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

1. Introduction

Finding particle trajectories is usually the most time consuming part of modern experiments in high energy physics [1]. In many present experiments with high track densities and complicated event topologies a Kalman filter [1,2] based track fit is used already at this combinatorial part of the event reconstruction. Therefore speed of the track fitting algorithm becomes very important for the total processing time.

CBM [3] is a dedicated heavy-ion experiment with fixed target to investigate the properties of highly compressed baryonic matter as it is produced in nucleus-nucleus collisions at the Facility for Antiproton and Ion Research (FAIR) in Darmstadt, Germany. Large track densities (on average 500 tracks in the main tracker for a typical central Au + Au collision) together with the presence of a non-homogeneous magnetic field make the reconstruction of events challenging. The track reconstruction procedure in the CBM experiment is based on the cellular automaton track finder and the Kalman filter track fitter [4,5]. To achieve a high track finding efficiency the Kalman filter fitting algorithm is intensively used within the track finder.

[*] Corresponding author.
E-mail address: ikisel@kip.uni-heidelberg.de (I. Kisel).

0010-4655/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2007.10.001

Motivated by the idea of using the SIMD unit of modern processors (e.g., [6]), we have investigated here a chain of modifications of the Kalman filter based track fitting algorithm in order to increase the speed of the track finding stage of the event reconstruction. In the CBM experiment the track fitting algorithm based on the conventional Kalman filter is implemented using scalar instructions. Thus we have started with the double precision scalar version of the conventional Kalman filter based track fitting algorithm.

The algorithm uses the 70 MB large map of the magnetic field and, therefore, permanently accesses the main memory which is slow relative to the cache. But similar to other high energy physics experiments, the non-homogeneous magnetic field of the CBM experiment is smooth enough to be locally approximated by a polynomial of fourth order. In the case of the polynomial approximation of the magnetic field the algorithm operates within the cache which results in a significant increase of the speed without degradation of the tracking precision.

In order to further optimize memory usage, precision of all data in the algorithm has been changed from double to single. As a result, twice more data can be stored in the cache and, as well, twice more data can be later packed into a SIMD register, effectively doubling the throughput. The conventional Kalman filter algorithm exhibits an unstable behavior when using only single precision numbers (see also [7,8]). Therefore the Kalman filter algorithm has been specially investigated and modified in order to avoid such instability due to roundoff errors. In addition, the algorithm has been mathematically and numerically optimized, especially in the parts of initial track parameters estimation and also propagation in the magnetic field [5].

In a next step, the algorithm has been adapted for use of a SIMD instruction set. The adaptation has been done by inline operator overloading. This keeps the source code of the algorithm unchanged. Therefore, both versions, scalar and SIMDized, are equivalent and can be selected by a compile time option. Furthermore, this approach gives a unified way of dealing with different CPU families which implement different SIMD instruction sets.

Finally, the SIMDized version of the algorithm has been ported to the Cell processor [9,10]. Initially designed for a game console, the Cell processor promises extremely high computing capabilities. The Cell processor consists of a general-purpose PowerPC processor core (PPE) connected to eight special-purpose streamlined coprocessing synergistic processing elements (SPE), which greatly accelerate multimedia and vector processing applications, as well as many other forms of dedicated computation. Cell combines the considerable floating point resources required for demanding numerical algorithms with a power efficient software-controlled memory hierarchy. The current implementation of Cell is most often noted for its extremely high performance single precision arithmetic. Even though single precision is widely considered insufficient for many scientific applications, it is fully adequate for the reformulated Kalman algorithm. The Cell processor is particularly compelling because it is expected to be produced in high volumes and to be cost competitive with commodity PC CPUs.

Using the IBM Cell Broadband Engine SDK [9,10], the algorithm has been first ported to the PPE and modified for use of the AltiVec vector instructions [11], and then ported to the SPE with the corresponding SPE specific vector instructions. After extensive tests on a Cell simulator, the algorithm has been run on a Cell Blade computer.

In the end, the performance of the SIMDized version of the Kalman filter based track fitting routine has been evaluated on three different computer architectures: Intel Xeon, AMD Opteron and Cell Broadband Engine.

2. SIMD architecture

There are three important classes of computer architectures based upon the number of concurrent instruction and data streams:

- Single instruction, single data stream (SISD)—a single instruction stream on scalar data.
- Single instruction, multiple data streams (SIMD)—multiple data streams against a single instruction stream to perform operations which may be naturally parallelized.
- Multiple instruction, multiple data (MIMD)—many functional units perform different operations on different data.

The basic data unit of SIMD is the vector, which is why SIMD computing is also known as vector processing. These vectors are represented in a packed data format. Data is grouped into bytes or words, and packed into a vector to be operated on. One of the biggest issues in designing a SIMD implementation is how many data elements will it be able to operate on in parallel. For instance, using a 4-element, 128-bit vector one can do four-way single-precision (32-bit) floating-point calculations in parallel (see Fig. 1).

Today SIMD instructions can be found on most CPUs, including the PowerPC's AltiVec [11] and Intel's MMX, SSE, SSE2, SSE3 and SSE4 [6]:

- AltiVec is a floating point and integer SIMD instruction set designed by Apple Computer, IBM and Motorola, and implemented on the PowerPC processors.
- MMX (MultiMedia eXtension) is a SIMD instruction set designed by Intel, introduced in 1997 in their Pentium

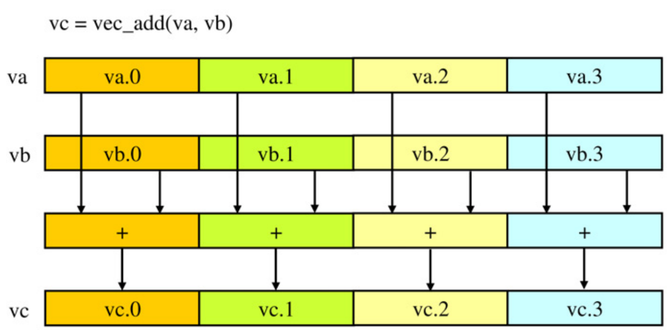


Fig. 1. Four concurrent add operations [10].

