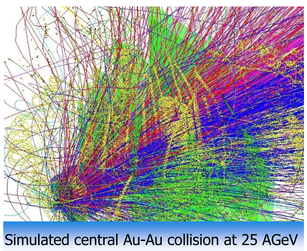
# Parallel Algorithms for Track Reconstruction in the CBM Experiment

I. Kisel, I. Kulakov*, M. Zyzak
(for the CBM Collaboration) * I.Kulakov@gsi.de

## Tracking Challenge in the CBM Experiment



Simulated central Au-Au collision at 25 AGeV

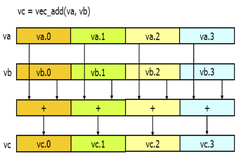
- Fixed-target heavy-ion experiment
- 1000 charged particles/collision
- Non-homogeneous magnetic field
- 85% fake combinatorial space points in STS
- $10^7$ events per second
- Full event reconstruction is required in the first trigger level

- CBM event reconstruction is based on Kalman Filter and Cellular Automaton algorithms
- High speed and efficiency of the reconstruction algorithms are required
- The algorithms have to be highly parallelized and scalable in order to use full potential of the modern processors

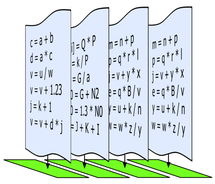
## Parallelization

Computer architectures :

- SISD - Single Instruction, Single Data stream
- SIMD - Single Instruction, Multiple Data streams
- MIMD - Multiple Instruction, Multiple Data streams

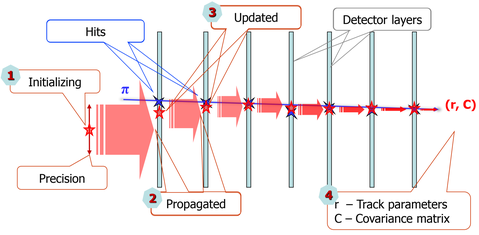



SIMD parallelization
(Data level parallelism)



MIMD parallelization
(Task level parallelism)

## Kalman Filter (KF) Based Track Fitter


CBM track model:

$r = \{ x\ y\ t_x\ t_y\ q/p \}$

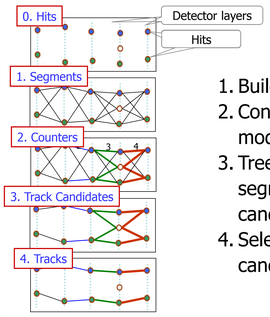
x, y – coordinates
$t_x$ - slope dx/dz*
$t_y$ - slope dy/dz
q – charge
p – momentum
*z axis is along the beam line

Track fit:

1. Start with an arbitrary initialization
2. Add one hit after another
3. Improve the state vector
4. Get the optimal parameters after the last hit

- Highly optimized code with respect to time
- Single precision
- Highly parallelized code

## Cellular Automaton (CA) Track Finder




Cellular Automaton:

1. Build short track segments
2. Connect according to the track model
3. Tree structures appear, collect segments into track candidates
4. Select the best track candidates

- Kalman filter for track segments fit
- Highly optimized code with respect to both efficiency and time
- Highly parallelized code
  - Data level (SIMD instructions, 4 single-precision floating point calculations in parallel)
  - Task level (ITBB, parallelization between cores)

## KF Library

Tracking tools:

- KF track fitter
- KF track smoother
- Deterministic Annealing Filter

KF approaches:

- Conventional KF
- U-D-Filtering
- Square root KF
  - Carlson
  - Potter

Track propagation:

- Runge-Kutta
- Analytic formula

Parallelization:

data level:

- Headers
- Vc
- ArBB

task level:

- ITBB
- ArBB
- Open MP

Precision:

- single
- double

| residuals ($\rho = r^{reco} - r^{mc}$) | | | | |
|---|---|---|---|---|
| x, μm | y, μm | $t_x$, $10^{-3}$ | $t_y$, $10^{-3}$ | p, % |
| 43 | 39 | 0.3 | 0.25 | 0.93 |

Track fit; conventional; analytic formula; single precision

High track parameters resolution at all stations

## SIMDization with C++ Header Files

Header files overload SIMD instructions implementing all operands and inlining basic arithmetic and logic functions, that makes a code compact and easily readable.

Overloading of + and * operators, example:

```
friend fvec operator+(const fvec &a, const fvec &b) {
  return _mm_add_ps(a,b); }
friend fvec operator*(const fvec &a, const fvec &b) {
  return _mm_mul_ps(a,b); }
```

Line function, example:

w/o headers

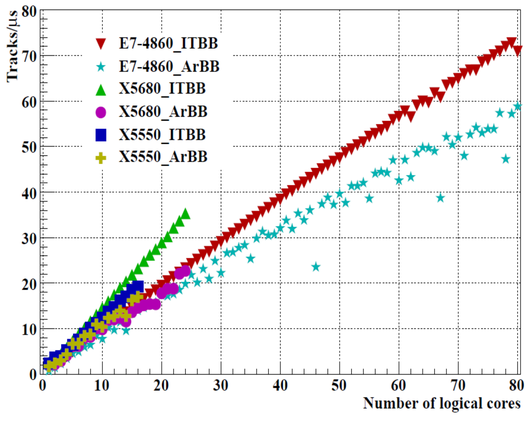
```
__m128 y = _mm_add_ps(_mm_mul_ps(a,x),b);
```

with headers:

```
fvec y = a*x + b;
```

a SIMDized code with the headers is as simple as a scalar code

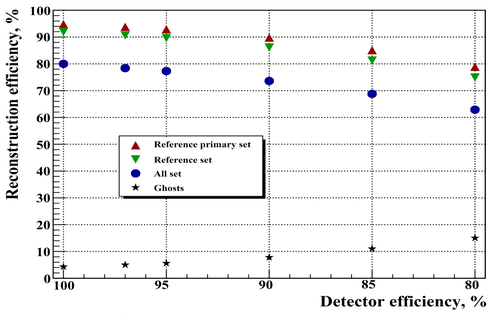
## Kalman Filter Many-Core Scalability




Track fit; conventional; analytic formula; single precision

Strong scalability on many-core systems

## Track Reconstruction Efficiency




All set: p ≥ 0.1 GeV/c
Reference set: p ≥ 1 GeV/c
Ghost: purity < 70%

Reconstructable track:
Number of consecutive MC points ≥ 4

Au+Au central events at 25 AGeV, 8 STS, geometry with 0 & 15 strip angles.

- High reconstruction efficiency
- Stable w.r.t. the detector inefficiency
- 199 ms/event
- Strong scalability on many-core systems

## Conclusions

- ✓ Reconstruction in the CBM experiment is based on the Kalman Filter and Cellular Automaton algorithms
- ✓ The algorithms are effective and fast (up to 13 ns/track/node for KF track fitting and up to 200 ms/event/core for CA track finding)
- ✓ The algorithms are highly parallelized both on data and on task level.
- ✓ The algorithms scales linearly with number of cores.