Science & Technology Facilities Council
Rutherford Appleton Laboratory

A GPU-based Kalman filter for ATLAS Level 2 Trigger

Dmitry Emeliyanov
Particle Physics Department
Rutherford Appleton Laboratory

Outline

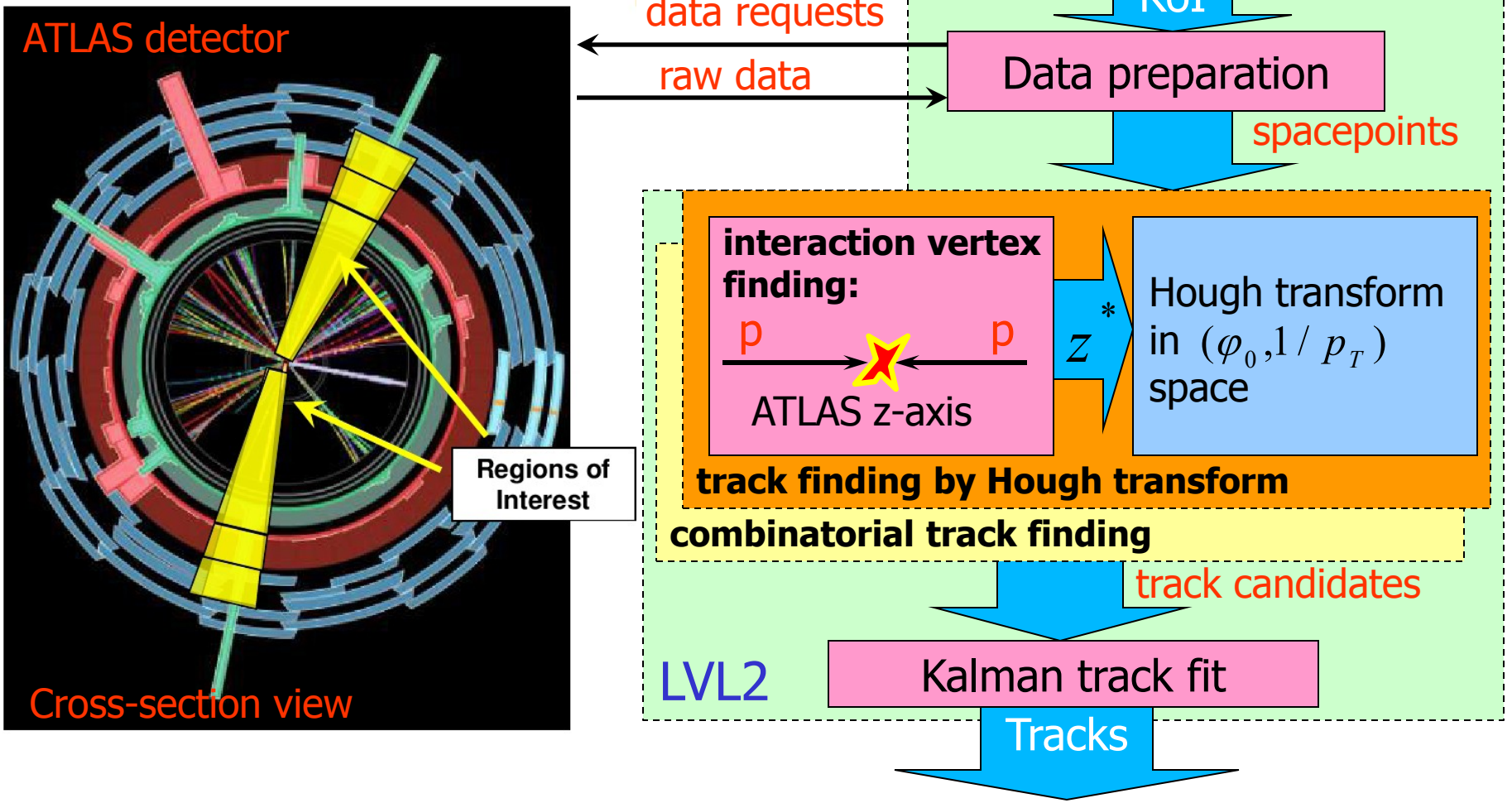
- GPUs for ATLAS High-Level Trigger (HLT)
- Inner Detector Tracking at Level 2 Trigger
- GPU-based implementation for the track fitting algorithm
- Timing results
- GPU code optimization
- Discussion
- Conclusion and outlook