MMX microprocessors. MMX added 8 new registers to the architecture. Each of the MMX registers are 64-bit integers.
- SSE (Streaming SIMD Extensions) has been introduced in 1999 in the Pentium III processors. SSE adds 8 new 128-bit registers known as XMM registers. Each register packs together four 32-bit single-precision floating point numbers.
- Introduced with the Pentium 4, SSE2 adds new math instructions for double-precision (64-bit) floating point and 8/16/32-bit integer data types, all operating on the same 128-bit XMM registers. SSE2 enables the programmer to perform SIMD math of virtually any type (from 8-bit integer to 64-bit float) entirely with the XMM registers.
- SSE3 is an incremental upgrade to SSE2, adding DSP-oriented math instructions like addition and subtraction of multiple values stored within a single vector register.
- SSE4 is the fourth iteration of the SSE instruction set developed for the Intel Core microarchitecture.

Both AltiVec and SSE feature 128-bit vector registers that can represent:

- sixteen 8-bit signed or unsigned chars,
- eight 16-bit signed or unsigned shorts,
- four 32-bit integers, or
- four 32-bit floating point variables.

Both provide cache-control instructions intended to minimize cache pollution when working on streams of data.

They also exhibit important differences. Unlike SSE2, AltiVec:

- does not operate on 64-bit double precision floats;
- there is no way to move data directly between scalar and vector registers: the vector registers, like the scalar registers, can only be loaded from and stored to memory.

However, AltiVec:

- provides a much more complete set of “horizontal” operations that work across all the elements of a vector;
- is also unique in its support for a flexible vector permute instruction, in which each byte of a resulting vector value can be taken from any byte of either of two other vectors, parametrized by yet another vector, that allows for sophisticated manipulations in a single instruction;
- the allowable combinations of data type and operations are much more complete;
- 32 128-bit vector registers are provided, compared to 8 for SSE and SSE2;
- most AltiVec instructions take three register operands compared to only two register/register or register/memory operands on IA-32.

Both AltiVec and SSE instruction sets aim the same purpose, but having many shared features are implemented in different ways. Therefore programming common for both platforms is possible, but must be done carefully using matching instructions and constructing equivalent blocks of instructions from both instruction sets.

3. Cell Broadband Engine

Cell[1] [9,10] is a microprocessor architecture jointly developed by a Sony, Toshiba, and IBM alliance known as STI. Cell combines a general-purpose Power-architecture core of modest performance with multiple streamlined coprocessing elements which greatly accelerate multimedia and vector processing applications, as well as many other forms of dedicated computation. The resulting architecture emphasizes efficiency/watt, prioritizes bandwidth over latency, and favors peak computational throughput over simplicity of program code. For these reasons, Cell is widely regarded as a challenging environment for software development. The major commercial application of Cell is in Sony’s PlayStation 3 game console. Although the Cell Broadband Engine is initially intended for application in game consoles and media-rich consumer-electronics devices such as high-definition televisions, the architecture and the Cell Broadband Engine implementation have been designed to enable fundamental advances in processor performance.

The Cell Broadband Engine (Fig. 2) is a single-chip multiprocessor with nine processors operating on a shared, coherent memory. The Cell processor can be split into four components:

- the main processor called the Power Processing Element (PPE) (a two-way SMT multithreaded Power 970 architecture compliant core),
- eight fully-functional co-processors called the Synergystic Processing Elements (SPEs),
- a specialized high-bandwidth circular data bus connecting the PPE, input/output elements and the SPEs, called the Element Interconnect Bus (EIB),
- external input and output structures.

The first type of processor, the PPE, is not intended to perform all primary processing for the system, but rather to act as a controller for the other eight SPEs, which handle most of the computational workload. The second type of processor, the SPE, has RISC architecture with a fixed-width 32-bit instruction format. It is optimized for running compute-intensive applications, and it is not optimized for running an operating system. In one typical usage scenario, the system will load the SPEs with small programs, chaining the SPEs together to handle each step in a complex operation. Another possibility is to partition the input data set and have several SPEs performing the same kind of operation in parallel.

The PPE, is a 64-bit PowerPC Architecture core. It is fully compliant with the 64-bit PowerPC Architecture and can run 32-bit and 64-bit operating systems and applications. The PPE contains a 64-bit general purpose register set, a 64-bit floating point register set, and a 128-bit VMX (AltiVec) register set. The

[1] Cell is a shorthand for Cell Broadband Engine Architecture, commonly abbreviated CBEA in full or Cell BE in part.

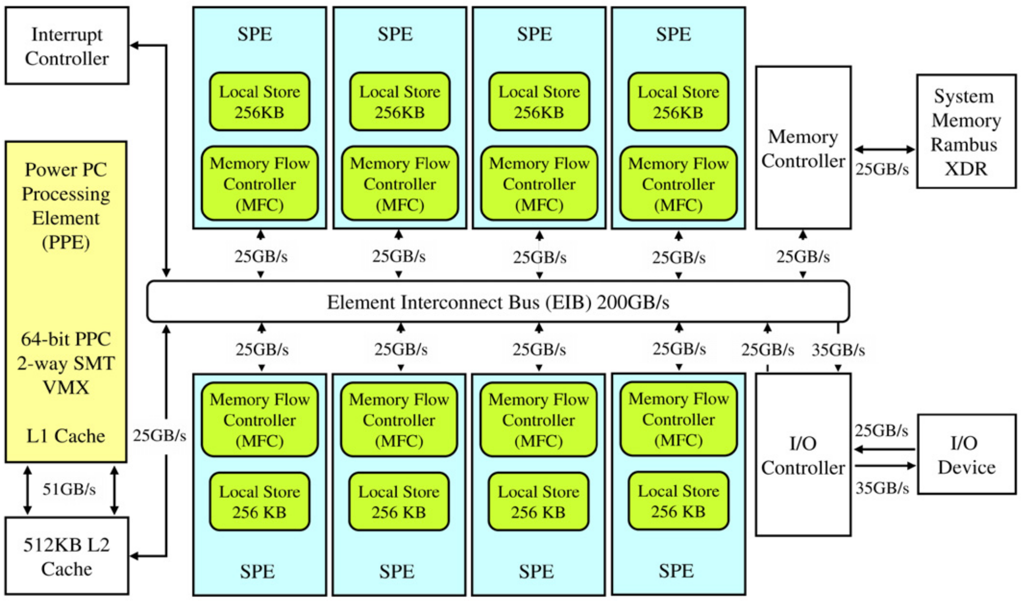


Fig. 2. Cell Broadband Engine overview [10].

PPE contains a 16 kB instruction and data Level 1 cache and a 512 kB Level 2 cache.

