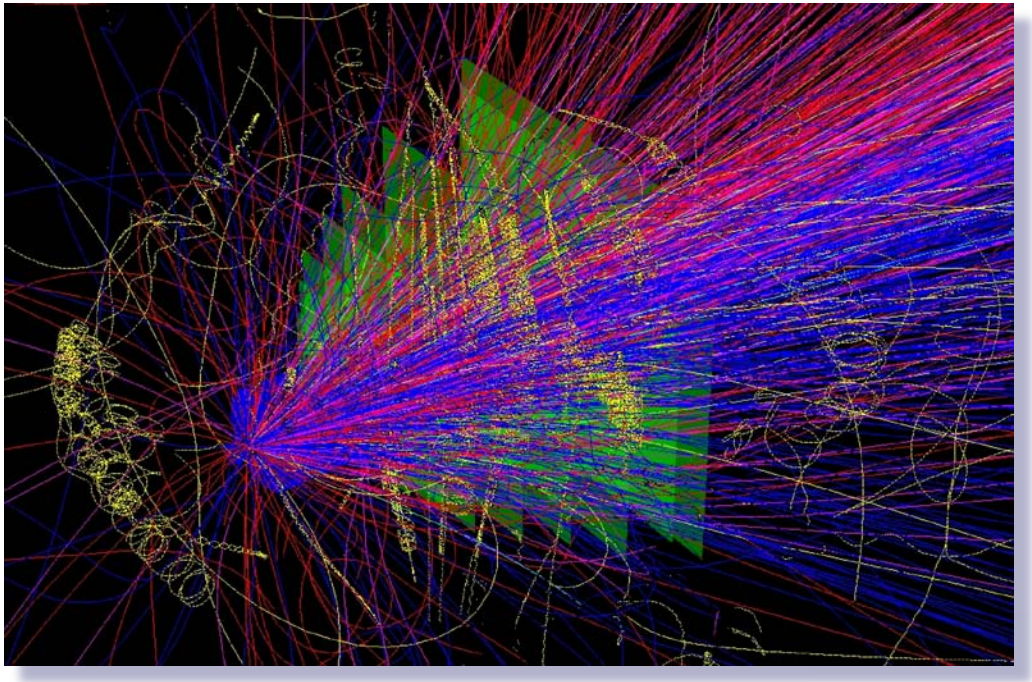
Parallel Approach to Online Event Reconstruction in the CBM Experiment

Ivan Kisel
GSI, Darmstadt, Germany
(for CBM Collaboration)

ACAT-2010, Jaipur, India
February 26, 2010

Tracking Challenge in CBM (FAIR/GSI, Germany)

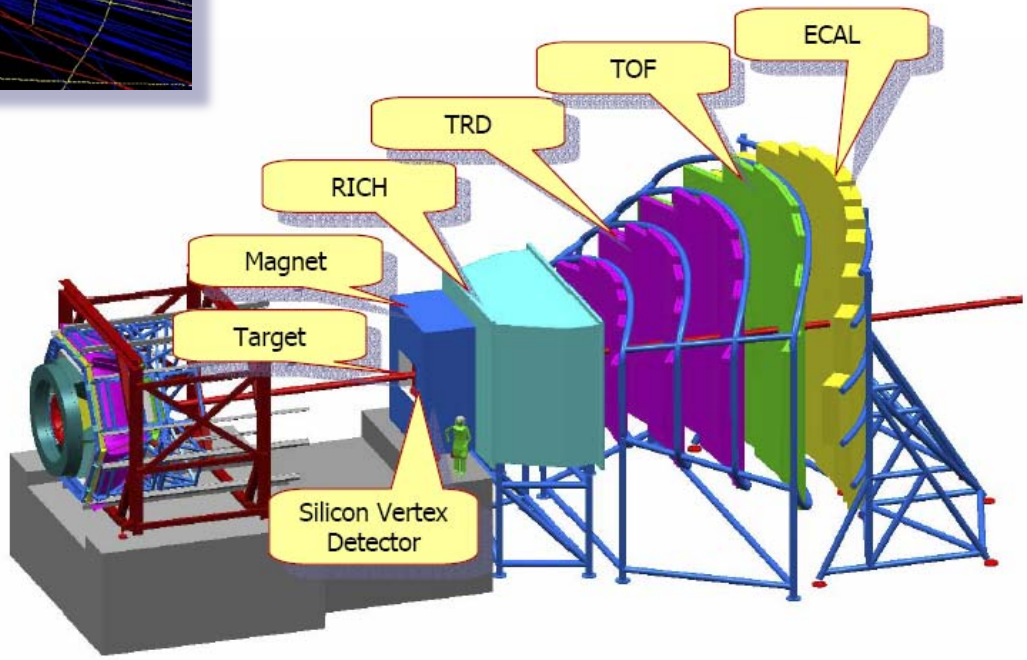


* Fixed-target heavy-ion experiment
* 10^7 Au+Au collisions/sec
* ~ 1000 charged particles/collision
* Non-homogeneous magnetic field
* Double-sided strip detectors
  (85% fake space points)

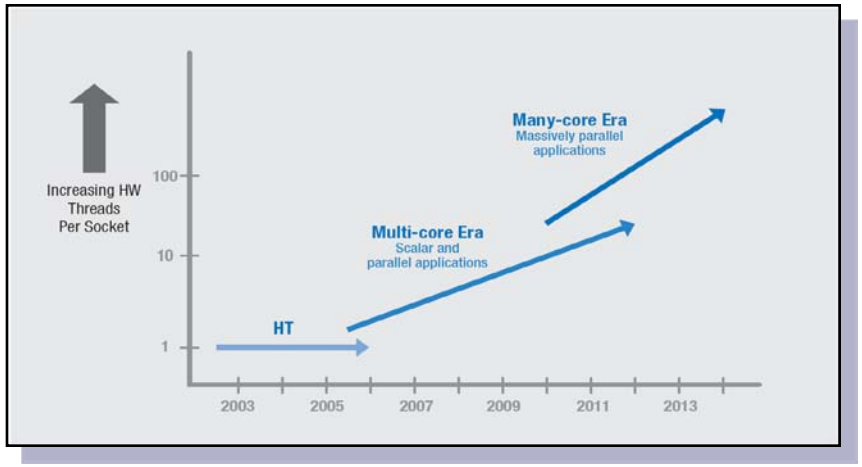
Track reconstruction in STS/MVD
and displaced vertex search
are required in the first trigger level.

