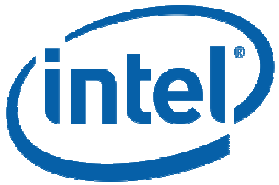


Parallelization of the SIMD Kalman Filter for Track Fitting

Rama Malladi

R. Gabriel Esteves

Ashok Thirumurthi

Xin Zhou

Michael D. McCool

Anwar Ghuloum

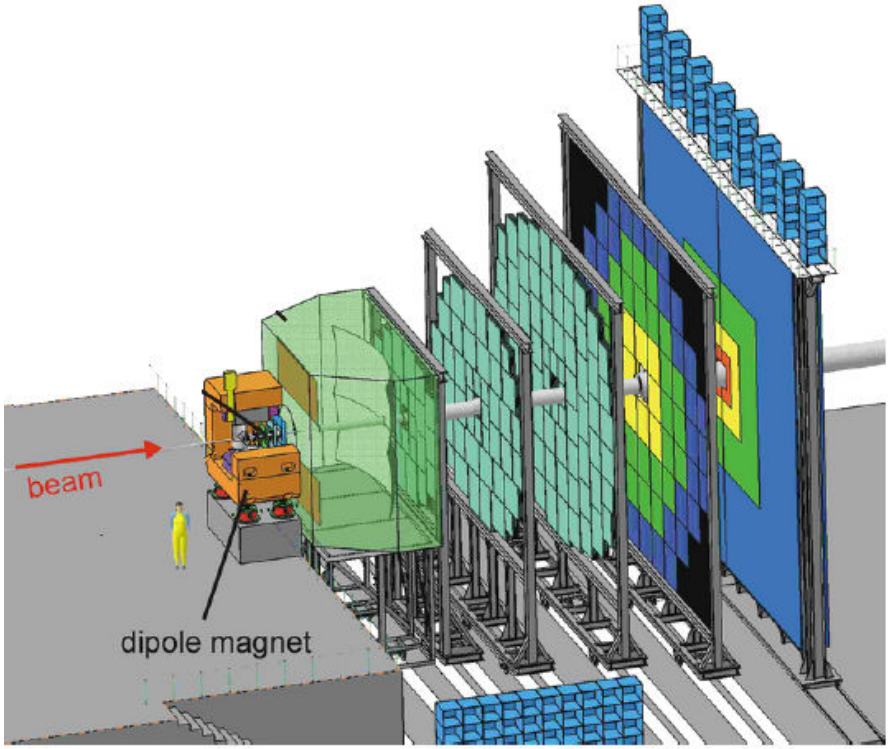
Outline

- Track fitting: introduction and motivation
- Kalman filter-based track fitting
  - Algorithm
  - Optimization
- Ct implementation
  - Results
  - Future work

