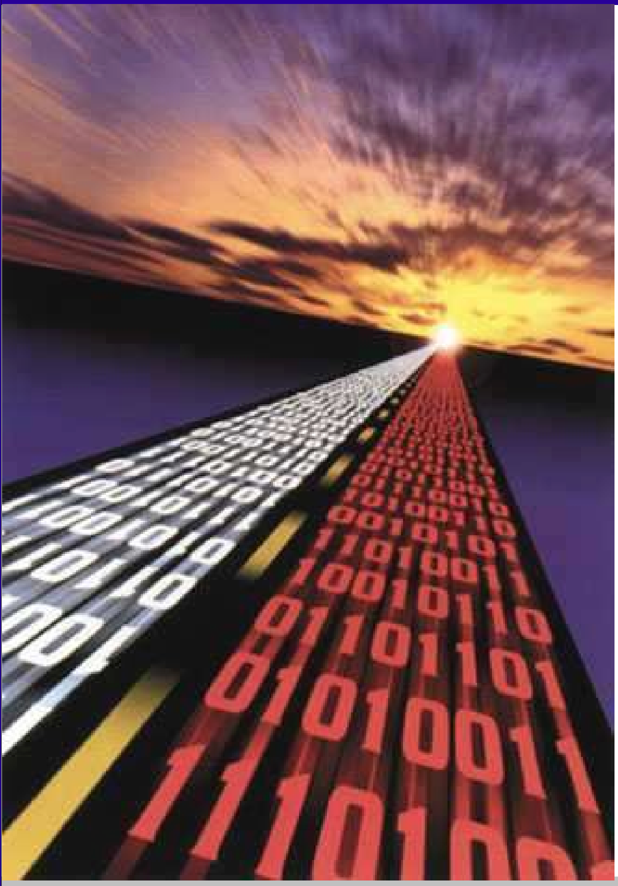


Track Reconstruction Algorithms for High Energy Physics on Many-Core Systems

Udo Kebschull
Ivan Kisel

Kirchhoff Institute for Physics
Computer Engineering in Physics (TIP)
University Heidelberg, Germany
Phone: +49 6221 54 9800
Fax: +49 6221 54 9809
Email: udo@kebschull.org
WWW: www.ti.uni-hd.de

24. April 2009

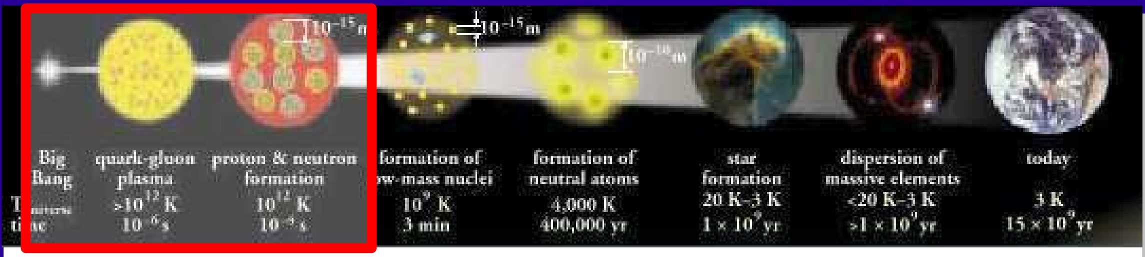
Agenda

- High Energy Physics Experiments
  - ALICE Experiment @ CERN
  - CBM @ FAIR/GSI
- Tracking Algorithms
  - Track Finding: Cellular Automaton
  - Track Fitting: Kalman Filter
- Multi- and Many- Core Architectures
- Results
- Conclusions

High Energy Physics Experiments

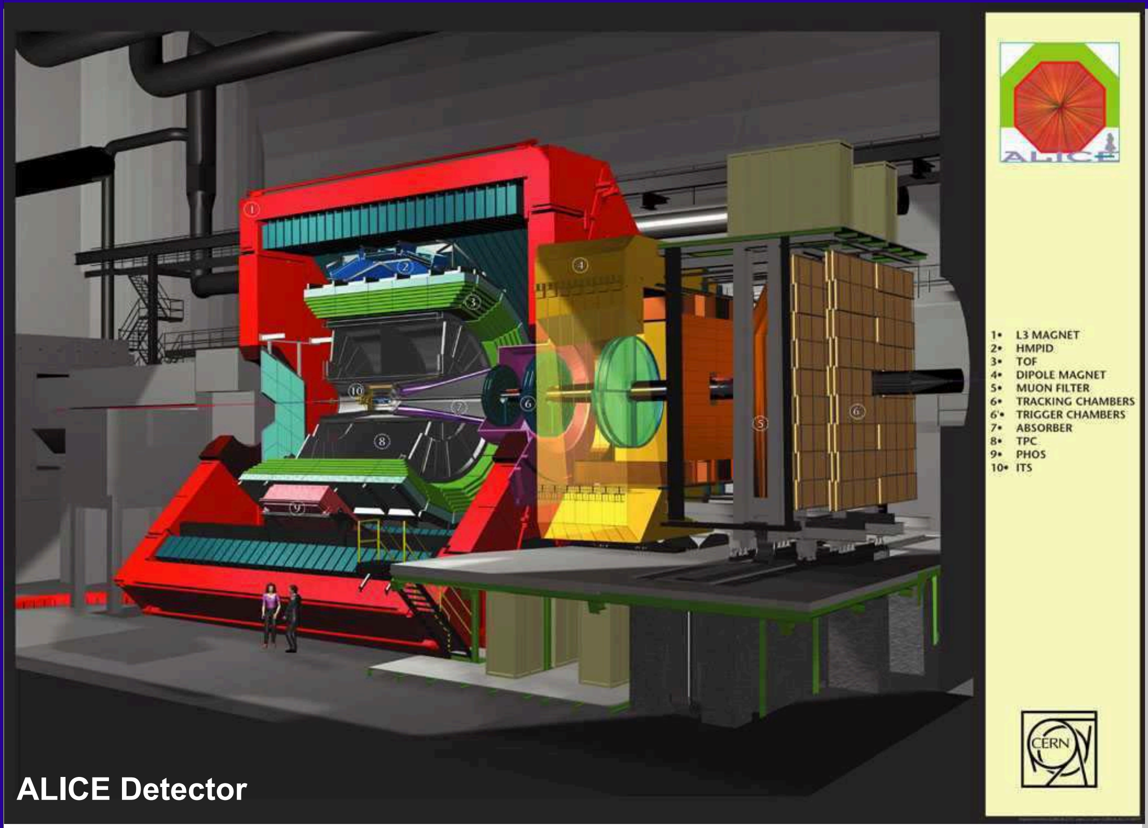
Large Hadron Collider at CERN

Let's go backwards to the Big Bang

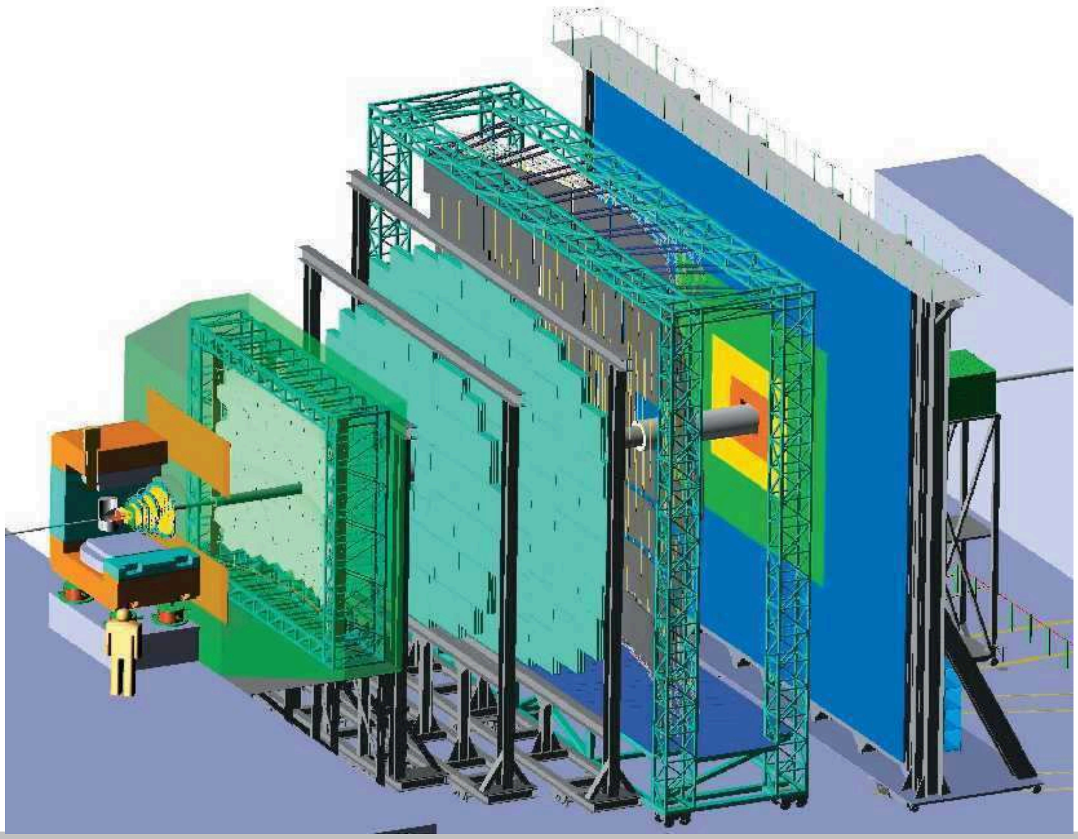


- Why ?
  - Observe the strong interaction in action
  - How do elementary particles interact?
  - How did the interaction give rise to the composite particles which constitute the universe?