Reconstruction packages:

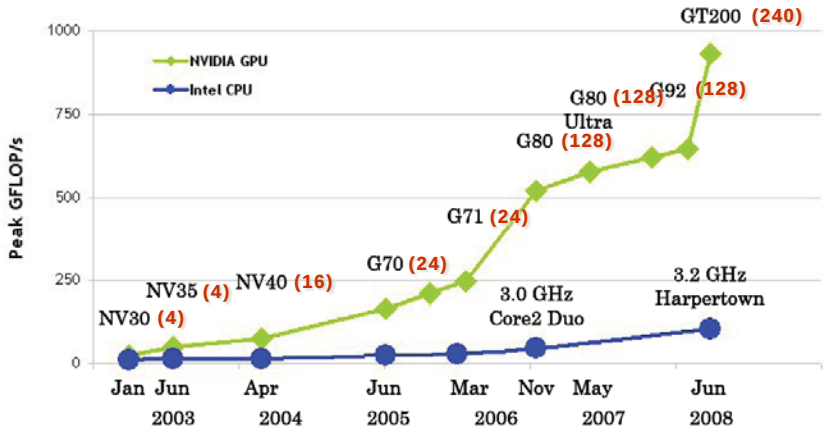
track finding	Cellular Automaton (CA)
track fitting	Kalman Filter (KF)
vertexing	KF Particle



Many-Core CPU/GPU Architectures

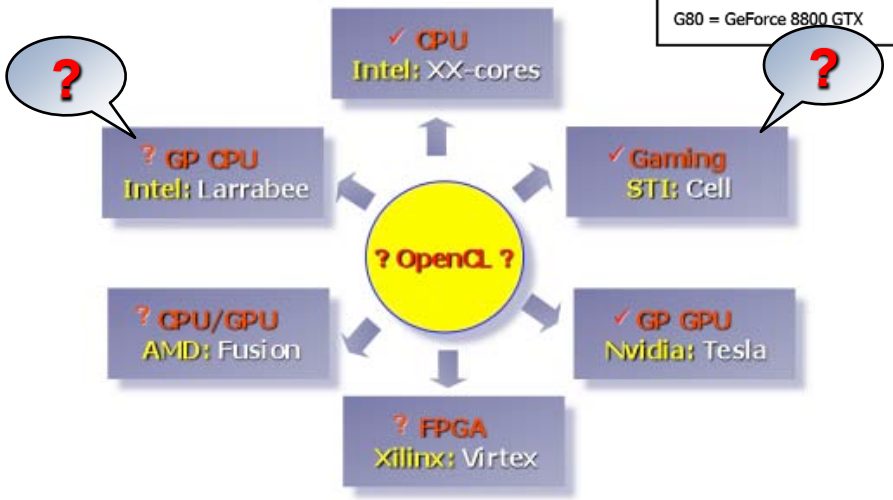


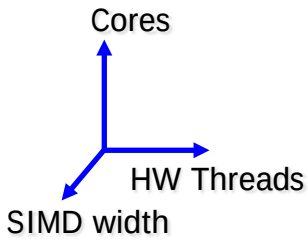
S. Borkar et al. (Intel), "Platform 2015: Intel Platform Evolution for the Next Decade", 2005.



GT200 = GeForce GTX 280	G71 = GeForce 7900 GTX	NV35 = GeForce FX 5950 Ultra
G92 = GeForce 9800 GTX	G70 = GeForce 7800 GTX	NV30 = GeForce FX 5800
G80 = GeForce 8800 GTX	NV40 = GeForce 6800 Ultra	

Source: NVIDIA



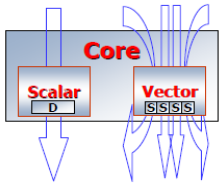


- On-line event selection
- Mathematical and computational optimization
- Optimization of the detector

