Письма в ЭЧАЯ. 2010. Т. 7, № 4(160). С. 473–482

FAST PARALLEL TRACKING ALGORITHM FOR THE MUON DETECTOR OF THE CBM EXPERIMENT AT FAIR

A. Lebedev [a,b], *C. Höhne* [a], *I. Kisel* [a], *G. Ososkov* [b]
(for the CBM Collaboration)

[a] Gesellschaft für Schwerionenforschung mbH, Darmstadt, Germany
[b] Joint Institute for Nuclear Research, Dubna

Particle trajectory recognition is an important and challenging task in the Compressed Baryonic Matter (CBM) experiment at the future FAIR accelerator at Darmstadt. The tracking algorithms have to process terabytes of input data produced in particle collisions. Therefore, the speed of the tracking software is extremely important for data analysis. In this contribution, a fast parallel track reconstruction algorithm which uses available features of modern processors is presented. These features comprise a SIMD instruction set (SSE) and multithreading. The first allows one to pack several data items into one register and to operate on all of them in parallel thus achieving more operations per cycle. The second feature enables the routines to exploit all available CPU cores and hardware threads. This parallel version of the tracking algorithm has been compared to the initial serial scalar version which uses a similar approach for tracking. A speed-up factor of 487 was achieved (from 730 to 1.5 ms/event) for a computer with 2×Intel Core i7 processors at 2.66 GHz.

Реконструкция траекторий заряженных частиц — это важная и сложная задача в эксперименте CBM (Compressed Baryonic Matter) на будущем ускорителе FAIR в Дармштадте. Алгоритм реконструкции треков должен обработать терабайты входных данных, вырабатываемых в столкновениях частиц. Следовательно, скорость работы алгоритма реконструкции треков чрезвычайно важна для обработки данных. В работе представлен быстрый параллельный алгоритм реконструкции треков, использующий возможности современных процессоров, в частности набор системных команд SIMD (SSE) и многопоточность. Первая особенность позволяет упаковывать несколько элементов данных в один регистр и выполнять операции над всеми данными параллельно, достигая таким образом большего количества операций за такт. Вторая особенность позволяет использовать все возможные ядра и аппаратные потоки процессора. Проведено сравнение параллельной версии алгоритма реконструкции треков с первоначальной последовательной скалярной версией, использующей аналогичный подход для реконструкции треков. Достигнуто ускорение в 487 раз (с 730 до 1,5 мс/событие) для компьютера с двумя процессорами Intel Core i7 (2,66 ГГц).

PACS: 29.20.db; 29.40.Gx; 29.85.Fj

INTRODUCTION

The Compressed Baryonic Matter (CBM) [1] experiment will be a dedicated setup for the measurement of fixed target heavy-ion collisions at the future FAIR accelerator at Darmstadt. It is being designed for the investigation of the properties of highly compressed baryonic

matter [2]. High reaction rates (up to 10 MHz), events with large track multiplicity (about 800 charged particles per central Au + Au collision in the CBM detector acceptance) and high hit density are expected in the CBM experiment. This leads to special requirements for the tracking software, which has to perform fast and stable in such an environment. Currently, two different CBM setups are under investigation, one for electron and the other for muon measurements [3]. In the work presented here only the CBM setup for muon measurements is considered. In this case, the CBM setup consists of several detectors including as tracking detectors the silicon tracking system (STS), consisting of 8 silicon microstrip detector stations, located inside a large acceptance dipole magnet for track and vertex reconstruction and momentum determination, and the muon detector (MUCH). The layout of these detectors is still in the process of optimization. For the work presented here, the muon system consists of 6 hadron absorber layers made of iron of variable thickness, of 2 tracking detectors between each of the gaps and of 3 tracking detectors after the last absorber (see Fig. 1). The detector technology is still under discussion. GEM technology is the most promising candidate for regions with high hit density. For the outer parts at lower hit density, straw tubes or MWPCs are under discussion. For the work presented here, the detectors are implemented as GEM detectors with a minimum pad size of 0.28×0.28 cm for the inner regions of the first station and with a maximum pad size of 4.48×4.48 cm for the outer regions of the last station. No detector inefficiency is currently implemented in the simulation. For the measurement of muons from the decay of low-mass vector mesons (ρ, ω, ϕ) muons are required to pass only an iron absorber thickness of 125 cm ($7.5\lambda_I$, with λ_I being the nuclear interaction length), whereas for muons from the decay of charmonia the full absorber length of 225 cm is used (total thickness of $13.4\lambda_I$). This difference in muon identification is related to the fact that muons from low-mass vector measons only have momenta of 1 GeV/c on average, while the mean momentum for muons from charmonia is 6 GeV/c. The shorter absorber length which is required for muons from low-mass vector measons enhances their identification probability, however on account of a larger background.