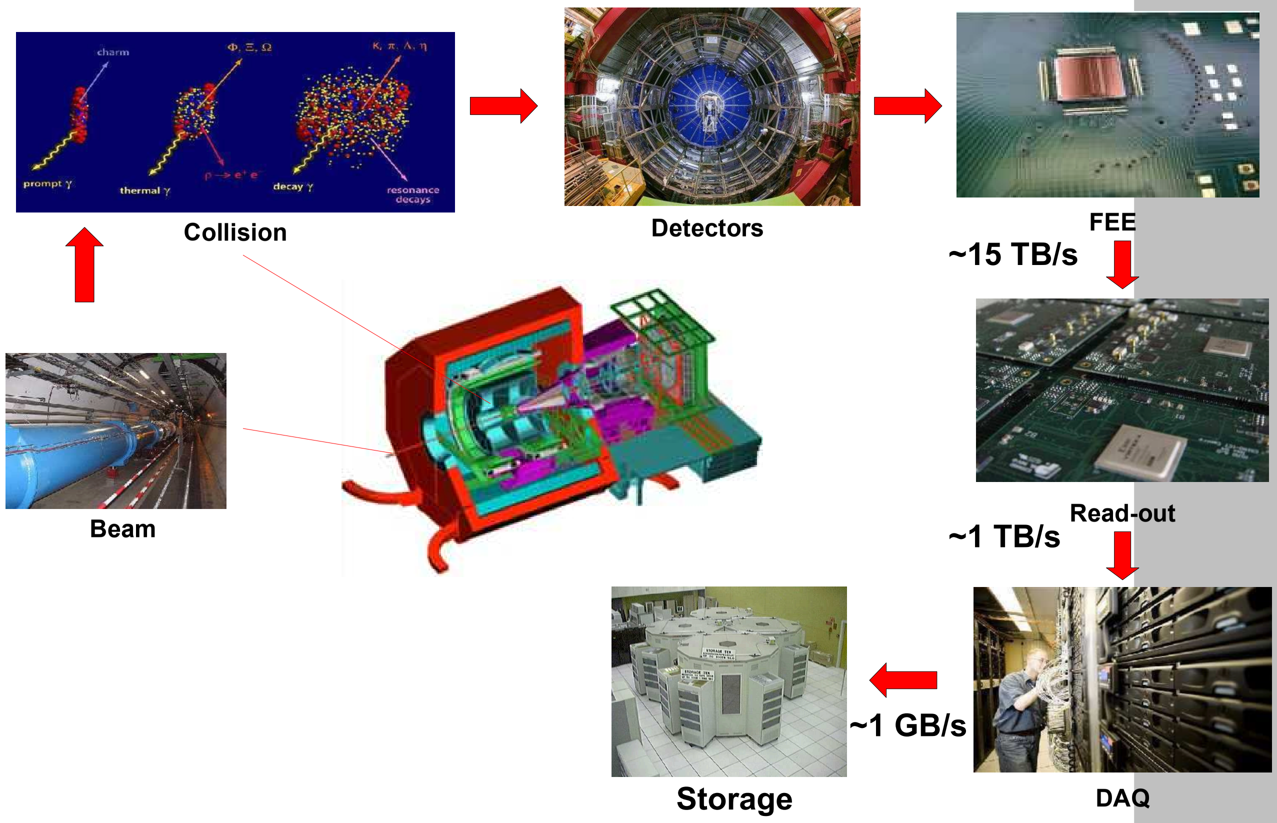
The ALICE Experiment



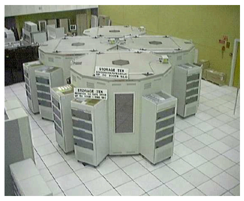
CBM Experiment @ FAIR/GSI, Darmstadt

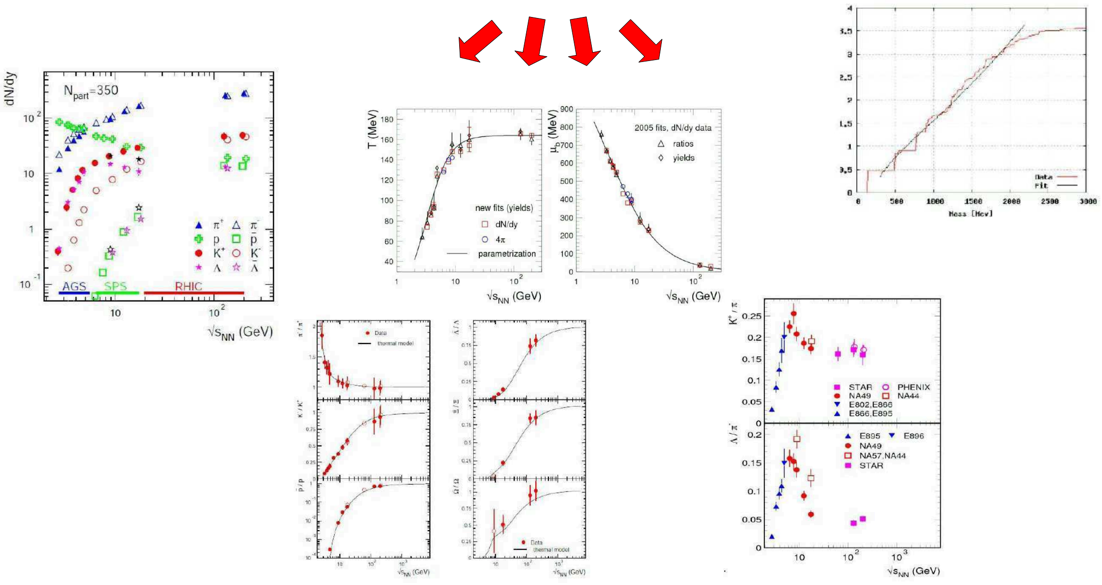


CBM Data Flow: Online Processing



CBM Data Flow: Offline Processing

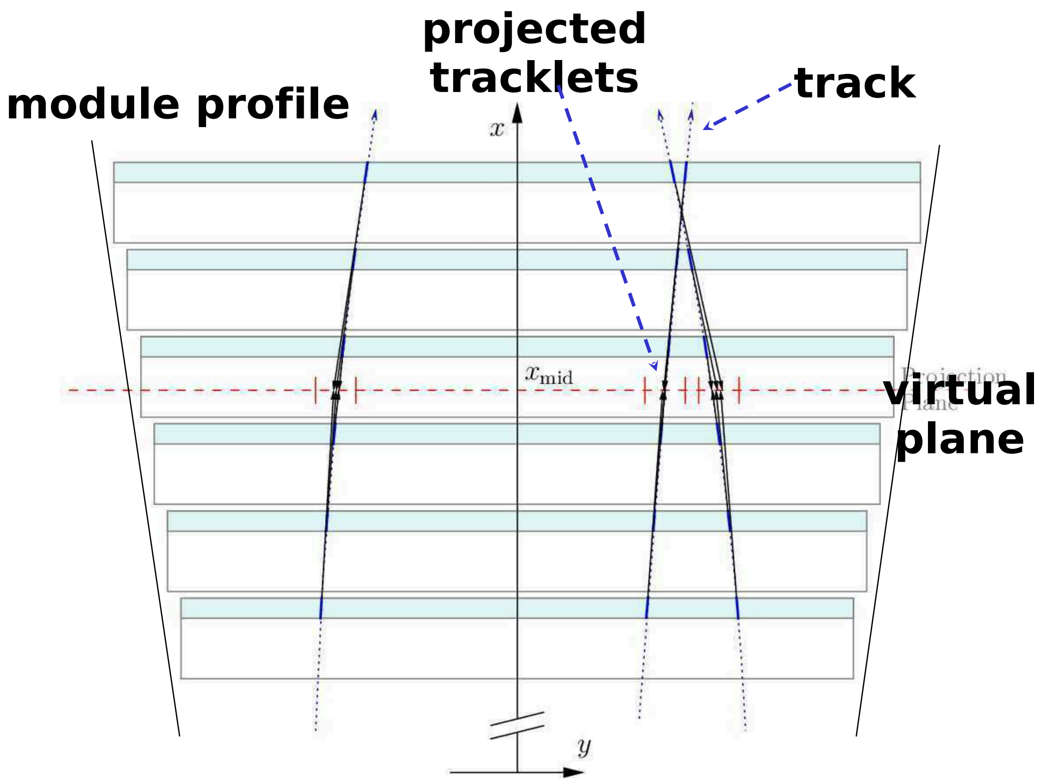




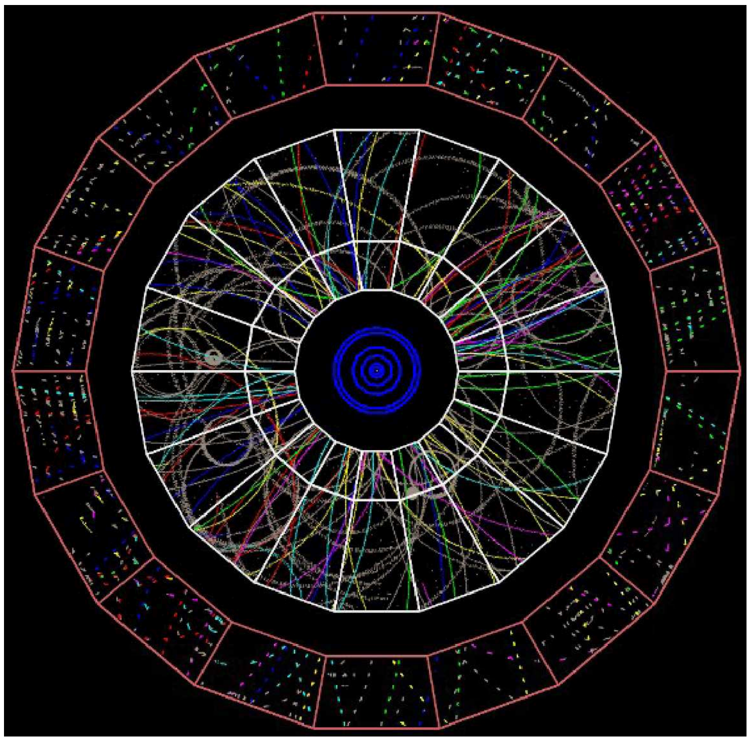
Tracking Algorithms

From Raw Data to Tracks

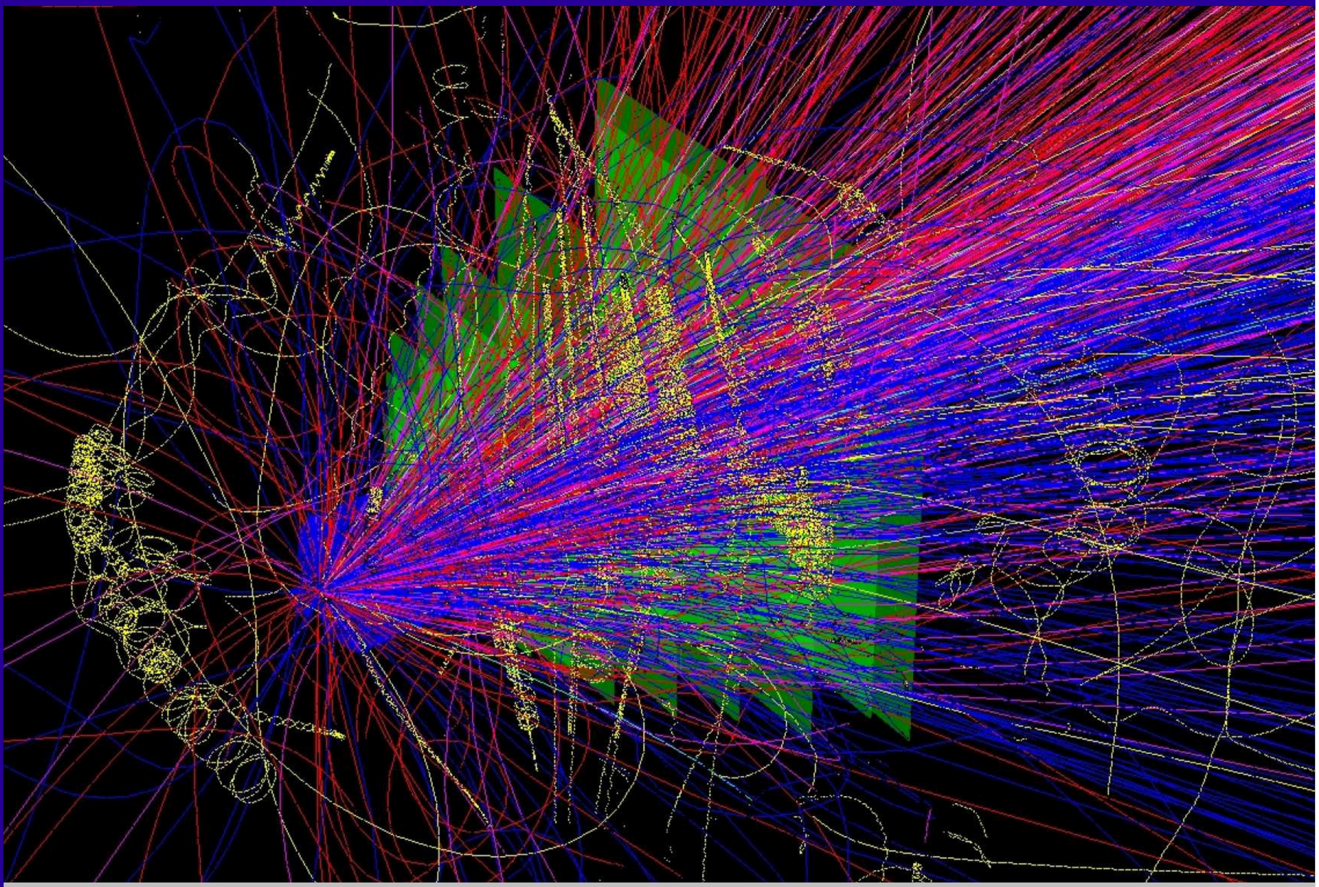
Track Re-assembly (Transition Radiation Detector)