The SPEs are independent processors, each running its own individual application programs. They are designed to be programmed in high-level languages and support a rich instruction set that includes extensive SIMD functionality. The SPE is a RISC processor with 128-bit SIMD organization for single and double precision instructions. The SPE contains a 128×128 register file. The SPE can be used for scalar data types ranging from 8-bits to 128-bits in size or for SIMD computations on a variety of integer and floating point formats: 16 8-bit integers, 8 16-bit integers, 4 32-bit integers, or 4 single precision floating-point numbers in a single clock cycle. The SPEs are capable of performing double precision calculations, albeit with an order of magnitude performance penalty. Each SPE has full access to coherent shared memory, including the memory-mapped I/O space. Each SPE is composed of a Streaming Processing Unit (SPU), and a Memory Flow Controller (MFC) unit (DMA, MMU, and bus interface). With the current generation of the Cell, each SPE contains a 256 kB instruction and data local memory area, called Local Store (LS), which is visible to the PPE and can be addressed directly by software. The local store does not operate like a conventional CPU cache since it is neither transparent to software nor does it contain hardware structures that predict what data to load.

A significant difference between the SPEs and the PPE is how they access memory. The PPE accesses main storage (the effective-address space that includes main memory) with load and store instructions that go between a private register file and main storage (which may be cached). However, the SPEs access main storage with direct memory access (DMA) commands that go between main storage and a private local memory used to store both instructions and data. SPE's fetch, load and store instructions access this private local store, rather than shared main storage. This 3-level organization of storage (register file, local store, main storage), with asynchronous DMA transfers between local store and main storage, is a radical break with conventional architecture and programming models, because it explicitly parallelizes computation and the transfers of data and instructions.

At 3.2 GHz, each SPE gives a theoretical 25.6 GFLOPS of single precision performance. The PPE's VMX (AltiVec) unit is fully pipelined for double precision floating point and can complete two double precision operations per clock cycle, which translates to 6.4 GFLOPS at 3.2 GHz; or eight single precision operations per clock cycle, which translates to 25.6 GFLOPS at 3.2 GHz. IBM already presented a blade server prototype based on two Cell processors, running the 2.6.11 Linux kernel. The processors run at 2.4–2.8 GHz. IBM expects soon to run them at 3.0 GHz, providing 200 GFLOPS single-precision floating point performance per CPU (or 400 GFLOPS per board). IBM also expects to arrange seven blades in a single Rackmount chassis (similar to their BladeCenter product line) for a total performance of 2.8 TFLOPS (or 284 GFLOPS in double precision) per chassis. It will also become available in a blade configuration from Mercury Computer Systems, which has released preproduction blades with cell microprocessors that are currently shipping.

Software adoption remains a key issue in whether Cell ultimately delivers on its performance potential. IBM provides a comprehensive Linux-based Cell development platform to assist developers in confronting these challenges.

4. Kalman filter method

The Kalman filter method [1,2] is intended for finding the optimum estimation $\mathbf{r}$ of an unknown vector $\mathbf{r}^t$ according to the measurements $\mathbf{m}_k$, $k = 1 \ldots n$, of the vector $\mathbf{r}^t$.

The Kalman filter starts with a certain initial approximation $\mathbf{r} = \mathbf{r}_0$ and refines the vector $\mathbf{r}$, consecutively adding one mea-

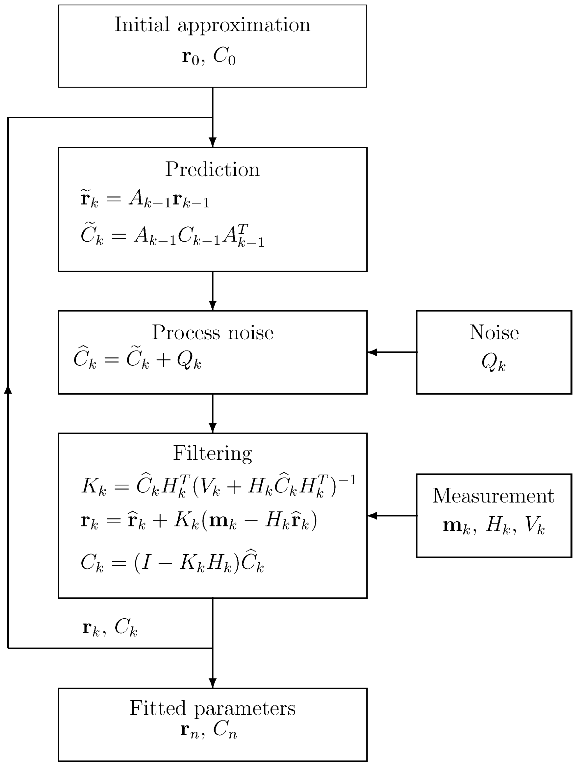

Initial approximation
$\mathbf{r}_0, C_0$
Prediction
$\tilde{\mathbf{r}}_k = A_{k-1}\mathbf{r}_{k-1}$
$\widetilde{C}_k = A_{k-1}C_{k-1}A^T_{k-1}$
Process noise
$\widehat{C}_k = \widetilde{C}_k + Q_k$
Noise
Q_k
Filtering
$K_k = \widehat{C}_k H^T_k (V_k + H_k \widehat{C}_k H^T_k)^{-1}$
$\mathbf{r}_k = \hat{\mathbf{r}}_k + K_k(\mathbf{m}_k - H_k\hat{\mathbf{r}}_k)$
$C_k = (I - K_k H_k)\widehat{C}_k$
Measurement
$\mathbf{m}_k, H_k, V_k$
$\mathbf{r}_k, C_k$
Fitted parameters
$\mathbf{r}_n, C_n$

Fig. 3. Block diagram representation of the conventional Kalman filter.

surement after the other. The optimum value is attained after the addition of the last measurement.

The vector $\mathbf{r}^t$ can change from one measurement to the next:

$$\mathbf{r}^t_k = A_k \mathbf{r}^t_{k-1} + \boldsymbol{\nu}_k, \tag{1}$$

where A_k—a linear operator, $\boldsymbol{\nu}_k$—a process noise between $(k-1)$th and kth measurements.

The measurement $\mathbf{m}_k$ linearly depends on $\mathbf{r}^t_k$:

$$\mathbf{m}_k = H_k \mathbf{r}^t_k + \boldsymbol{\eta}_k, \tag{2}$$

where $\boldsymbol{\eta}_k$—an error of the kth measurement.