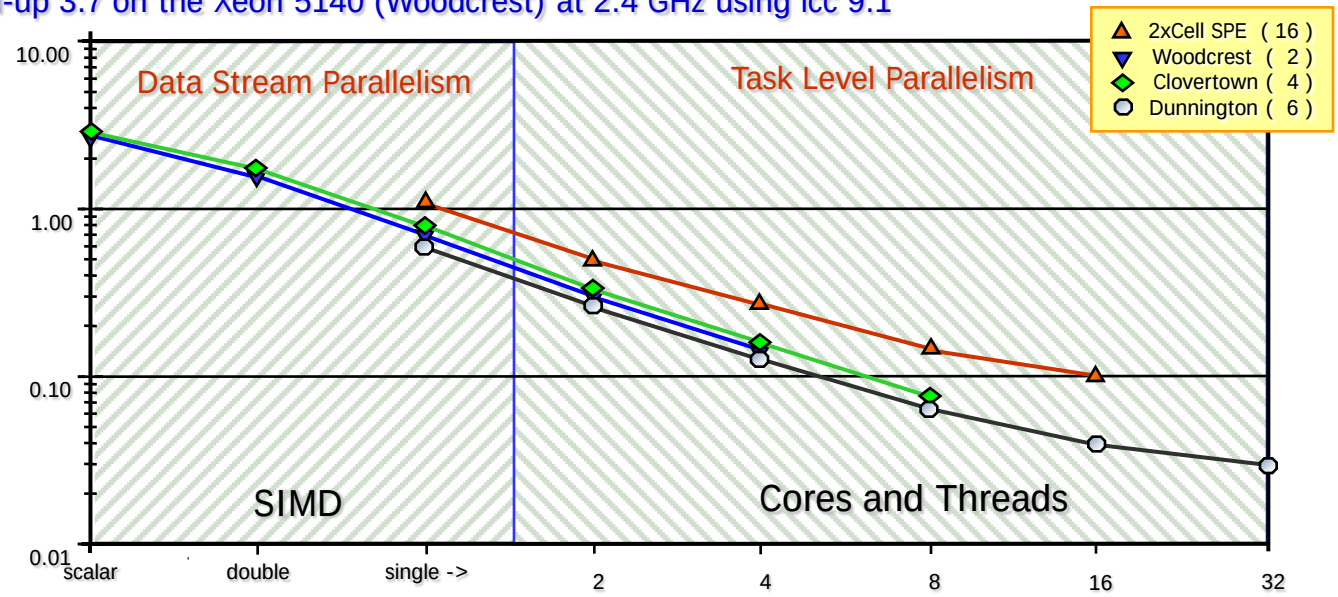
- Heterogeneous systems of many cores
- Uniform approach to all CPU/GPU families
- Similar programming languages (CUDA, Ct, OpenCL)
- Parallelization of the algorithm (vectors, multi-threads, many-cores)

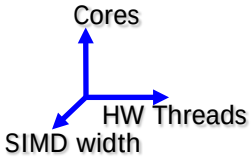
Performance of the KF Track Fit on CPU/GPU Systems

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

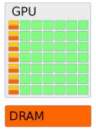




Scalability (μs) on different CPU architectures – speed-up 100 with 32 threads

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

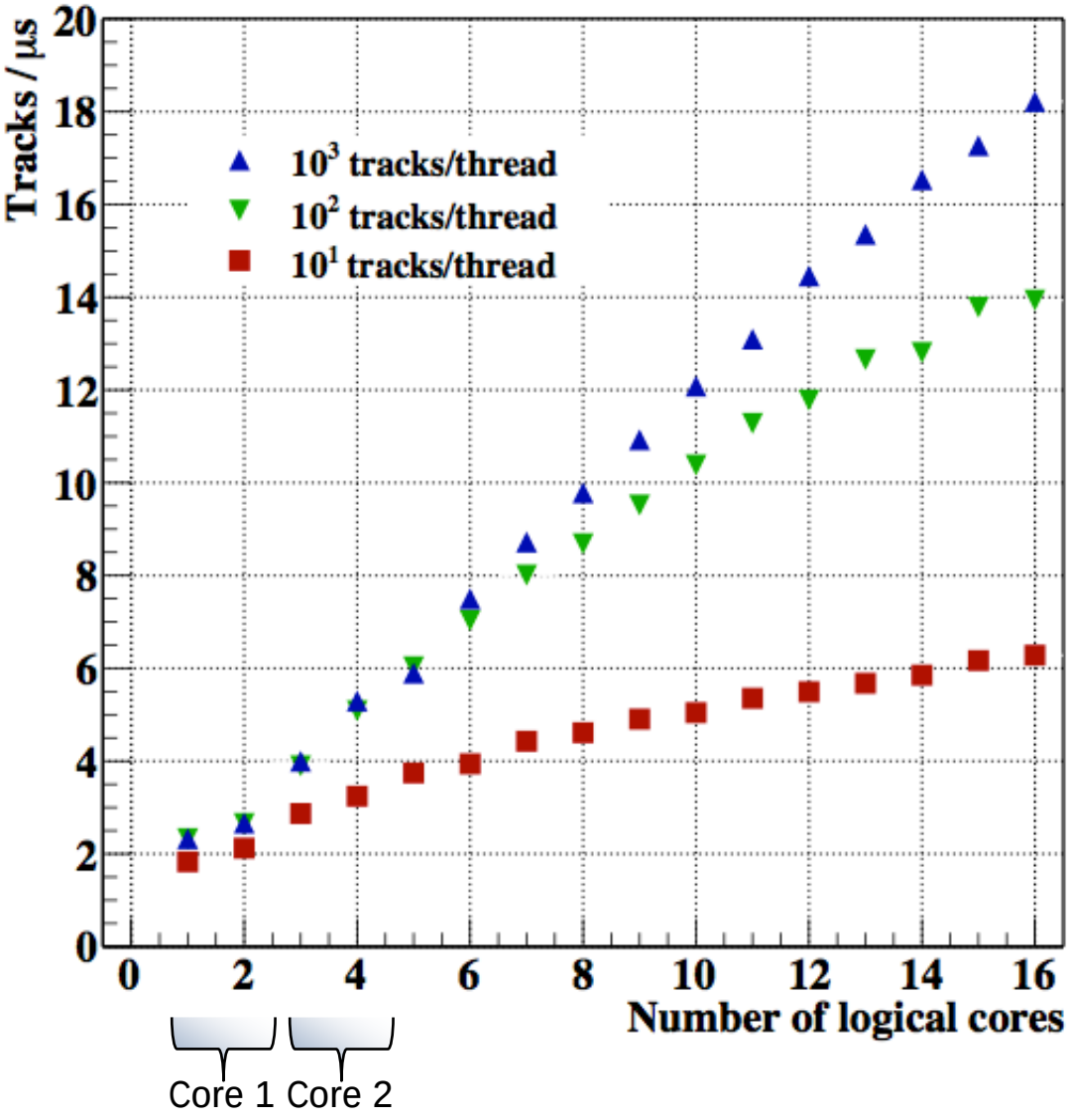
Real-time performance on different Intel CPU platforms



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

Scalability of the KF Track Fit on 2xNehalem = 8 Cores



Each physical core of Nehalem has 2 HW threads (read/exe), which are considered by the operating system as logical cores.

