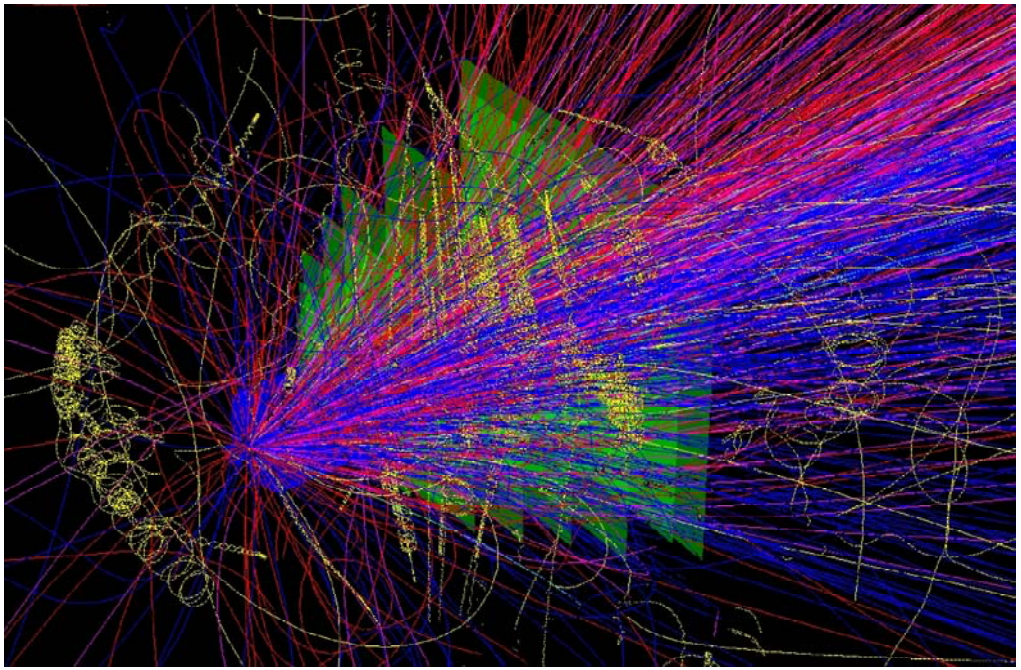
First Level Event Selection Package of the CBM Experiment

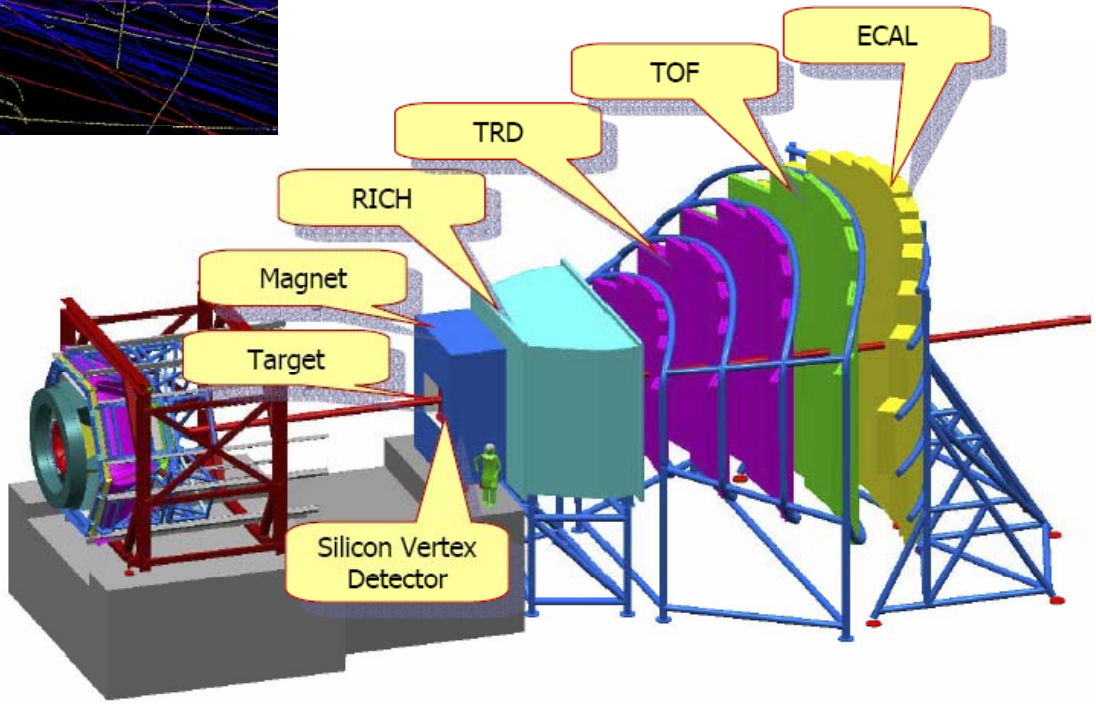
S. Gorbunov, I. Kisel, I. Kulakov, I. Rostovtseva, I. Vassiliev
(for the CBM Collaboration)