ALICE Online Event Display

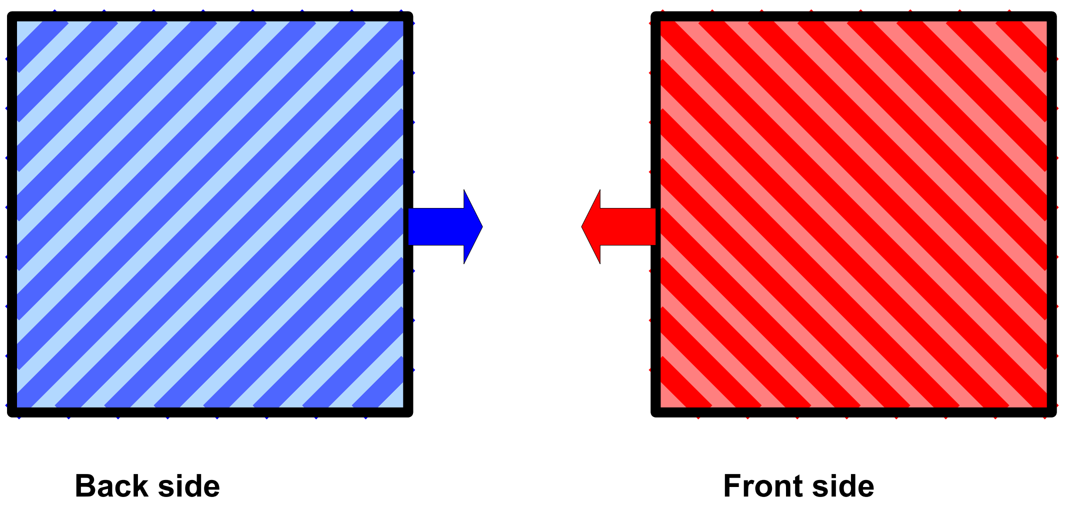


CBM: Simulated Tracks

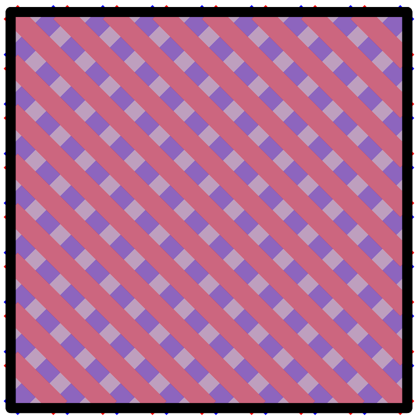


Strip Sensors

How to detect particles

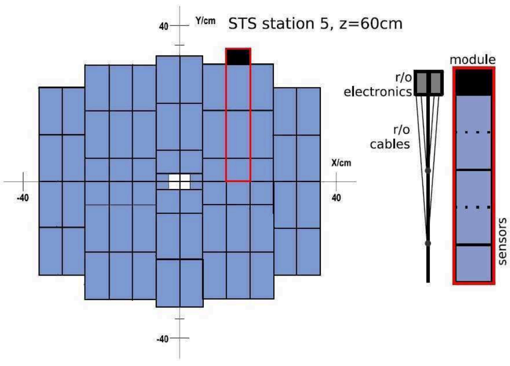


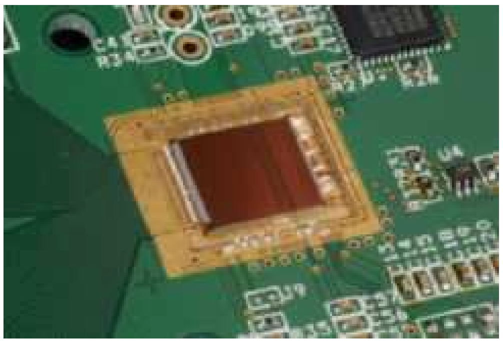
Strip Sensor



Sensor Dimension
Size 6x6 cm
Pitch 58µm, 1024 lines

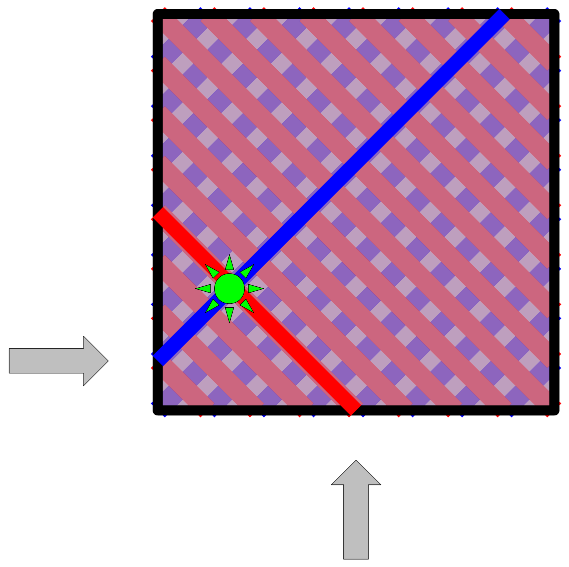
STS Detector



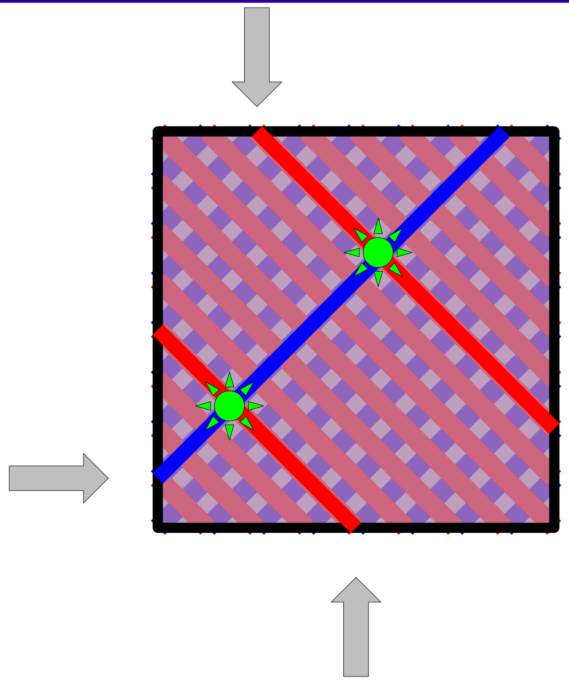


[J. M. Heuser, GSI]