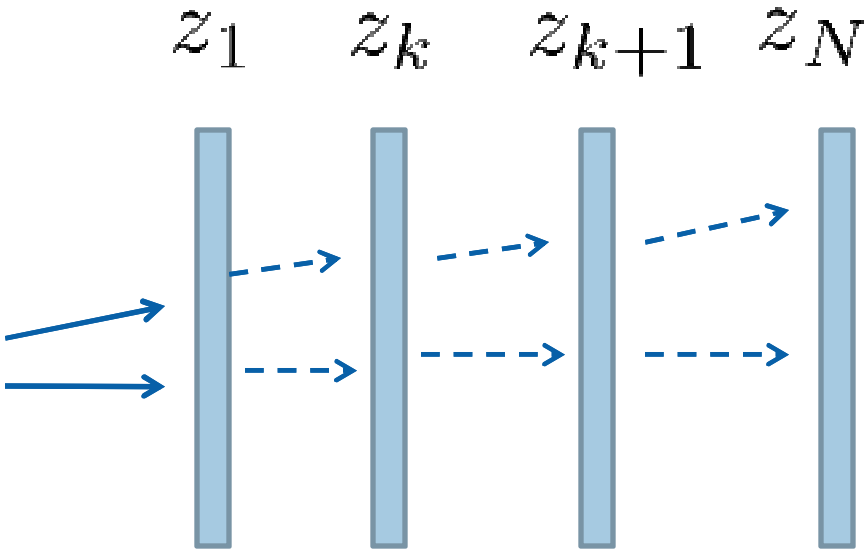
Goal of HEP experiments

- Simulate extreme conditions of matter
  - Experiments involving particle collisions
- Particles are subject to several physics processes:
  - Energy loss
  - Multiple Coulomb scattering
  - Magnetic field
- Measurements taken along the trajectory (noisy)
- Data analysis:
  - **Track finding:**
    - Associating a subset of measurements to each particle.
  - **Track fitting:**
    - Computing the state of the particle at each measurement in the subset.

Track fitting: Measurement process



From Friese et al "CBM experiment at FAIR"



Track fitting: Problem statement

- Given a track (sequence of observations z_k) → determine the state of the particle at each observation point.
- Measurement varies with each experiment
- State is usually given by tuple of values:

$$r_k = (x, y, t_x, t_y, q/p)$$

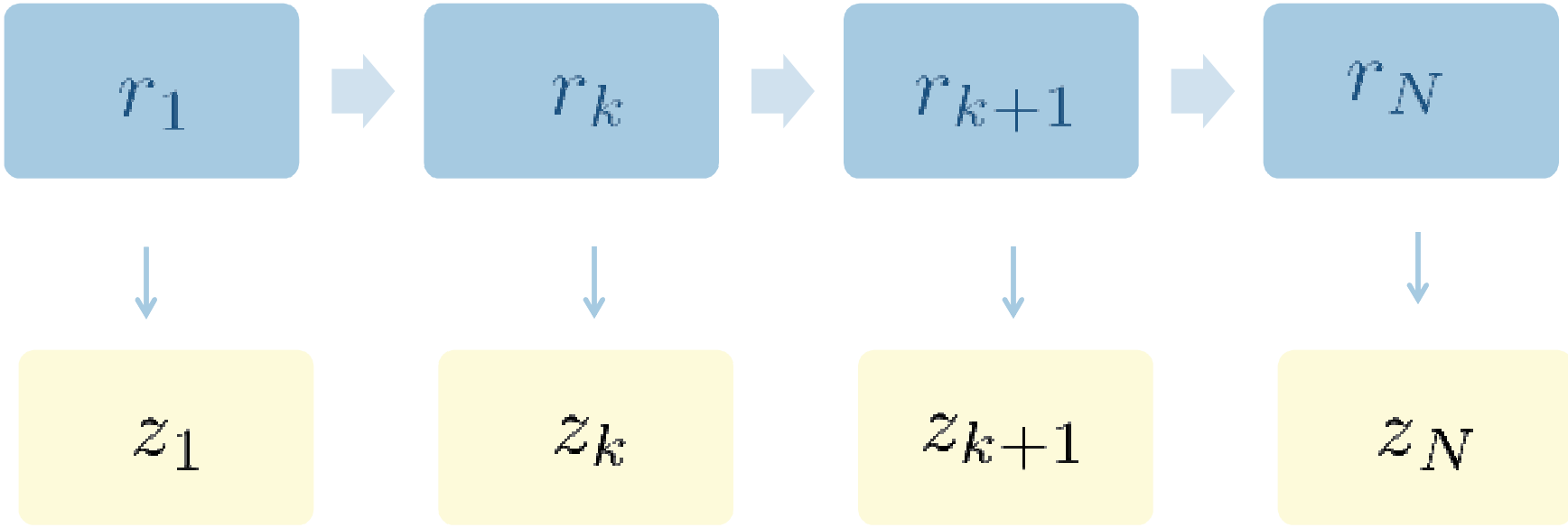
$$r_k \in \mathbb{R}^m$$

Progressive methods: Kalman filter

- Proposed by Rudolf Kalman (1960)
- Recursive least-squares formulation, based on system's state-space representation
- Applied to track fitting independently by Früwirth and Billoir
- Used to analyze data from LHCb, CBM, most detectors