The speed of the tracking algorithm is extremely important for any analysis of data in CBM as CBM will be a high-rate experiment collecting terabytes of data. In this contribution, a fast parallel track reconstruction algorithm which uses available features of modern processors is presented. These features comprise a SIMD instruction set and multithreading. The first allows one to pack several data items into one register and to operate on all of them in parallel

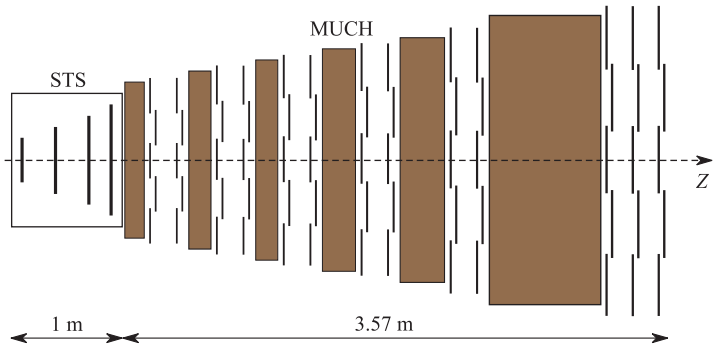

Fig. 1. Sketch of the silicon tracking system (STS) and the muon detector (MUCH)

thus achieving more operations per cycle. The second feature enables the routines to exploit all available CPU cores and hardware threads. The use of this modern CPU technologies allows one to achieve a high calculation speed of the tracking algorithm.

1. TRACK RECONSTRUCTION ALGORITHM

MUCH track reconstruction in CBM is based on track following using reconstructed tracks in the STS as seeds. In the STS track reconstruction is based on the cellular automaton method [4] and STS track parameters are used as starting point for the following track prolongation. This track following is based on the standard Kalman Filter technique [5] and is used for the estimation of track parameters [6] and trajectory recognition.

Track Propagation. The track propagation algorithm estimates the trajectory and its errors in a covariance matrix while taking into account three physics effects which influence the trajectory, i.e., energy loss, multiple scattering and the influence of a magnetic field. The influence of the material on the track momentum is taken into account by calculating the expected average energy loss due to ionization (Bethe–Bloch formula), bremsstrahlung (Bethe–Heitler formula) and direct pair production [7]. The influence on the error, i.e., the covariance matrix due to multiple scattering is included by adding process noise in the track propagation. Here, a Gaussian approximation using the Highland formula [7] is used to estimate the average scattering angle. The propagation of the trajectory is done according to the equation of motion. If the track passes a magnetic field the equation of motion for a charged particle is solved applying the 4th order Runge–Kutta method [8]. If passing a field free region a straight line is used for propagation and the transport matrix calculation. The transport matrix is calculated by integrating the derivatives along the so-called zero trajectory [9]. A detailed description of the developed track propagation can be found in [10, 11].

Track Finding. In the track finding algorithm tracks are prolongated subsequently from one detector station to the next adding additional hits in each detector. For the attachment of hits a validation gate is calculated, according to the chosen probability for rejecting the correct hit. The nearest neighbor approach is used to choose the hit to be assigned to a track, i.e., the algorithm attaches the nearest hit if lying in the validation region at all. After the hit is attached, track parameters are updated with the Kalman Filter. The algorithm allows for 2 missing hits out of 13 in total, taking into account that for each station approximately 4% of the area is covered by detector frames. The allowed number of missing hits will have to be reconsidered when implementing realistic detector inefficiencies. Currently, 2 missing hits are a compromise between efficiency and speed of the tracking algorithm.