Aiming to develop a scheduler, we investigate scalability and CPU load.

Measure tracks throughput rather than time per track.

Given n threads each filled with 10^m tracks, run them on specific n logical cores with 1 thread per 1 logical core.

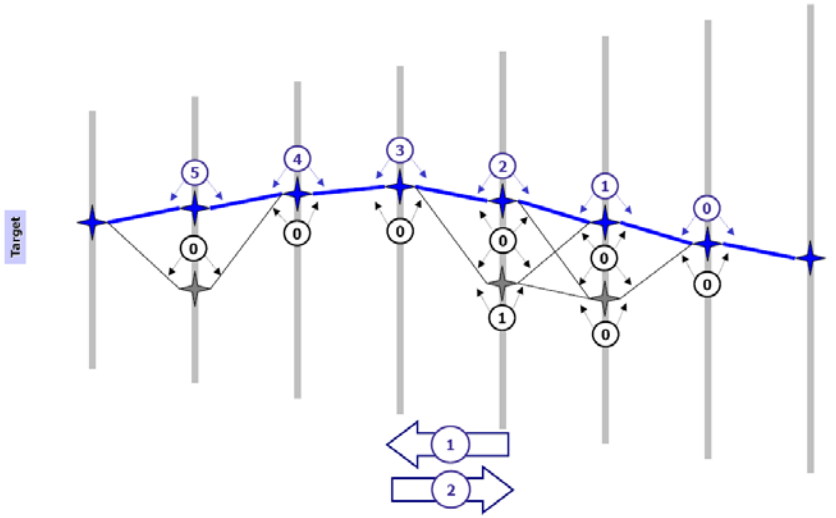
For small groups of tracks the overhead becomes significant, while large groups of tracks use CPU more efficient.

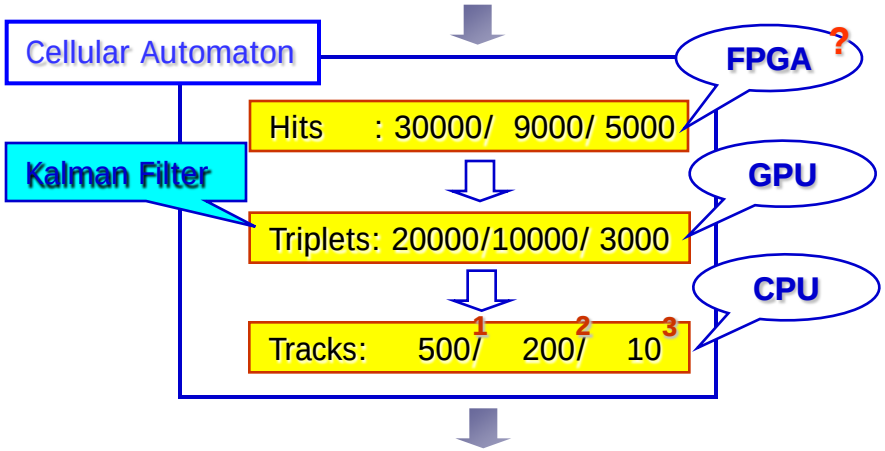
CBM Cellular Automaton Track Finder

```
01 void CATrackFinder(){
02   for (step = 0; step <= 2; step++){
03     if (step == 0) MinMom = 1.0; else MinMom = 0.2;
04     if (step <= 1) TargetConstrHit(ht); else NoTargetConstrHit(ht);
05     for (sta = NStations-3; sta >= 0; sta--){
06       for (hl = FirstHit[sta]; hl <= LastHit[sta]; hl++){
07         if (hl.used) continue;
08         Triplet trl = Triplet(ht, hl, MinMom);
09         trl.Propagate(sta+1);
10         for (hm = FirstHit[sta+1]; hm <= LastHit[sta+1]; hm++){
11           if (hm.used) continue;
12           Triplet trm = Triplet(trl);
13           trm.AddHit(hm);
14           if (trm.chi2 > MaxChi2) continue;
15           trm.Propagate(sta+2);
16           for (hr = FirstHit[sta+2]; hr <= LastHit[sta+2]; hr++){
17             if (hr.used) continue;
18             Triplet trr = Triplet(trm);
19             trr.AddHit(hr);
20             if (trr.chi2 > MaxChi2) continue;
21             for (tr = FirstTriplet[hm]; tr <= LastTriplet[hm]; tr++){
22               if (trr.hr != tr.hm) continue;
23               if (fabs(trr.p-tr.p)/trr.errp > MaxDistP) continue;
24               if (trr.level <= tr.level) trr.level = tr.level+1;
25             }
26             trr.Store();
27             hl.StorePointer(trr);
28           }
29         }
30       }
31     }
32     for (level = NStations-3; level >= 0; level--){
33       for (tri = FirtsTriplet; tri <= LastTriplet; tri++){
34         if (tri.level != level) continue;
35         if ((tri.hl.used)||(tri.hm.used)||(tri.hr.used)) continue;
36         Track tra = FindBestBranchRecursive(tri);
37         if ((tra.chi2 > MaxChi2)||(GhostTrack(tra))) continue;
38         tra.StoreAsCandidate();
39       }
40       SortTrackCandidates();
41       for (tra = FirstTrackCand; tra <= LastTrackCand; tra++){
42         if (tra.NUsedHits() != 0) continue;
43         tra.MarkHitsAsUsed();
44         tra.Store();
45       }
46     }
47     MergeClones();
48     GatherHits();
49   }
50 }
```