CHEP'09
Prague, March 26, 2009

Tracking Challenge in CBM (FAIR/GSI, Germany)

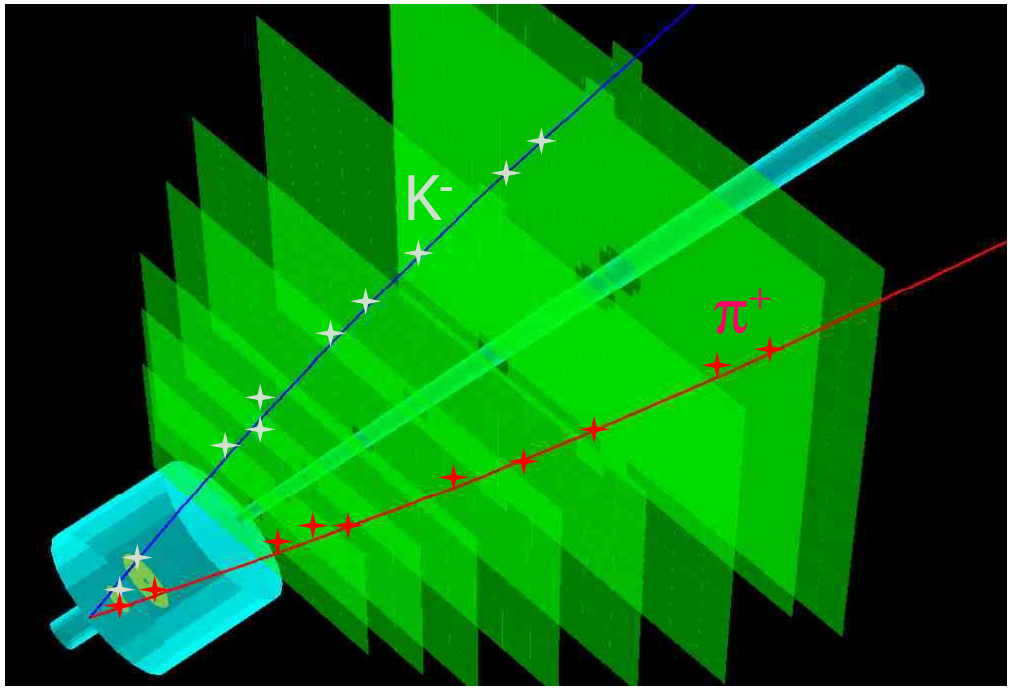


* Fixed-target heavy-ion experiment
* 10^7 Au+Au collisions/sec
* ~ 1000 charged particles/collision
* Non-homogeneous magnetic field
* Double-sided strip detectors
  (85% fake space points)



Track reconstruction in STS/MVD and displaced vertex search required in the first trigger level

Open Charm Event Selection



$D^{\pm}$ ($c\tau$ = 312 μm):

$D^{+} \rightarrow K^{-}\pi^{+}\pi^{+}$ (9.5%)

D^{0} ($c\tau$ = 123 μm):

$D^{0} \rightarrow K^{-}\pi^{+}$ (3.8%)

$D^{0} \rightarrow K^{-}\pi^{+}\pi^{+}\pi^{-}$ (7.5%)

$D^{\pm}_{s}$ ($c\tau$ = 150 μm):

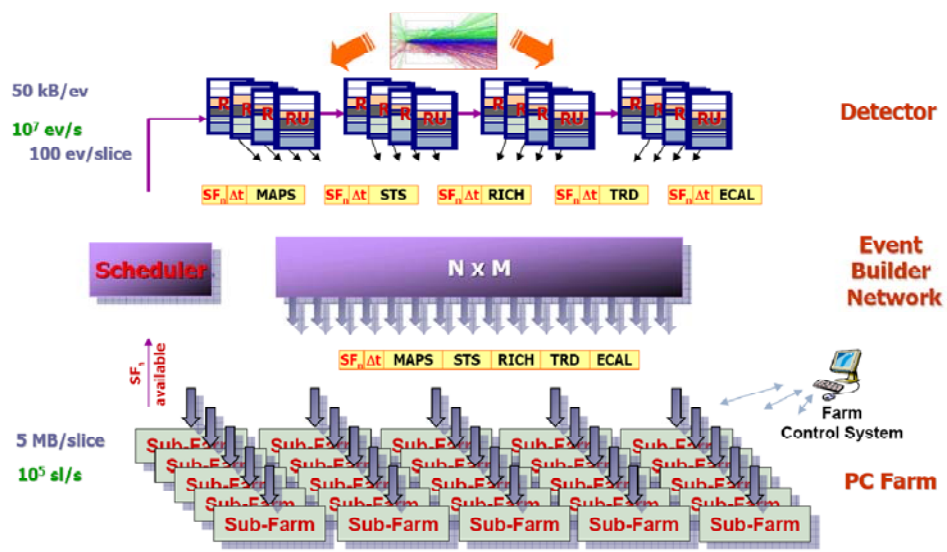
$D^{+}_{s} \rightarrow K^{+}K^{-}\pi^{+}$ (5.3%)

