Porting Reconstruction Algorithms to the Cell Broadband Engine

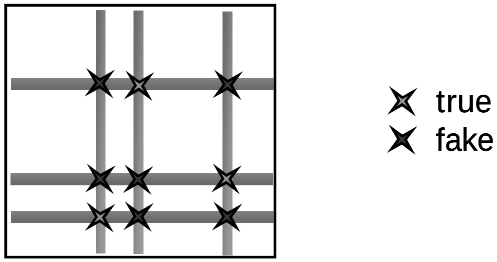
S. Gorbunov[1], U. Kebschull[1], I. Kisel[1,2], V. Lindenstruth[1], W.F.J. Müller[2]

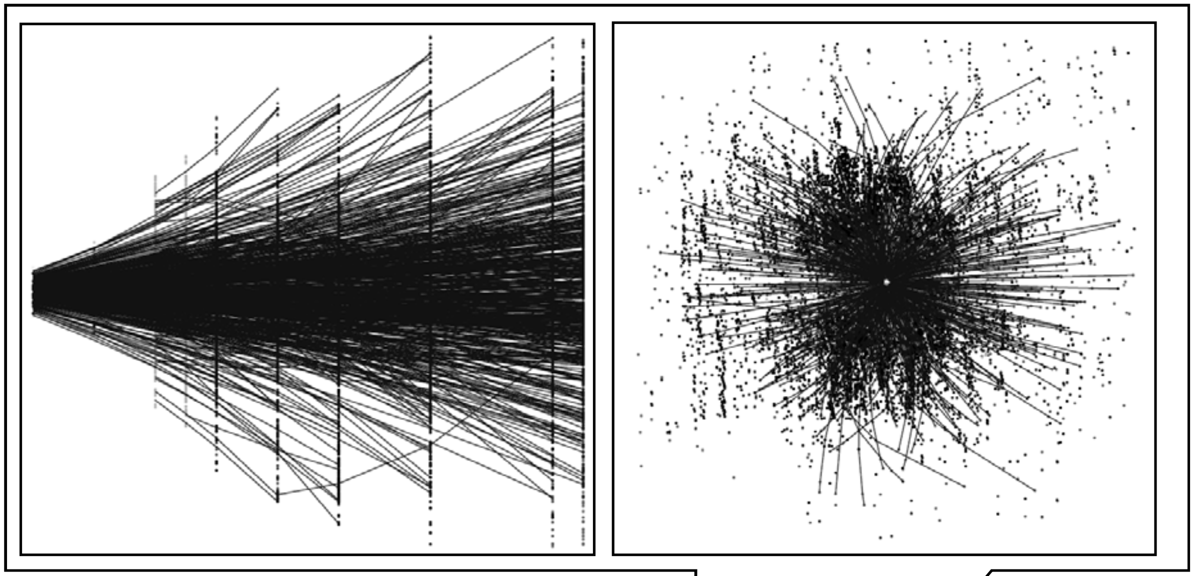
[1]Kirchhoff Institute for Physics, University of Heidelberg, Germany
[2]Gesellschaft für Schwerionenforschung mbH, Darmstadt, Germany

ACAT08, Erice, Sicily
November 3-7, 2008

CBM Experiment (FAIR/GSI): Tracking Challenge

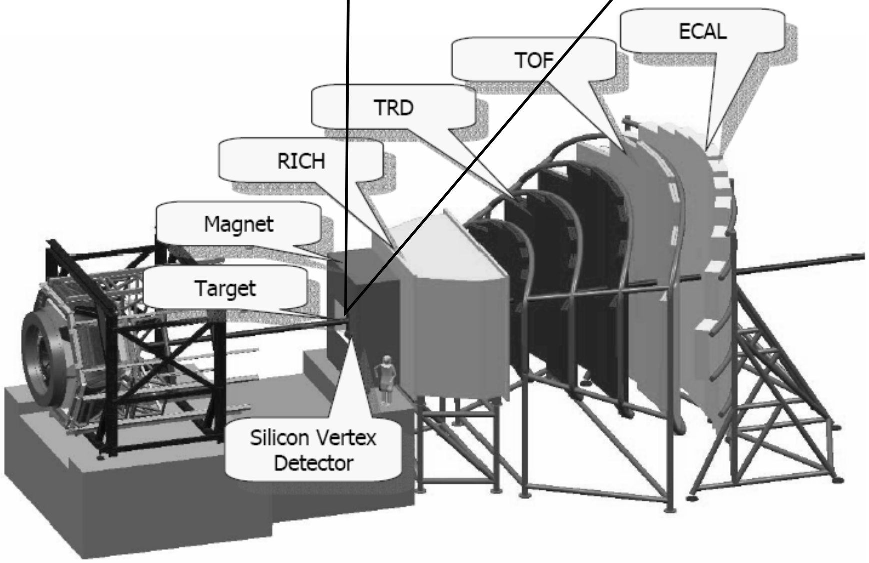
- future heavy ion experiment
- ~ 1000 charged particles/collision
- 10^7 Au+Au collisions/sec
- inhomogeneous magnetic field
- double-sided strip detectors





Chamber		1	2	3	4	5	6	7	8
	true	126	133	169	179	175	185	165	153
Min. bias	fake	4	4	0	0	650	606	395	257
	total	130	137	169	179	825	791	560	410
	true	588	615	775	821	803	805	760	696
Central	fake	18	18	2	2	5095	4158	3014	1953
	total	606	636	777	823	5898	4963	3774	2649

Processing time per event increased from 1 sec to 5 min
for double-sided strip detectors with 85% of fake space points !



Modern (Pentium, AMD, PowerPC, ...) CPUs have vector units operating 2 d.p. or 4 s.p. scalars in one go !
SIMD = Single Instruction Multiple Data

Cell Processor: Supercomputer-on-a-Chip

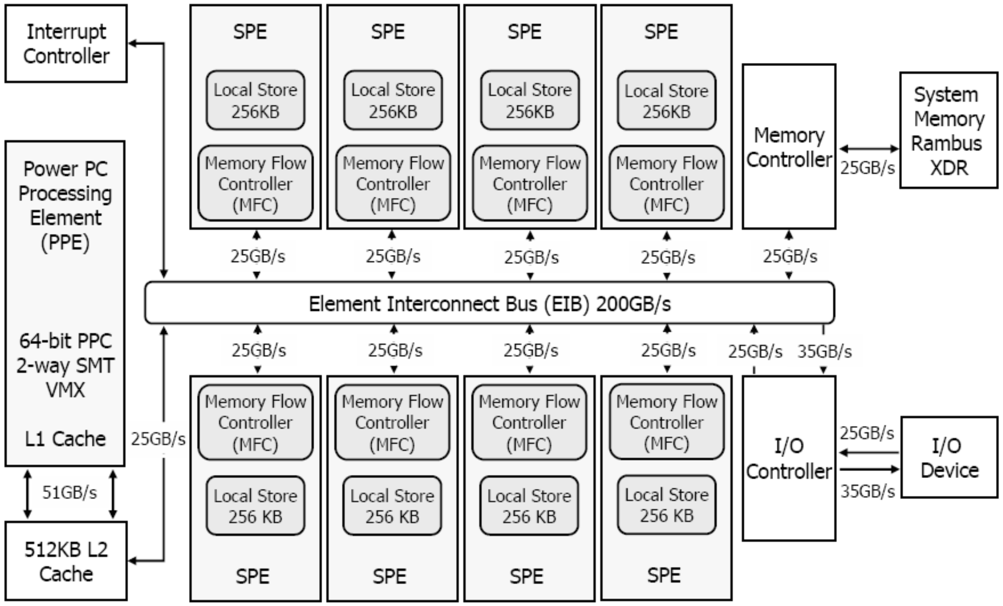
Power Processor Element (PPE):

- •General Purpose, 64-bit RISC Processor (PowerPC AS 2.0)
- •2-Way Hardware Multithreaded
- •L1 : 32KB I ; 32KB D
- •L2 : 512KB
- •Coherent load/store
- •VMX
- •3.2 GHz

Synergistic Processor Elements (SPE):

- •8 per chip
- •128-bit wide SIMD Units
- •Integer *and* Floating Point capable
- •256KB Local Store
- •Up to 25.6 GF/s per SPE --- 200GF/s total *

** At clock speed of 3.2GHz*



Internal Interconnect:

- •Coherent ring structure
- •300+ GB/s total internal interconnect bandwidth
- •DMA control to/from SPEs supports >100 outstanding memory requests

External Interconnects:

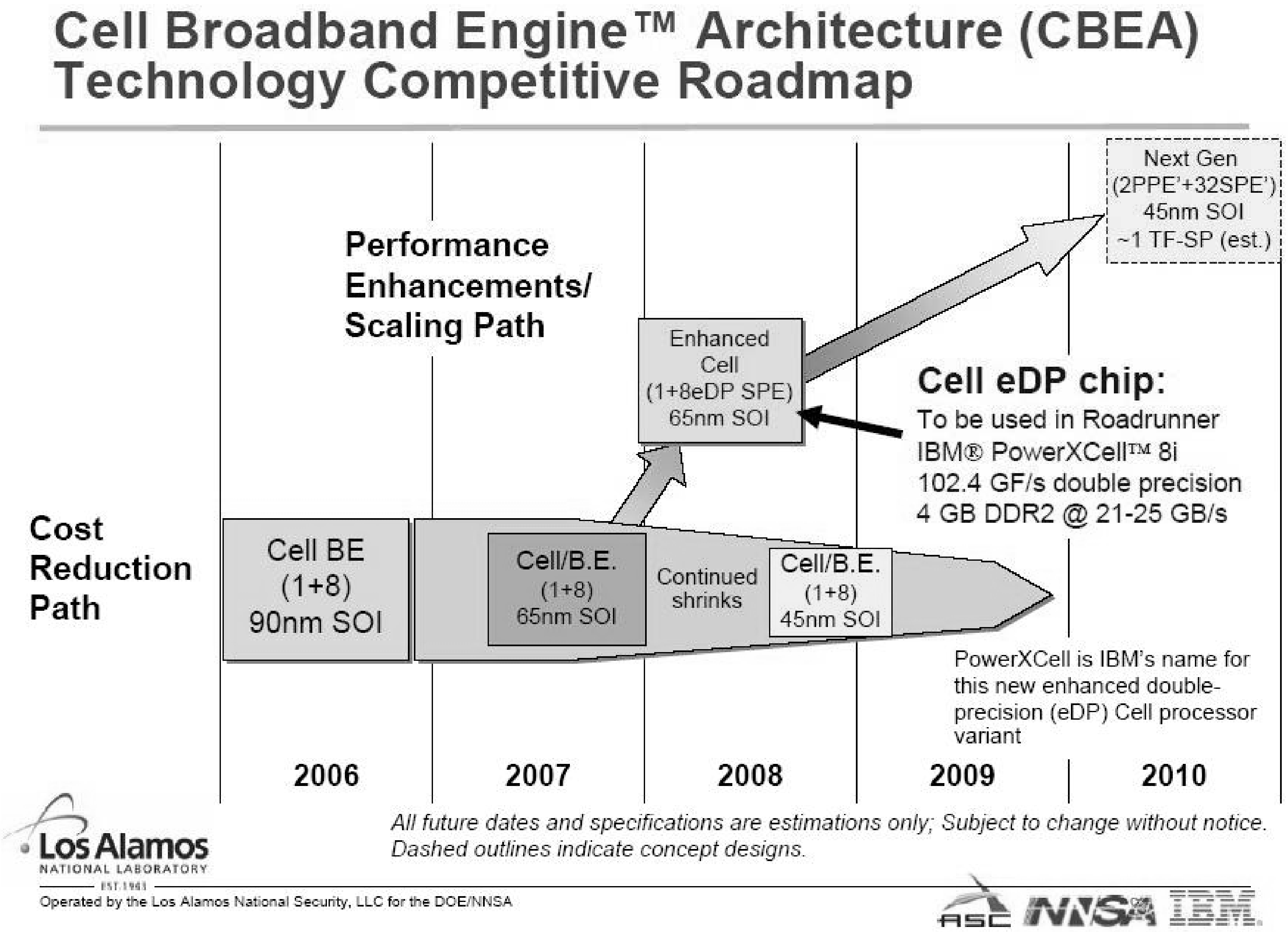
- •25.6 GB/sec BW memory interface
- •2 Configurable I/O Interfaces
  - •Coherent interface (SMP)
  - •Normal I/O interface (I/O & Graphics)
  - •Total BW configurable between interfaces
  - •Up to 35 GB/s out
  - •Up to 25 GB/s in

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

Memory Management & Mapping

- •SPE Local Store aliased into PPE system memory
- •MFC/MMU controls SPE DMA accesses
  - •Compatible with PowerPC Virtual Memory architecture
  - •S/W controllable from PPE MMIO
- •Hardware or Software TLB management
- •SPE DMA access protected by MFC/MMU

Enhanced Cell with Double Precision



Cell Broadband Engine™ Architecture (CBEA) Technology Competitive Roadmap

Performance Enhancements/ Scaling Path

Cost Reduction Path

Next Gen (2PPE'+32SPE') 45nm SOI ~1 TF-SP (est.)

Enhanced Cell (1+8eDP SPE) 65nm SOI

Cell BE (1+8) 90nm SOI

Cell/B.E. (1+8) 65nm SOI

Continued shrinks

Cell/B.E. (1+8) 45nm SOI

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

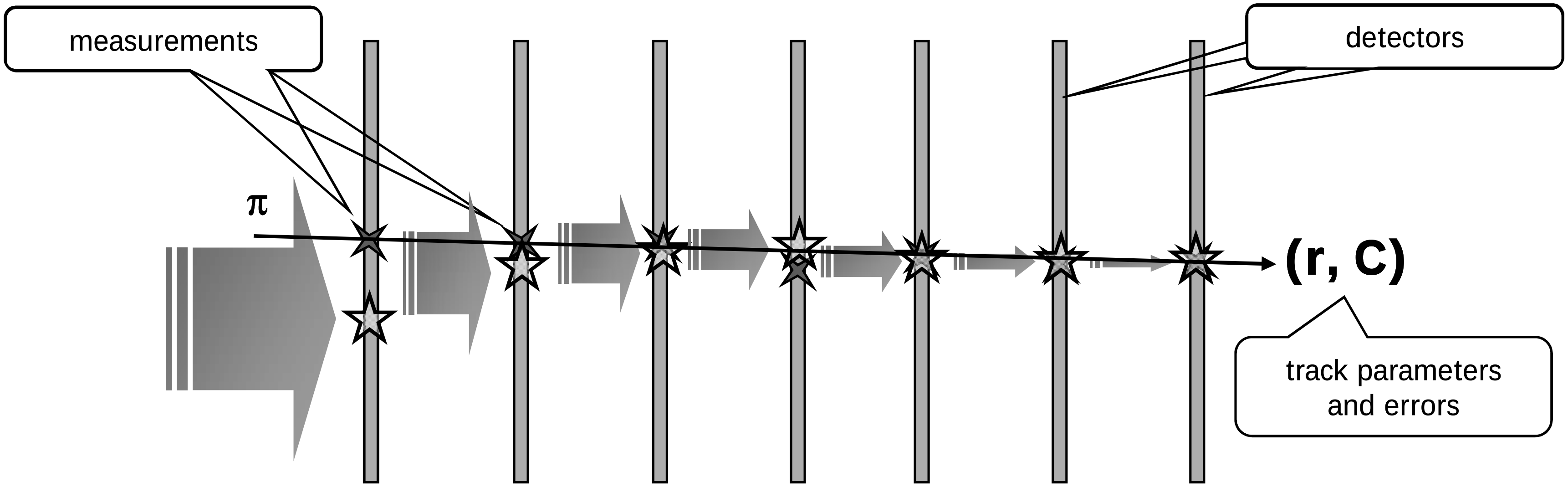
All future dates and specifications are estimations only; Subject to change without notice. Dashed outlines indicate concept designs.