1 Create triplets

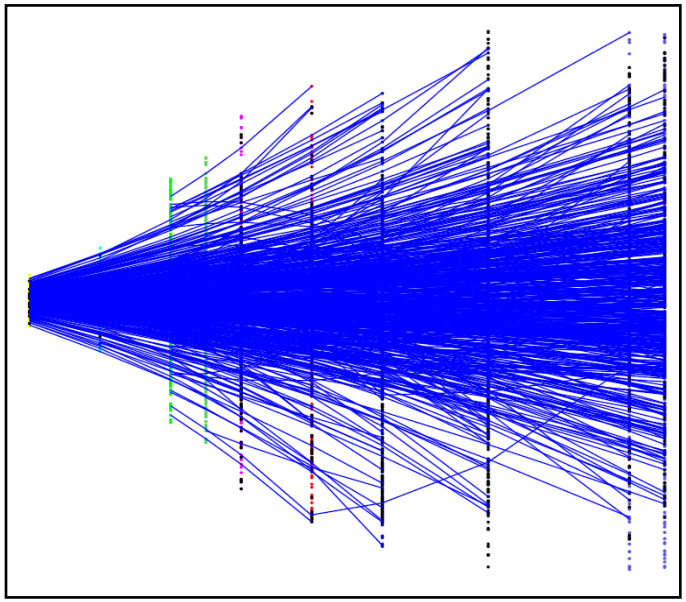
2 Collect tracks

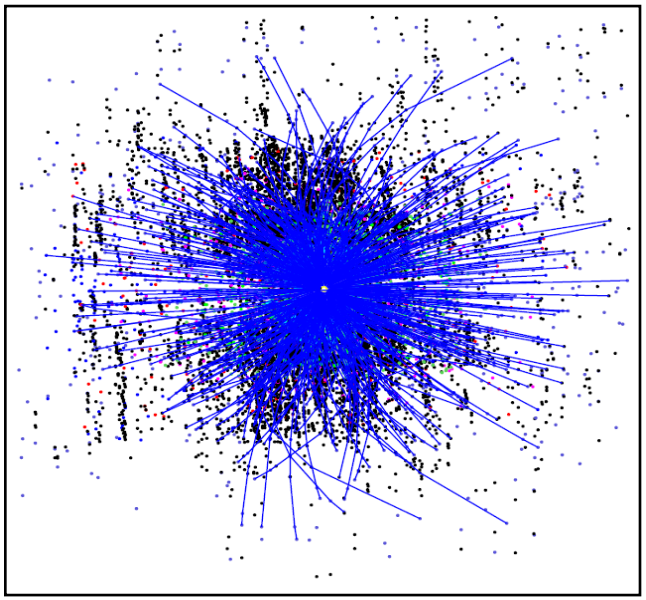


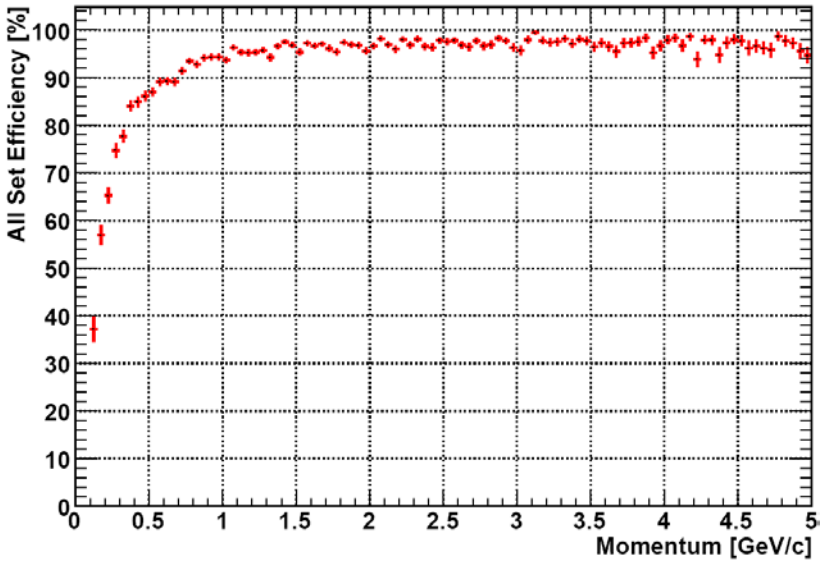


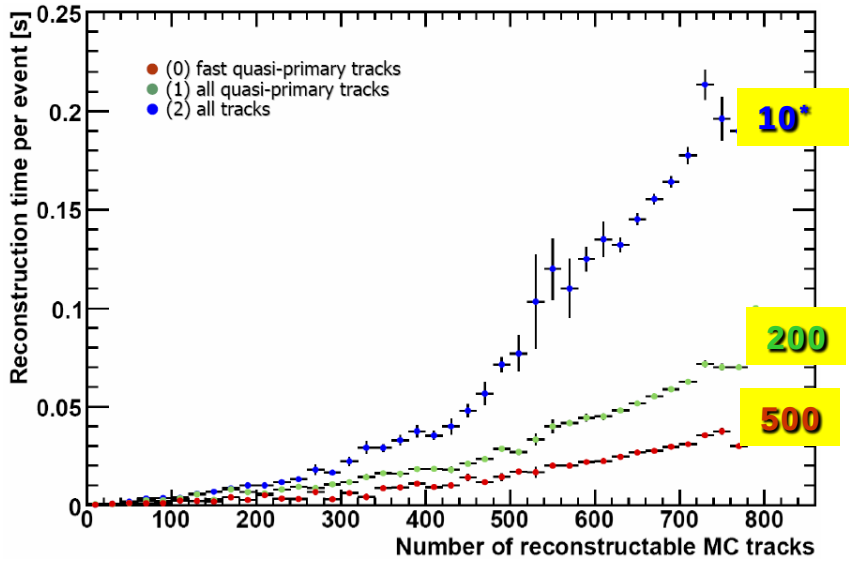
Three types of tracks and three stages of track finding:
fast primary tracks[1] / all primary tracks[2] / all tracks[3]

CBM Cellular Automaton Track Finder



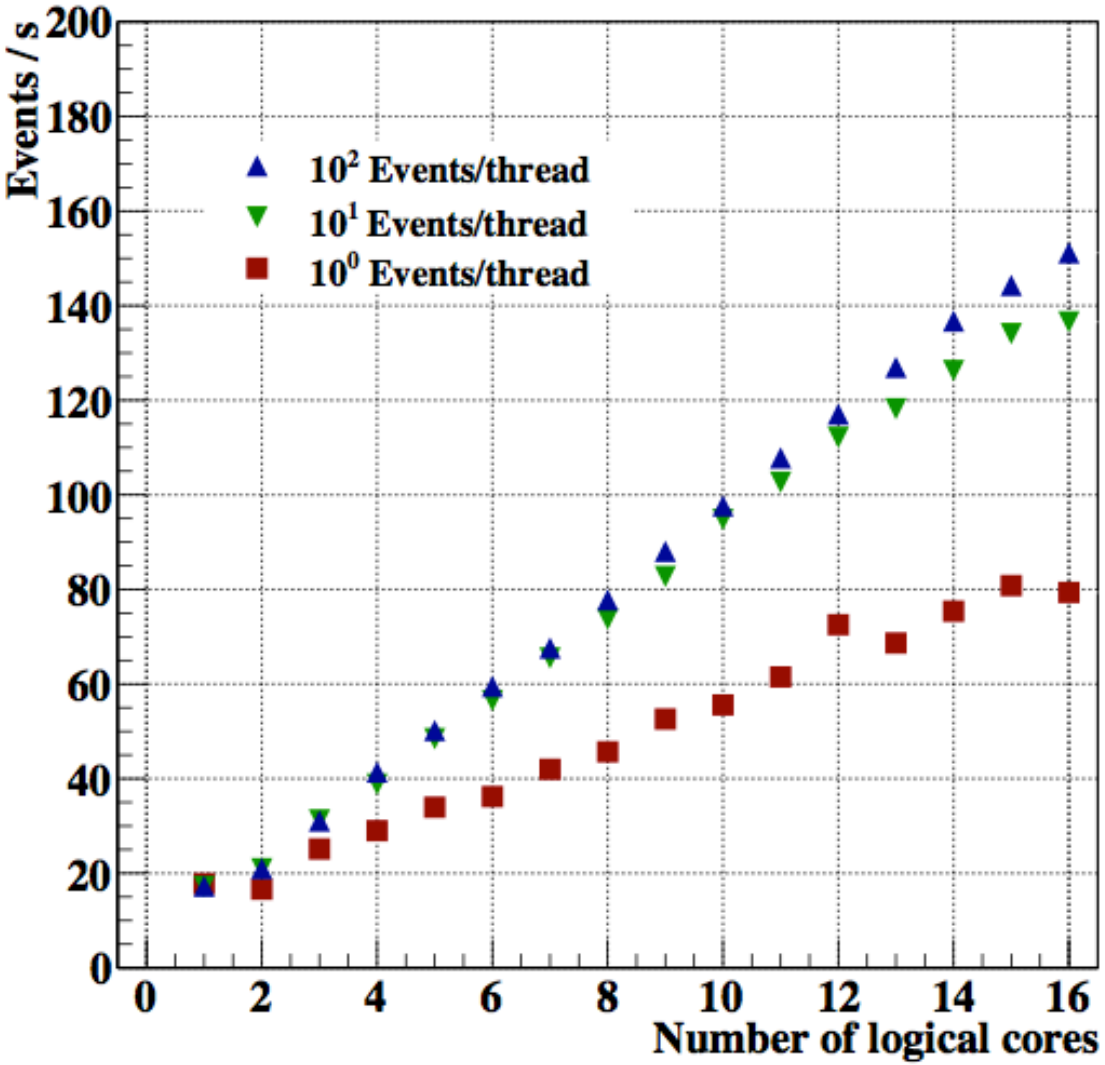






*Avoid random access to data -> appropriate sorting of the measurements is necessary

Scalability of the CA Track Finder on 2xNehalem = 8 Cores



Each physical core of Nehalem has 2 HW threads (read/exe), which are considered as logical cores by the operating system

Aiming to develop a scheduler, we investigate scalability and CPU load.

Measure events throughput rather than time per event.

Given n threads each filled with 10^m central events (~700 tracks/event), run them on specific n logical cores with 1 thread per 1 logical core.

For single events the overhead becomes significant, while large groups of events use CPU more efficient.