Track Selection. After track finding, so-called clone tracks (consisting of a very similar set of hits) and ghost tracks (consisting of a set of hits from different tracks) have to be rejected while keeping correctly found tracks with high efficiency. The selection algorithm works in two steps. First, tracks are sorted by their quality which is defined by the track length and χ^2. Then, starting from the highest quality tracks all hits belonging to a track are checked. In particular, the number of hits shared with other tracks is calculated and the track is rejected if more than 15% of the hits are shared.

2. PARALLELISM: SIMD AND MULTITHREADING

SIMD is short for Single Instruction Multiple Data. It refers to a computing method that enables processing of multiple data with a single instruction. Using a simple summation as an example, the difference between scalar and SIMD operations is illustrated in Fig. 2. With conventional scalar operations, four add instructions must be executed one after another to obtain the sums as shown in Fig. 2, *a*. Meanwhile, SIMD uses only one add instruction to achieve the same result, as shown in Fig. 2, *b*. Requiring fewer instructions to process a given amount of data, SIMD operations yield higher computational speed than scalar operations. The Intel Streaming SIMD Extension (SSE) technology [12] is supported by modern CPUs. A set of SSE Intrinsics allows the use of SSE instructions directly from C++ code. Processors with Intel SSE support have a set of 128-bit registers, each of which may contain four 32-bit single-precision floating-point numbers. SSE is a set of instructions which allows one to load the floating-point numbers to 128-bit registers, performs the arithmetic and logical operations with them and writes the result back to memory. In a C++ program the use of SSE Intrinsics allows one not to care about registers. A `__m128` data type and a set of functions are provided to perform the arithmetic and logical operations. It is up to the C++ compiler to decide which SSE register to use and to make code optimizations. However, C++ code using SSE instructions uses different expressions compared to code with the corresponding scalar instructions, e.g., `c = _mm_add_ps(a, b)` instead of `c = a + b`. Rewriting the code using vector instructions would require to provide support for both, scalar and vector, versions, duplicating modifications, debugging and testing. Therefore, the SSE vector instructions were set in a header file, overloading all operands and inlining several functions [13]. In this way the source code remains the same, and possible changes of the code in the future will be valid for both, scalar and vector, versions. Moreover, overloading the operands provides a more elegant and familiar way to program with SSE using C++.

Multicore processors have made parallel programming more and more mainstream and it is no longer optional. Motivated by this fact the possibilities to use multithreading in the track reconstruction were investigated. For this the Intel Threading Building Blocks (TBB) library [14] was used. TBB is a C++ template library for parallelism that extends C++ by abstracting away thread management and allowing straightforward parallel programming. TBB supports scalable parallel programming, which means that programs using TBB will run on systems with a single-processor core, as well as on systems with multiple-processor cores taking advantage of all available cores.

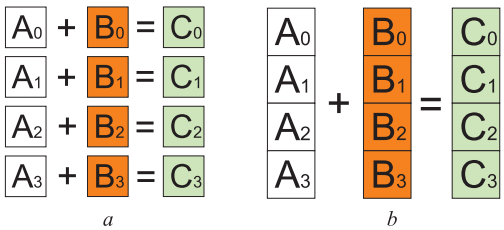

Fig. 2. Comparison between scalar (*a*) and SIMD (*b*) summation. With scalar summation four add instructions are executed one after another, meanwhile, SIMD uses only one add instruction to achieve the same result

3. SPEED-UP OF THE ALGORITHM

Based on the possibilities offered by SIMD and multithreading the tracking algorithm for the MUCH detector in CBM has been modified yielding an increased speed by a factor 100. Changes will be discussed in this paragraph. They include necessary modifications in the track propagation and track fitter in order to SIMDize the algorithm. Also, the track finder algorithm was modified to use a SIMDized track propagator and track fitter and the multithreading capabilities have been implemented to the tracking algorithm.