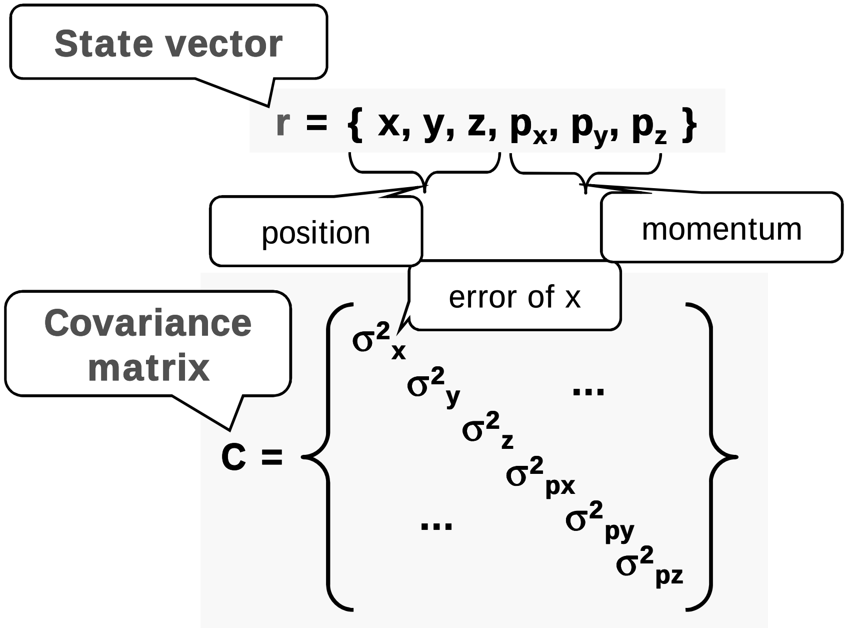
Los Alamos NATIONAL LABORATORY EST.1943

Operated by the Los Alamos National Security, LLC for the DOE/NNSA

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

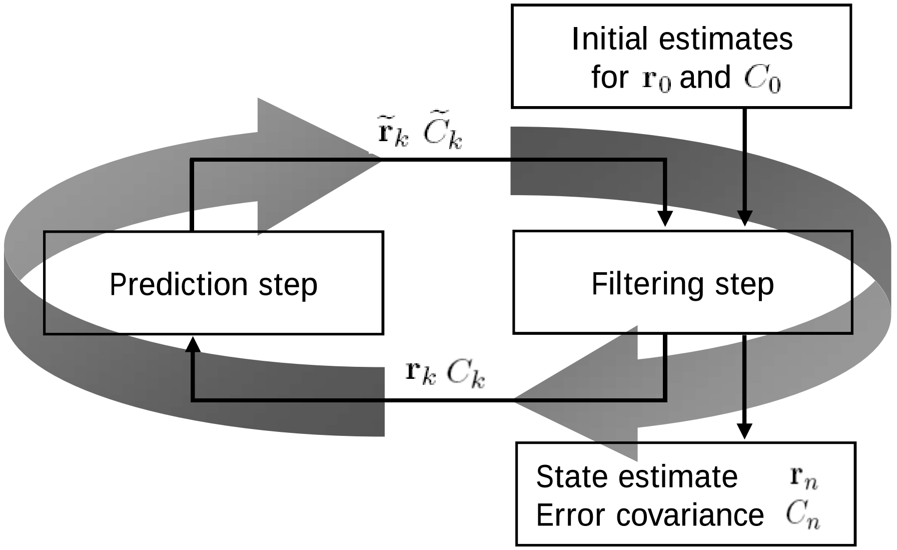
Core of Reconstruction: Kalman Filter based Track Fit



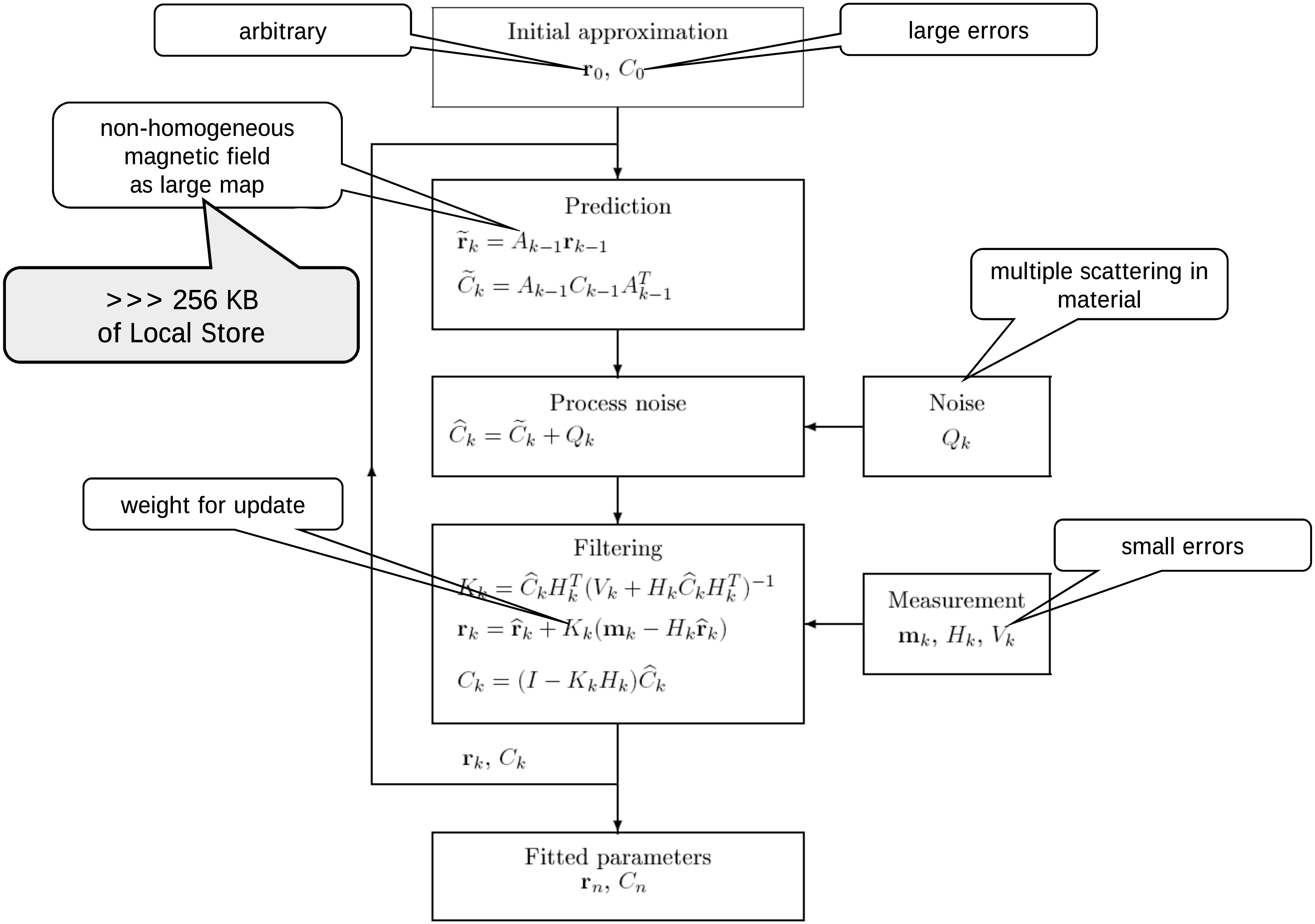


$$\mathbf{r} = \{\, \underbrace{x, y, z}_{\text{position}},\ \underbrace{p_x, p_y, p_z}_{\text{momentum}} \,\}$$

$$C = \left\{ \begin{matrix} \sigma^2_x & & & & & \\ & \sigma^2_y & & \dots & & \\ & & \sigma^2_z & & & \\ & & & \sigma^2_{px} & & \\ & \dots & & & \sigma^2_{py} & \\ & & & & & \sigma^2_{pz} \end{matrix} \right\}$$


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 1/2



arbitrary

Initial approximation

$\mathbf{r}_0,\ C_0$

large errors

non-homogeneous magnetic field as large map

>>> 256 KB of Local Store

Prediction

$\tilde{\mathbf{r}}_k = A_{k-1}\mathbf{r}_{k-1}$

$\tilde{C}_k = A_{k-1}C_{k-1}A^T_{k-1}$

multiple scattering in material

Process noise

$\hat{C}_k = \tilde{C}_k + Q_k$

Noise

Q_k

weight for update

Filtering

$K_k = \hat{C}_k H_k^T (V_k + H_k \hat{C}_k H_k^T)^{-1}$

$\mathbf{r}_k = \hat{\mathbf{r}}_k + K_k(\mathbf{m}_k - H_k\hat{\mathbf{r}}_k)$

$C_k = (I - K_k H_k)\hat{C}_k$

small errors

Measurement

$\mathbf{m}_k,\ H_k,\ V_k$

$\mathbf{r}_k,\ C_k$

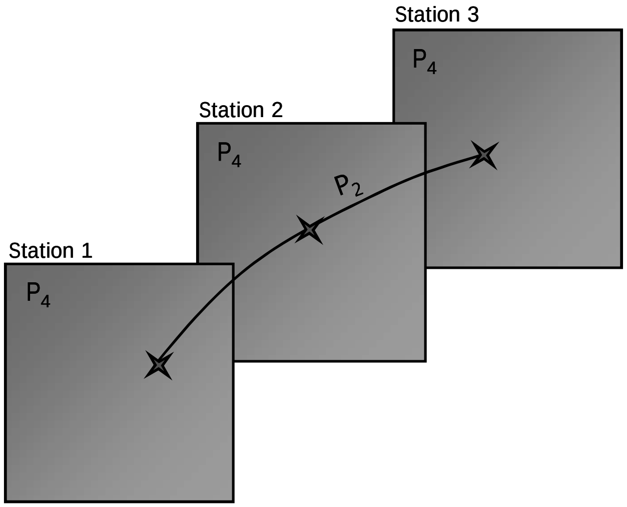
Fitted parameters

$\mathbf{r}_n,\ C_n$

“Local” Approximation of the Magnetic Field

Problem:

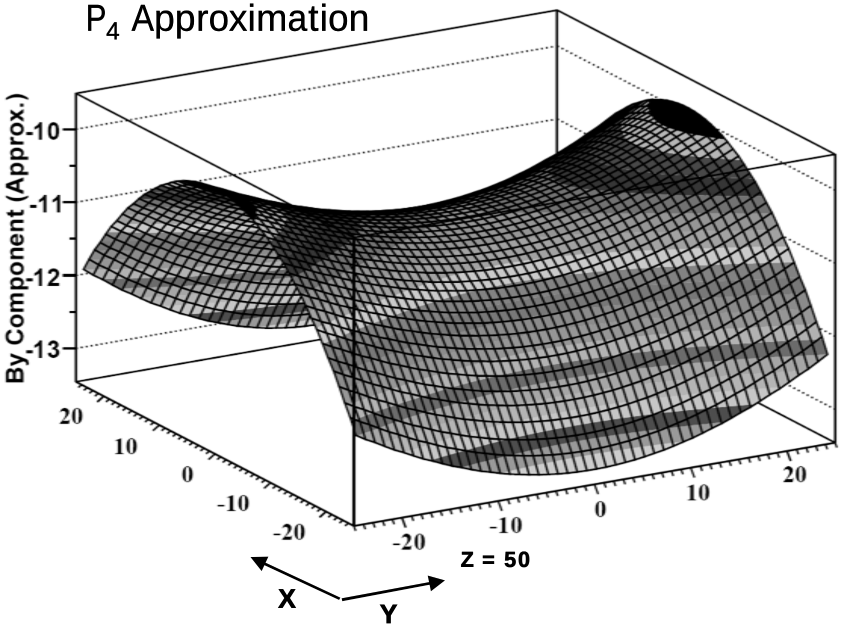
- Full reconstruction must work within 256 kB of the Local Store.
- The magnetic field map is too large for that (70 MB).
- A position (x,y), to which the track is propagated, is unknown in advance.
- Therefore, access to the magnetic field map is a blocking process.



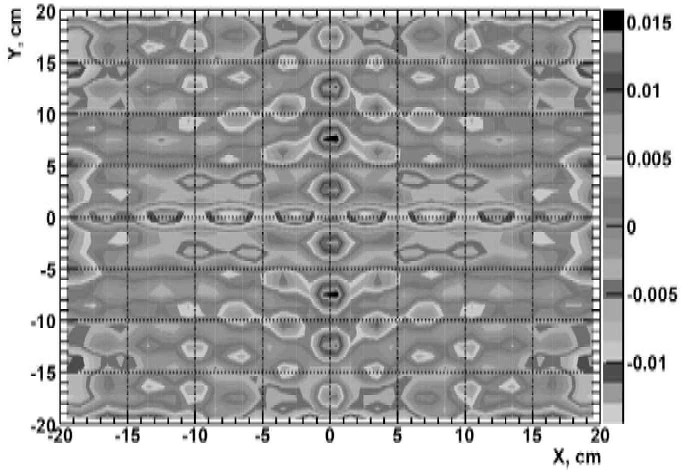
Solution:

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

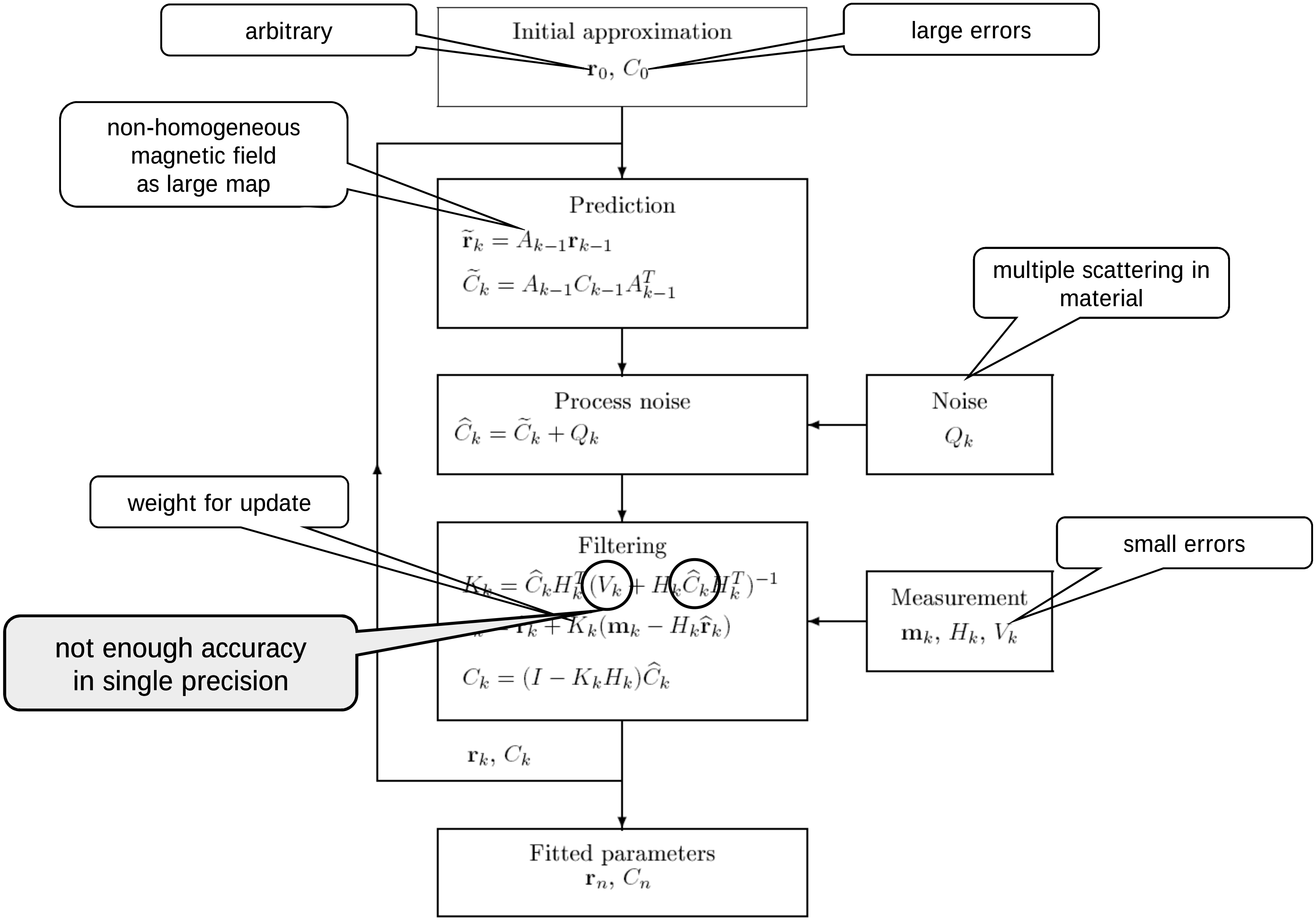


Difference



Kalman Filter for Track Fit

2/2



arbitrary

Initial approximation

$\mathbf{r}_0,\ C_0$

large errors

non-homogeneous magnetic field as large map

Prediction

$\widetilde{\mathbf{r}}_k = A_{k-1}\mathbf{r}_{k-1}$

$\widetilde{C}_k = A_{k-1}C_{k-1}A^T_{k-1}$

multiple scattering in material

Process noise

$\widehat{C}_k = \widetilde{C}_k + Q_k$

Noise

Q_k

weight for update

Filtering

$K_k = \widehat{C}_k H^T_k (V_k + H_k \widehat{C}_k H^T_k)^{-1}$

$\mathbf{r}_k = \widetilde{\mathbf{r}}_k + K_k(\mathbf{m}_k - H_k \widehat{\mathbf{r}}_k)$