Λ^{+}_{c} ($c\tau$ = 60 μm):

$\Lambda^{+}_{c} \rightarrow pK^{-}\pi^{+}$ (5.0%)

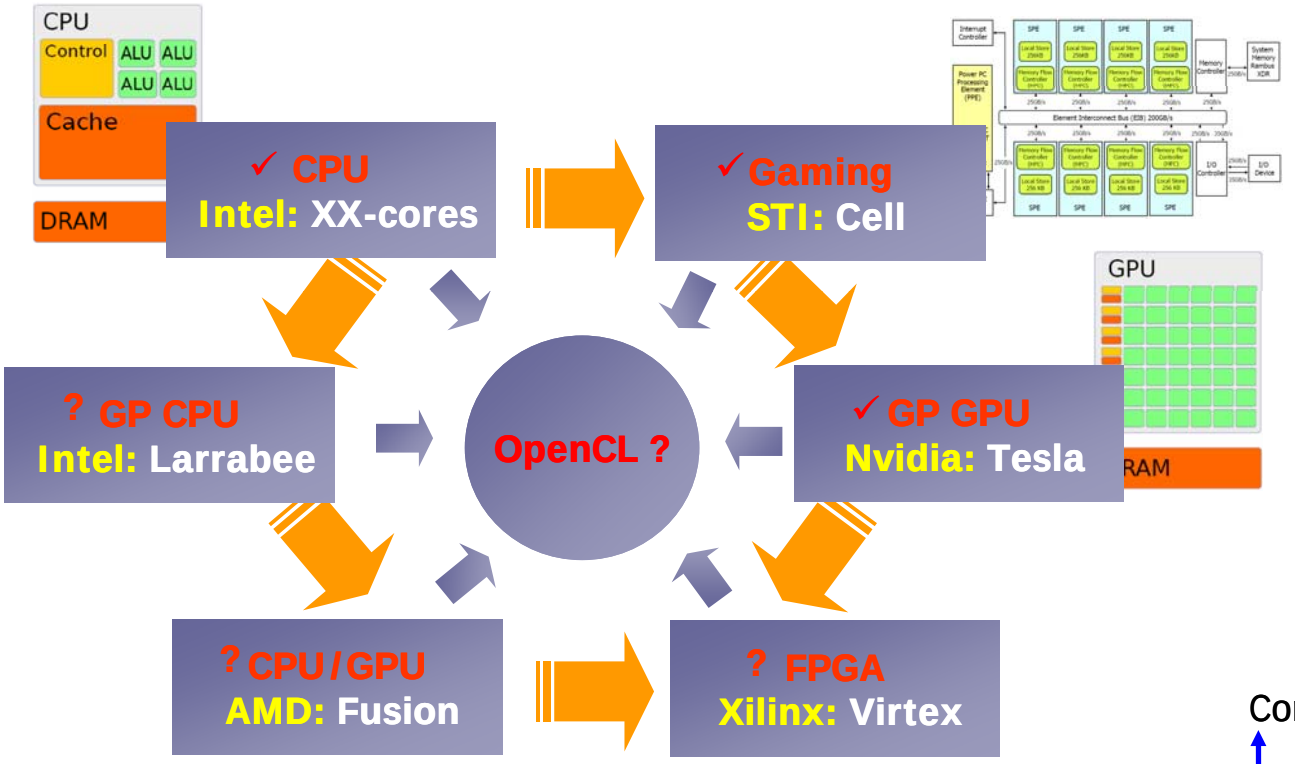
No simple trigger primitive, like high p_t, available to tag events of interest. The only selective signature is the detection of the decay vertex.

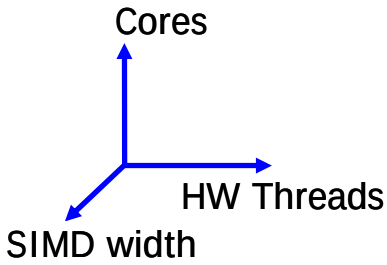
First level event selection is done in a processor farm fed with data from the event building network



Many-core HPC

- On-line event selection
- Mathematical and computational optimization
- Optimization of the detector