It is assumed that measurement errors $\boldsymbol{\eta}_i$ and the process noise $\boldsymbol{\nu}_j$ are uncorrelated, unbiased ($\langle\boldsymbol{\eta}_i\rangle = \langle\boldsymbol{\nu}_j\rangle = \mathbf{0}$) and those covariance matrices V_k, Q_k are known:

$$\langle \boldsymbol{\eta}_i \cdot \boldsymbol{\eta}_i^{\mathrm{T}} \rangle \equiv V_i, \qquad \langle \boldsymbol{\nu}_j \cdot \boldsymbol{\nu}_j^{\mathrm{T}} \rangle \equiv Q_j. \tag{3}$$

The Kalman filter starts with an initial vector $\mathbf{r}_0$, then for each measurement $\mathbf{m}_k$ a vector $\mathbf{r}_k$ is calculated, which is the optimum estimation of the vector $\mathbf{r}^t$ according to the first k measurements.

The conventional Kalman filter algorithm consists of four stages (see also a block diagram in Fig. 3):

1. Initialization step. Choose an approximate value of the vector $\mathbf{r}_0$. Its covariance matrix is set to $C_0 = \mathrm{I} \cdot \mathrm{inf}^2$, where inf denotes a large positive number.
2. Prediction step.
$$\tilde{\mathbf{r}}_k = A_k \mathbf{r}_{k-1}, \qquad \widetilde{C}_k = A_k C_{k-1} A_k^{\mathrm{T}}. \tag{4}$$
3. Process noise. In contrast to the prediction step, describing deterministic changes of the vector $\mathbf{r}^t$ in time, the process noise describes probabilistic deviations of the vector $\mathbf{r}^t$:
$$\hat{\mathbf{r}}_k = \tilde{\mathbf{r}}_k, \qquad \widehat{C}_k = \widetilde{C}_k + Q_k. \tag{5}$$
4. Filtration step. At this step the state vector $\hat{\mathbf{r}}_k$ is updated with the new measurement $\mathbf{m}_k$ to get the optimal estimate of $\mathbf{r}_k$ and its covariance matrix C_k:
$$\begin{aligned} K_k &= \widehat{C}_k H_k^{\mathrm{T}} \big(V_k + H_k \widehat{C}_k H_k^{\mathrm{T}}\big)^{-1}, \\ \mathbf{r}_k &= \hat{\mathbf{r}}_k + K_k(\mathbf{m}_k - H_k \hat{\mathbf{r}}_k), \\ C_k &= \widehat{C}_k - K_k H_k \widehat{C}_k, \\ \chi^2_k &= \chi^2_{k-1} + (\mathbf{m}_k - H_k \hat{\mathbf{r}}_k)^{\mathrm{T}} \big(V_k + H_k \widehat{C}_k H_k^{\mathrm{T}}\big)^{-1} \\ &\quad \times (\mathbf{m}_k - H_k \hat{\mathbf{r}}_k). \end{aligned} \tag{6}$$

The following designations are used in Eqs. (4)–(6): $\mathbf{r}_{k-1}$, C_{k-1}—the optimum estimation, obtained at the previous step and the error covariance matrix; the matrix A_k relates the state at step $k-1$ to the state at step k; $\tilde{\mathbf{r}}_k$, $\widetilde{C}_k$—predicted estimation of $\mathbf{r}^t_k$ before the process noise; $\hat{\mathbf{r}}_k$, $\widehat{C}_k$—predicted estimation of $\mathbf{r}^t_k$ after the process noise; $\mathbf{m}_k$, V_k—the kth measurement and its covariance matrix; the matrix H_k—the model of measurement; the matrix K_k is the so-called gain matrix; the value χ^2_k is the total χ^2-deviation of the obtained estimation $\mathbf{r}_k$ from the measurements $\mathbf{m}_1, \ldots, \mathbf{m}_k$.

The vector $\mathbf{r}_n$ obtained after the filtration of the last measurement is the desired optimal estimation of the $\mathbf{r}^t_n$ with the covariance matrix C_n.

In track fitting applications, the state vector $\mathbf{r}_k$ is vector of the track parameters, the prediction matrix A_k describes extrapolation of the track in the magnetic field from one detector to another, and the matrix of noise Q_k describes the effect of multiple scattering in the material.

5. Speedup of the algorithm

The Kalman filter method is used both in the track finding and track fitting routines of the CBM experiment. The track finder is based on the cellular automaton method [4]. The algorithm creates short track segments (triplets) locally in neighboring detector planes and links them into track candidates, which are then selected using the χ^2-criterion. The Kalman filter based routines are used at all stages of the track finder in order to reliably estimate parameters of the track segments and quality of the track candidates. The track fitting routine in the CBM experiment realizes the Kalman filter in its conventional approach and includes all necessary implementations of the Kalman filter method, like extrapolation, update and smoother. All variables in the routine are scalars and most of them have floating point representation in double precision. The track fitting routine performs three iterations in order to achieve maximum momentum resolution and to take accurately into account multiple scattering in material of the detectors. As in the track fitter the Kalman filter is more isolated comparing to the track finder, we have chosen the Kalman filter based

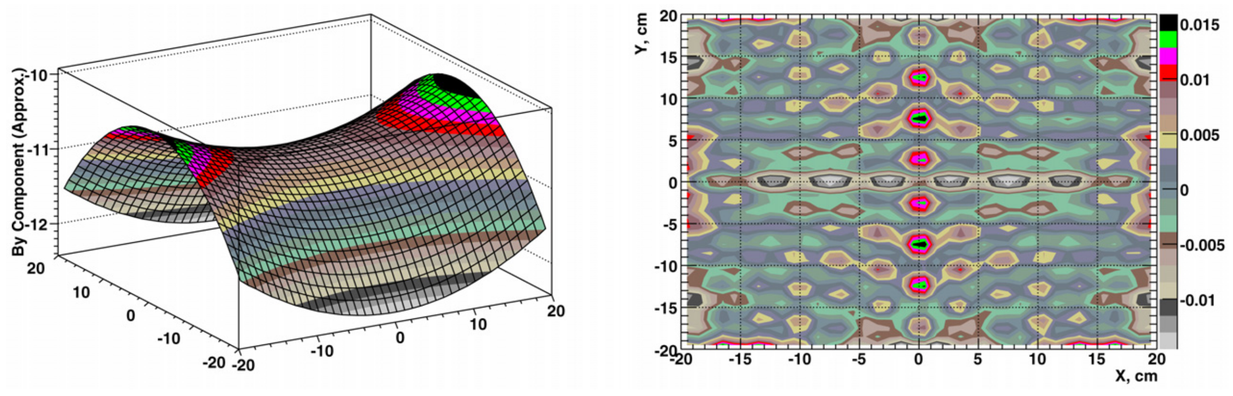


Fig. 4. The most significant (B_y) component of the active magnetic field in the middle of the detector system ($z = 50$ cm) calculated using the polynomial approximation (left) and difference between two alternative field representations (right).

track fitting procedure for investigation of its vectorization ability and further implementation in the Cell processor. Collecting of measurements into tracks has been done based on the Monte-Carlo information requiring for hits in a track to have the same particle ID. Thus the problem becomes well localized for further analysis.