$C_k = (I - K_k H_k)\widehat{C}_k$

not enough accuracy in single precision

small errors

Measurement

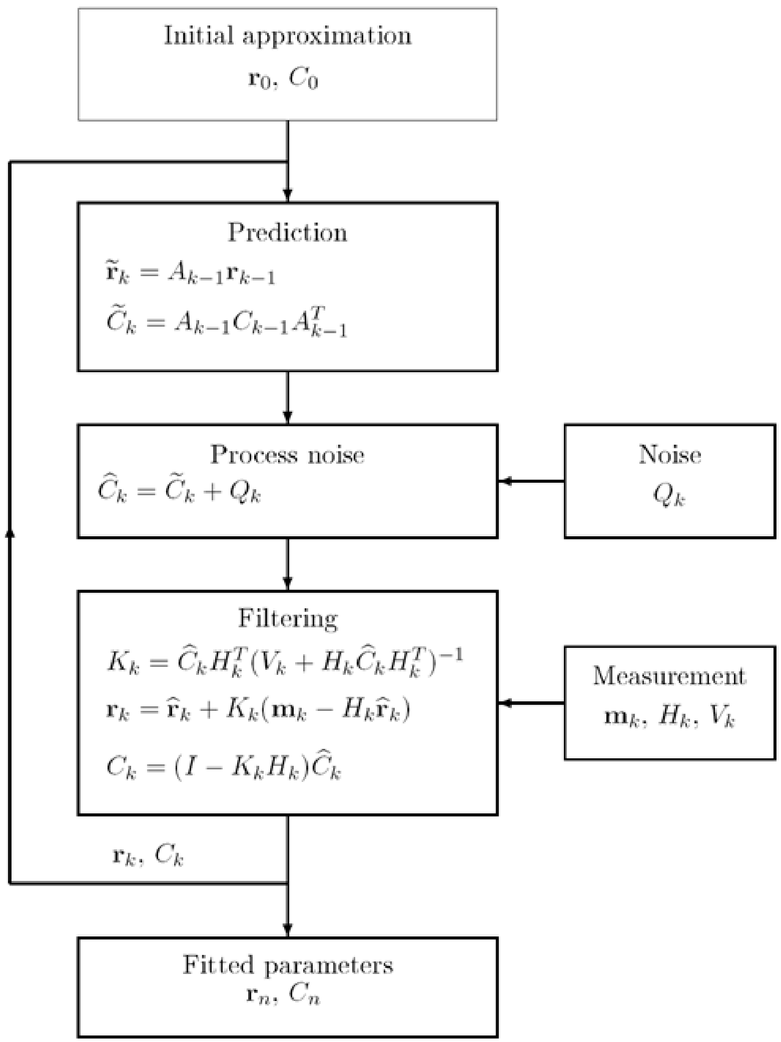
$\mathbf{m}_k,\ H_k,\ V_k$

$\mathbf{r}_k,\ C_k$

Fitted parameters

$\mathbf{r}_n,\ C_n$

Kalman Filter: Conventional and Square-Root



Initial approximation

$\mathbf{r}_0,\ C_0$

Prediction

$\widetilde{\mathbf{r}}_k = A_{k-1}\mathbf{r}_{k-1}$

$\widetilde{C}_k = A_{k-1}C_{k-1}A_{k-1}^T$

Process noise

$\widehat{C}_k = \widetilde{C}_k + Q_k$

Noise

Q_k

Filtering

$K_k = \widehat{C}_k H_k^T (V_k + H_k \widehat{C}_k H_k^T)^{-1}$

$\mathbf{r}_k = \widehat{\mathbf{r}}_k + K_k(\mathbf{m}_k - H_k\widehat{\mathbf{r}}_k)$

$C_k = (I - K_k H_k)\widehat{C}_k$

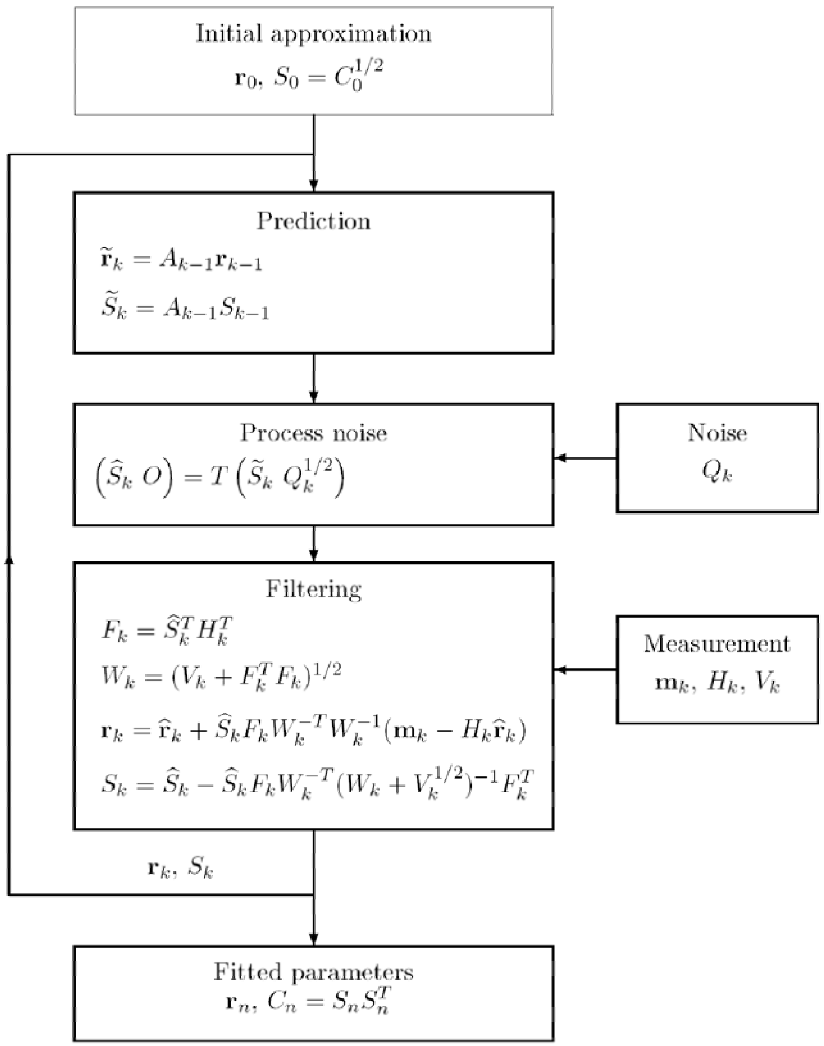
Measurement

$\mathbf{m}_k,\ H_k,\ V_k$

$\mathbf{r}_k,\ C_k$

Fitted parameters

$\mathbf{r}_n,\ C_n$



Initial approximation

$\mathbf{r}_0,\ S_0 = C_0^{1/2}$

Prediction

$\widetilde{\mathbf{r}}_k = A_{k-1}\mathbf{r}_{k-1}$

$\widetilde{S}_k = A_{k-1}S_{k-1}$

Process noise

$\left(\widehat{S}_k\ O\right) = T\left(\widetilde{S}_k\ Q_k^{1/2}\right)$

Noise

Q_k

Filtering

$F_k = \widehat{S}_k^T H_k^T$

$W_k = (V_k + F_k^T F_k)^{1/2}$

$\mathbf{r}_k = \widehat{\mathbf{r}}_k + \widehat{S}_k F_k W_k^{-T} W_k^{-1}(\mathbf{m}_k - H_k\widehat{\mathbf{r}}_k)$