GPUs for ATLAS Trigger

- R&D programme for ATLAS higher luminosity upgrade includes Trigger system and HLT software
- In general, various upgrade approaches are possible:
  - using more/better CPUs for HLT farms
  - vectorization of HLT software to process more than one event per CPU core
  - using GPUs for time-critical parts of HLT code which are suitable for SIMT parallelization
- The GPU-based option is feasible since ATLAS HLT uses dedicated farms which can be, in principle, equipped with GPU cards.
- To prove this feasibility a few GPU-accelerated algorithms for Level 2 Trigger (LVL2) tracking have been developed.

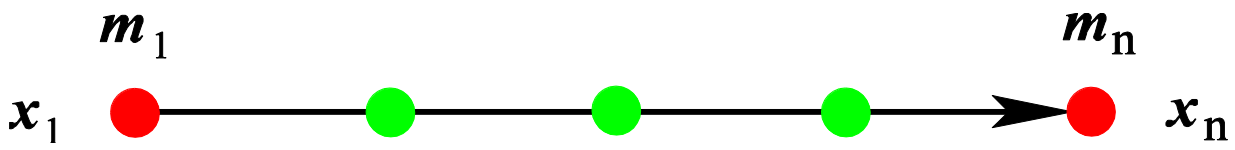
Tracking at LVL2 Trigger

- LVL2 operates independently on Region-of-Interests (RoI) identified by the Level 1 Trigger:

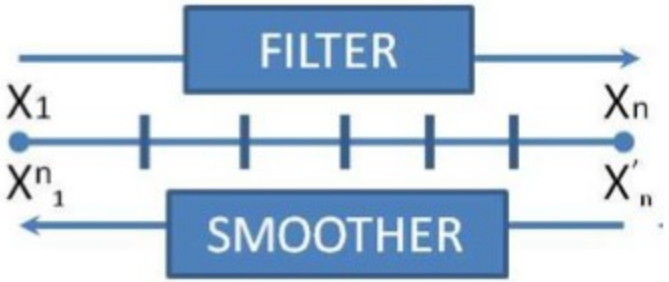


Track fitting at LVL2 Trigger

- The track fit provides track parameter estimates $\hat{x}_1, \hat{x}_n$ and covariance matrices C_1, C_n at both ends of a track:

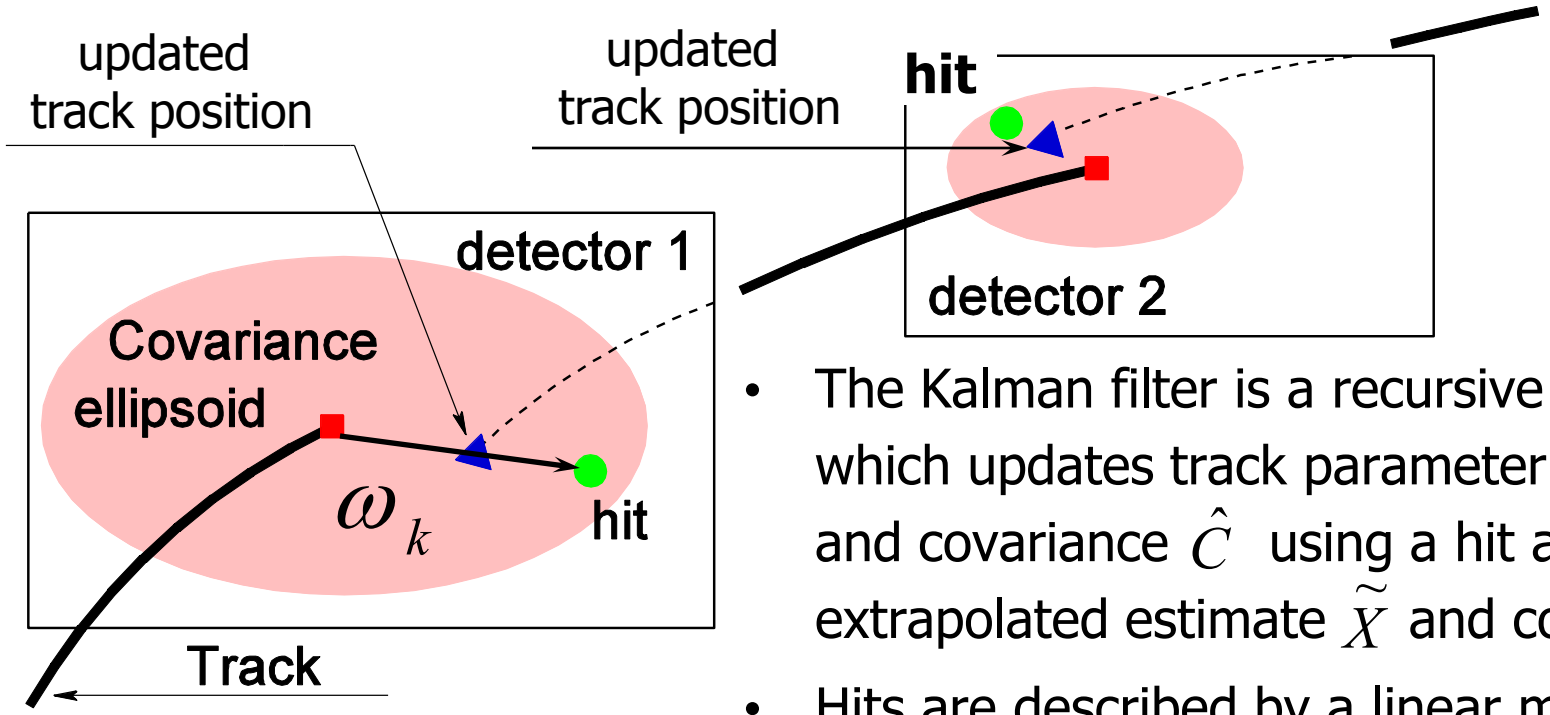


- The track fitter consists of a forward Kalman filter and a backward smoother:



- Implemented in C++ using OO techniques and STL containers

The Kalman filter algorithm



- The Kalman filter is a recursive algorithm which updates track parameter estimate $\hat{X}$ and covariance $\hat{C}$ using a hit and the extrapolated estimate $\tilde{X}$ and covariance $\tilde{C}$
- Hits are described by a linear model:

$u_k = HX + \eta_k$, u_k – hit coordinates, H – measurement matrix, η_k – gaussian noise with zero mean and covariance V_k

- The track parameter and covariance update by the Kalman filter:

$$\hat{X}_k = \tilde{X} + K_k \omega_k \qquad \hat{C}_k = \tilde{C} - K_k H \tilde{C}$$

- K_k – the Kalman gain matrix, ω_k – "hit-track "residual: $\omega_k = u_k - H\tilde{X}$

Adapting the fitter for GPU

- The Kalman filter/smoother code is sequential – use **track-level parallelism** :
  - one thread per track – N tracks are fitted by N threads in parallel
- Limitations/requirements:
  - no dynamic memory allocation is possible
  - arrays and structs used instead of STL vectors and C++ objects
- Data structures: structs of arrays (SoA):
  - Array1 : [data for thr #1][data for thr #2]…[data for thr #N]
- Vector data types (float4) for compact representation of data: detector plane geometry and magnetic field:
  - plane centre *c* (3 floats), rotation matrix *m* (9 floats), field **B** at the centre (3 floats), radiation thickness *x0* (1 float) – packed into 4 float4 – and each float4 can be read at once