For these studies central Au + Au collisions at 35 AGeV have been simulated. In the simulations we used the main tracking detector of 7 silicon pixel stations positioned at 10, 20, 30, 40, 60, 80 and 100 cm from the target. The first 2 stations have thickness of 150 μm, while others—400 μm. All detectors have idealized response (no fake hits, efficiency losses, pile-up, etc.). The non-homogeneous active magnetic field has been used to trace particles through the detector.

At the first stage we have optimized memory access in the algorithm. The magnetic field of the CBM experiment is non-homogeneous and stored in a 70 MB large map, thus requiring permanent access to the main memory. It is obvious, that running on a conventional computer performance of an algorithm with data located in the main memory is significantly slower comparing to that working within the cache. This is especially true for the Cell processor with the size of the local store of SPEs comparable with the cache size, where unpredictable access to the magnetic field map in the main memory of PPE is a blocking process which stalls the algorithm. But the magnetic field of the CBM experiment is relatively smooth and, therefore, can be locally approximated by polynomials. It was found to be sufficient for the propagation step in the Kalman filter to use a polynomial of fourth order to approximate the field in planes of each station (see Fig. 4 for comparison of two alternative field representations). Field behavior between stations is approximated by a parabola with coefficients calculated from three closest hits of the current track, since we need the field only along the track to be fitted [5]. Track parameters taken with the polynomial approximation of the magnetic field are as precise as those calculated using the full magnetic field map, showing there is no degradation.

At the second stage, the fitting algorithm has been significantly modified in two directions: changing precision of the variables in the algorithm from double to single and computationally optimal implementation of the Kalman filter method. Usually, the conventional Kalman filter requires calculations with double precision [1]. On the other hand, operating with data in single precision has several advantages. Using single precision reduces twice the size of data and, therefore, size of required memory to store it. In this case, more data can be read into cache of a conventional CPU or into local store of SPE that results in faster performance of the algorithm. In addition, twice more data with single precision can be later packed into a vector thus automatically doubling the speed of the SIMDized algorithm. Moreover, the current implementation of the Cell processor is optimal for SIMD operations with single precision, but is an order of magnitude slower performing double precision calculations. Changing precision of all floating point variables from double to single precision, we have realized that 32 bits of single precision is not enough for the conventional Kalman filter to be numerically stable. It produces results, which are completely unacceptable not only due to poorer quality of the track parameters, but also because of bad numerical properties of the covariance matrix, like negative diagonal elements. It is possible to keep several variables (like the covariance matrix) in double precision and also process some critical calculations in double precision, but having a significant extra charge for operations in double precision on SPE, we have decided to rewrite the entire algorithm in single precision. Therefore, a numerically stable and accurate single precision approach of the Kalman filter was necessary. There are several methods to keep the Kalman filter stable and accurate in single precision [7]. One of the best is the square root implementation of the Kalman filter [7,8,12,13], where calculations are performed on square root of the covariance matrix ($C_k \equiv S_k S_k^{\mathrm{T}}$). Although equivalent algebraically to the conventional approach, the square root filter exhibits improved numerical characteristics, particularly in ill-conditioned problems. The square root filter provides in single precision the same accuracy as the conventional Kalman filter in double precision. The square root filter has additional transformations compared to the conventional approach and therefore requires about 30% more processing time. Such additional overhead is usually considered as acceptable for the benefit of improved numerical stability. But in the tracking application of the Kalman filter, a comprehensive analysis has shown, that the only source of instability is filtration of the first measurements,

Table 1
Timing (in %) for different steps of the Kalman filter based fitting routine

Step	Description	Timing, %
1	Initialization	11
2	Prediction	45
3	Process noise	8
4	Filtration	36

where errors of the initial track parameters can be a few orders of magnitude larger than errors of the measurements, thus causing significant roundoff errors (see Eqs. (6)) and, therefore, loss of precision of the method. We have found that the direct comparison of the errors before the filtration step and neglecting the errors which are more than 4 times larger than the mean measurement error keeps the algorithm stable and accurate in single precision without any extra calculations like in the square root approach.

In addition, the Kalman filter algorithm has been mathematically optimized. Usually it starts with arbitrary initial track parameters and a large covariance matrix, and then repeats few iterations in order to converge to an optimum solution. In our case the initial track parameters are directly estimated from the input data, thus only one iteration of the Kalman procedure is necessary. Furthermore, as a result of the polynomial approximation of the magnetic field, the propagation step of the Kalman filter can be performed directly from measurement to measurement without necessity of additional intermediate steps. Other optimizations have been also implemented, like replacement of matrix multiplications by direct operations on only non-trivial matrix elements. The algorithm has been also extensively analyzed with respect to its numerical optimization, for instance: most of loops have been unrolled in order to provide additional instructions for interleaving; most branches have been eliminated from the algorithm to avoid branch misprediction penalty; calculations have been reordered for better use of the processor pipeline.

Table 1 gives relative timing for different steps (see Eqs. (4)–(6) and Fig. 3) of the Kalman filter based fitting routine. One can see that even without reading the magnetic field map the propagation of the track parameters (the prediction step) is still the most time consuming part of the fitting procedure because of complexity of the propagation in a non-homogeneous magnetic field.

At the third stage the tracking data has been vectorized. As all tracks are independent and fitted by the same algorithm, one can fit them in parallel after packing the corresponding track parameters of each four consecutive tracks into vectors. They are packed such that, for instance, x-coordinates of four tracks are represented as vector of four elements $(x_i, x_{i+1}, x_{i+2}, x_{i+3})$. Thus data processing remains identical both for scalar and vector representations.

