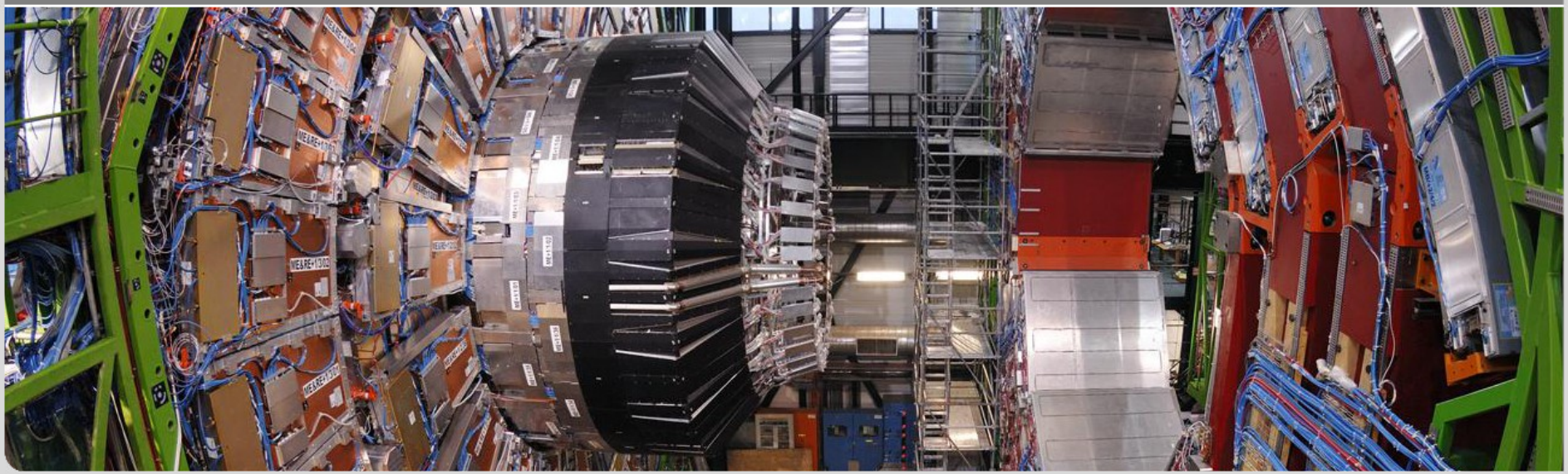
Status and future plans for software technology demonstrators in CMS

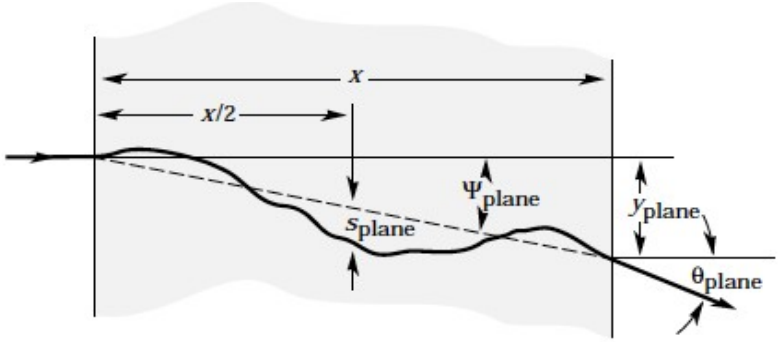
Forum on Concurrent Programming Models and Frameworks, 10.10.2012