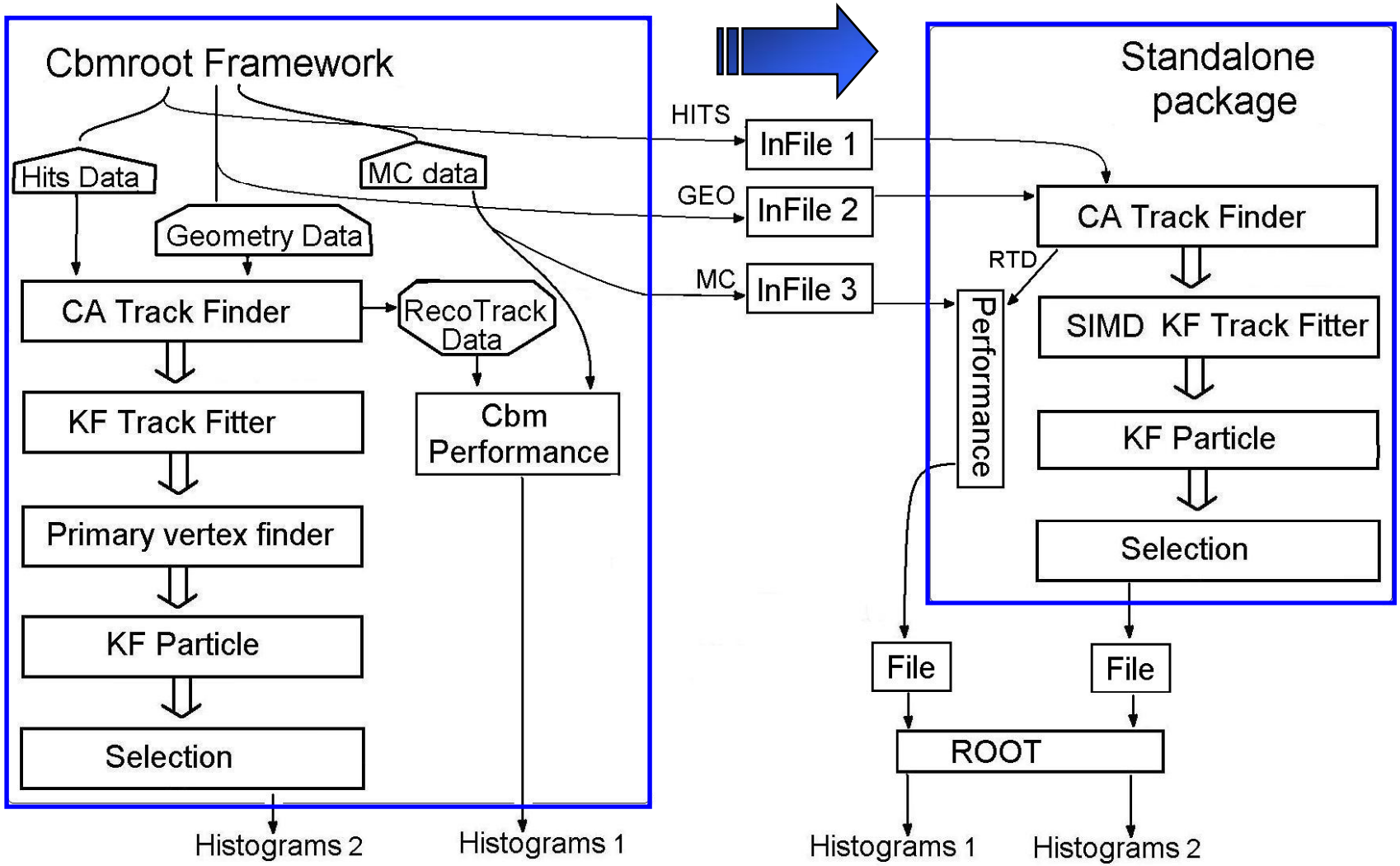




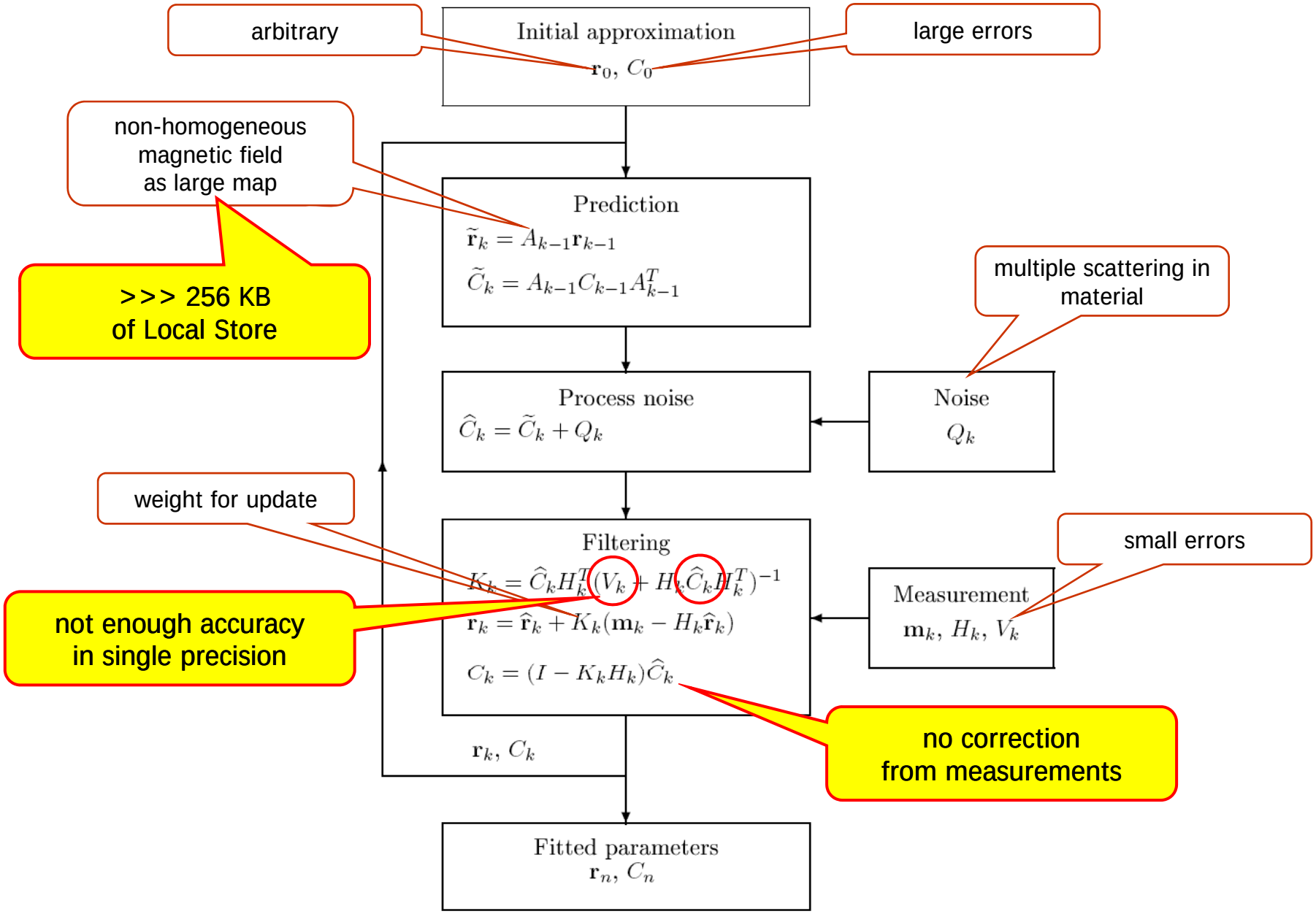
- Heterogeneous systems of many cores
- Uniform approach to all CPU/GPU families
- Similar programming languages (CUDA, Ct, OpenCL)
- Parallelization of the algorithm (vectors, multi-threads, many-cores)

$$N_{speed\text{-}up} = N_{cores} * (N_{threads}/2) * W_{SIMD}$$

Standalone Package for Event Selection

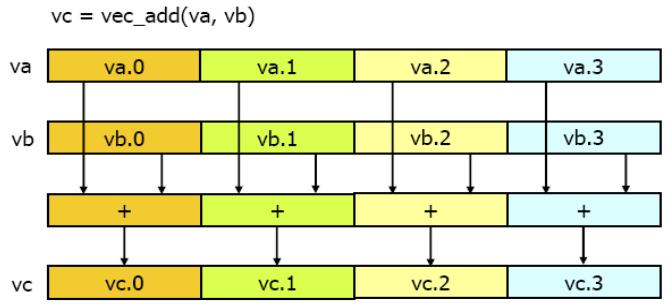


Kalman Filter for Track Fit



Code (Part of the Kalman Filter)

```
inline void AddMaterial( TrackV &track, Station &st, Fvec_t &qp0 )
{
  cnst mass2 = 0.1396*0.1396;

  Fvec_t tx = track.T[2];
  Fvec_t ty = track.T[3];
  Fvec_t txtx = tx*tx;
  Fvec_t tyty = ty*ty;
  Fvec_t txtx1 = txtx + ONE;
  Fvec_t h = txtx + tyty;
  Fvec_t t = sqrt(txtx1 + tyty);
  Fvec_t h2 = h*h;
  Fvec_t qp0t = qp0*t;

  cnst c1=0.0136, c2=c1*0.038, c3=c2*0.5, c4=-c3/2.0, c5=c3/3.0, c6=-c3/4.0;

  Fvec_t s0 = (c1+c2*st.logRadThick + c3*h + h2*(c4 + c5*h +c6*h2) )*qp0t;

  Fvec_t a = (ONE+mass2*qp0*qp0t)*st.RadThick*s0*s0;

  CovV &C = track.C;

  C.C22 += txtx1*a;
  C.C32 += tx*ty*a; C.C33 += (ONE+tyty)*a;
}
```



Use headers to overload +, -, *, / operators --> the source code is unchanged !