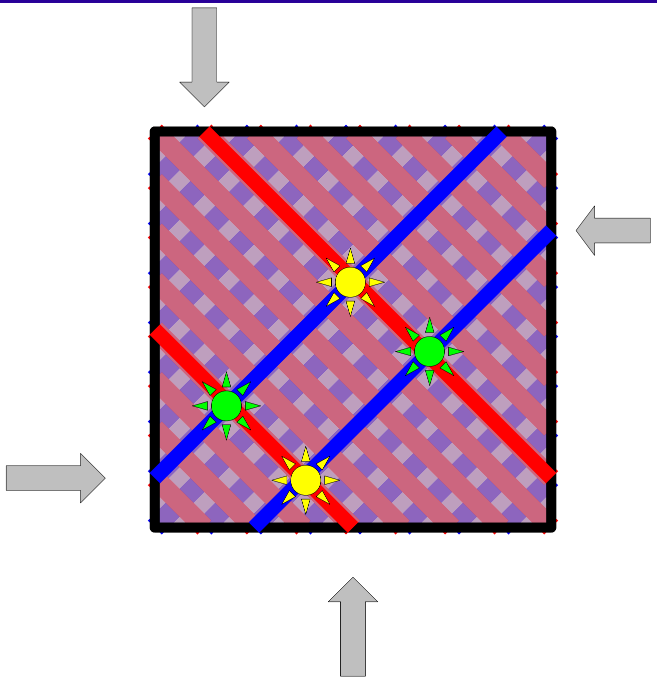
Track Finding Basic Principle



Track Finding

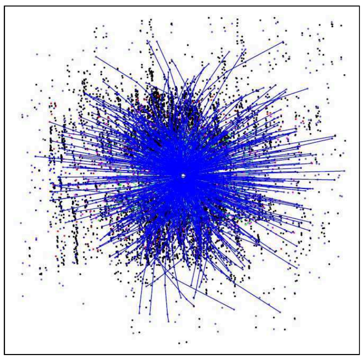


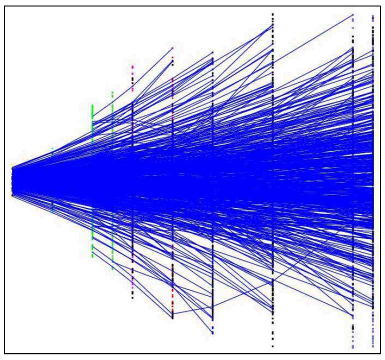
Track Finding



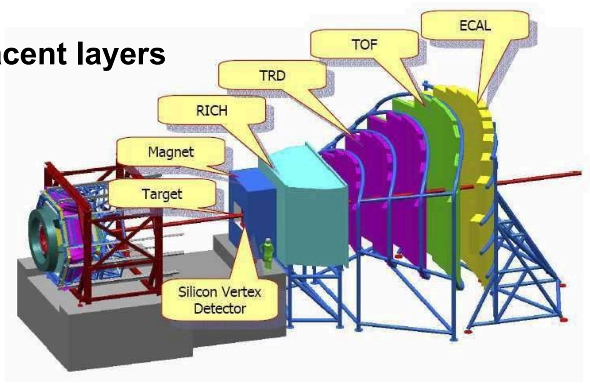
Track finding problem:
How to separate valid from ghost hits?

Track Finding

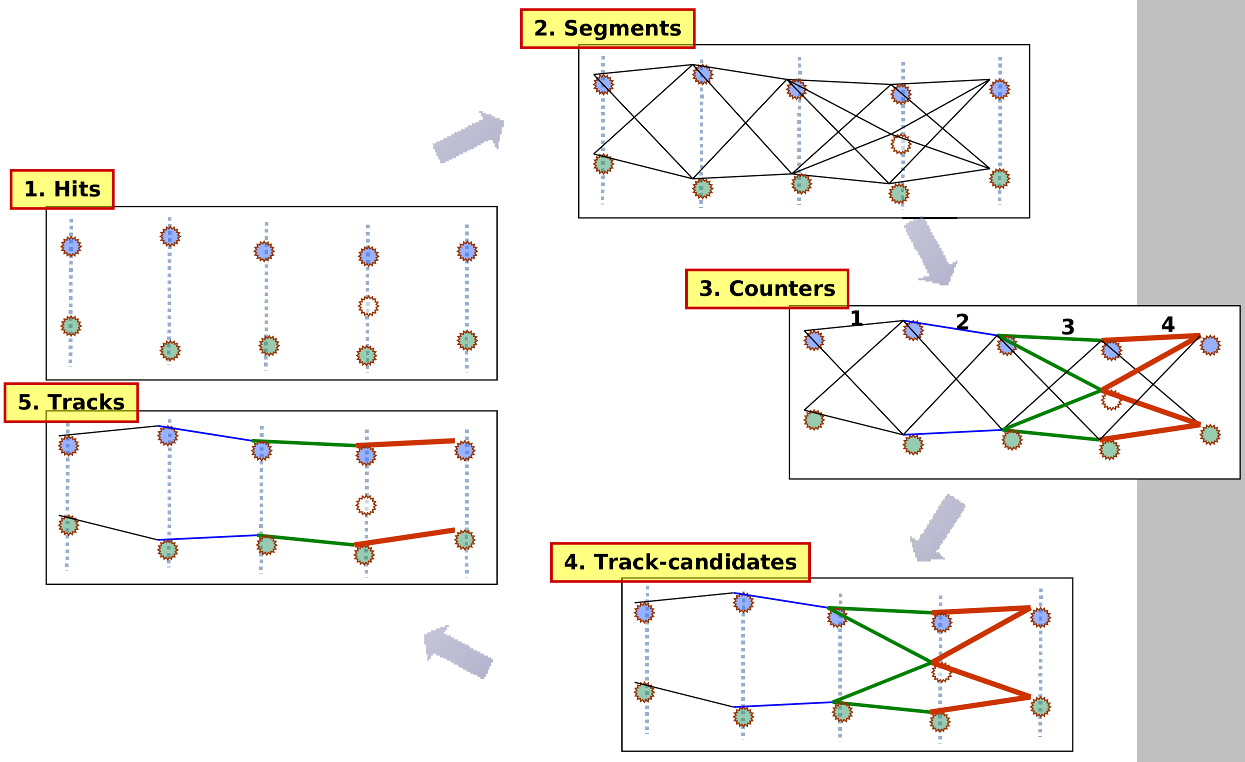




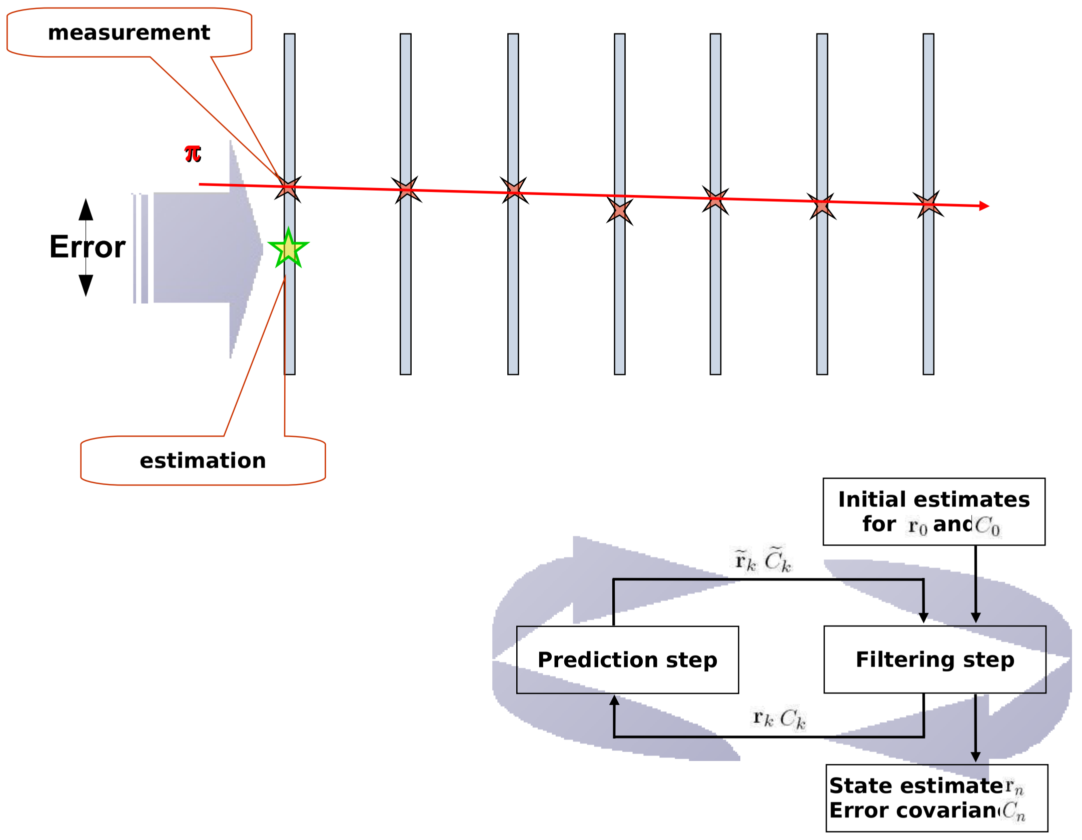
Track finding challenge:
- **Find corresponding hits on adjacent layers**
- **Time information**