Then, the track fitting algorithm written in C++ language has been vectorized at first in order to use the SSE2 instruction set. The problem is that the vector instructions look completely different from the corresponding scalar instructions: for instance, the scalar operation `c = a + b` becomes `c = vec_add(a, b)`. Rewriting the code using vector instructions would require in the future to provide support for both, scalar and vector, versions, duplicating modifications in both versions and initiating another loop of debugging and testing. Therefore, we have decided to implement the SSE2 vector instruction set in a header file, overloading all operands and inlining several functions.[2] In this way the source code remains untouched, and possible changes of the code in the future will be valid for both, scalar and vector, versions. Quality of the track parameters and the covariance matrix of the SIMDized version of the track fit has been shown to be the same as in the scalar version.

At the fourth stage the algorithm has been ported into the Cell simulator and run on the PPE. For that we have installed the Red Hat Linux (Fedora Core 4), the only Linux version which is supported by the Cell simulator at the time of the investigation. Implementing the AltiVec instruction set of the PPE in another header was relatively easy because of similarity between SSE2 and AltiVec. It was still possible to run both, scalar and vector, versions also on the PPE, thus examining consistency of results.

The last step was porting the code into the SPE. Again, this has been done by writing another header, which implements the specialized SIMD instruction set of the SPE. In addition, we have slightly modified the code in order to provide communication between the PPE and the SPE and to exchange data between the main memory and the local store of the SPE. The total size of SPE code is only 50 kB, which fits very well into the local store of the SPE, leaving the remaining 200 kB for data.

The SPU statistics of the Cell simulator is given in Fig. 5. It shows that the algorithm achieves a very good overall cycles per instruction (CPI) performance of 1.03. It has 15.5% dual-issue (odd and even pipeline use) rates, almost no stall due to branch miss (1.9%) and low dependency stalls (19.3%) that is good for such complicated algorithm. In addition, all 128 registers have been used.

After extensive tests on the simulator the algorithm has been run on a Dual Cell-Based Blade computer running at 2.4 GHz. There were no significant problems observed at this step.

At the last stage all 16 SPEs of the two Cell processors available on the Cell Blade have been running in parallel to process different data samples.

6. Results and discussion

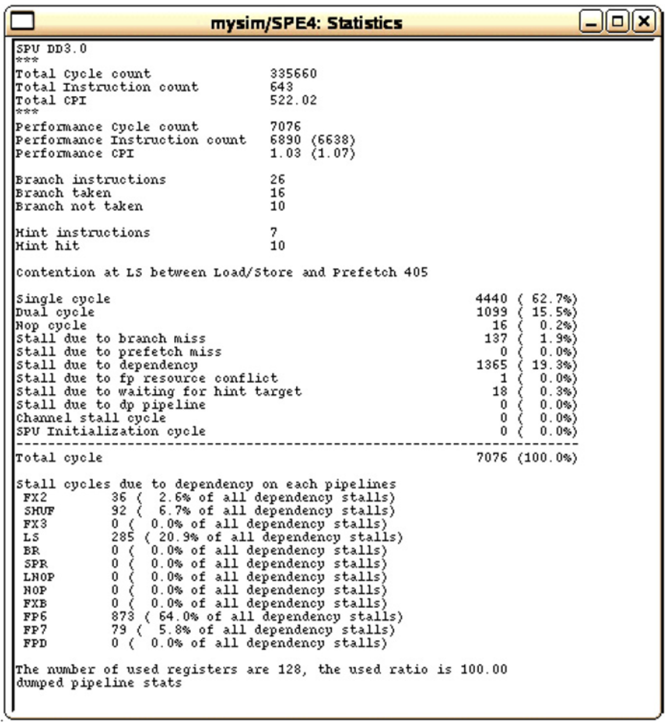
The Kalman filter based track fitting algorithm has been tested on simulated data of the CBM experiment [3,4].

In the CBM experiment with forward geometry the natural choice of the state vector[3] is:

$$\mathbf{r} = \{x, y, t_x, t_y, q/p\}, \tag{7}$$

[2] In case no SIMD instruction set is supported by a computer, the vector type is substituted by the pseudo-vector array of four scalars.

[3] The z-coordinate points downstream the spectrometer axis, and x and y are transverse coordinates.