Thomas Hauth, Danilo Piparo, Vincenzo Innocente



Using OpenCL for CMS Algorithms

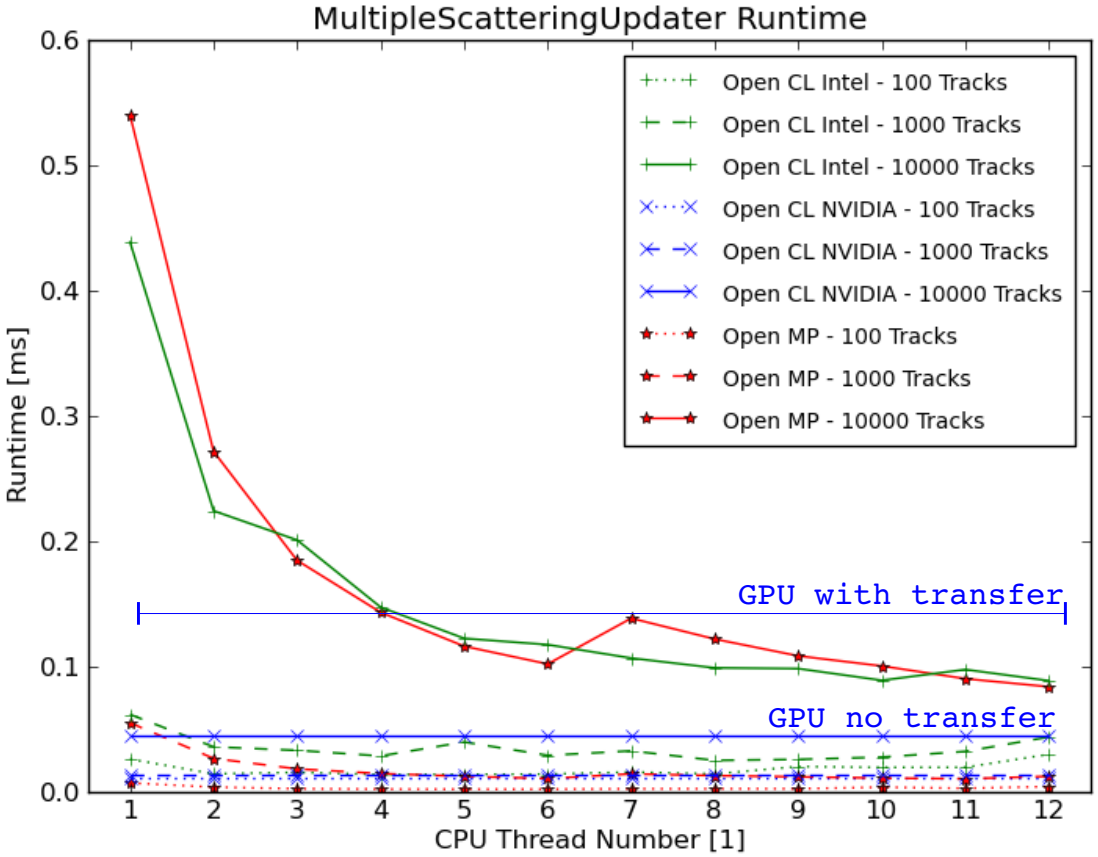
- The multiple scattering algorithm from the CMS track reconstruction was picked from the CMSSW source code[1] and run in an stand-alone application using OpenCL:
  - Calculate the maximum scattering angle of a particle passing through a material layer
  - Implementation of the Highland formula for multiple coulomb scattering
- OpenCL verision of this algorithm is not intended for insertion in CMS production code but served as a “sandbox” to test OpenCL with CMS input data in an isolated fashion
- Called several times during the reconstruction of a single track
- Useful figure: 500 to 1000 Tracks during the 2012 run period
- In terms of mathematical operations:
  - Multiplications, divisions, sums and a logarithm.
  - I/O: 4 double precision floating points in, 3 of them out.
  - About 40 lines of code, 1 branching
  - Same source is run on the CPU and the GPU
- Reference implementations were created using OpenMP and TBB to be able to compare the OpenCL results



[1] TrackingTools/MaterialEffects/interface/MultipleScatteringUpdator.h

OpenCL Performance Results

- The algorithm has been computed for various amount of input tracks (100, 1.000, 10.000)
- A varying amount of threads have been used on the CPU (Intel Core i7-3930K – 6 cores)
- The measurements on the GPU always use the full device (NVIDIA GeForce GTX 560)