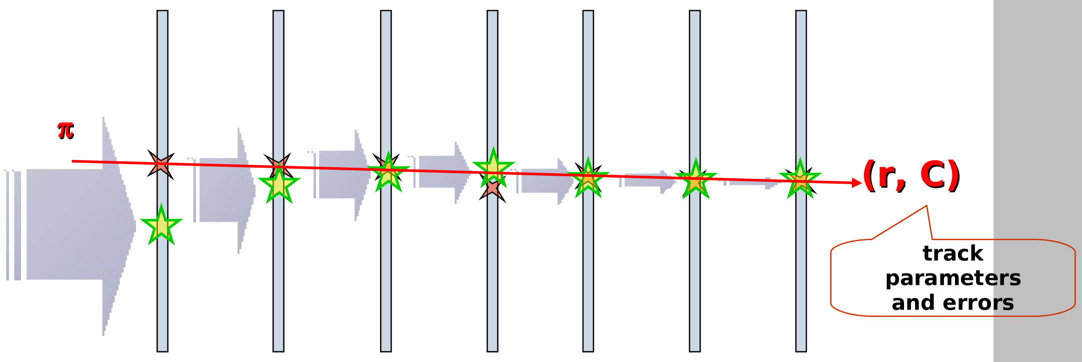
Track Finding Algorithm

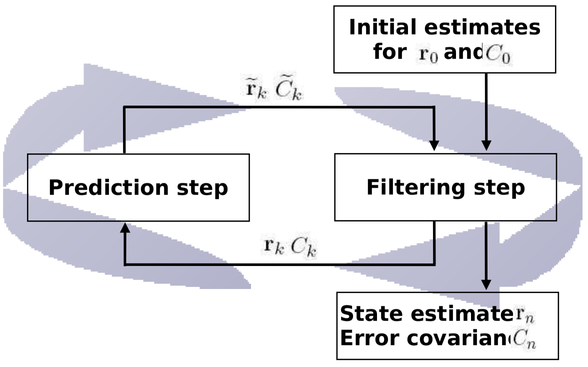


Kalman Filter for Track Fit



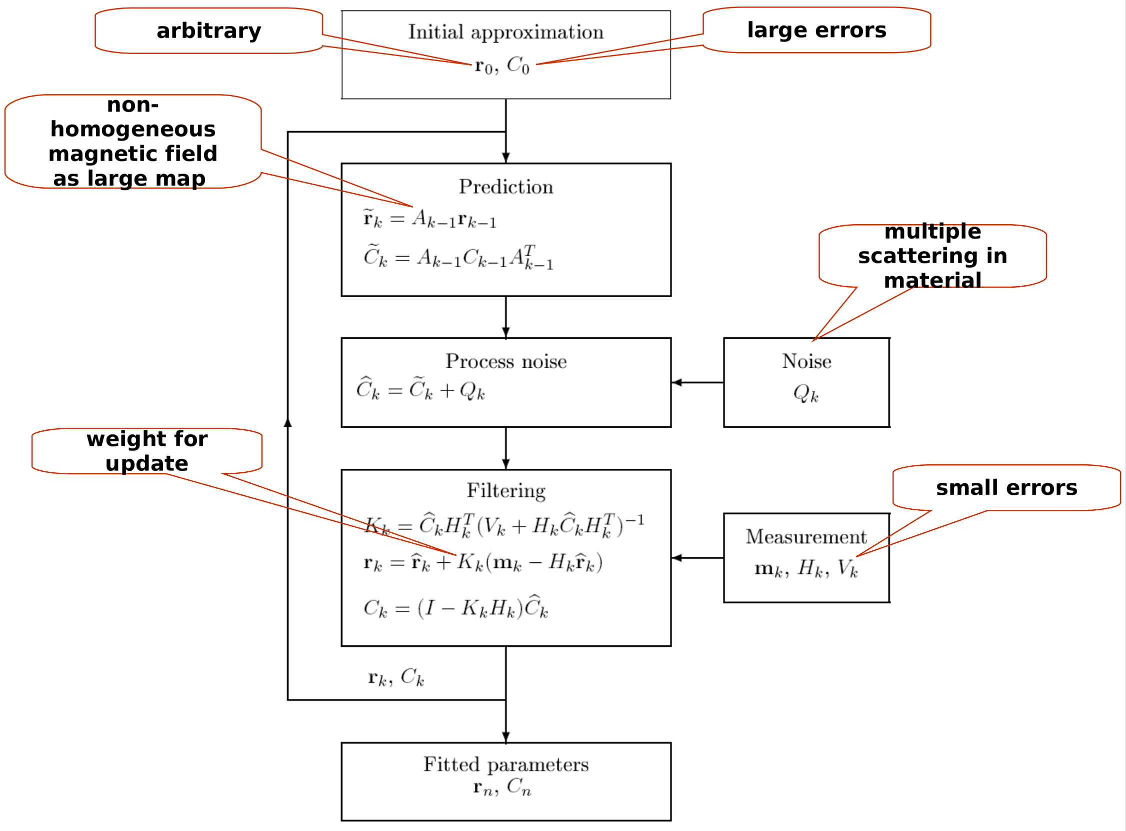
Kalman Filter for Track Fit





Initial estimates for $\mathbf{r}_0$ and C_0
$\tilde{\mathbf{r}}_k\ \tilde{C}_k$
Prediction step
Filtering step
$\mathbf{r}_k\ C_k$
State estimate $\mathbf{r}_n$
Error covariance C_n

Kalman Filter for Track Fit

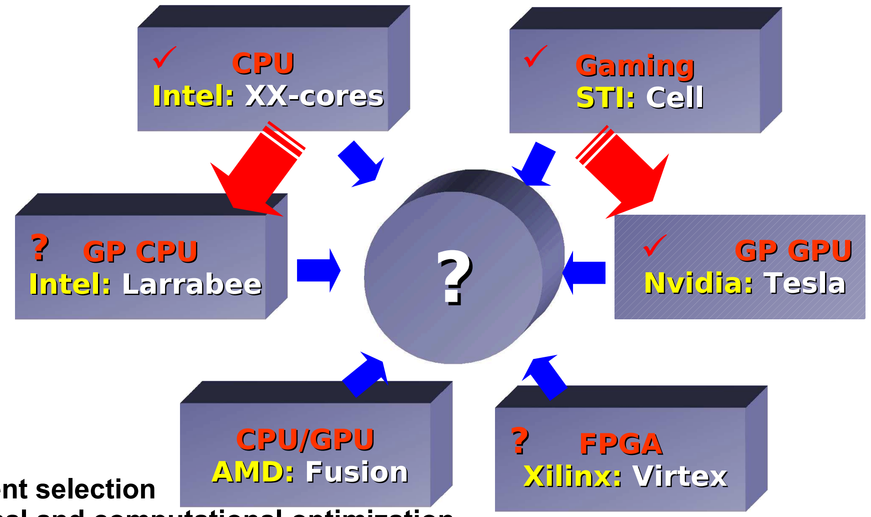


arbitrary
Initial approximation
$\mathbf{r}_0,\ C_0$
large errors
non-homogeneous magnetic field as large map
Prediction
$\tilde{\mathbf{r}}_k = A_{k-1}\mathbf{r}_{k-1}$
$\tilde{C}_k = A_{k-1}C_{k-1}A_{k-1}^T$
multiple scattering in material
Process noise
$\widehat{C}_k = \tilde{C}_k + Q_k$
Noise
Q_k
weight for update
Filtering
$K_k = \widehat{C}_k H_k^T (V_k + H_k \widehat{C}_k H_k^T)^{-1}$
$\mathbf{r}_k = \widehat{\mathbf{r}}_k + K_k(\mathbf{m}_k - H_k\widehat{\mathbf{r}}_k)$
$C_k = (I - K_k H_k)\widehat{C}_k$
small errors
Measurement
$\mathbf{m}_k,\ H_k,\ V_k$
$\mathbf{r}_k,\ C_k$
Fitted parameters
$\mathbf{r}_n,\ C_n$

Multi- and Many-Core Architectures

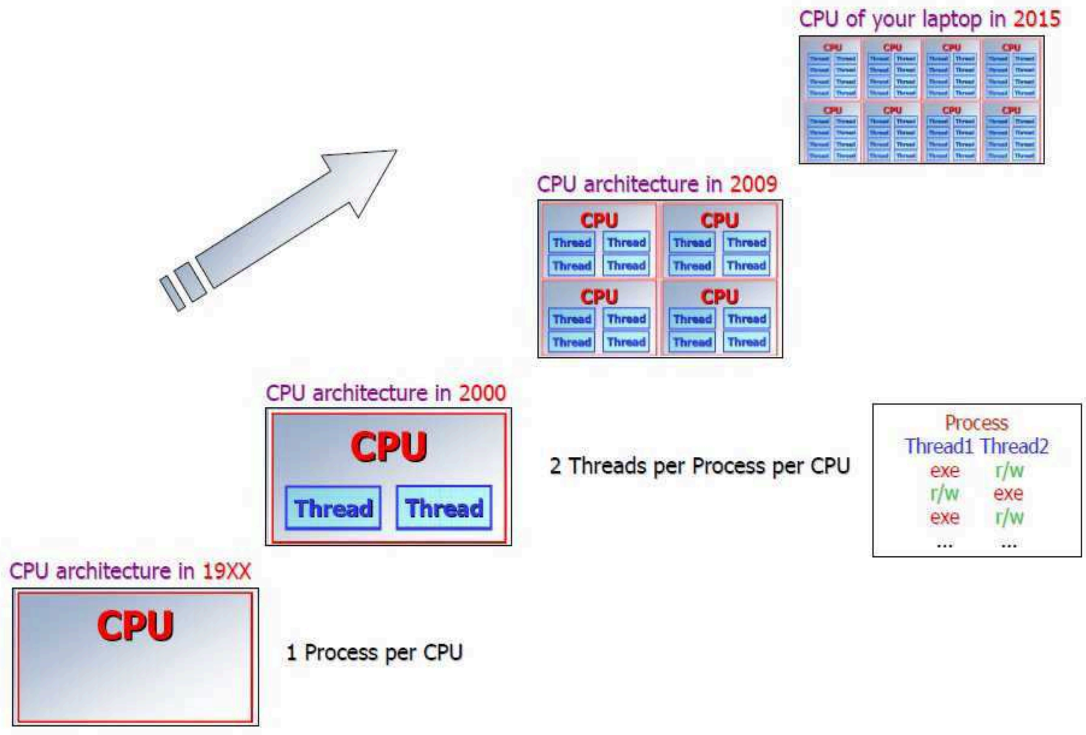
Multi- / Many-Core HPC

- High performance computing (HPC)
- Highest clock rate is reached
- Performance/power optimization
- Heterogeneous systems of many (>8) cores
- Similar programming languages (OpenCL, Ct and CUDA)
- We need a uniform approach to all CPU/GPU families



- On-line event selection
- Mathematical and computational optimization
- SIMDization of the algorithm (from scalars to vectors)
- MIMDization (multi-threads, many-cores)
- Optimize the STS geometry (strips, sector navigation)
- Smooth magnetic field

Cores and Threads

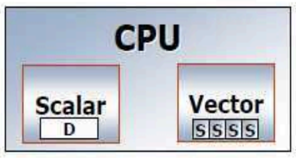


1 Process per CPU

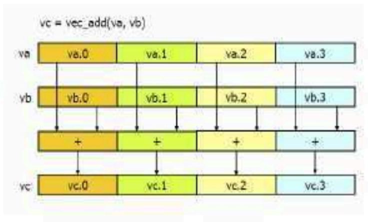
2 Threads per Process per CPU

Process	
Thread1	Thread2
exe	r/w
r/w	exe
exe	r/w
...	...

SIMD Width



SIMD = Single Instruction Multiple Data
SIMD uses vector registers
SIMD exploits data-level parallelism



Faster or Slower ?

