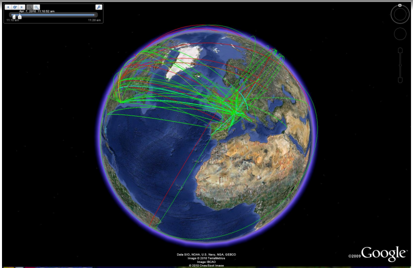
After triggering the LHC experiments still produce vast amounts of data!
We developed worldwide LHC computing grid infrastructure

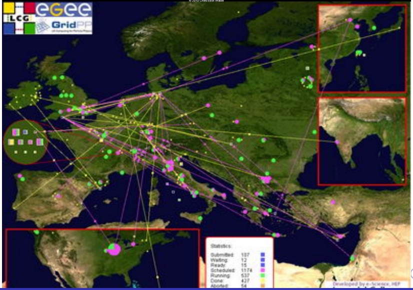
- Approximately 15 PB of data recorded per annum
- Currently >100,000 processors across Grid
- 130 sites in 34 countries

Geant4 simulation of detector response ($\sim$ 1000 cpu seconds per event)

Up to ten million events simulated daily

G4 failure rate is less than 10^{-6}





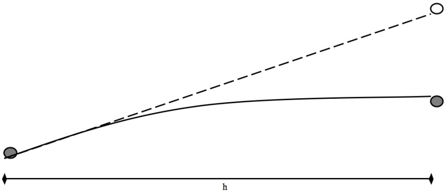
Particle tracking in a magnetic field

Preliminary GPGPU test case study

Particle tracking in a magnetic field

Preliminary GPGPU test case study

- Tracking charged particles in the magnetic field



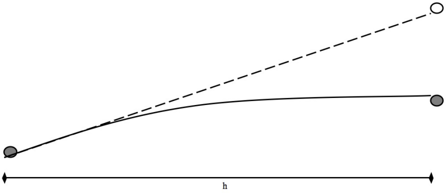
Particle tracking in a magnetic field

Preliminary GPGPU test case study

- Tracking charged particles in the magnetic field
- Lorentz force (perpendicular to plane of magnetic field)

$$\mathbf{F} = m\mathbf{a} = q \cdot (\mathbf{E} + \mathbf{v} \times \mathbf{B})$$

$$\frac{d\mathbf{v}}{dt} = \mathbf{a} = \frac{q}{m} \cdot (\mathbf{E} + \mathbf{v} \times \mathbf{B})$$



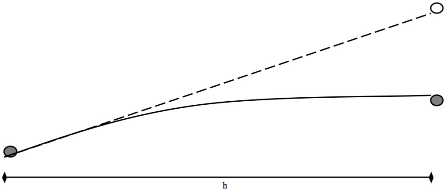
Particle tracking in a magnetic field

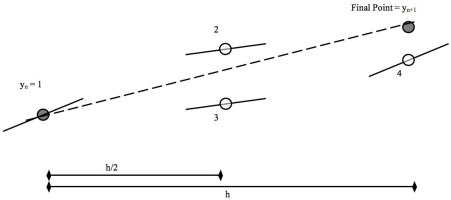
Preliminary GPGPU test case study

- Tracking charged particles in the magnetic field
- Lorentz force (perpendicular to plane of magnetic field)

$$\mathbf{F} = m\mathbf{a} = q \cdot (\mathbf{E} + \mathbf{v} \times \mathbf{B})$$
$$\frac{d\mathbf{v}}{dt} = \mathbf{a} = \frac{q}{m} \cdot (\mathbf{E} + \mathbf{v} \times \mathbf{B})$$

- Solve the differential equation with 4th order Runge Kutta Integration (called " Stepper" algorithm)





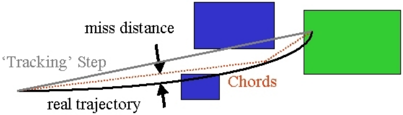
Steppers

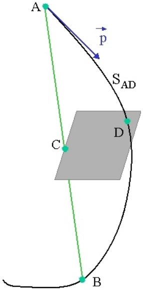
Steppers (EM field integration steps)
Various performance requirements:

- Miss Distance (chord sagitta),
- Boundary Intersection Error,
- Tolerable Integration Error, . . .

Lots of simulation time was spent on field calls...
Introduced adaptive stepper & caching
Different steppers:

- G4ClassicalRK4
  (EM field map: 10 calls per step)
- AtlasRK4
  (EM field map:2 calls per step





Acceleration with GPGPUs

Studied GPU acceleration of the standard 4th order Runge-Kutta (G4ClassicalRK4)

1. Using the GPGPU, pre-calculated a "look-up" table of derivative calculations for a space point matrix
   - Calculation time not a limiting factor (abandoned this idea)
   - Also lost accuracy due to rounding to nearest look up point

Acceleration with GPGPUs

Studied GPU acceleration of the standard 4th order Runge-Kutta (G4ClassicalRK4)

1. Using the GPGPU, pre-calculated a “look-up” table of derivative calculations for a space point matrix
   - Calculation time not a limiting factor (abandoned this idea)
   - Also lost accuracy due to rounding to nearest look up point
2. Increased calculation complexity to use adaptive stepping
   - Adjusting step size to be within an error tolerance
   - Still slower than the CPU. . .