State-space formulation

Hidden (unobserved/unobservable) process generates the observed sequence of observations:



Kalman filter (KF): Assumptions

- Independence structure: Markovian assumption

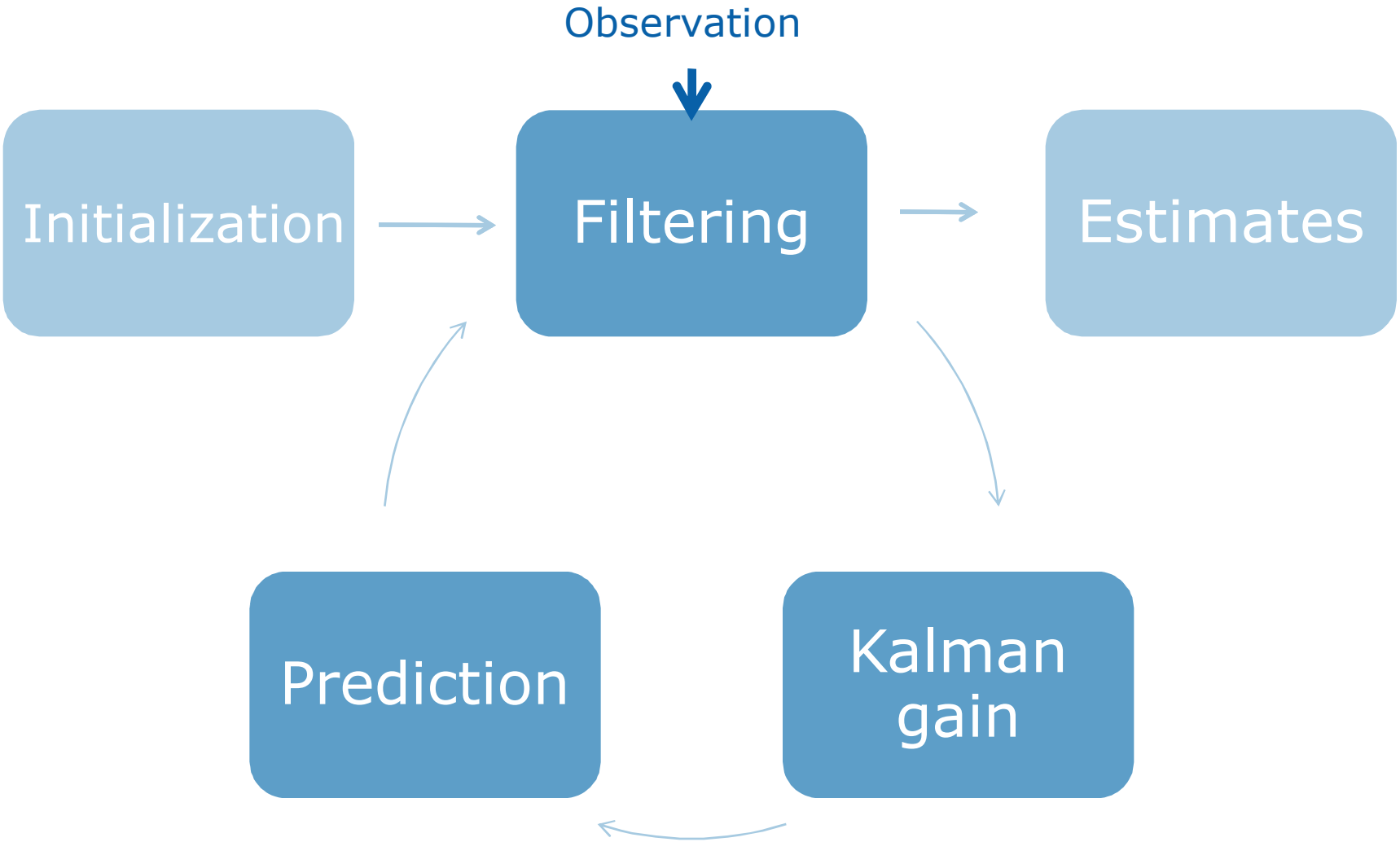
$$P(r_k|r_{1:k-1}) = P(r_k|r_{k-1})$$
$$P(z_k|r_{1:k}, z_{1:k}) = P(z_k|r_k)$$