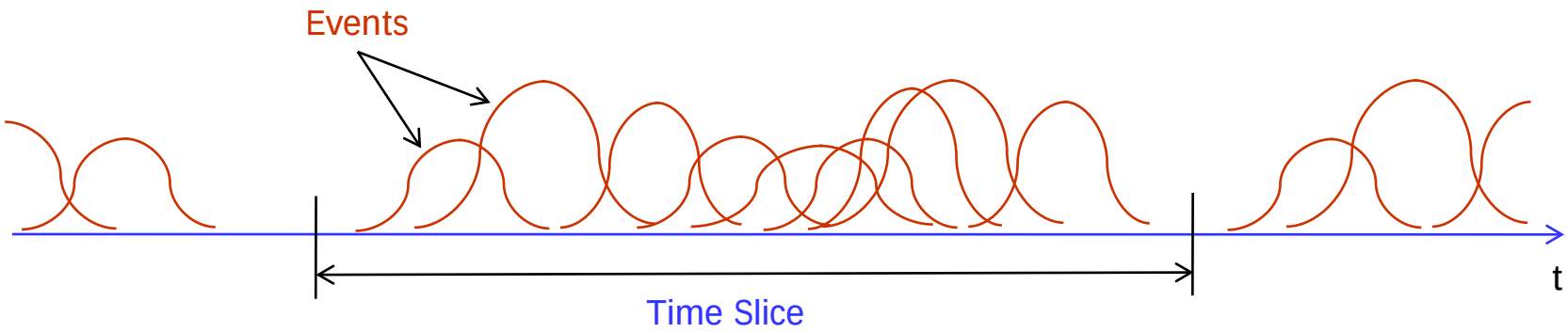
150 central ev/s/PC = 1500 mbiasev/s/PC

10^7 input rate-> $60*10^3$ cores

Time Slices

We observe, that central events seem too small to be parallelized between cores, one core per central event is optimum – therefore, event level parallelism between cores.

Minimum bias events are too small even for a single core, therefore several minimum bias events must be packed together and reconstructed as a single event with measurements having 3 coordinates and event index.



A beam in the CBM experiment will have no bunch structure, but continuous. Separation of events will not be possible, instead groups of events close or overlapped in time (time slices) will be provided. Measurements in this case will be 4D (x, y, z, t). Such complex event topology will efficiently load CPU cores, but will make the online event reconstruction and selection extremely sophisticated:

- high interaction rate,
- non-homogeneous magnetic field,
- large track density,
- 4D measurements,
- many-core CPU/GPU architectures.

Parallelization is now a Standard in the CBM Reconstruction

CBM Tracking Workshop (15-17 June 2009, GSI):

- Tracking Presentations
- Tracking Discussion
- "Future Intel CPU Architecture"
- "OpenCL Parallel Language"
- Training Day on SIMD/MT

Next Tracking Workshop in May 2010

Algorithm	Vector SIMD	Multi-Threading	NVIDIA CUDA	OpenCL	Speedup	Speed/PC
STS	+	+	+	+	10000	6.5 ms
MuCh	+	+			500*	1.5 ms
TRD	+	+			500*	1.5 ms
RICH	+	+			100**	3.0 ms
Vertexing	+		Future		1.5***	20µs
Open Charm Analysis	+				1.5***	20µs
User Reco/Digi		Future				
User Analysis						

+March 2009
+October 2009

*Single hit access should be avoided
**Reformulation of the algorithm is probably necessary
***Avoid accessing the main memory –> approximation of the magnetic field map

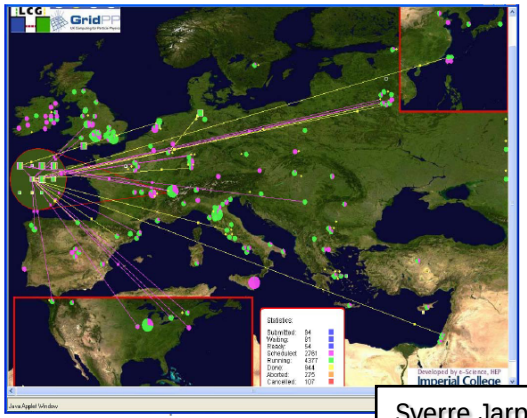
First Level Event Selection (FLES) Complexity

Bergen – 9 February 2009

World-wide LHC Computing Grid

- **Largest Grid service in the world !**

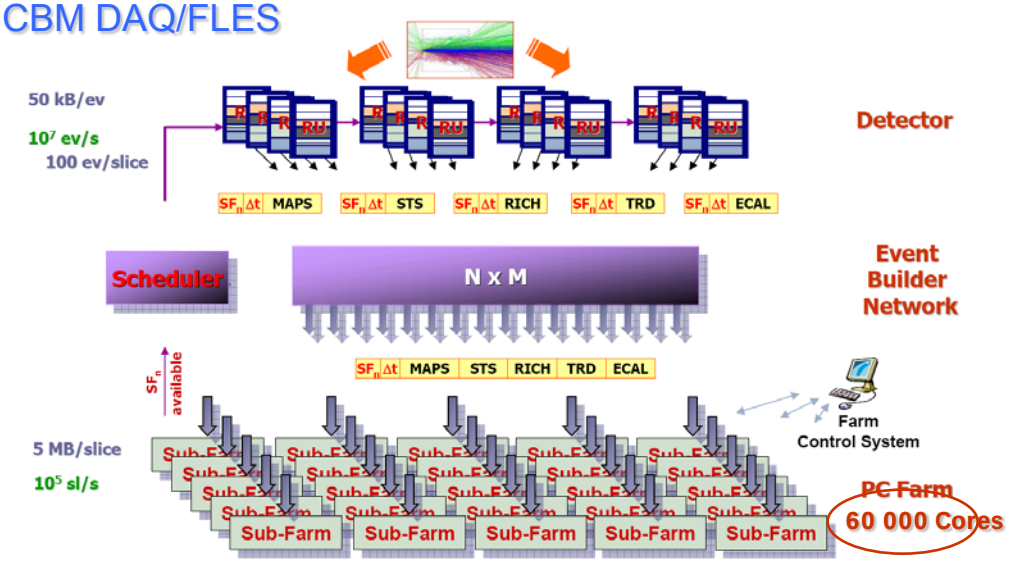
- Around 140 sites in 35 countries
- Tens of thousands of Linux PC servers (over 100'000 cores)
- Tens of petabytes of storage