Acceleration with GPGPUs

Studied GPU acceleration of the standard 4th order Runge-Kutta (G4ClassicalRK4)

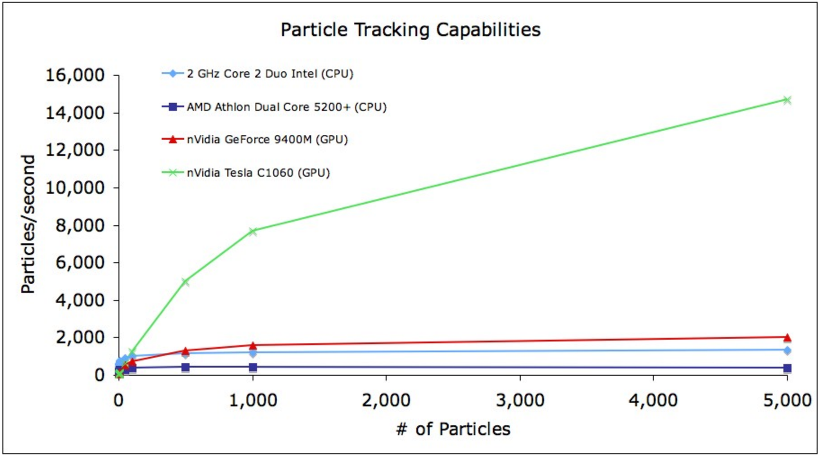
1. Using the GPGPU, pre-calculated a “look-up” table of derivative calculations for a space point matrix
   - Calculation time not a limiting factor (abandoned this idea)
   - Also lost accuracy due to rounding to nearest look up point
2. Increased calculation complexity to use adaptive stepping
   - Adjusting step size to be within an error tolerance
   - Still slower than the CPU. . .
3. Treated x,y,z coordinates in parallel (3 threads in block)
   - Cross-product ($\mathbf{v} \times \mathbf{B}$) calculation needs perp. coordinates
   - Set up the threads in the block to use shared memory
   - Speed was now closer to CPU

Acceleration with GPGPUs

Studied GPU acceleration of the standard 4th order Runge-Kutta (G4ClassicalRK4)

1. Using the GPGPU, pre-calculated a “look-up” table of derivative calculations for a space point matrix
   - Calculation time not a limiting factor (abandoned this idea)
   - Also lost accuracy due to rounding to nearest look up point
2. Increased calculation complexity to use adaptive stepping
   - Adjusting step size to be within an error tolerance
   - Still slower than the CPU...
3. Treated x,y,z coordinates in parallel (3 threads in block)
   - Cross-product ($\mathbf{v} \times \mathbf{B}$) calculation needs perp. coordinates
   - Set up the threads in the block to use shared memory
   - Speed was now closer to CPU
4. Next stage was to do many particle tracks in parallel...

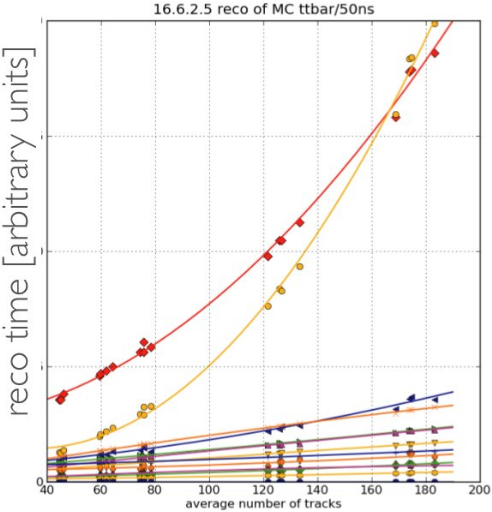
Magnetic Field Integration results



- Rapidly achieved a factor 32 speedup (more in progress)

Reconstruction: Tracking

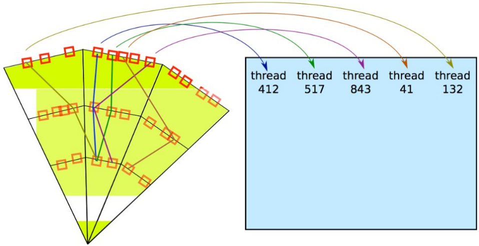
- Reconstruction time depends on multiplicity in the detector
- Track finding has worst combinatorial behaviour (expected) and starts to dominate already at modest multiplicities.



Reconstruction: Tracking

Christian Schmidt

- Initial prototyping of tracking on GPUs being done
- Single thread for each combination in every segment combination
- Assigns GPU-global hit data via thread index
- Preliminary results are showing GPU is at least 10 times faster than CPU



Summary

- The ATLAS trigger, particle tracking & simulation algorithms are key areas where GPUs can be used to improve performance.
- Significant enhancements to the trigger and reconstruction algorithms could prove invaluable for dealing with the rates from the LHC upgrade.
- Observed an initial *32x* speed-up for parallel Runge Kutta integration.
- Best case optimisation of *35x* speed-up for the Z Finder routine.
- Port of OO-based Kalman Filter algorithm showed GPU acceleration is feasible and scales to thousands of tracks.
- For much more information please see the talks at the recent workshop Future computing for Particle Physics