The Kalman Filter-based track fit is in intensive use in the track reconstruction algorithm. Therefore, its speed is very critical for the track reconstruction. Following the developments of the SIMDized Kalman Filter-based track fit for the STS tracking in CBM [13], several modifications of the Kalman Filter-based track fit algorithm in the MUCH detector have also been investigated here for improved speed. The development is based on a tracking algorithm which has been developed using a double precision scalar version of the Kalman Filter [11]. In the following this is referred to as the «initial scalar version».

Profiling the scalar version of the track propagation procedure it was found that the bottle neck is the access to the field map. The reason is that for the application in CBM the algorithm uses a 70 MB large field map, therefore permanently accesses the main memory. It is obvious, that running on a conventional computer the performance of an algorithm with data located in the main memory is significantly slower comparing to one working with the cache. The magnetic field of the CBM magnet is smooth and can be locally approximated by polynomials. It was found to be sufficient for the track propagator to use a polynomial of the third order to approximate the field in the planes of each station (see Fig. 3). The field behavior between stations is approximated by a straight line or parabola with coefficients calculated from the two or three closest stations, since the field is only needed along the track to be propagated. Track parameters taken with the polynomial approximation of the magnetic field are as precise as those calculated using the full magnetic field map.

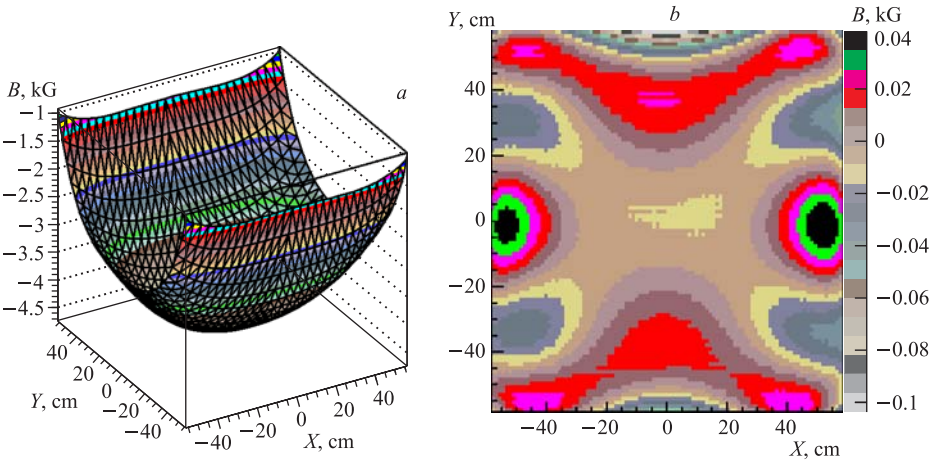

Fig. 3. The most significant magnetic field component B_y in the detector station after the first iron absorber ($z = 125$ cm) calculated using polynomial approximation (a) and the difference between the approximation and the field map (b)

The initial scalar version of the track propagator uses a geometry navigation based on the ROOT geometry package. It is a very precise method for geometry navigation, however, not efficient in terms of calculation speed. First, because it uses the detailed Monte-Carlo detector geometry, currently consisting of 800 000 nodes and probably more in the future implementing more detailed detector layouts. Second, because this algorithm is rather general and not optimized for the CBM setup. Thus, a simplified detector geometry and an optimized geometry navigation algorithm were developed. The simplified geometry is created by converting the detailed Monte-Carlo detector geometry into a reduced one, which only consists of planes perpendicular to the beam direction. The navigation in such a simplified geometry has been significantly optimized.

In the initial serial scalar version of the tracking algorithm [11] tracks are propagated through the detector sequentially one by one, picking up the nearest hits on the detector stations. The use of such an approach does not allow one to apply directly SIMDized track propagation and fitting routines. Thus, the track finder algorithm was significantly changed in order to be SIMDized. In Listing 1 the pseudocode of the modified version of the tracking algorithm is presented. Since all tracks are independent and are propagated by the same algorithm, one can propagate four tracks in parallel after packing the correspondent four scalar track parameters from different tracks into vectors. This reduces the loop size four times.