Using single FP precision

- Double-precision arithmetic is available on modern GPU
  - but double precision performance is typically half that of single one
  - using single precision halves the volume of data to/from GPU
- The original fitter code is sensitive to single precision:
  - backward smoother suffers from divergence of covariance

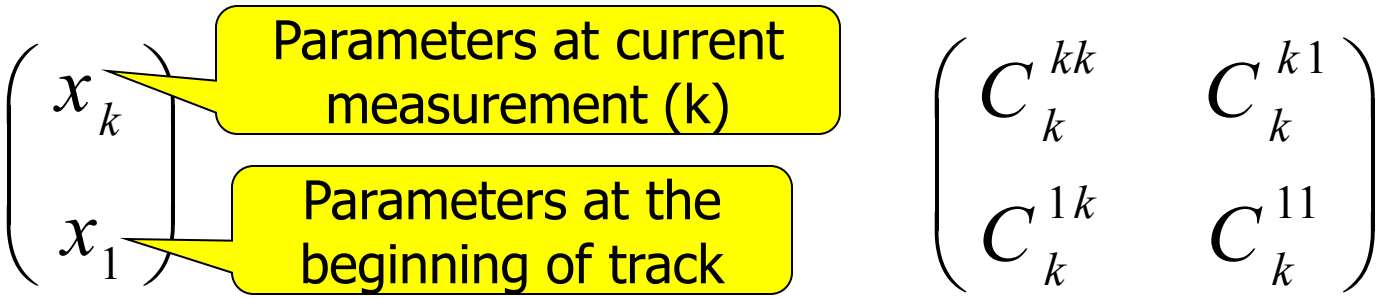
$$\hat{C}_1^S = \hat{C}_1^F - G_1\left(\tilde{C}_2 - \hat{C}_2^S\right)G_1^T$$

BIG **small** => Loss of precision

- A trivial fix:
  - using GeV instead of MeV for track momentum reduces dynamic range of covariance matrix elements
- Using a different smoothing algorithm to improve numerical stability of the fitter