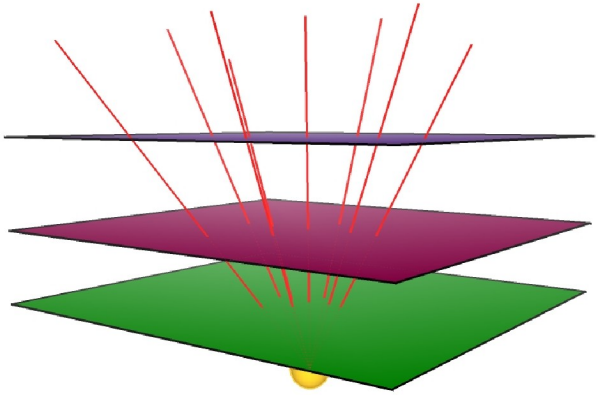


- Transferring the data from and to the graphic cards adds ~0.1ms to the runtime on the GPU
- Reuse of data transferred to the GPU is desirable
- The Intel OpenCL implementation performs as well as the classic OpenMP does
- The same OpenCL kernel code can be also run on the GPU, if available
- For less then 10.000 track input size, the OpenCL scheduling overhead must be considered
- TBB performs equally well as OpenMP (see backup)

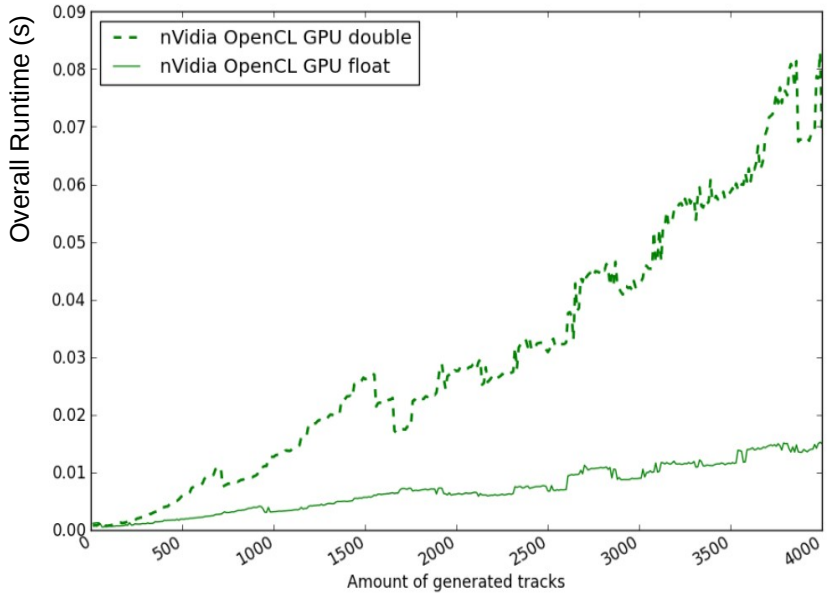
Note: Switching to the new Intel OpenCL SDK 2012 reduced the scheduling overhead of this platform considerably

OpenCL for CMS Tracking

- A summer student project (finished in August 2012) had the goal to start this effort and demonstrate the feasibility of OpenCL for track seeding in CMS
- Students: Grazina Laurinaviciute and Darius Miskinis, both from Vilnius University, Lithuania
- A simplified representation for the geometry of the CMS tracker in OpenCL memory was created
- Various types of track seeding algorithms were investigated on GPU and CPU
- OpenCL kernels were found to be portable between the different devices without changes
- All OpenCL platforms were able to handle the increasing amount of tracks without a significant drop in the time per track
- Building on the initial work of Grazina and Darius, we intend to build a more complete OpenCL tracking technology demonstrator



Simplified visualization of particle tracks crossing the first three detector layer. These three hits are used for the initial seeding.



Double precision floating point on GPU: huge price to pay

Bottom Line: Computation on GPU

- The promises of OpenCL are maintained: the same kernels run smoothly and without modifications on CPUs and GPUs. No-vendor lock-in!
- The same code will run on regular x86-64 CPUs (for end-user analysis), HLT and other dedicated compute farms with GPUs or Intel MICs.