Scalar double precision (64 bits)

D

Vector (SIMD) double precision (128 bits) — 2 or 1/2

D1	D2

Vector (SIMD) single precision (128 bits) — 4 or 1/4

S1	S2	S3	S4

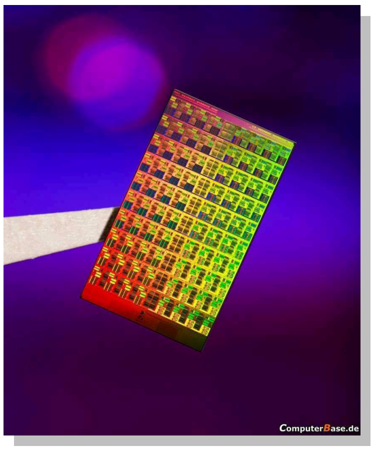
Intel AVX (2010) vector single precision (256 bits) — 8 or 1/8

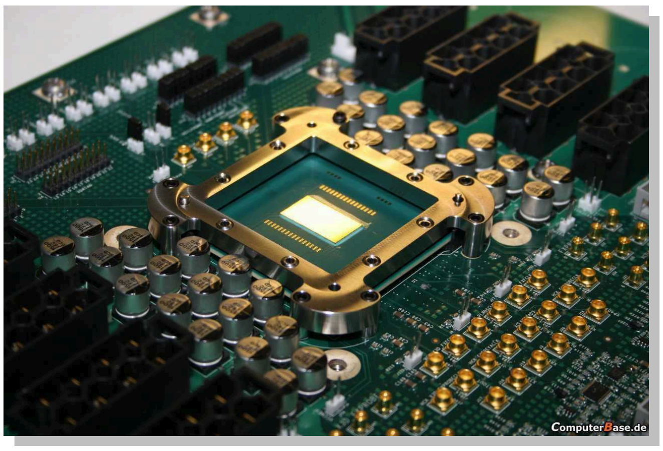
S1	S2	S3	S4	S5	S6	S7	S8

Intel LRB (2010) vector single precision (512 bits) — 16 or 1/16

S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16

Intel Polaris: 80 Cores

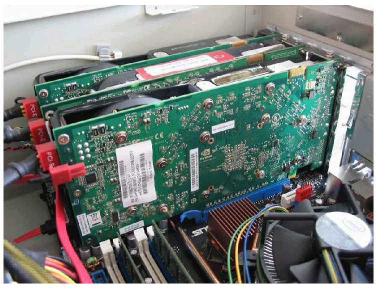




3.16 GHz, 0.95 Volt, 62 Watt -> 1.01 Teraflops

NVIDIA GeForce GTX 280

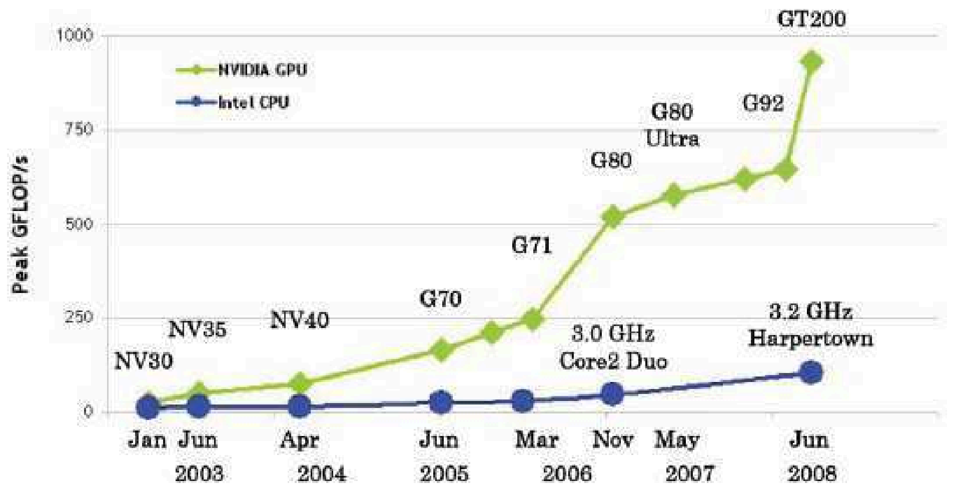
- **NVIDIA GT200**
  GeForce GTX 280 1024MB.
- **933 GFlops single precision (240 FPUs).**
- **236 Watts**
- **finally double precision support, but only ~ 90 GFlops (8 core Xeon ~80 Gflops).**
- **Currently under investigation:**
  - **·Tracking**
  - **·Linpack**
  - **·Image Processing**



CUDA (Compute Unified Device Architecture)

Sebastian Kalcher

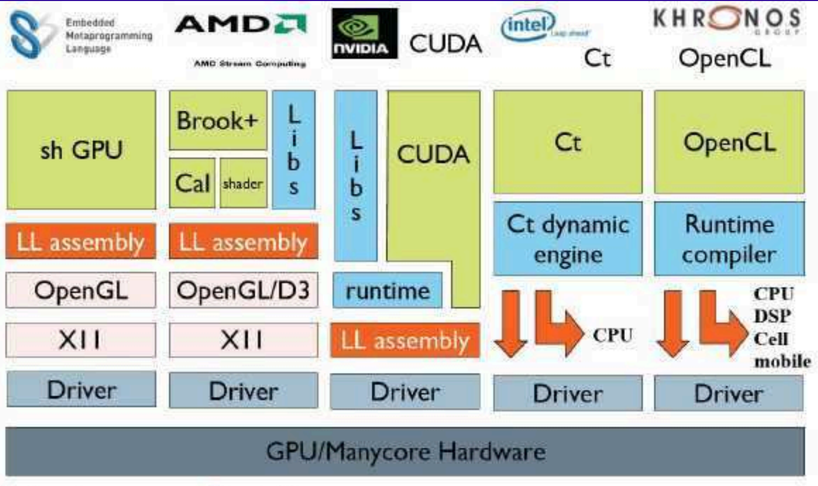
General Purpose Graphic Processing Units vs. CPU



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	

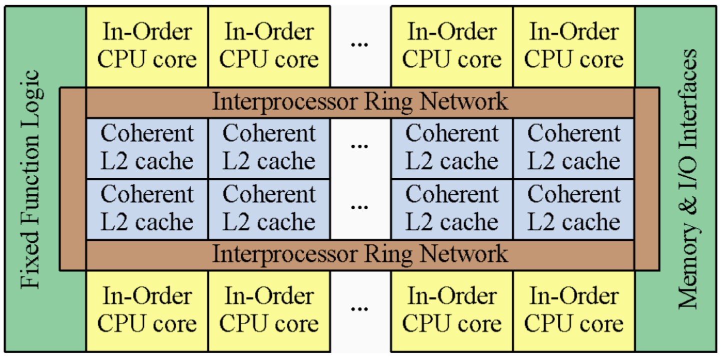
Source: NVIDIA

CPU/GPU Programming Frameworks



- Cg, OpenGL Shading Language, Direct X
  - Designed to write shaders
  - Require problem to be expressed graphically
- AMD Brook
  - Pure stream computing
  - No hardware specific
- AMD CAL (Compute Abstraction Layer)
  - Generic usage of hardware on assembler level
- NVIDIA CUDA (Compute Unified Device Architecture)
  - Defines hardware platform
  - Generic programming
  - Extension to the C language
  - Explicit memory management
  - Programming on thread level
- Intel Ct (C for throughput)
  - Extension to the C language
  - Intel CPU/GPU specific
  - SIMD exploitation for automatic parallelism
- OpenCL (Open Computing Language)
  - Open standard for generic programming
  - Extension to the C language
  - Supposed to work on any hardware
  - Usage of specific hardware capabilities by extensions

Intel Larrabee: 32 Cores



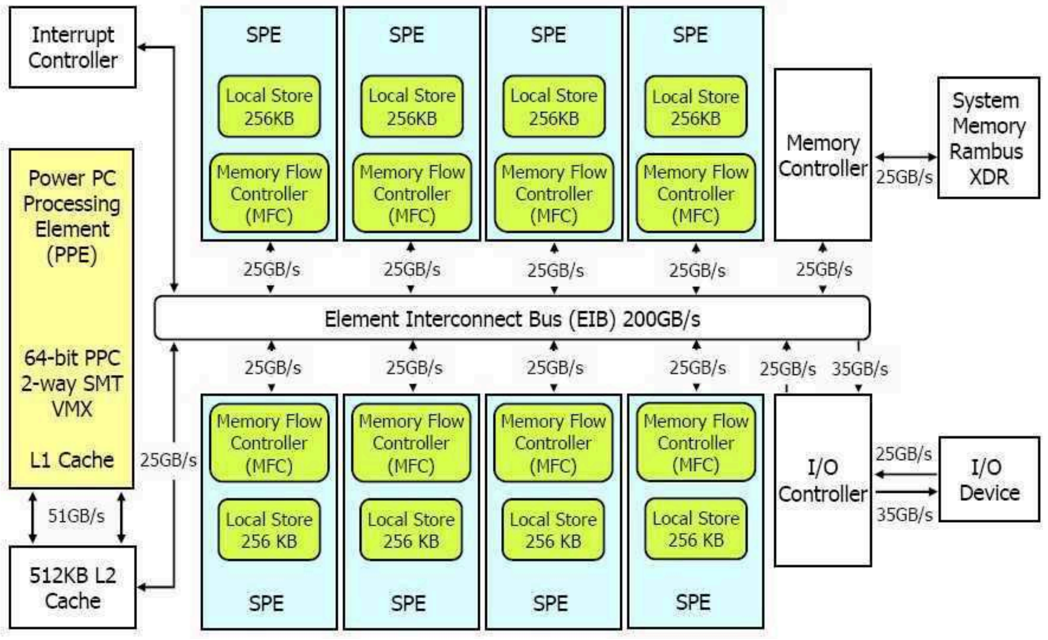
Larrabee will differ from other discrete GPUs currently on the market such as the GeForce 200 Series and the Radeon 4000 series in three major ways:

- **use the x86 instruction set with Larrabee-specific extensions;**
- **feature cache coherency across all its cores;**
- **include very little specialized graphics hardware.**

The x86 processor cores in Larrabee will be different in several ways from the cores in current Intel CPUs such as the Core 2 Duo:

- **LRB's x86 cores will be based on the much simpler Pentium design;**
- **each core contains a 512-bit vector processing unit, able to process 16 single precision floating point numbers at a time;**
- **LRB includes one fixed-function graphics hardware unit;**
- **LRB has a 1024-bit (512-bit each way) ring bus for communication between cores and to memory;**
- **LRB includes explicit cache control instructions;**
- **each core supports 4-way simultaneous multithreading, with 4 copies of each processor register.**