mysim/SPE4: 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
```

Fig. 5. A dynamic timing analysis of the SPE using the IBM Full System Simulator for the Cell Broadband Engine.

where x and y are track coordinates at the reference z-plane, $t_x = \tan\theta_x$ is the track slope in the xz plane, $t_y = \tan\theta_y$ is the track slope in the yz plane, q/p is the inverse particle momentum, signed according to charge.

Quality of the track parameters has been monitored at all stages of the speedup of the algorithm. Fig. 6 shows residuals and normalized residuals (pulls) of the track parameters at the production vertex obtained on a Cell Blade computer.

Errors (residuals) ρ of the track parameters, for instance, of the x-coordinate, are determined as:

$$\rho_x = x^{\text{reco}} - x^{\text{mc}}, \tag{8}$$

where x^{reco}—reconstructed and x^{mc}—true Monte-Carlo values of the x-coordinate.

A measure of the reliability of the fit are the normalized residual (pull) distributions of the fitted track parameters. Normalized residuals are determined according to the formula:

$$P(x) = \frac{\rho_x}{\sqrt{C_{xx}}}, \tag{9}$$

where C_{xx}—the corresponding diagonal element of the covariance matrix, obtained in the track fit. In the ideal case the normalized error distributions of coordinates and slopes of the track should be unbiased and Gaussian distributed with width of 1.0.

Fig. 6 gives also the RMS of the Gaussian fits to the residual and normalized residual distributions at the production vertex. The reconstructed track parameters and covariance matrix at the vertex where the track originates are obtained by propagating the track parameters at the measurement position closest to the vertex, taking into account the remaining material traversed. All pulls are centered at zero indicating that there is no systematic shift in the reconstructed track parameter values. The pull distributions are well fitted using Gaussian functions with small tails caused by the various non-Gaussian contributions to the fit. The q/p pull shows slightly underestimated errors. This can be a result of several approximations made in the fitting procedure mainly in the part of material treatment.

Table 2 summarizes all stages and gives timing analysis and speedup after each stage of the development. One can see, that elimination of the magnetic field map and, as a result, no need to access the main memory increase the speed of the algorithm up to 50 times. Optimization of the algorithm resulted in the 35 times faster performance. The vectorization stage, in contrast to clear software improvements at the first two stages, required both software and hardware changes giving 4.5 times speedup. Porting to SPE resulted in 1.5 increase of the speed with respect to a Pentium 4 processor used at the previous stages, probably because of increased number of registers. The last stage is another hardware improvement due to use of all 16 SPEs of the Cell Blade computer and, because of such simple parallelization as in the case of track fitting, gives another 10 times speedup. In total the speed of the algorithm had been increased in 120,000 times.

In addition, we have compared the timing performance of the SIMDized version of the Kalman filter based track fitting routine on three different computers based on:

- 2 Intel Xeon Processors with Hyper-Threading enabled and 512 kB cache at 2.66 GHz[4];
- 2 Dual Core AMD Opteron Processors 265 with 1024 kB cache at 1.8 GHz[5];
- 2 Cell Broadband Engines with 256 kB local store at 2.4 GHz.[6]

Both, Intel and AMD based, personal computers are treated by the operating system as having 4 processors each.

Table 3 gives a real time performance for fitting a single track on different computers. Only one processing unit (CPU or SPU) is active, while others are in the idle state or running the operating system. Since only one track of about 0.5 kB size is fitted, all computations are located within the cache or the local memory. The Cell Blade computer has the fastest performance requiring only half the clock cycles per track when compared to the Intel Xeon based computer.

Fig. 7 gives real time per track for the SIMDized version of the Kalman filter fitting routine for the Intel Xeon, AMD Opteron and Cell based computers running different number of processes in parallel. A very large sample of tracks exceeding many times the size of the cache or the local store has been processed in order to include the effect of communication to

[4] `lxg1411` at the Gesellschaft für Schwerionenforschung mbH, 64291 Darmstadt, Germany.

[5] `eh102` at the Kirchhoff Institute for Physics, University of Heidelberg, 69120 Heidelberg, Germany.

[6] `blade11bc4` at the IBM Laboratory Böblingen, Schönaicher Str. 220, 71032 Böblingen, Germany.

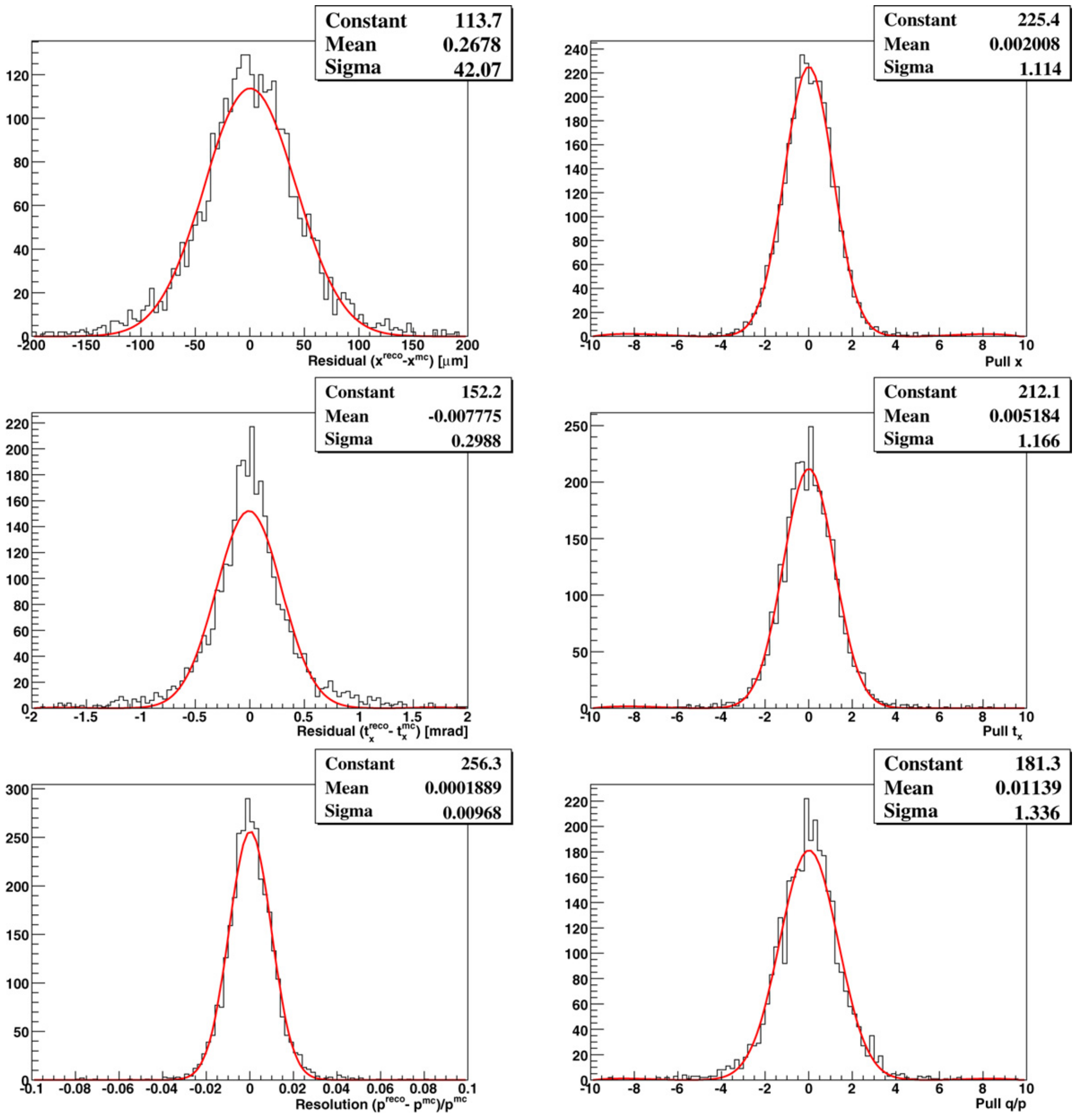


Fig. 6. Residuals and normalized residuals (pulls) of the estimated track parameters at the production vertex for central Au + Au collisions at 35 *A*GeV in the approximated active magnetic field of the CBM experiment [3,4] obtained on the Cell Blade computer.

Table 2
Summarized stages of the porting procedure

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 (2 Cells)	0.1 μs	10
	Final SIMDized version	0.1 μs	120,000

Table 3
Real time performance of the SIMDized version of the Kalman filter based fitting routine for a single track fitted on three different CPU families

Processing unit	Clock, GHz	Time/track, μs	kCycle/track
Intel Xeon	2.66	1.47	3.91
AMD Opteron	1.8	1.86	3.35
Cell SPE	2.4	0.87	2.09