Header (Intel's SSE)

SIMD instructions

```
typedef F32vec4 Fvec_t;
/* Arithmetic Operators */
friend F32vec4 operator +(const F32vec4 &a, const F32vec4 &b) { return _mm_add_ps(a,b); }
friend F32vec4 operator -(const F32vec4 &a, const F32vec4 &b) { return _mm_sub_ps(a,b); }
friend F32vec4 operator *(const F32vec4 &a, const F32vec4 &b) { return _mm_mul_ps(a,b); }
friend F32vec4 operator /(const F32vec4 &a, const F32vec4 &b) { return _mm_div_ps(a,b); }
/* Functions */
friend F32vec4 min( const F32vec4 &a, const F32vec4 &b ){ return _mm_min_ps(a, b); }
friend F32vec4 max( const F32vec4 &a, const F32vec4 &b ){ return _mm_max_ps(a, b); }
/* Square Root */
friend F32vec4 sqrt ( const F32vec4 &a ){ return _mm_sqrt_ps (a); }
/* Absolute value */
friend F32vec4 fabs( const F32vec4 &a){ return _mm_and_ps(a, _f32vec4_abs_mask); }
/* Logical */
friend F32vec4 operator&( const F32vec4 &a, const F32vec4 &b ){ // mask returned
  return _mm_and_ps(a, b);
}
friend F32vec4 operator|( const F32vec4 &a, const F32vec4 &b ){ // mask returned
  return _mm_or_ps(a, b);
}
friend F32vec4 operator^( const F32vec4 &a, const F32vec4 &b ){ // mask returned
  return _mm_xor_ps(a, b);
}
friend F32vec4 operator!( const F32vec4 &a ){ // mask returned
  return _mm_xor_ps(a, _f32vec4_true);
}
friend F32vec4 operator||( const F32vec4 &a, const F32vec4 &b ){ // mask returned
  return _mm_or_ps(a, b);
}
/* Comparison */
friend F32vec4 operator<( const F32vec4 &a, const F32vec4 &b ){ // mask returned
  return _mm_cmplt_ps(a, b);
}
```

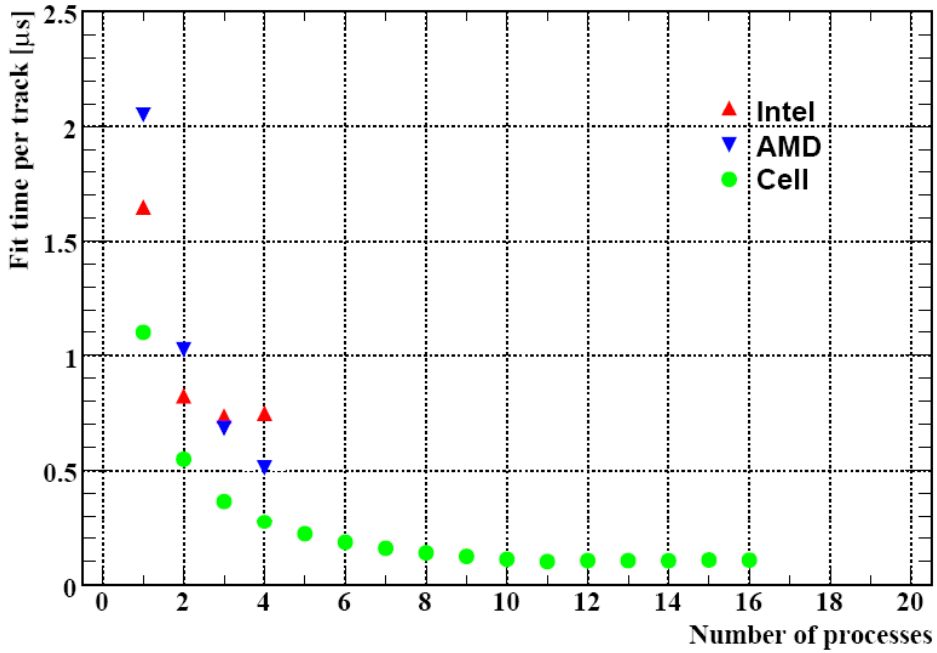
Kalman Filter Track Fit on Intel Xeon, AMD Opteron and Cell

	Stage	Description	Time/track	Speedup
		Initial scalar version	12 ms	–
Intel P4	1	Approximation of the magnetic field	240 μs	50
Intel P4	2	Optimization of the algorithm	7.2 μs	35
Intel P4	3	Vectorization	1.6 μs	4.5
Cell	4	Porting to SPE	1.1 μs	1.5
Cell	5	Parallelization on 16 SPEs	0.1 μs	10
		Final simdized version	0.1 μs	120000

10000 faster!

Motivated, but not restricted by Cell !