CELL Architecture

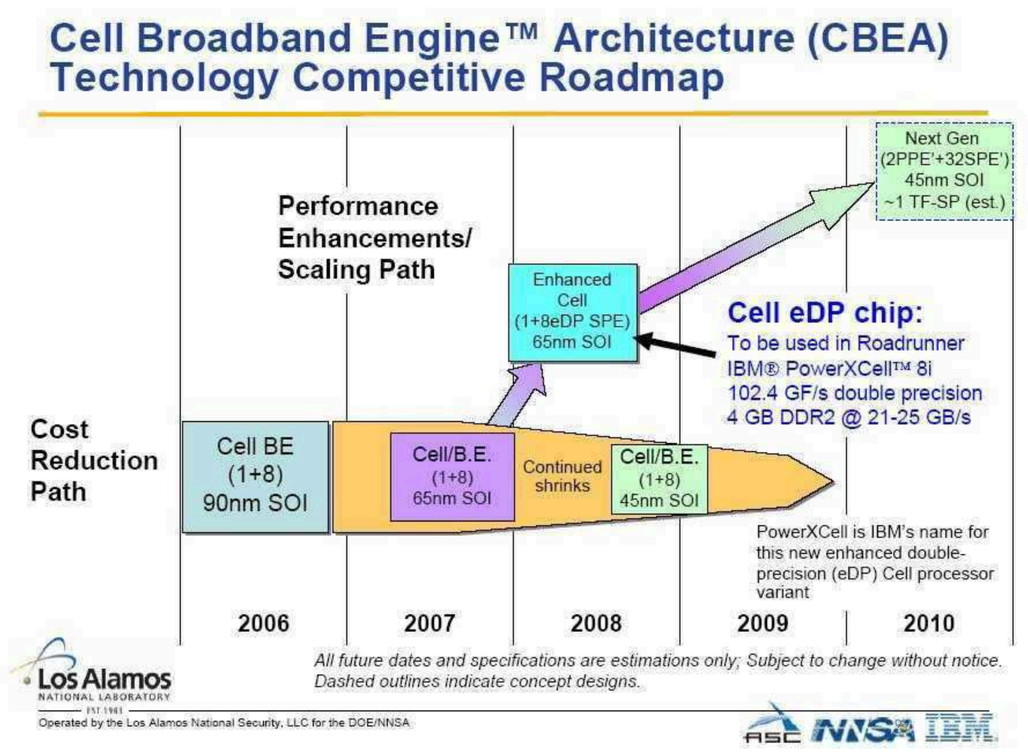


CELL Facts

- Power Processor Element
  - General Purpose 64-bit RISC (PPC AS 2.0)
  - 2-Way Hardware Multithread
  - L1: Inst 32 KB, Data 32 KB
  - L2: 512 KB
  - Coherent Load/Store
  - 3.2 GHz
- External Interconnects:
  - 25.6 GB/s memory interface
  - 2 configurable I/O interfaces
    - Coherent interface (SMP)
    - Standard I/O (I/O & graphics)
  - Total bandwidth
    - 35 GB/s out
    - 25 GB/s in
- Memory management and Mapping
  - SPE local store aliased into PPE system memory
  - MFC/MMU controls DMA access
  - Hardware or software TLB management
  - SPE DMA access protected by MMU
- Synergistic Processor Elements (SPE):
  - 8 per chip
  - 128 bit wide SIMD units
  - Integer and floating point (single precision)
  - 256 KB local store
  - Up to 25.8 GF/s per SPE
  - 200 GF/s total
- Internal interconnect
  - Coherent ring structure
  - 300+ GB/s total internal bandwidth
  - DMA control to/from SPEs supports > 100 outstanding memory requests

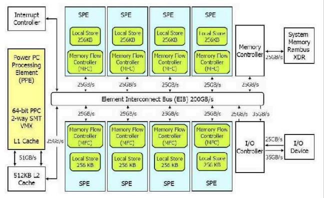
- **Sony PlayStation-3 -> cheap**
- **32 (8x4) times faster !**

Enhanced Cell with Double Precision


Cell Broadband Engine™ Architecture (CBEA)
Technology Competitive Roadmap
Performance Enhancements/ Scaling Path
Cost Reduction Path
Cell BE (1+8) 90nm SOI
Cell/B.E. (1+8) 65nm SOI
Continued shrinks
Cell/B.E. (1+8) 45nm SOI
Enhanced Cell (1+8eDP SPE) 65nm SOI
Next Gen (2PPE'+32SPE') 45nm SOI ~1 TF-SP (est.)
Cell eDP chip:
To be used in Roadrunner
IBM® PowerXCell™ 8i
102.4 GF/s double precision
4 GB DDR2 @ 21-25 GB/s
PowerXCell is IBM's name for this new enhanced double-precision (eDP) Cell processor variant
2006
2007
2008
2009
2010
All future dates and specifications are estimations only; Subject to change without notice.
Dashed outlines indicate concept designs.
Los Alamos NATIONAL LABORATORY EST.1943
Operated by the Los Alamos National Security, LLC for the DOE/NNSA
ASC NNSA IBM

- **128 (32x4) times faster !**
- **future: 512 (32x16) ?**

Porting Algorithms to CELL



Approach:

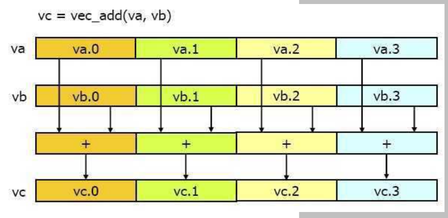
- **Universality (any multi-core architecture)**
- **Vectorization (SIMDization)**
- **Run SPEs independently (one collision per SPE)**

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

Data Types:

- **Scalar double**
- **Scalar float**
- **Pseudo-vector (array)**
- **Vector (4 float)**

c = a + b



Track
|Tr|Tr|Tr|Tr|

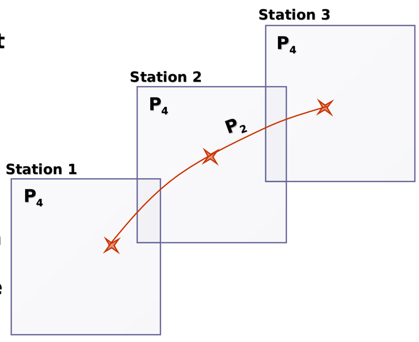
Platform:

1. **Linux** — **SSE2**
2. **Virtual machine:** — **SSE2**
   - ✓ Red Hat (Fedora Core 4)
   - ✓ Cell Simulator:
     - ❖ PPE — **AltiVec**
     - ❖ SPE
3. **Cell Blade** — **Specialized SIMD**

Approximation of Magnetic Field

Problem:

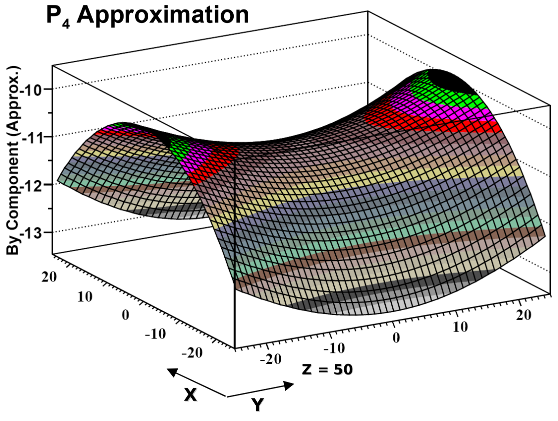
- **Full reconstruction must work within 256 kB of the Local Store.**
- **The magnetic field map is too large for that (70 MB).**
- **A position (x,y), to which the track is propagated, is unknown in advance.**
- **Therefore, access to the magnetic field map is a blocking process.**



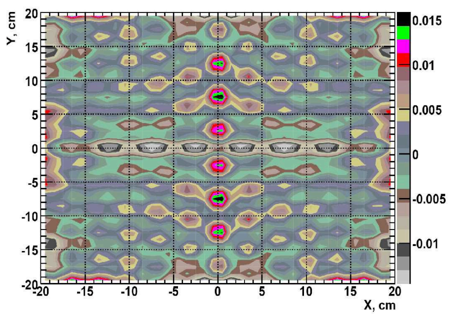
Solution:

1. **Use a polynomial approximation (4-th order) of the field in XY planes of the stations.**
2. **Assuming a parabolic behavior of the field between stations calculate the magnetic field along the track based on 3 consecutive measurements.**