$S_k = \widehat{S}_k - \widehat{S}_k F_k W_k^{-T}(W_k + V_k^{1/2})^{-1} F_k^T$

Measurement

$\mathbf{m}_k,\ H_k,\ V_k$

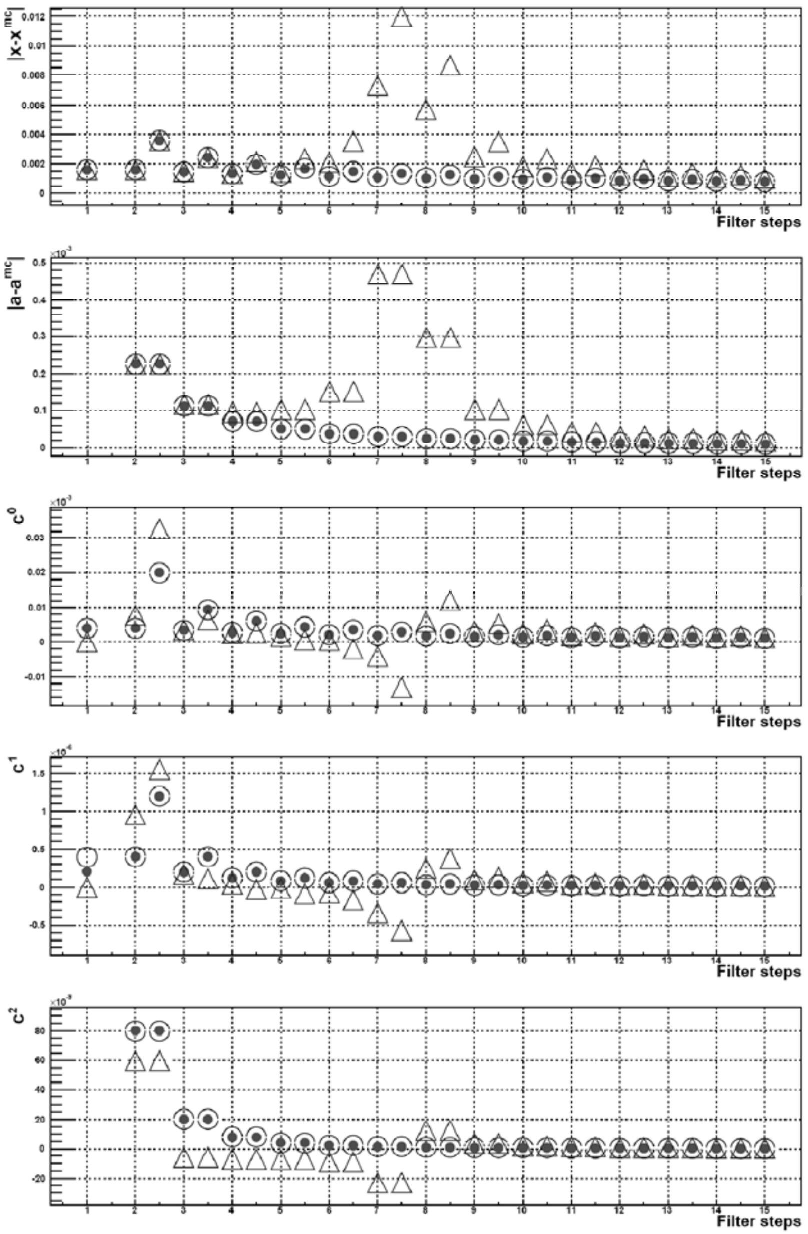
$\mathbf{r}_k,\ S_k$

Fitted parameters

$\mathbf{r}_n,\ C_n = S_n S_n^T$

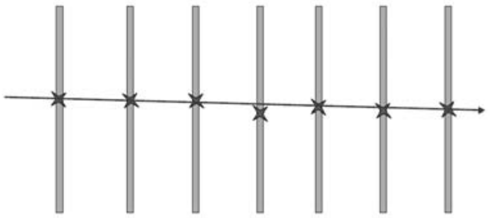
The square-root KF provides the same precision as the conventional KF, but roughly 30% slower.

Kalman Filter Instability in Single Precision



◌ d.p. conventional KF
△ s.p. conventional KF
● s.p. square-root KF

Illustrative example:
2D fit of straight line, no MS, 15 detectors

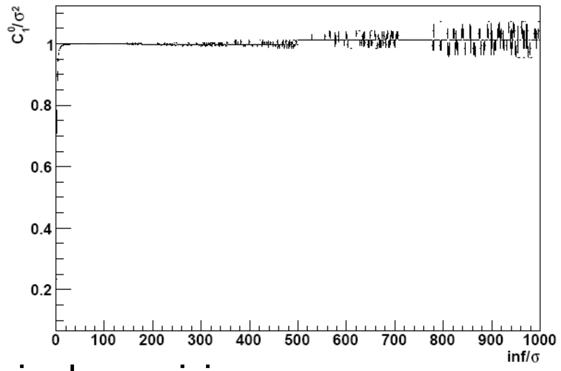


$$\mathbf{r}_k = (x_k, a_k)^T ,$$

$$C_k = \begin{pmatrix} c_k^0 & c_k^1 \\ c_k^1 & c_k^2 \end{pmatrix} .$$

Parameter	Double precision	Single precision	Square root
Resolution $x_{15}[\mu\text{m}]$	9.83	12.42	9.83
Resolution $a_{15}[\mu\text{rad}]$	11.93	21.48	11.93
Pull x_{15} $[\Delta x_{15}/\sqrt{c_{15}^0}]$	1.00	1.13	1.00
Pull a_{15} $[\Delta a_{15}/\sqrt{c_{15}^2}]$	1.00	1.28	1.00

Single precision conventional KF



Conclusion for single precision:

Use the square-root version of KF or

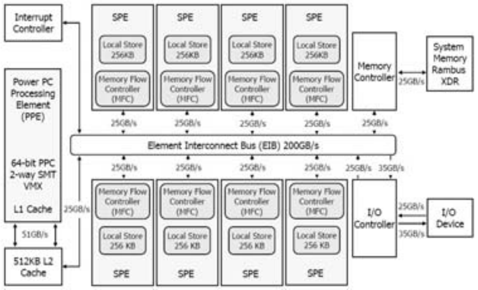
Use double precision for the critical part of KF or

➡ Use a proper initialization in the conventional KF

Porting Algorithms to Cell

Approach:

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

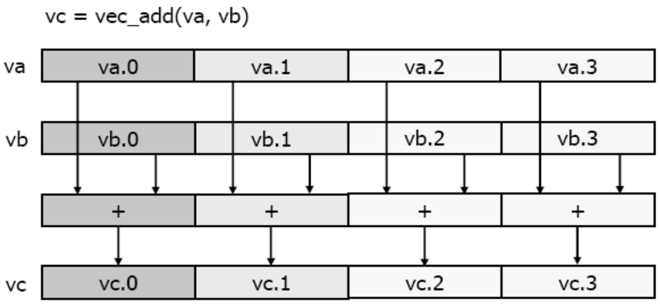


Use headers to overload +, -, *, / operators --> the source code is unchanged !

Data Types:

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

c = a + b



Platform:

1. Linux
2. Virtual machine:
   - ✓ Red Hat (Fedora Core 4)
   - ✓ Cell Simulator:
     - ❖ PPE
     - ❖ SPE
3. Cell Blade

SSE2

SSE2

AltiVec

Specialized SIMD

Track
|Tr|Tr|Tr|Tr|

Code (Part of the Kalman Filter)

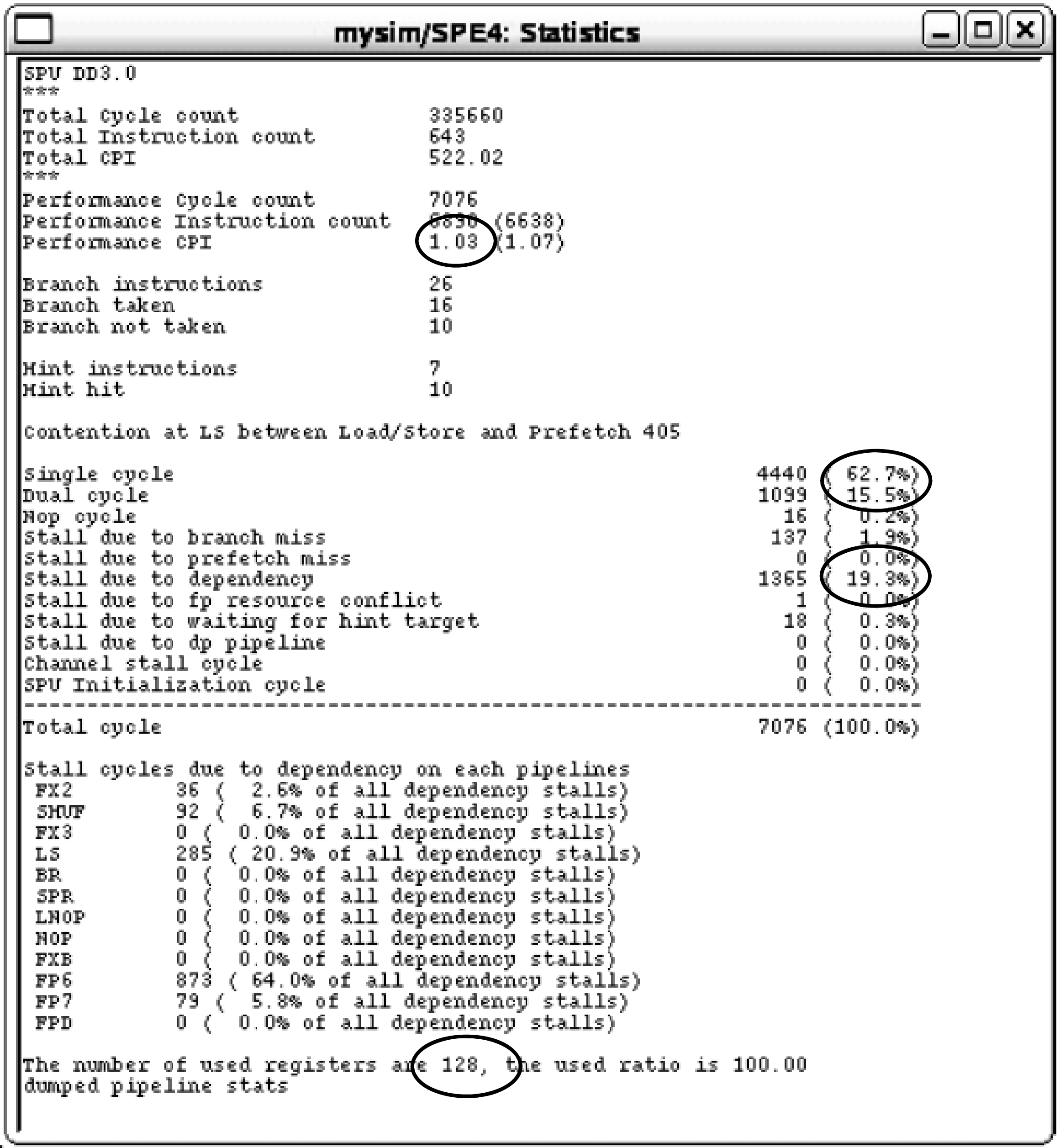
```
inline void AddMaterial( TrackV &track, Station &st, Fvec_t &qp0 )
{
  cnst mass2 = 0.1396*0.1396;

  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;
}
```