- 2 Intel Xeon Processors with Hyper-Threading enabled and 512 kB cache at 2.66 GHz;
- 2 Dual Core AMD Opteron Processors 265 with 1024 kB cache at 1.8 GHz;
- 2 Cell Broadband Engines with 256 kB local store at 2.4G Hz.

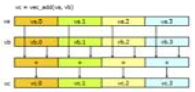


lxg1411@GSI
eh102@KIP
blade11bc4 @IBM

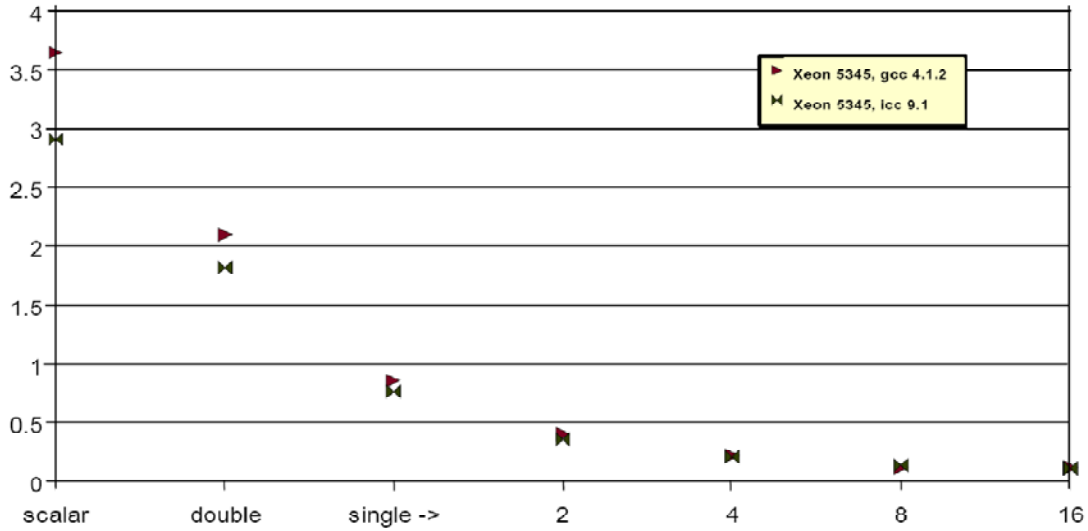
Comp. Phys. Comm. 178 (2008) 374-383

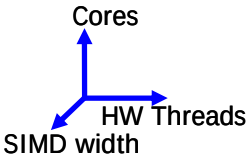
Performance of the KF Track Fit on CPU/GPU Systems

Type	Time/track, μs	Speed-up
Scalar double	2.6	–
Vector double	1.6	1.6
Vector single	0.7	2.3

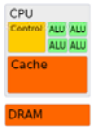


Speed-up 3.7 on the Xeon 5140 (Woodcrest) at 2.4 GHz using icc 9.1





Real-time performance on the quad-core Xeon 5345 (Clovertown) at 2.4 GHz – speed-up 30 with 16 threads



Type	Cores	Clock, GHz	Time/track, μs
Core 2	2	2.66	0.26
Core i7	4	3.2	0.1

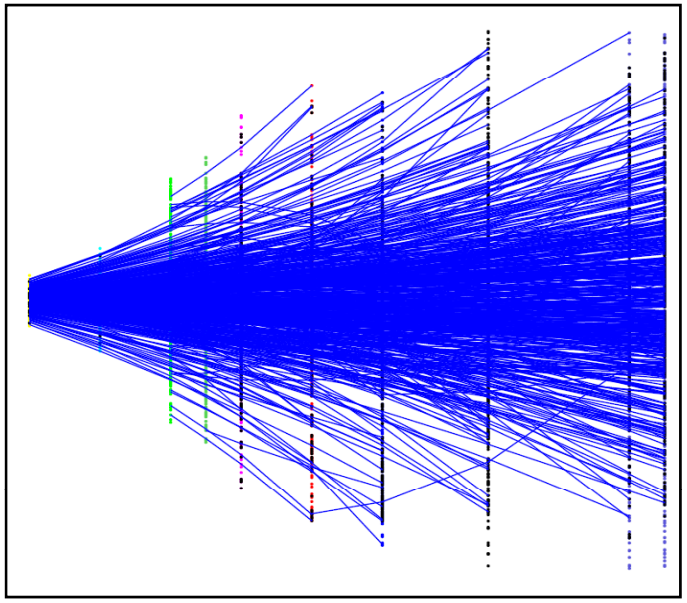
Real-time performance on different Intel CPU platforms