- Dynamical system formulation: Process/measurement eqs.

$$r_k = Fr_{k-1} + \epsilon_k$$
$$z_k = Hr_k + \nu_k$$

$$\epsilon_k \sim \mathcal{N}(0, R) \qquad \nu_k \sim \mathcal{N}(0, Q)$$

KF: Sequentially incorporate observations



KF: Prediction/correction

- Prediction
  - Geometrical/physical considerations
    - Movement of particle in magnetic field
  - Material considerations
    - Energy loss
    - Multiple Coulomb scattering
- Correction
  - Bayesian update

KF: Prediction/correction

- Prediction

$$\hat{r}^p_{k+1} = F\hat{r}_k$$
$$P^p_{k+1} = FP_kF' + R$$

- Kalman gain update

$$K_{k+1} = \tilde{P}_{k+1}H'(H\tilde{P}_{k+1}H' + Q)^{-1}$$

- Correction

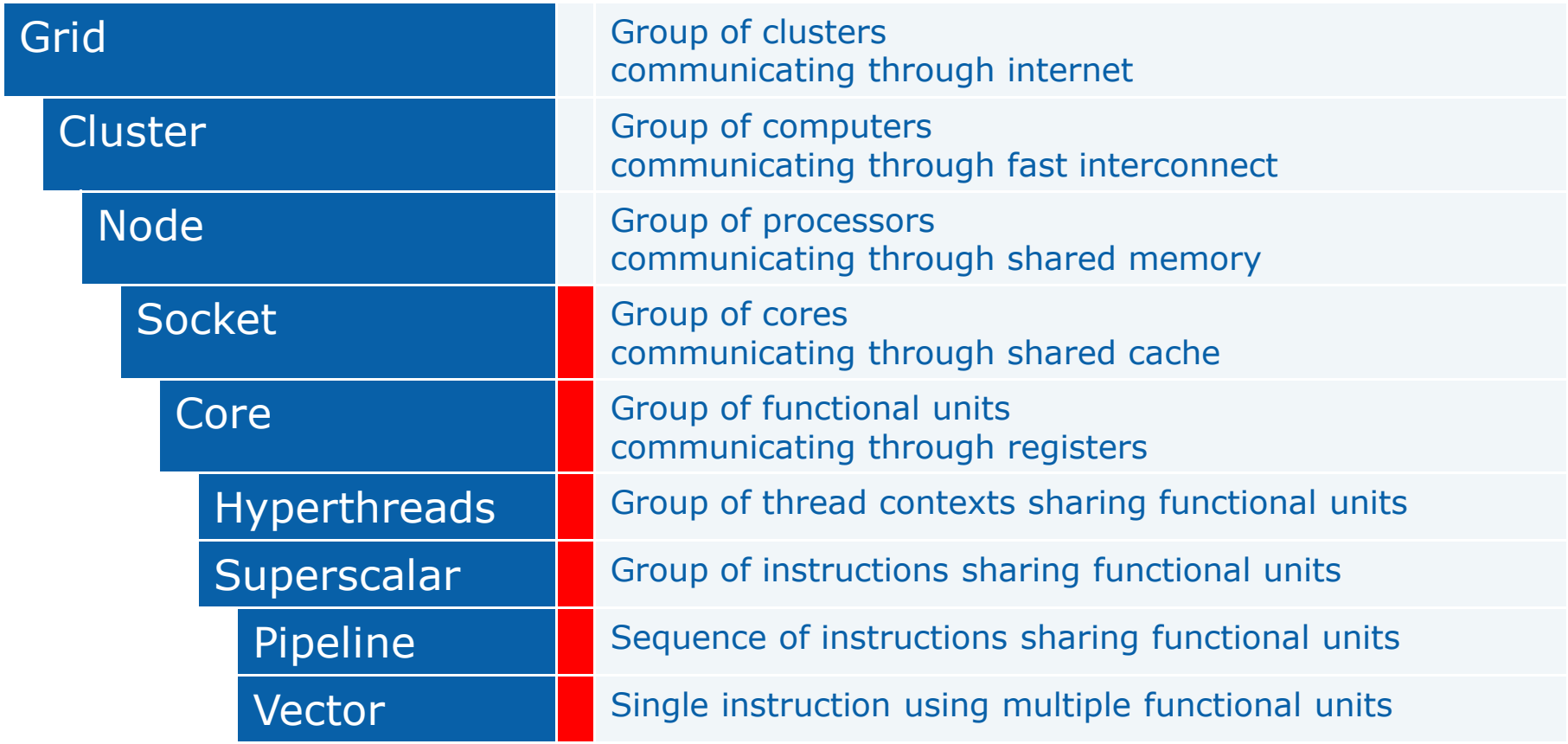
$$\hat{r}_{k+1} = \hat{r}^p_{k+1} + K_{k+1}(z_k - H\hat{r}^p_{k+1})$$
$$P_{k+1} = (I - K_{t+1}H)P^p_{k+1})$$