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);
 }
```

SPE Statistics



```
SPU DD3.0
***
Total Cycle count              335660
Total Instruction count        643
Total CPI                      522.02
***
Performance Cycle count        7076
Performance Instruction count  6890 (6638)
Performance CPI                1.03 (1.07)

Branch instructions            26
Branch taken                   16
Branch not taken               10

Hint instructions              7
Hint hit                       10

Contention at LS between Load/Store and Prefetch 405

Single cycle                                               4440 ( 62.7%)
Dual cycle                                                 1099 ( 15.5%)
Nop cycle                                                    16 (  0.2%)
Stall due to branch miss                                    137 (  1.9%)
Stall due to prefetch miss                                    0 (  0.0%)
Stall due to dependency                                    1365 ( 19.3%)
Stall due to fp resource conflict                             1 (  0.0%)
Stall due to waiting for hint target                         18 (  0.3%)
Stall due to dp pipeline                                      0 (  0.0%)
Channel stall cycle                                           0 (  0.0%)
SPU Initialization cycle                                      0 (  0.0%)
-----------------------------------------------------------------------
Total cycle                                                7076 (100.0%)

Stall cycles due to dependency on each pipelines
 FX2        36 (  2.6% of all dependency stalls)
 SHUF       92 (  6.7% of all dependency stalls)
 FX3        0 (  0.0% of all dependency stalls)
 LS         285 ( 20.9% of all dependency stalls)
 BR         0 (  0.0% of all dependency stalls)
 SPR        0 (  0.0% of all dependency stalls)
 LNOP       0 (  0.0% of all dependency stalls)
 NOP        0 (  0.0% of all dependency stalls)
 FXB        0 (  0.0% of all dependency stalls)
 FP6        873 ( 64.0% of all dependency stalls)
 FP7        79 (  5.8% of all dependency stalls)
 FPD        0 (  0.0% of all dependency stalls)