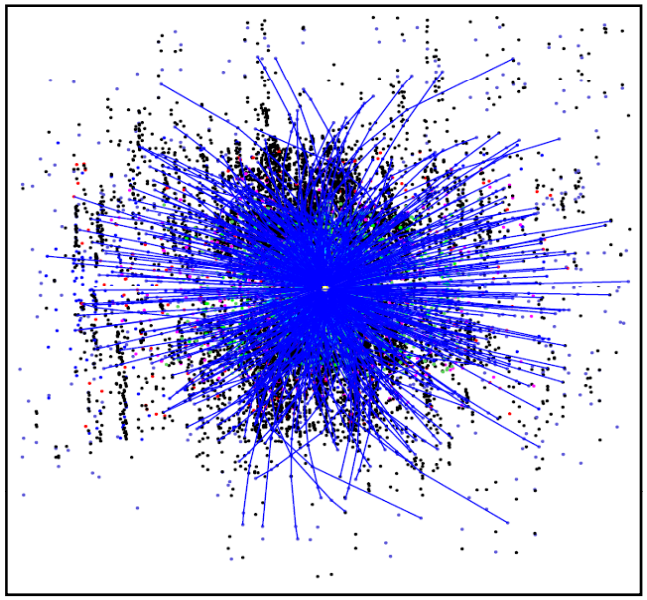
NVIDIA Unit	Clock, GHz	Throughput, 10^6 tr./s
8800 GTS 512	1.6	13.0
GTX 280	1.3	21.7

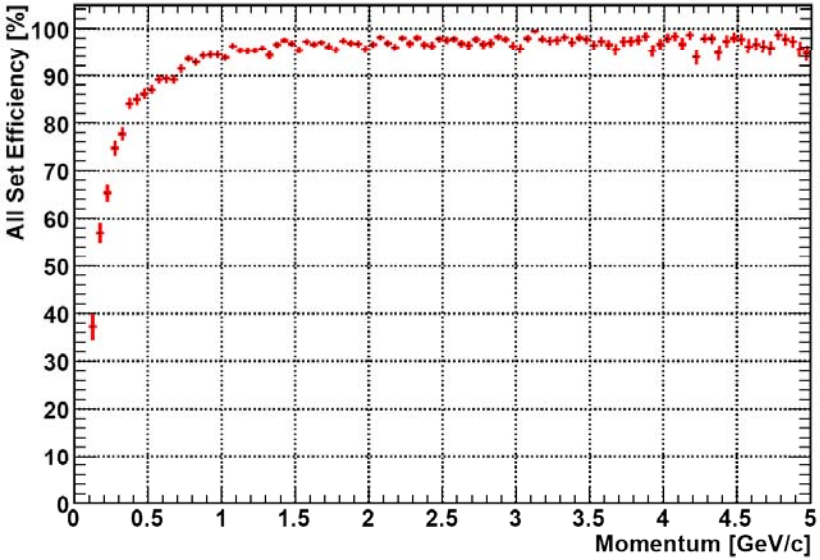
Real-time performance on NVIDIA for a single track

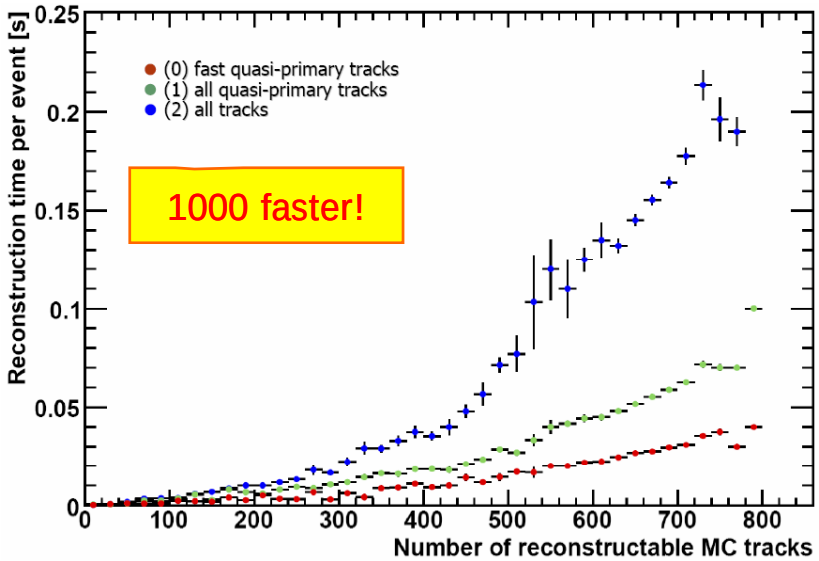
CBM Progress Report, 2008

Cellular Automaton Track Finder: Pentium 4









Summary

- Standalone package for online event selection is ready for investigation
- Cellular Automaton track finder takes 5 ms per minimum bias event
- Kalman Filter track fit is a benchmark for modern CPU/GPU architectures
- SIMDized multi-threaded KF track fit takes 0.1 μs/track on Intel Core i7
- Throughput of $2.2 \cdot 10^7$ tracks /sec is reached on NVIDIA GTX 280