FPS: Fixed-Point Smoother

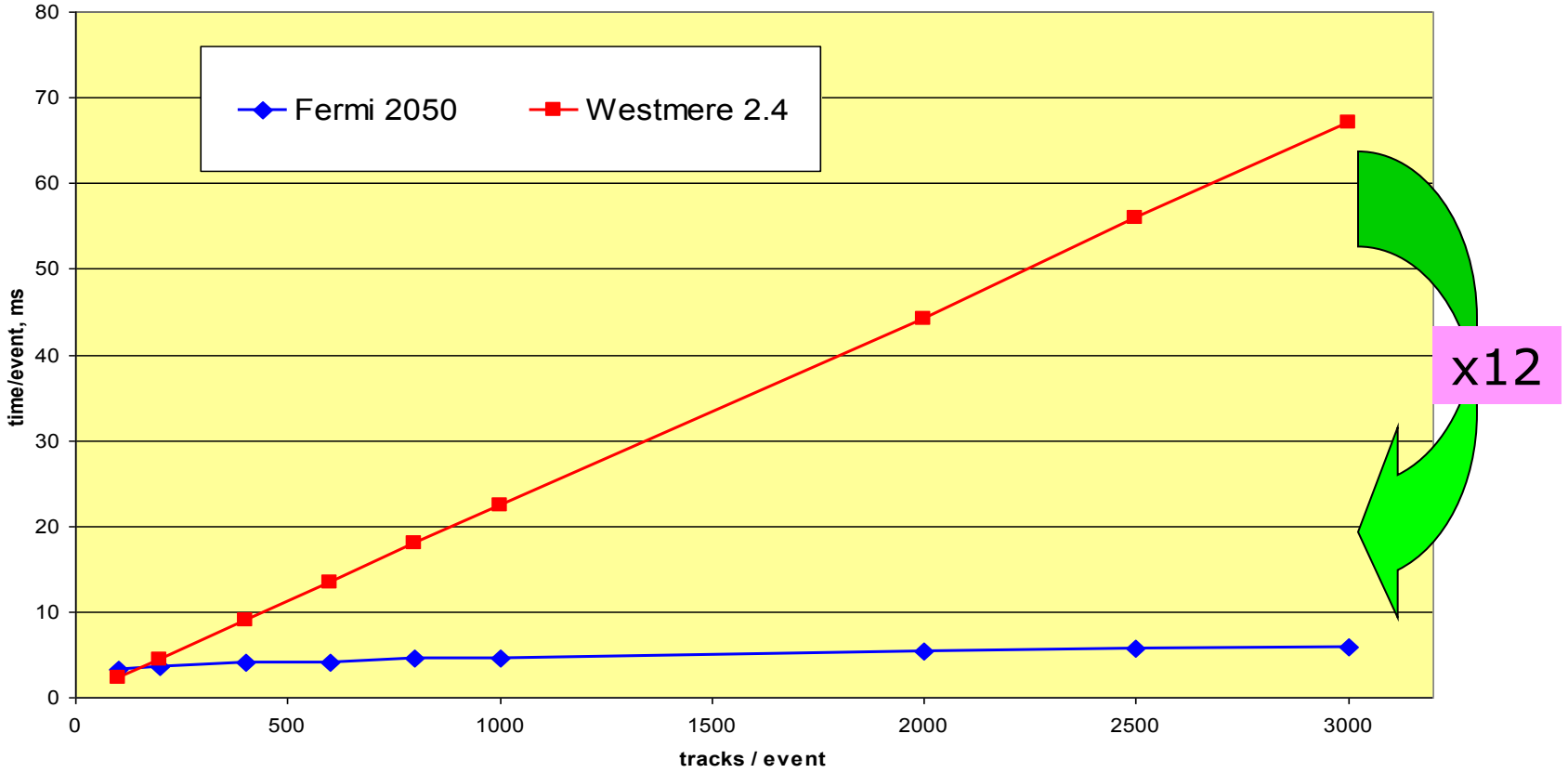
- An FPS is just a Kalman filter which operates on an extended track parameter vector and covariance :



- Advantages:
  - Single-pass algorithm which does not require covariance inversion (needed for traditional smoother)
  - No need to store track parameter estimates and covariances for intermediate measurements – great saving of GPU memory
  - Bonus: a cross-covariance $\mathrm{cov}(x_0, x_n)$ is readily available – comes very handy if we want to add more measurements / merge two fitted track segments