The number of used registers are 128, the used ratio is 100.00
dumped pipeline stats
```

Timing profile !

No need to check the assembler code !

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

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

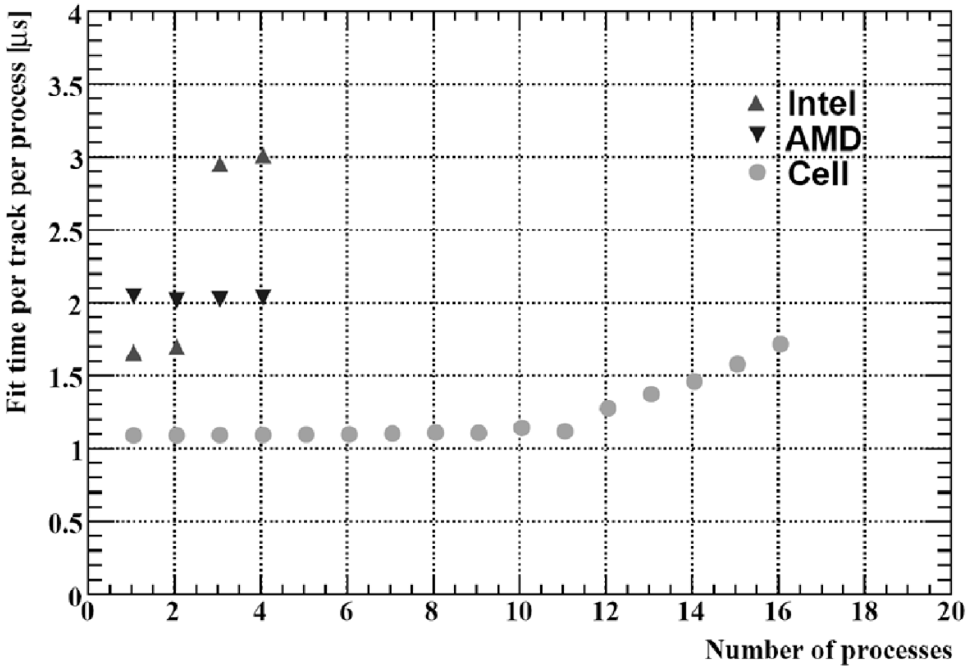
Intel P4

Cell

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.

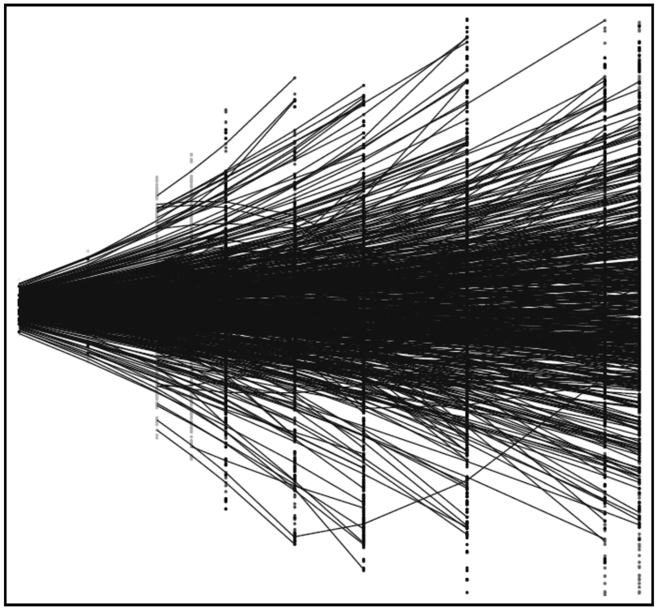


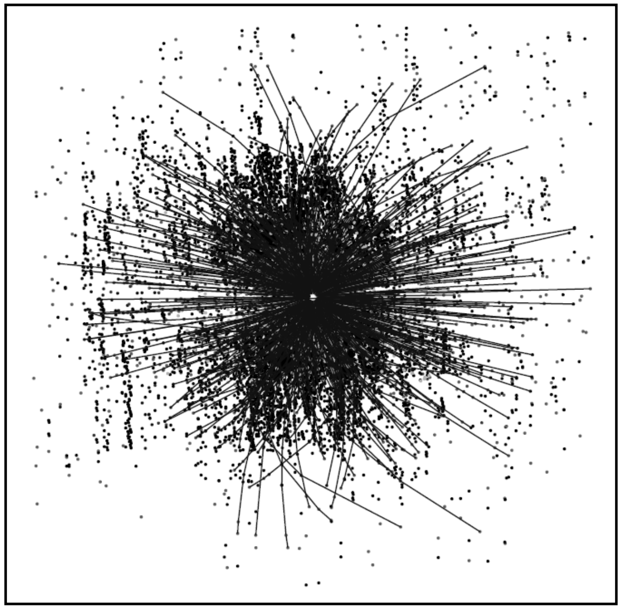
lxg1411@GSI

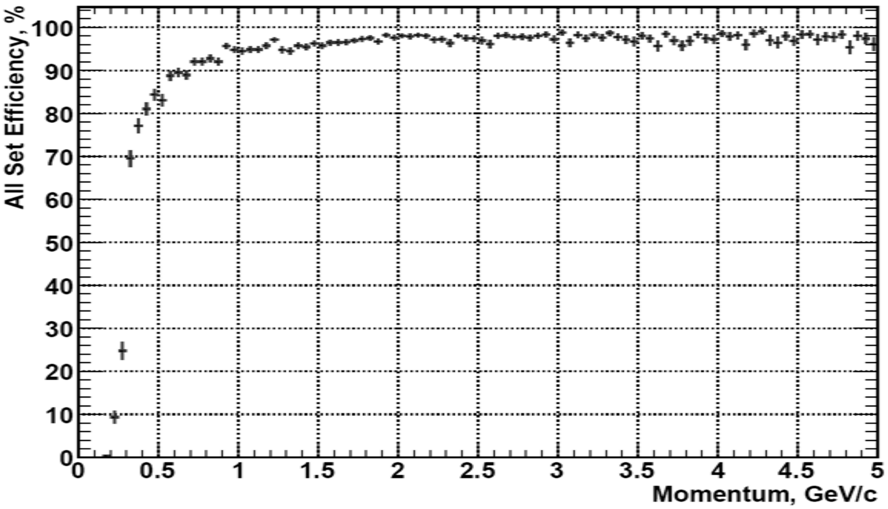
eh102@KIP

blade11bc4 @IBM

SIMDized Cellular Automaton Track Finder: Pentium 4





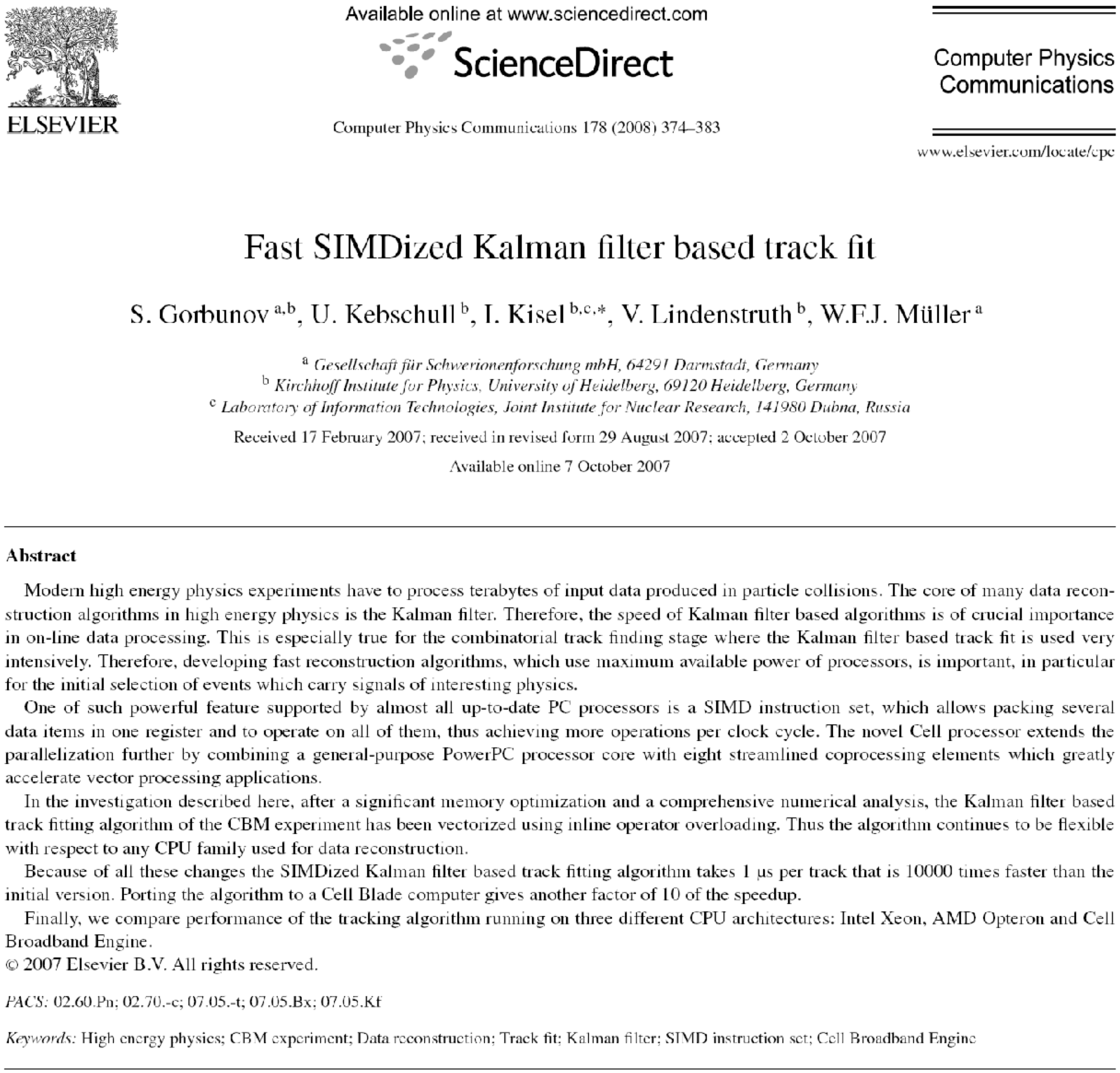


Efficiency, %	Track category	Efficiency, %
97.04	Reference set (>1 GeV/c)	98.21
93.26	All set	94.65
79.95	Extra set (<1 GeV/c)	82.02
1.15	Clone	1.07
2.28	Ghost	0.46
523	MC tracks/event found	93
78 ms	CA time/event	5 ms

1000 faster!

SIMD KF: Code and Article

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



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.

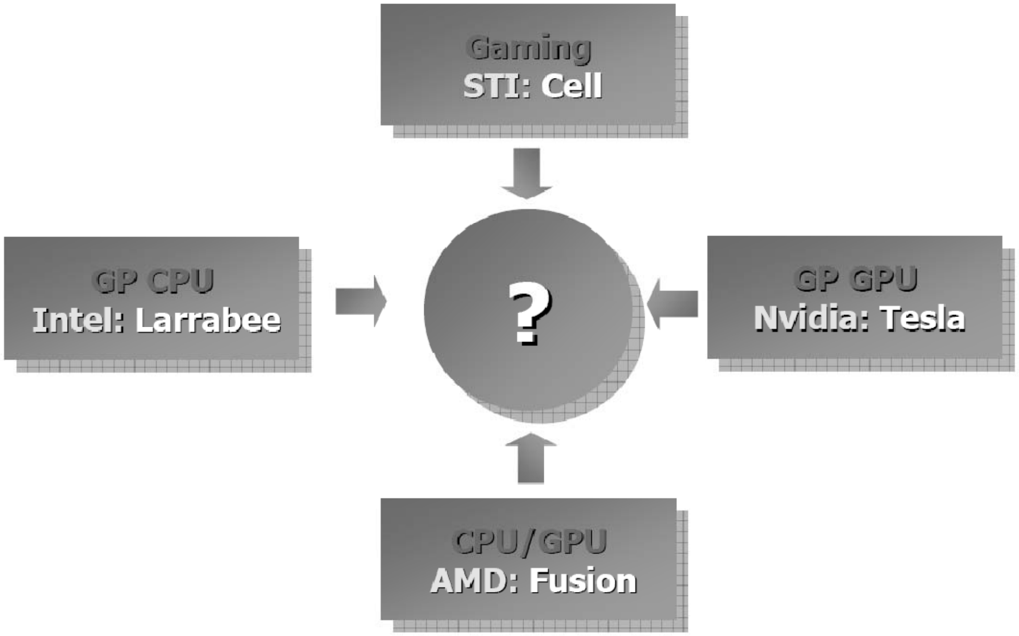
Because of all these changes the SIMDized Kalman filter based track fitting algorithm takes 1 μs per track that is 10000 times faster than the initial version. Porting the algorithm to a Cell Blade computer gives another factor of 10 of the speedup.

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

Summary



- Think about using 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: Ct, CUDA, ...
- Keep portability of the code (Intel, AMD, Cell, GPGPU, ...)
- Try the auto-vectorization option of the compilers