Finally, multithreading was implemented in the tracking. Since the parallel SIMDized propagation of each four tracks is independent of each other, each such track propagation can be executed concurrently in different threads. The number of threads depends on the number of CPU cores and the number of propagated tracks.

Listing 1: Pseudocode of the parallel tracking algorithm

```
for (number of station groups) {
    parallel_for (number of tracks / 4) {
        pack track parameters;
        SIMDized propagation of 4 tracks in parallel through the absorber;
    }
    for (number of stations) {
        parallel_for (number of tracks / 4) {
            pack track parameters;
            SIMDized propagation of 4 tracks in parallel to the station;
            attach the nearest hit to the track;
        }
    }
}
```

Some other neccesary computational optimizations have been implemented, like the replacement of the matrix operations in the Kalman Filter by explicit calculations only on nontrivial matrix elements. For better performance most of the loops have been unrolled and branches (`if`, `continue`, `break`, etc.) have been eliminated.

Optimization of Combinatorics. The hit-to-track assignment procedure in the track finding algorithm was optimized. The simplest way is to loop over all hits in the detector station, and for each hit update the track parameters with the Kalman Filter method, calculate the validation gate and check whether the hit falls into the validation gate or not. However, these loops are very time-consuming, and moreover, most of them are running without success

as no hit is attached to the track. To solve this issue a fast search of hit-candidates has been developed. The method is executed before the described loop and selects a certain range of hits corresponding to the most probable track position for the more precise check performed afterwards.

4. PERFORMANCE OF THE PARALLEL TRACKING ALGORITHM

Performance of the SIMDized Track Fitter. The quality of the track fit is evaluated using two sets of values:

1. *Residuals* which show the deviation between the simulated and the estimated values:

$$\Delta x = x_{\mathrm{MC}} - x_{\mathrm{reconstr}},$$

x_{MC} is the true Monte-Carlo track parameter and x_{reconstr} is the estimated one;

2. *Pulls (or normalized residuals)* which evaluate the correctness of the error propagation:

$$\mathrm{pull}_x = \frac{x_{\mathrm{MC}} - x_{\mathrm{reconstr}}}{\sigma_x},$$

σ_x is the value from the propagated covariance matrix. Therefore, assuming a correct prediction, pulls should be distributed with a variance of 1 around the mean value 0.

The algorithm has been tested using simulated tracks which have been transported through the CBM setup using GEANT3 [15]. First, only muons from the primary vertex have been investigated: $5 \cdot 10^4 \mu^+$ and $5 \cdot 10^4 \mu^-$ with a flat distribution in momentum p ($p = 2.5{-}25$ GeV/c), polar angle θ ($\theta = 2.5{-}25^\circ$) and azimuthal angle ϕ ($\phi = 0{-}360^\circ$).

The widths of Gaussians fitted to the residual and pull distributions are presented in Table 1. Some distributions are shown in Fig. 4. All pulls are centered at zero indicating that there is no systematic shift in the estimated track parameter values. The pull distributions are well fitted by a Gaussian with small tails caused by various non-Gaussian contributions to the fit. The q/p pull shows slightly underestimated errors as a result of approximations made in the material treatment of the track fitting procedure. The residual and pull values are similar for the scalar and SIMDized versions of the track fitter.

Table 2 shows the computational times and speed-up factors for the track fitter algorithm in the muon detector for two computers: 1) `PC1` with an Intel Core 2 Duo processor at 2.26 GHz, and 2) `PC2` with $2 \times$ Core i7 processors at 2.66 GHz. For `PC1` the 60 times acceleration is achieved by the substantial optimization of the algorithms described above.