Evaluation for CMS

\+ Fast code execution on a wide variety of platforms

\+ Scales very well with the available hardware

\- Existing CMSSW code must be reimplemented as OpenCL kernels

\- The scheduling overhead and transfer time GPU to CPU must be considered

  - Sending data buffers cumulated over more than one event to OpenCL becomes necessary (implications on the framework)

Parallel Track Seeding

- The Intel Threading Building Blocks library was used to to exploit concurrency within reconstruction algortithms in CMSSW: In this case Triplet Seeding
- The seeding part of the CMS track reconstruction was parallelized with only minor changes to the framework (atomics for ref counting, thread pool beyond module boundaries)

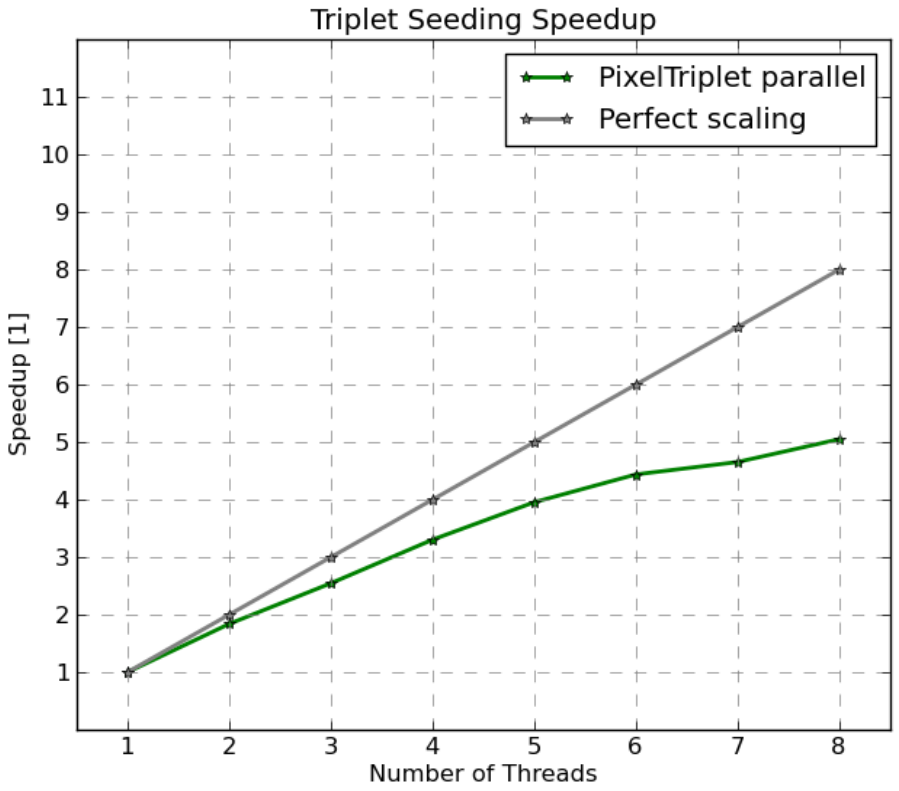
Performance Measurements

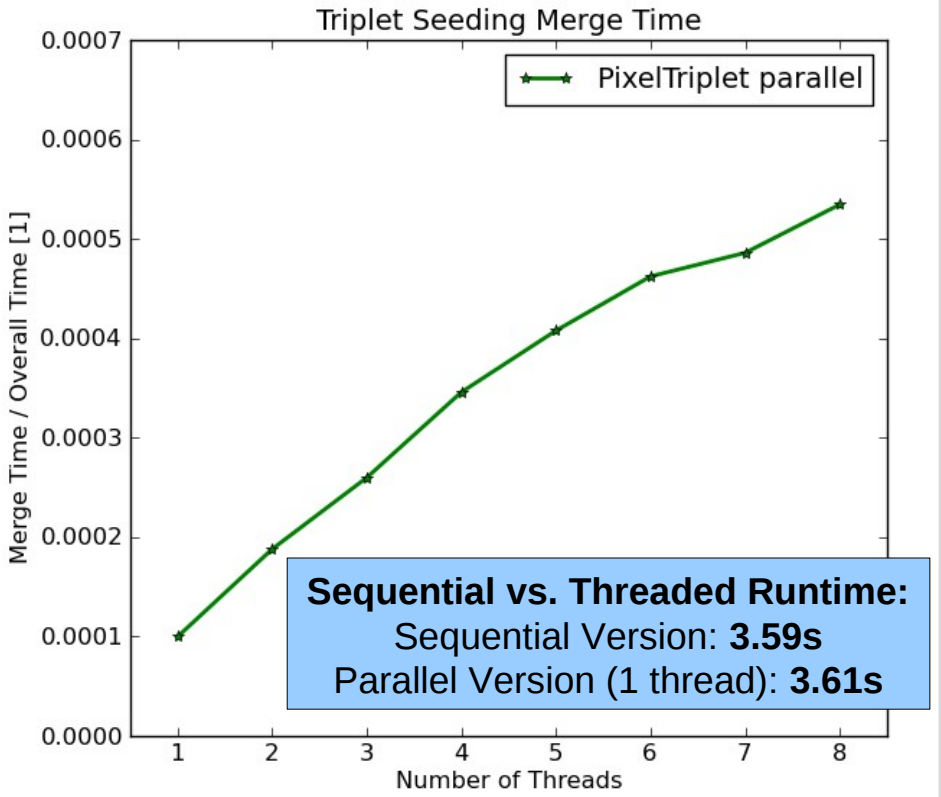
- The full CMS reconstruction chain was run with different numbers of threads
- Input: 50 events of the highest pile-up sample recorded with the CMS detector in 2011
- On average, one event contains ~40 collisions
- Test Setup:
  - Intel(R) Core(TM) i7 CPU X 980 @ 3.33GHz with 6 physical cores (12 HyperThr.)
  - 6 GB RAM
  - Scientific Linux 5.8
  - CMSSW 5.2 official release (with modifications for the multi-threading code)
  - The measurements labeled *Serial* refer to an unchanged version of CMSSW (no TBB Service, no atomic operations)
- The triplet seeding takes about 14% of the runtime in the serial version
- Realistic and complex test setup: Large application, complex data flow in the RECO app etc.