the main memory. Comparing with Table 3 one can see that for all computers there is a little overhead of about 10% because of reading data from the main memory. It is also clear that the hyper-threading of the Intel Xeon processor does not contribute in this particular case of the fitting procedure, in contrast the dual core technology of the AMD Opteron processor shows stability of the timing performance due to its NUMA architecture. In the Cell Blade computer all 16 SPEs work completely independent and in parallel. They have the constant speed of the algorithm per processing unit up to 11 processes, then slightly reducing the speed probably due to large data flow through the element interconnect bus.

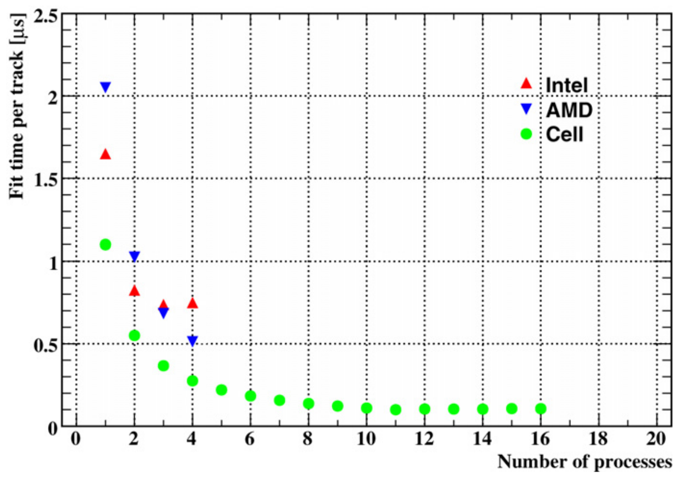


Fig. 7. Fit time per track of the SIMDized version of the Kalman filter based fitting routine measured in real time for the Intel Xeon, AMD Opteron and Cell based computers running different number of processes in parallel.

Having significant differences in the architectures and clock rate all computers have shown similar speed[7] of the algorithm per processing unit. This can be, first, because the final algorithm does not require large memory and most of calculations are done within the cache or the local store. Second, the algorithm implements the Kalman filter technique in the same source code which after compilation by a gcc compiler produces executables with similar performances.

The local store of the SPE requires a special consideration, but can give more freedom to the developer if comparing with the cache of Intel or AMD processors. The vector instruction set of the SSE2 has relatively limited capabilities to operate with the cache. In contrast, the instruction set of the SPE has considerable number of instructions for non-blocking transfers between local store and main memory. In our case, the track fitting algorithm does not require large exchange of data between the local store and the main memory, therefore this difference between the processors has not been observed.

The cellular automaton track finder of the CBM experiment has been also significantly reworked in order to be SIMDized. The SIMDized Kalman filter based fitting routines have been included into the track finder. First tests of the fully SIMDized cellular automaton track finder [14] show 1000 times increase of the reconstruction speed with respect to the initial scalar version running on the same Pentium 4 based computer.

7. Conclusion

The Kalman filter based track fitting algorithm, the core algorithm of the event reconstruction software in high energy physics experiments, has been significantly optimized and adapted to a vector form implementing different SIMD instruction sets: SSE2 of the Intel and AMD CPUs, AltiVec of the PPE and the specialized SIMD instruction set of the SPE of the Cell processor. Overloading basic scalar operators by corresponding vector instructions keeps the source of the algorithm unchanged, thus providing the unified approach for all computer architectures under investigation.

The overall speedup in 120 000 times has been obtained on the Cell Blade computer compared to the initial implementation on a Pentium 4.

[7] It should be noted, that there are processors with higher clock rates available.

Acknowledgements

We wish to thank M. Engler, J. Franz and J. Jordan from the IBM Laboratory, Böblingen, for giving us the possibility to test the algorithm on the Cell Blade system. We would like to thank also Drs. T. Steinbeck and R. Weis from the Kirchhoff Institute for Physics, University of Heidelberg, for their technical assistance.

We acknowledge the support of the European Community-Research Infrastructure Activity under the FP6 "Structuring the European Research Area" programme (HadronPhysics, contract number RII3-CT-2004-506078) and of the BMBF Project "FAIR-CBM: Entwicklung von schnellen und leistungsfähigen Methoden für Ereignisselektion".

References

[1] R. Frühwirth, et al., Data Analysis Techniques for High-Energy Physics, second ed., Cambridge Univ. Press, 2000.
[2] R.E. Kalman, A new approach to linear filtering and prediction problems, Trans. ASME, Series D, J. Basic Eng. 82 (1960) 35–45.
[3] CBM Collaboration, Compressed Baryonic Matter Experiment, Technical Status Report, GSI, Darmstadt, 2005; 2006 Update http://www.gsi.de/documents/DOC-2006-Feb-108-1.pdf.
[4] I. Kisel, Event reconstruction in the CBM experiment, Nucl. Instr. Methods A 566 (2006) 85–88.
[5] S. Gorbunov, I. Kisel, Analytic formula for track extrapolation in non-homogeneous magnetic field, Nucl. Instr. Methods A 559 (2006) 148–152.
[6] IA-32 Intel Architecture Optimization Reference Manual, Intel, June 2005.
[7] M.S. Grewal, A.P. Andrews, Kalman Filtering: Theory and Practice using MATLAB, second ed., John Wiley & Sons, NY New York, 2001.
[8] P.G. Kaminski, A.E. Bryson, S.F. Schmidt, Discrete square root filtering: A survey of current techniques, IEEE Trans. Auto. Control AC-16 (1971) 727–736.
[9] Cell Broadband Engine, Programming Tutorial. Ver. 1.0, IBM, October 21, 2005.
[10] Cell Broadband Engine, Programming Handbook. Ver. 1.0, IBM, April 19, 2006.
[11] I. Ollmann, AltiVec Tutorial. Ver. 1.2, 2002; http://www.simdtech.org.
[12] A.S. Householder, The Theory of Matrices in Numerical Analysis, Blaisdell, Waltham, MA, 1964; Reprinted by Dover, New York, 1974.
[13] A. Andrews, A square root formulation of the Kalman covariance equations, AIAA J. 6 (June 1968) 1165–1166.
[14] I. Kisel, L1 Event Reconstruction. CBM Collaboration Meeting, February 27–March 2, 2007, GSI, Germany; http://www.gsi.de/documents/DOC-2007-Mar-27-1.pdf.