Table 1. **Widths of the residual and pull distributions fitted by a Gaussian shape for the SIMDized track fitter algorithm at the last MUCH station behind the whole absorber**

x, cm	y, cm	t_x	t_y	q/p, GeV^{-1}
Residuals				
0.40	0.38	0.0073	0.0073	0.0041
Pulls				
1.0	0.99	1.08	1.09	1.53

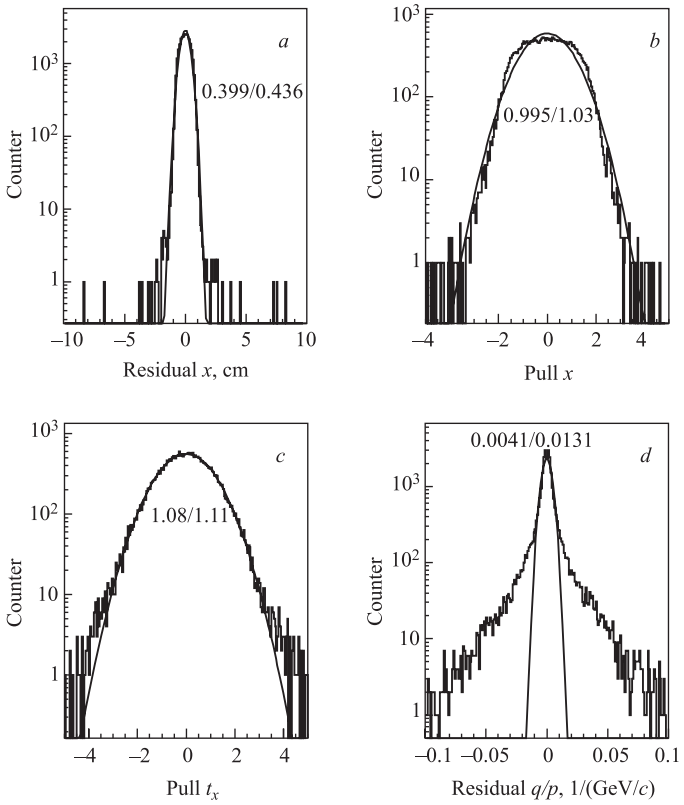

Fig. 4. Selected performance distributions for the SIMDized track fitter algorithm at the last MUCH station: *a*) x residual; *b*) x pull; *c*) t_x pull; *d*) q/p residual. Numbers show the width of a Gaussian fitted to the distributions (first number) and the root mean square value as second number

For PC2 the gain is even more — 101. Changing the data representation from the scalar to the vector (SIMD) format results in a 3 times faster algorithm. Running several threads of the SIMDized track fitter routine in parallel gaines speed up of 1.8 for the 2 core PC1 and 8.8 for the 8 core PC2. In total, the speed-up factor of 324 (from 1100 to 3.4 μs/track) is achieved for the PC1 computer and a factor of 2400 (from 1200 to 0.5 μs/track) for the PC2 computer.

Performance of the Track Finder. In order to test the parallel tracking algorithm, central Au + Au collisions at $25A$ GeV beam energy were simulated with UrQMD [16] and

Table 2. **Speed-up of the track fitter algorithm**

	PC1 with Core 2 Duo		PC2 with 2×Core i7	
	Time/track, μs	Speed-up	Time/track, μs	Speed-up
Initial	1100	—	1200	—
Optimization	18.4	60	13	92
SIMDization	6.2	3	4.4	3
Multithreading	3.4	1.8	0.5	8.8

Table 3. **Speed-up of the track finder algorithm**

	PC1 with Core 2 Duo		PC2 with 2×Core i7	
	Time/track, μs	Speed-up	Time/track, μs	Speed-up
Initial	636	—	730	—
Optimization	10.2	62	7.2	101
SIMDization	6.4	1.6	4.9	1.5
Multithreading	4.2	1.5	1.5	3.3