Timing 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

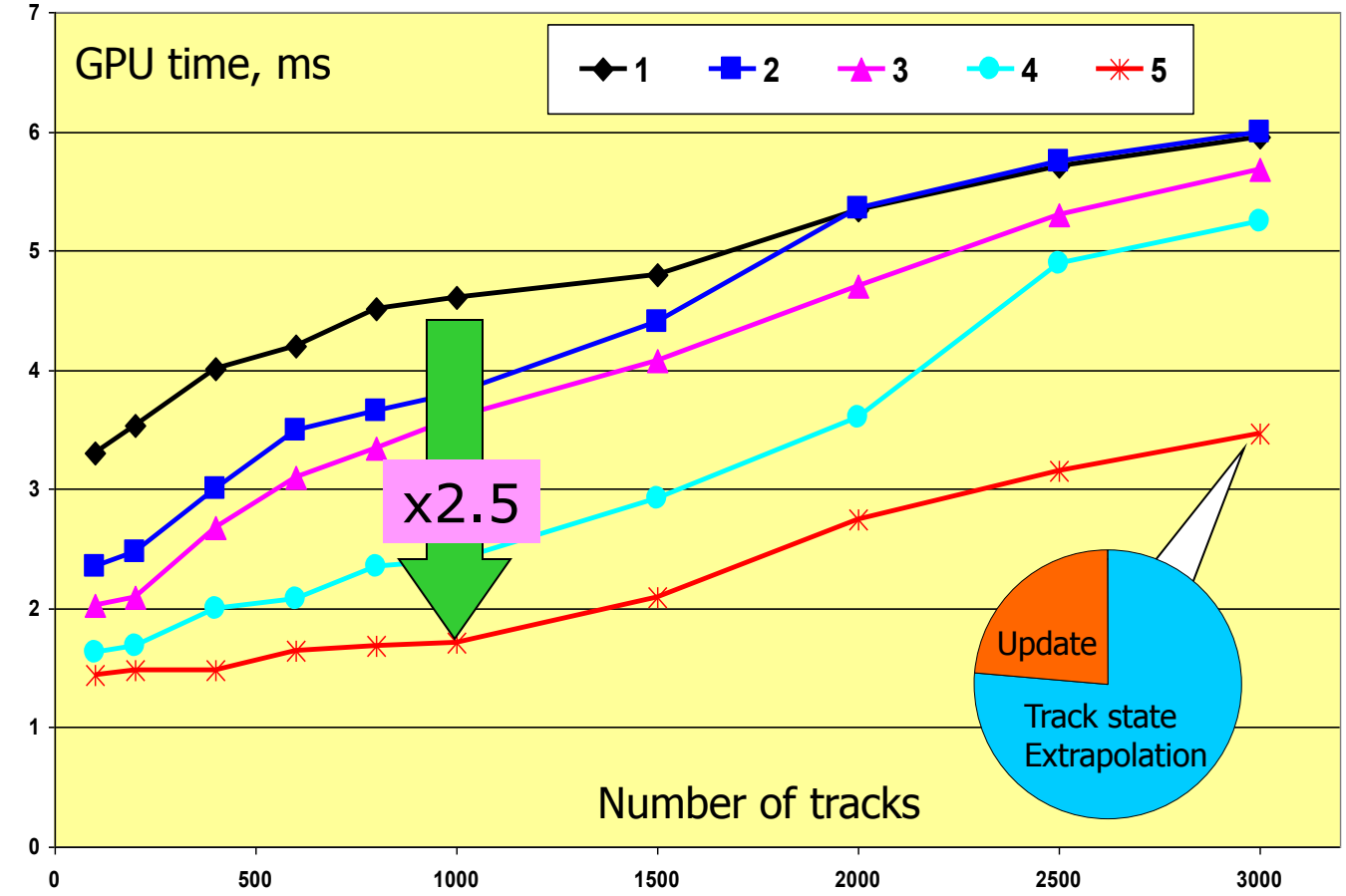


- I/O size (3000 tracks):

Input data	4.6 MB	Output tracks	0.3 MB

GPU code optimization

- A set of optimizations has been applied:



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

- The optimized code gives ~20x speed-up w.r.t. the CPU

Discussion

- 20x speed-up seen at 3000 tracks
- Data transfer via PCI-E (time, I/O size) is not a bottleneck
- For low number of tracks the GPU-based track fitting is only slightly faster than that on CPU :
  - N threads = N tracks – not enough to occupy all GPU cores
  - Track-level parallelism is not suitable it number of tracks is low
- Possible solutions for low multiplicity events:
  - process tracks from a few RoIs in parallel
  - use a different algorithm which would allow for **mixed parallelism:**
    - using separate threads for track parameters and track covariance extrapolations
    - using parallel matrix multiplications in track covariance extrapolation

Conclusion and outlook

- The design of GPU-based Kalman track fit for ATLAS Level 2 Trigger has been presented
- The techniques for the fitter code optimization have been demonstrated
- The optimized GPU-based track fitter shows a speed-up factor of 20 for high track multiplicity events
- Future work:
  - exploiting inherent parallelism of the track state extrapolation