P_4 Approximation

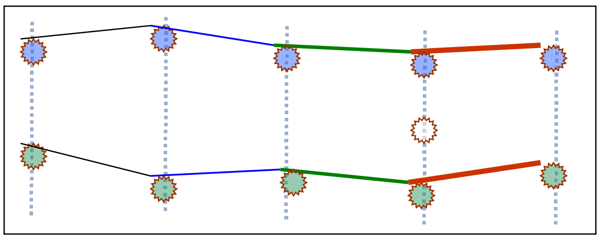


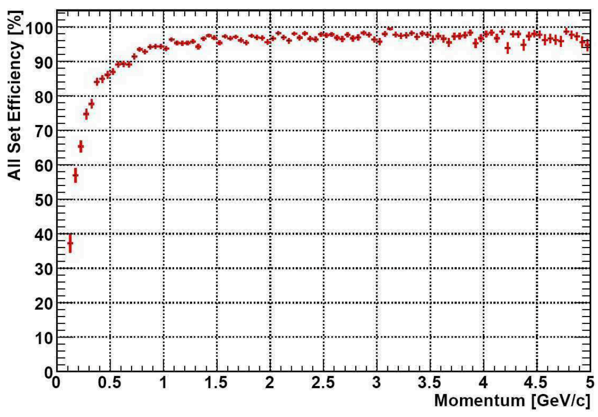
Difference

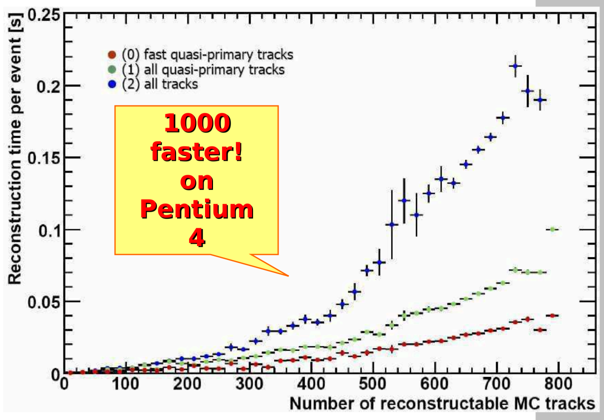


Results

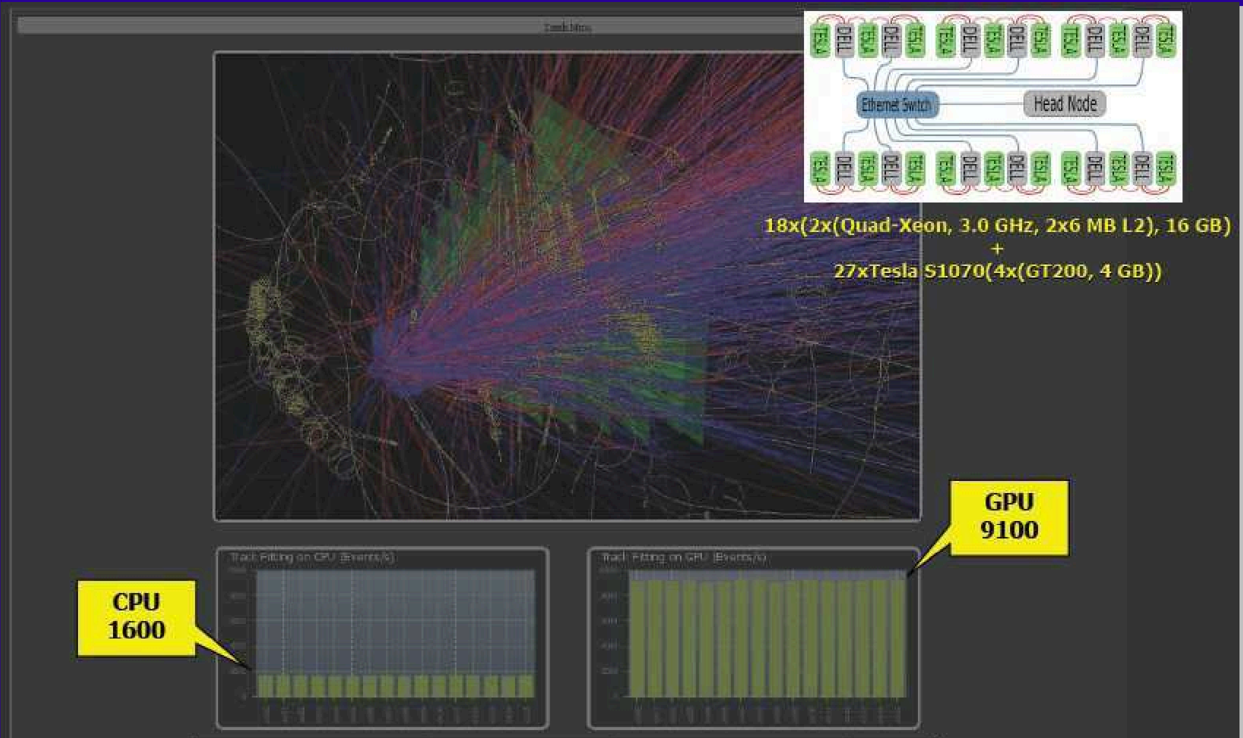
Track Finder: Single Core CPU







CPU vs. GPU Performance


Ethernet Switch
Head Node
TESLA
DELL
18x(2x(Quad-Xeon, 3.0 GHz, 2x6 MB L2), 16 GB)
+
27xTesla S1070(4x(GT200, 4 GB))
GPU
9100
CPU
1600
Track Fitting on CPU (Events/s)
Track Fitting on GPU (Events/s)

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

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

	Stage	Description	Time/track	Speedup
Intel P4		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

7800 faster!

- **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.**

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Computer Physics Communications 178 (2008) 374–383

Computer Physics Communications

www.elsevier.com/locate/cpc

Fast SIMDized Kalman filter based track fit

S. Gorbunov [a,b], U. Kebschull [b], I. Kisel [b,c,*], V. Lindenstruth [b], W.F.J. Müller [a]

[a] *Gesellschaft für Schwerionenforschung mbH, 64291 Darmstadt, Germany*
[b] *Kirchhoff Institute for Physics, University of Heidelberg, 69120 Heidelberg, Germany*
[c] *Laboratory of Information Technologies, Joint Institute for Nuclear Research, 141980 Dubna, Russia*

Received 17 February 2007; received in revised form 29 August 2007; accepted 2 October 2007

Available online 7 October 2007

Abstract

Modern high energy physics experiments have to process terabytes of input data produced in particle collisions. The core of many data reconstruction algorithms in high energy physics is the Kalman filter. Therefore, the speed of Kalman filter based algorithms is of crucial importance in on-line data processing. This is especially true for the combinatorial track finding stage where the Kalman filter based track fit is used very intensively. Therefore, developing fast reconstruction algorithms, which use maximum available power of processors, is important, in particular for the initial selection of events which carry signals of interesting physics.

One of such powerful feature supported by almost all up-to-date PC processors is a SIMD instruction set, which allows packing several data items in one register and to operate on all of them, thus achieving more operations per clock cycle. The novel Cell processor extends the parallelization further by combining a general-purpose PowerPC processor core with eight streamlined coprocessing elements which greatly accelerate vector processing applications.

In the investigation described here, after a significant memory optimization and a comprehensive numerical analysis, the Kalman filter based track fitting algorithm of the CBM experiment has been vectorized using inline operator overloading. Thus the algorithm continues to be flexible with respect to any CPU family used for data reconstruction.

Because of all these changes the SIMDized Kalman filter based track fitting algorithm takes 1 μs per track that is 10000 times faster than the initial version. Porting the algorithm to a Cell Blade computer gives another factor of 10 of the speedup.

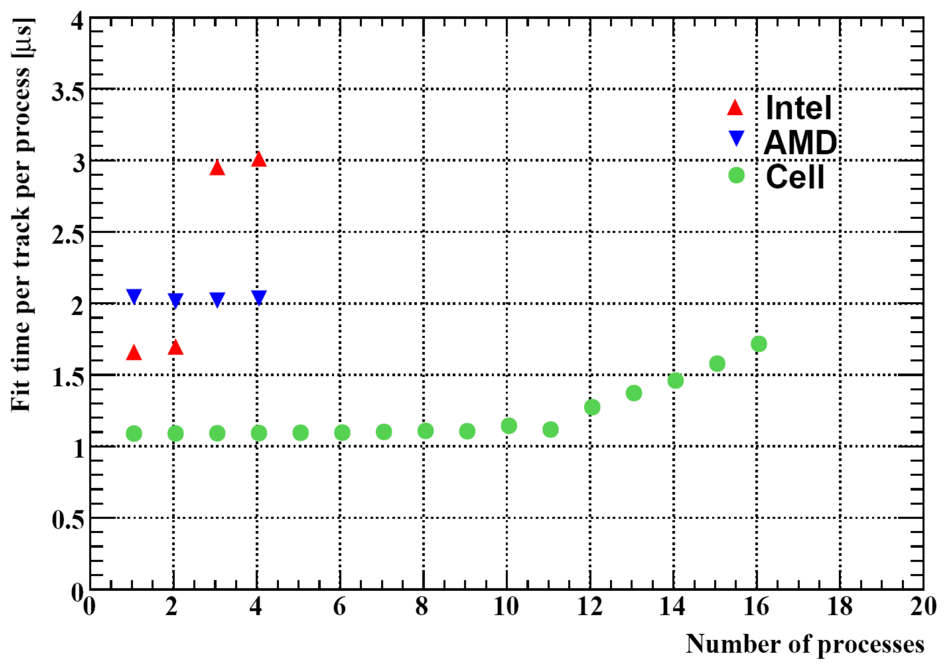
Finally, we compare performance of the tracking algorithm running on three different CPU architectures: Intel Xeon, AMD Opteron and Cell Broadband Engine.

PACS: 02.60.Pn; 02.70.-c; 07.05.-t; 07.05.Bx; 07.05.Kf

Keywords: High energy physics; CBM experiment; Data reconstruction; Track fit; Kalman filter; SIMD instruction set; Cell Broadband Engine

http://openlab-mu-internal.web.cern.ch/openlab-mu-internal/06_openlab-II/Platform_Competence_Centre/Optimization/Benchmarking/Benchmarks_print.htm

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



lxg1411@GSI
eh102@KIP
blade11bc4 @IBM

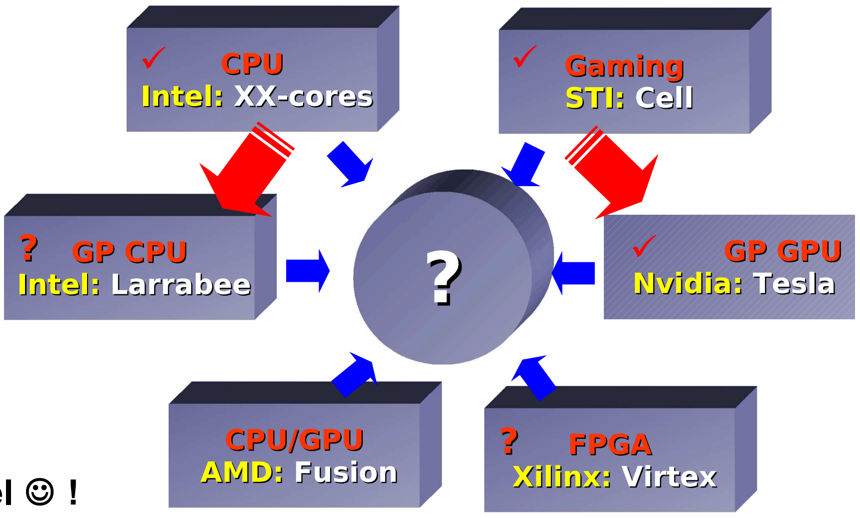
Kalman Filter for NVIDIA with CUDA

- Compute intensive single precision algorithm
- Algorithmic code is common with I. Kisel's SSE and Cell implementation using only different operator overloading and setup coding.
- Performance:
  - Core CPU (SSE): ~.735 us / Track / Core
  - Nvidia GTX 280: ~.305 us / Track including PCIe Transfer
    ~.072 us / Track if data already on device
- Utilised Bandwidth:
  - ~2.53 GB/s over PCIe (Gen1) which is ~ 79% of one way peak
  - ~24 GB/s on Device
- Speed is still memory bound
- Used Optimizations: pinned host memory to allow DMA
  (~ 50% boost), memory access coalescing using explicit on chip cache
  (~50% boost for device code), used bools instead of bitmasks to ease optimization for compiler

Some Lessions We Learned

- Optimization of Algorithms has a huge potential
  - A factor of 100 – 10,000 can be achieved on a single core machine only by rewriting and optimizing the algorithm
- Don't relay on caches
  - Main memory accesses are incredibly slow
- New architectures need to address fast read and write operations
  - Remember Amdahl's law

Conclusion



- Think parallel ☺ !
- Parallelizable algorithms
- Use SIMD units in the nearest future (many-cores, TF/s, …)
- Use single-precision floating point if possible
- In critical parts use double precision if necessary
- Avoid accessing main memory, no maps, no look-up-tables
- New parallel languages appear: OpenCL, Ct, CUDA, …
- Keep portability of the code (Intel, AMD, Cell, GPGPU, …)
- Try the auto-vectorization option of the compilers