propagated through the CBM setup using GEANT3 [15]. These events were used to estimate the background in which the interesting signal, i.e., muons from the primary vertex were embedded. In order to enhance statistics $5\mu^+$ and $5\mu^-$ were embedded in each event at the primary vertex. Compared to the average multiplicity of charged tracks in the acceptance (appr. 800) they do not distort the overall conditions. Hits were calculated from the MC information using semirealistic detector response. The above-described parallel tracking routines were then tested on this dataset. Reconstructed tracks in the MUCH detector were required to pass through the whole iron absorber of 2.25 m length. This is the condition which would be used for J/ψ reconstruction. The comparison of the initial serial scalar version and the parallel version of the tracking shows that on account of a minor loss in the momentum integrated track reconstruction efficiency (from 94.7 to 94.0%) the parallel tracking algorithm is much faster (see Table 3). A speed-up of 62 is achieved for PC1 by optimization of the algorithm and the use of the optimized Kalman Filter routines. For PC2 the speed-up is even more 92. SIMDization and multithreading give less speed-up factors than for the track fitter, which can be explained that some parts of the track finder are not completely SIMDized and parallel. In total, the parallel tracking algorithm is 151 times faster than the initial serial scalar algorithm (from 636 to 4.2 ms/event) for the PC1 computer and 487 times faster (from 730 to 1.5 ms/event) for the PC2 computer.

SUMMARY

Next generation experiments as the CBM detector planned at FAIR aim at data recording rates not experienced so far. In order to perform efficient analysis, fast tracking algorithms are essential. For CBM a parallel tracking algorithm for the muon detector was developed based on track seeds from the track reconstruction in the silicon tracking system located inside a dipole magnet. The algorithm uses two features of modern CPUs: a SIMD instruction set and multithreading. The comparison between the parallel tracking algorithm and the initial serial scalar one shows that on account of a minor loss in track reconstruction efficiency (from 94.7 to 94.0%) the parallel tracking algorithm is 487 times faster than the initial one (from 730 to 1.5 ms/event) for a computer with 2×Intel Core i7 processors at 2.66 GHz. The use of multithreading and SIMD will be much more obvious in the future with the next generation of many-core processors.

REFERENCES

1. Compressed Baryonic Matter Experiment. Technical Status Report. 2005. http://www.gsi.de/documents/DOC-2005-Feb-447-1.pdf
2. *Höhne C., Rami F., Staszel P.* // Nucl. Phys. News. 2006. V. 16, No. 1. P. 19–23.

3. *Höhne C.* // J. Phys. G: Nucl. Part. Phys. 2008. V. 35, No. 104160. 5 p.; doi: 10.1088/0954-3899/35/10/104160
4. *Kisel I.* Event Reconstruction in the CBM Experiment // Nucl. Instr. Meth. A. 2006. V. 566. P. 85–88.
5. *Kalman R.* A New Approach to Linear Filtering and Prediction Problems // Transactions of the ASME: J. of Basic Engin. Ser. D. 1960. V. 82. P. 35–45.
6. *Fruhwirth R.* Application of Kalman Filtering to Track and Vertex Fitting // Nucl. Instr. Meth. A. 1987. V. 262. P. 444–450.
7. *Amstel C. et al.* The Review of Particle Physics // Phys. Lett. B. 2008. V. 667.
8. *Press W. et al.* Numerical Recipes: The Art of Scientific Computing. Cambridge Univ. Press, 2007.
9. *Fruhwirth R. et al.* Data Analysis Techniques for High-Energy Physics. Cambridge Univ. Press, 2000.
10. *Lebedev A., Ososkov G.* LIT Track Propagation for CBM. CBM Note. Darmstadt, 2008; https://www.gsi.de/documents/DOC-2008-Dec-182-1.pdf
11. *Lebedev A. et al.* Track Reconstruction and Muon Identification in the Muon Detector of the CBM Experiment at FAIR // PoS. 2008. V. ACAT08, No. 068.
12. Intel® 64 and IA-32 Architectures Software Developer's Manual. V. 1: Basic Architecture. 2009.
13. *Gorbunov S. et al.* Fast SIMDized Kalman Filter Based Track Fit // Comp. Phys. Commun. 2008. V. 178. P. 374–383.
14. Intel® Threading Building Blocks. http://www.threadingbuildingblocks.org/
15. GEANT — Detector Description and Simulation Tool. CERN Program Library Long Writeup W5013.
16. *Bass S. A. et al.* // Prog. Part. Nucl. Phys. 1998. V. 41. P. 255–370.

Received on November 26, 2009.