Intel TBB website:
http://threadingbuildingblocks.org/

Triplet Seeding Runtime and Scaling

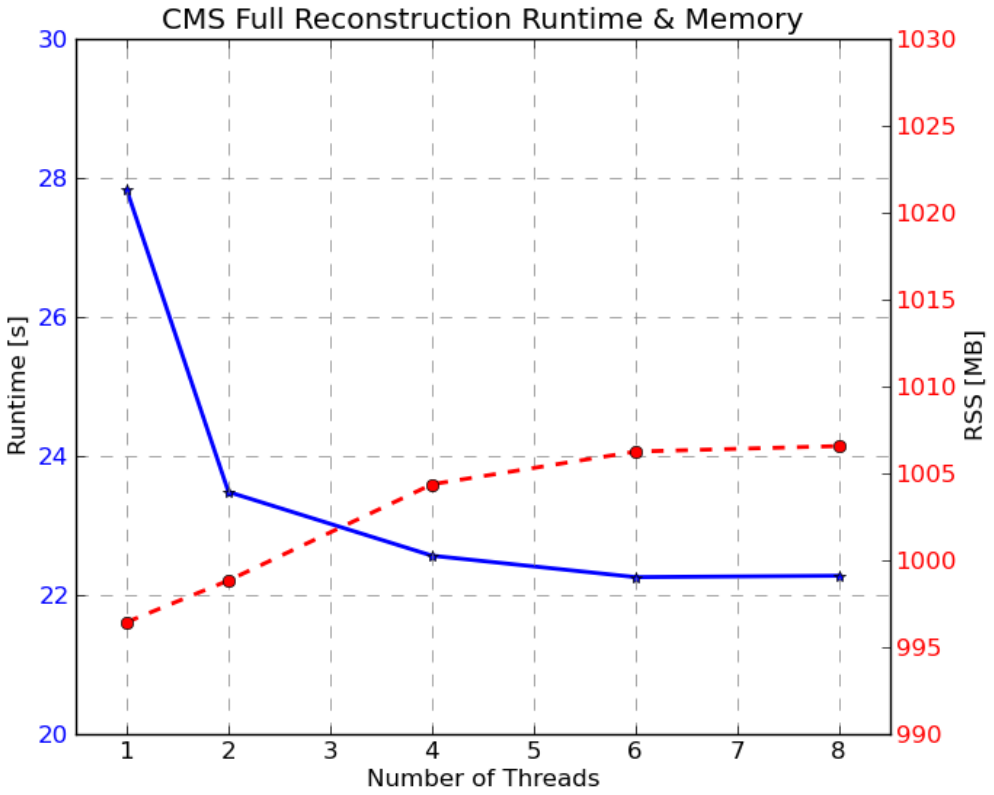
- Good scaling up to five cores
- Compared to the overall runtime of the algorithm, the final merge step only takes about .1 to .3 percent of the triplet seeding time
- This depends on the number of threads: for more threads more work blocks are partitioned
- Full reproducibility achieved, independent of the number of thread