KF: Optimization

- Physical
  - Initial model (homogeneous/non-homogeneous magnetic field)
  - Polynomial approximation of the magnetic field
- Numerical
  - Want to use single instead of double precision for performance
  - Still need to assure numerical stability (→ UD factorization)
- Implementation
  - Take advantage of modern hardware features: multicore + vectors
  - Obvious parallelization opportunity: over tracks

Consistent with approach described in Cell-based implementation (Gorbunov et al, 2007)

Levels of parallelism



Ct: C for Throughput

- Dynamic run-time environment
  - Managed data/code
  - Dynamic code generation
  - Autovectorization
- Domain-specific language embedded in C++

Ct implementation

Program structure

- Prolog
  - Copy (user) data into Ct-controlled space
  - Convert to structure of array (SoA) form for efficiency
- Compute
  - Define and invoke compute kernels that work on Ct types
  - Parallelized over multiple tracks with the same number of measurements
- Epilog
  - Copy data back into user space
  - Converts into array of structure (AoS) form for convenience

Performance measurements

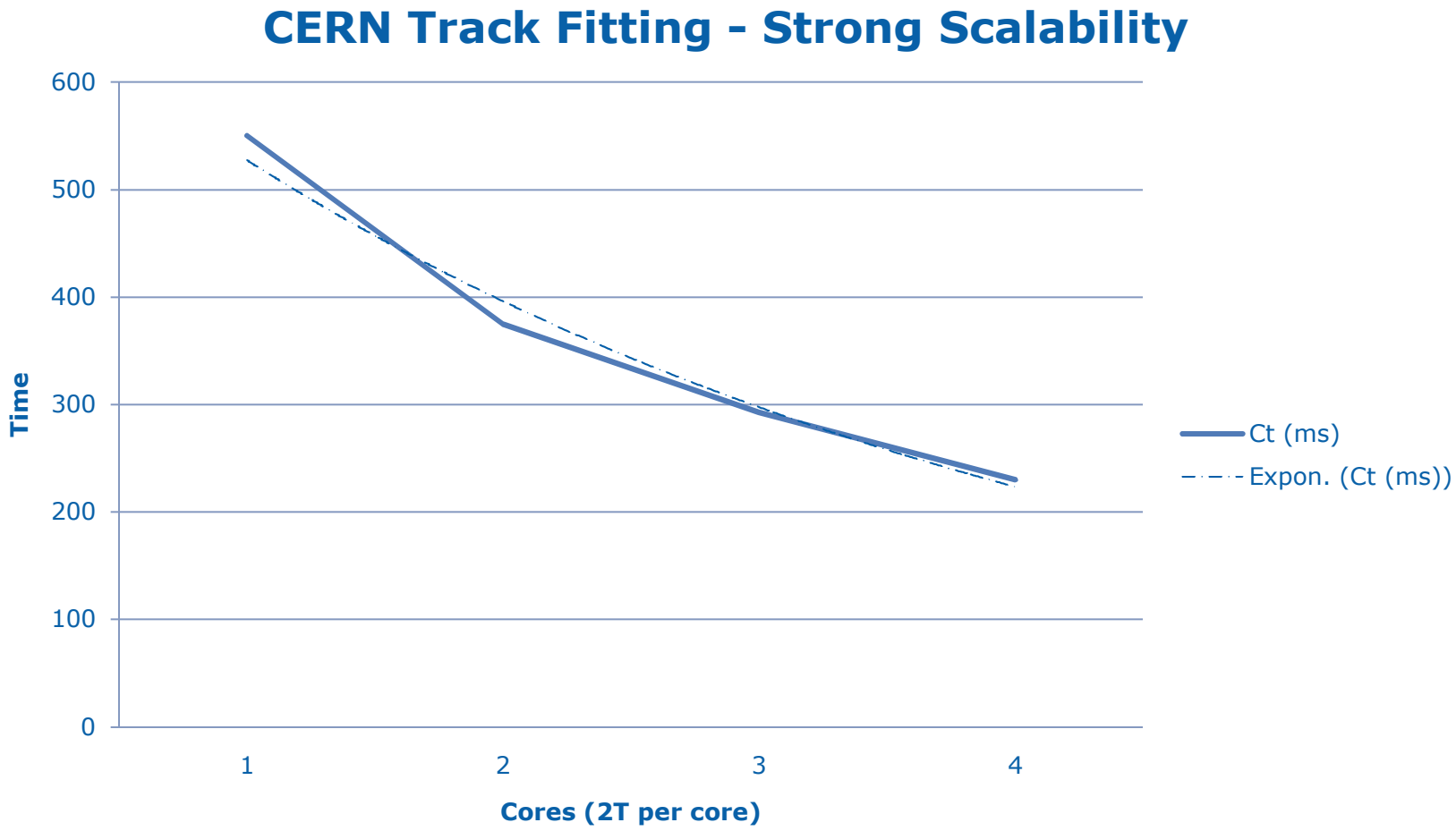
- Run-time compilation time ***excluded*** (can be amortized)
- SoA and AoS conversion time is ***included***