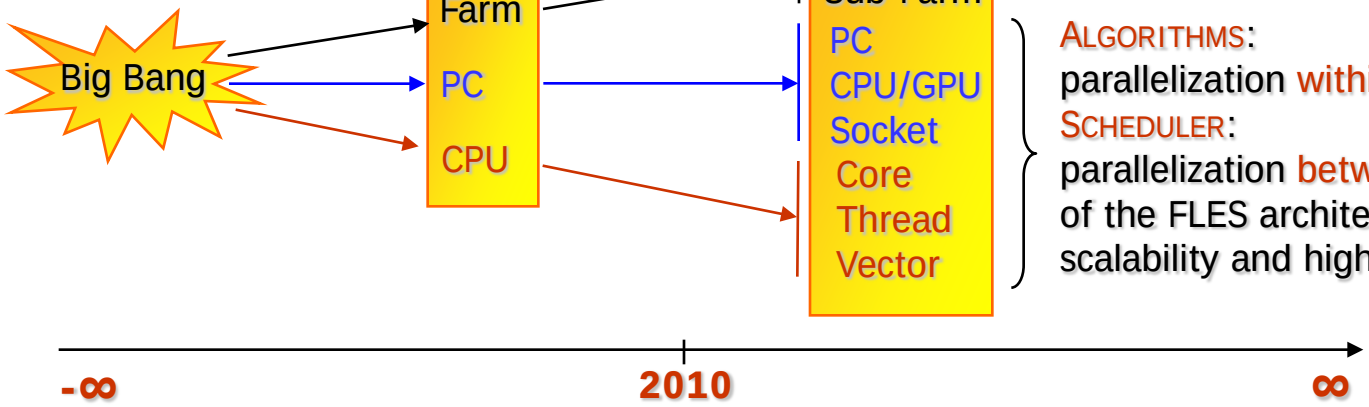


Sverre Jarp

9

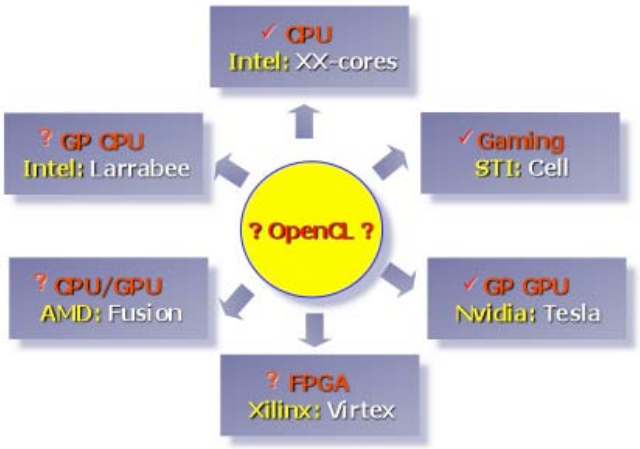
CBM DAQ/FLES

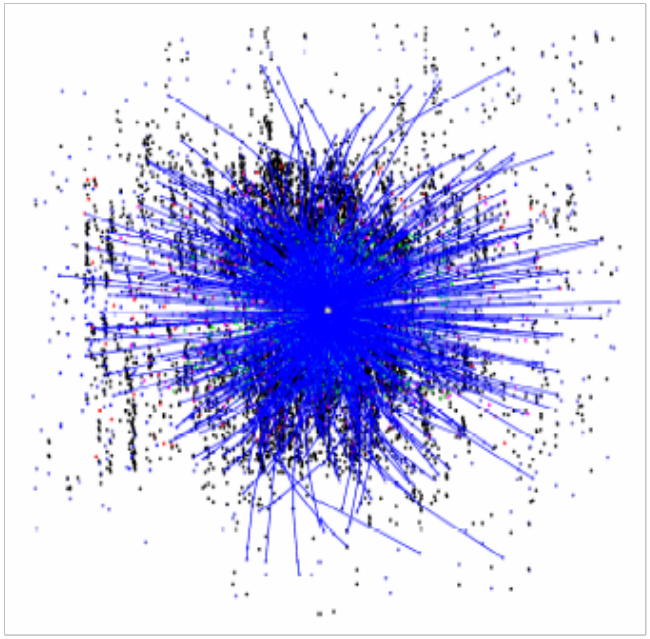




ALGORITHMS:
parallelization within the layers
SCHEDULER:
parallelization between the layers
of the FLES architecture to provide
scalability and high CPU/GPU load

Summary





- Parallel programming is the key to the full potential of the Tera-scale platforms
- Parallelization is now a standard in the CBM reconstruction
- Data parallelism vs. parallelism of the algorithm
- Algorithms optimal w.r.t. parallelism
- Use SIMD unit (many-cores, TF/s, ...)
- Multi-Threading to parallelize between cores
- Keep portability of the code on heterogeneous systems (CPU, GP GPU, ...)
- New parallel languages appear: CUDA, OpenCL, Ct
- First Level Event Selection (FLES) project