Sequential vs. Threaded Runtime:
Sequential Version: **3.59s**
Parallel Version (1 thread): **3.61s**

CMS Reconstruction Runtime and Memory



- Each thread adds about 1 MB to the overall memory consumption. Negligible compared to the memory footprint of the application (~ 1 GB) > lightweight scaling
- Higher-than-expected scaling from 1 to 2 cores, probably due to the positive effects of using the L1/L2 caches of two cores simultaneously

Bottom Line: Parallel Algorithms

- A multi-threaded track seeding using TBB was implemented within the CMS Software Framework
- Much more than a prototype: Tested and validated in a production environment with actual CMS proton-proton data
- Parallelism within an algorithm is a feasible way to speed-up long-running modules and serial module chains with very low memory overhead (~ 1 MB / Thread)
- This prototype has helped CMS to make an informed decision on the upcoming integration of multi-threading in the framework

Evaluation for CMS

+ Can be applied to existing code with minor changes
+ Prepares our software for next-generation accelerators (Intel MIC)
+ Wide varieties of processing can be run in parallel
  (Tracks, Hits, ...)

Future Steps

- The knowledge gained during the prototyping will be beneficial in the transformation process to a multi-threaded framework
- Once all necessary decisions on the framework changes have been made, time consuming parts of the reconstruction will be modified (namely the rest of the track reconstruction) to be able to run in parallel
  - We learned a lot during the seeding parallelization, especially in terms of reproducibility and thread-safe data access in large applications

- OpenCL on the CPU becomes mature enough to compete with “classical” methods like OpenMP
  - Further development on a fast CMS track reconstruction using GPU and CPU with OpenCL
  - Evaluation of upcoming hardware platforms (Intel MIC, AMD Bulldozer) in light of OpenCL and CMS

Note: Intel TBB 4.1 released with full support for a multi-threaded deterministic map-reduce implementation (`parallel_deterministic_reduce`)

BACKUP

Framework and Algorithm Parallelism

Beyond Event Level Parallelism

- Framework Parallelism
  - After modifications (declaring dependencies etc.), parallel execution of already existing serial modules is possible
  - Hides most of the multi-threading complexity from the module developer
  - Scales very well at the price of loading and writing multiple events at the same time. See the presentation by Chris Jones*
- Algorithm Parallelism
  - Changes mostly contained in one module
  - Very lightweight scaling (in terms of memory)
  - Transparent to subsequent Modules
  - Most profitable to apply on long-running Modules which can only operate sequentially (like CMS Iterative Tracking)

A great potential lies in combining these two levels of parallelism: scale with the amount of input data and the number of available computing cores.

* Forum on Concurrent Programming Models and Frameworks, 14.03.2012
http://indico.cern.ch/conferenceDisplay.py?confId=181721

Passage of particles through matter

26.3. Multiple scattering through small angles

A charged particle traversing a medium is deflected by many small-angle scatters. Most of this deflection is due to Coulomb scattering from nuclei, and hence the effect is called multiple Coulomb scattering. (However, for hadronic projectiles, the strong interactions also contribute to multiple scattering.) The Coulomb scattering distribution is well represented by the theory of Molière [32]. It is roughly Gaussian for small deflection angles, but at larger angles (greater than a few θ_0, defined below) it behaves like Rutherford scattering, having larger tails than does a Gaussian distribution.

If we define

$$\theta_0 = \theta^{\text{rms}}_{\text{plane}} = \frac{1}{\sqrt{2}}\,\theta^{\text{rms}}_{\text{space}} \,. \tag{26.9}$$

then it is sufficient for many applications to use a Gaussian approximation for the central 98% of the projected angular distribution, with a width given by [33,34]

$$\theta_0 = \frac{13.6\ \text{MeV}}{\beta cp}\, z\, \sqrt{x/X_0}\left[1 + 0.038 \ln(x/X_0)\right] \tag{26.10}$$

Source: PDG – Review of Particle Physics, 2010

OpenCL on CPU vs. TBB vs. OpenMP