Ct: Kernel invocation/definition

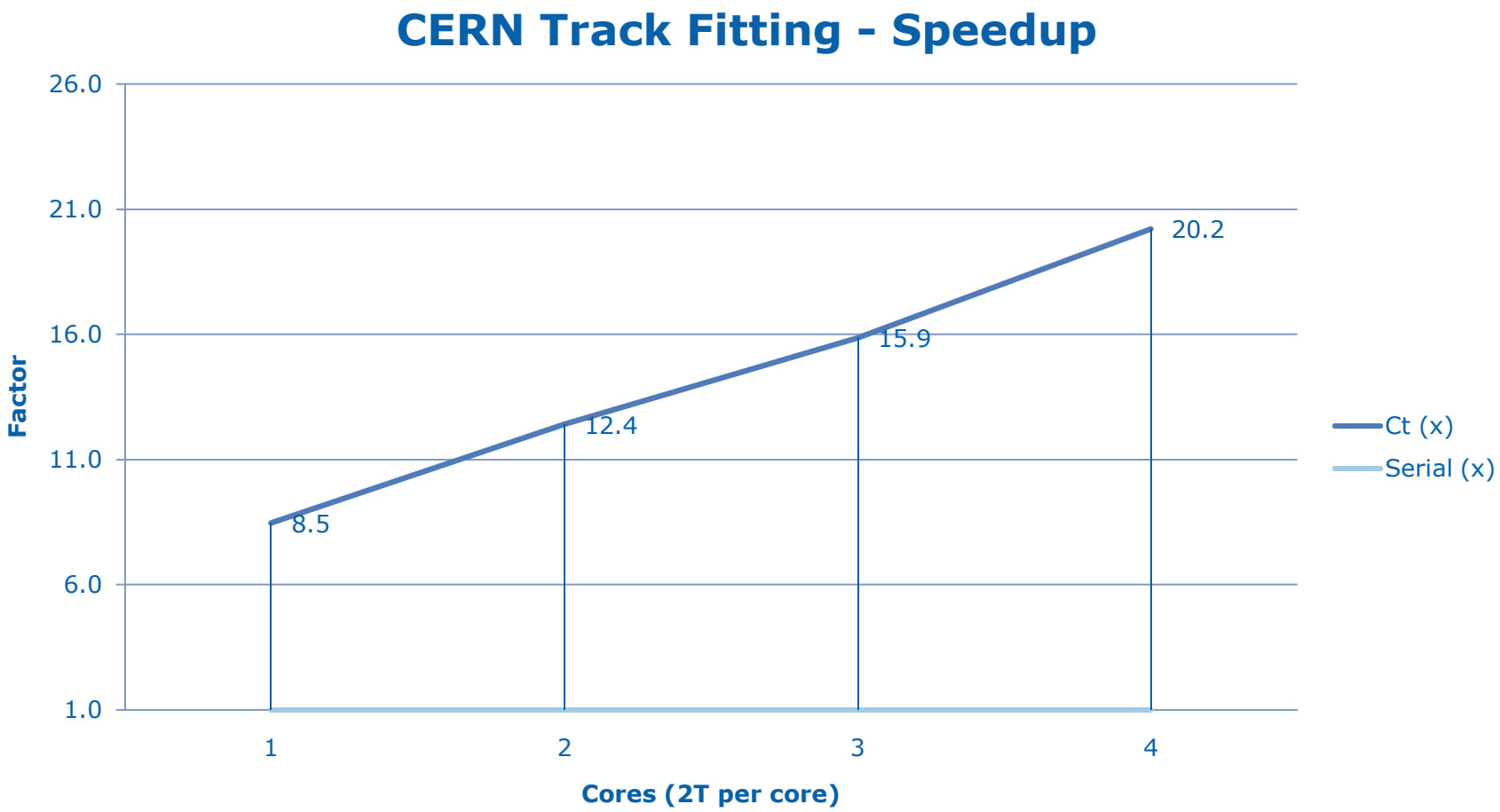
```
rcall(fitTracksCtEntry)(vSize, structTracks, structStations, vtT, vtC);
```

```
_for( i = N - 1, i >= 0, i-- ){
        // magnetic field setup
        filter( ts, ss, xInfo, ts.hitsX2.row( i ), w, T, C );
        filter( ts, ss, yInfo, ts.hitsY2.row( i ), w, T, C );
        for( int j = 0; j < 3; j++ ){
            H2[j] = H1[j];
            H1[j] = H0[j];
        }
        z2 = z1;
        z1 = z0;
    }_endFor;
```

Strong Scalability, with Hyperthreading



Strong Scalability with Hyperthreading



Future work

Parallelism allows use of more powerful algorithms:

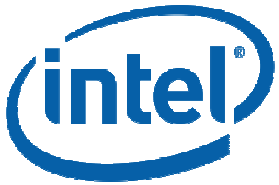
- Gaussian sum filter [Früwirth et al 1997]
  - Non-Gaussian noise; use parallel bank of Kalman filters
- Particle filters (aka Sequential Monte Carlo, Doucet et al, 2001)
  - Non-linear dynamics; naturally parallel

Explore other parallelisation strategies:

- Pipeline strategy if applicable

Integration with C++ frameworks:

- Ct allows construction of reusable frameworks
- Ct allows development of application-specific languages
- Explore parallelization of entire frameworks like `genfit` instead of experiment-specific kernels



Q/A

References

- Rudolf Früwirth: “Track fitting with non-Gaussian noise” , 1997
- Gorbunov et al: “Fast SIMDized Kalman filter based track fit”, Computer Physics Communications 178(5), 2008
- Doucet et al.: Sequential Monte Carlo Methods in Practice, Springer, 2001

Backup

Experimental setup

- Baseline characterization
  - Single-precision
  - UDKF
- Test platform

CPU	1 @ 4-core Intel Core i7-920 @ 2.67 GHz	6 GB RAM
OS	64-bit Windows Server 2008	
Compiler	icl.exe v11.1.054	