# High Energy Electromagnetic Particle Transportation on the GPU

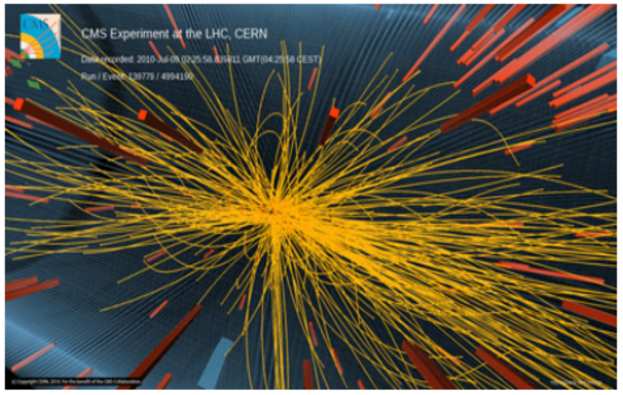
J. Apostolakis, P. Canal, D. Elvira, S.Y. Jun,
M. Paterno, J. Kowalkowski
CERN/Fermilab

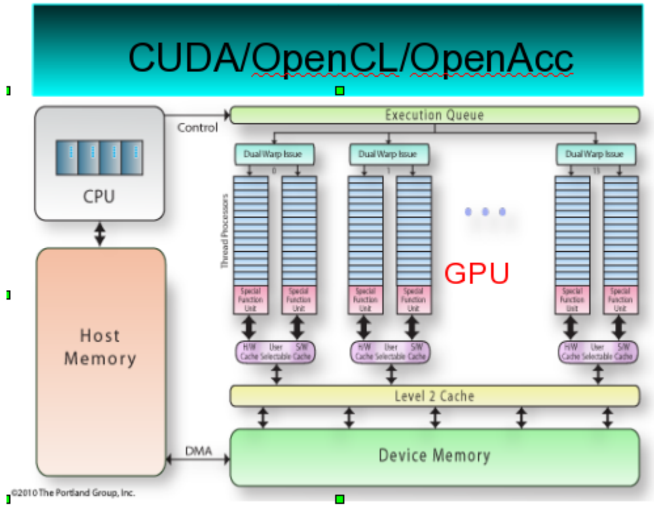
Conference on Computing in High Energy and Nuclear Physics
Amsterdam, Beurs van Berlage
October 14 – 18, 2013

## Introduction

- Future HEP software for HPC/HTC
  - hardware landscape is rapidly changing for power efficiency (advent of the many core era)
  - parallelism is no longer optional, but it must be explored thoroughly and present many challenges
  - maximize instruction throughput and data locality
- Our vision for HEP/HPC detector simulation
  - to have a massively parallelized particle (track level) transportation engine
  - comply with different architectures (GPU, MIC and etc.)
  - draw community interests for collateral efforts

# Detector Simulation in GPU as a show-case



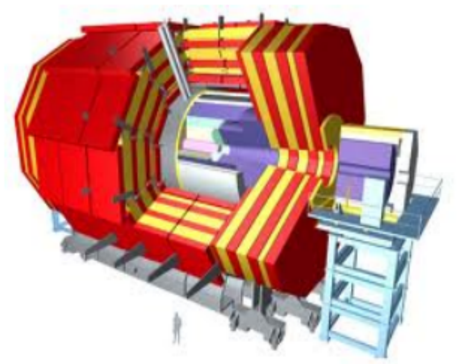


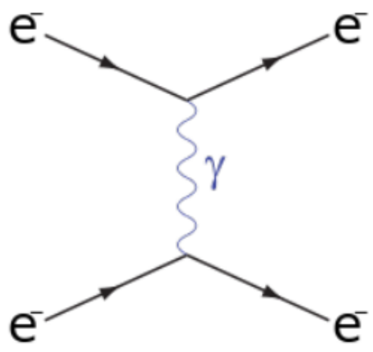
- Geant4 for detect simulation
  - highly sequential to reduce memory requirement (if-else)
  - event-level parallelism to take an advantage of using clusters
  - provided high-quality detector simulation for HEP
- GPU (CUDA) applications
  - require maximum SIMD/SIMT in conjunction with TLP
  - a good example of hybrid HPC (CPU/GPU work/load balancing)
  - many opportunities for challenging development in algorithms and efficient memory managements

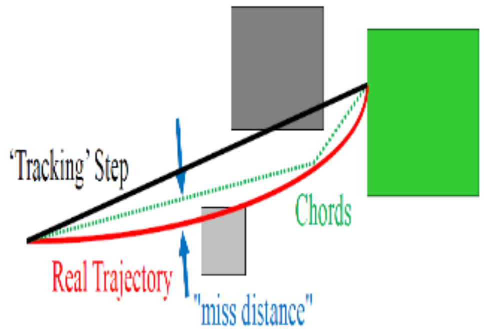
## Problem Statement

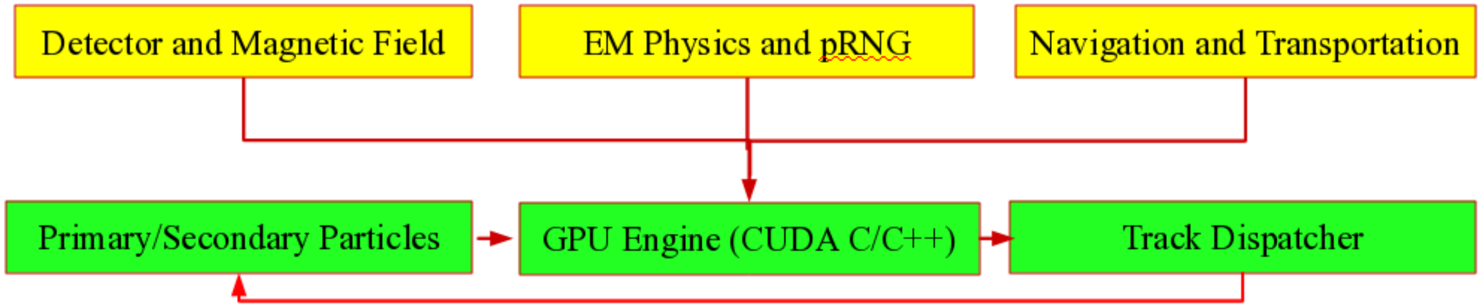
- Develop a massively parallelized EM particle transportation engine for many-core architects
- Key components for a (GPU) prototype
  - transportation (in a realistic magnetic field)
  - geometry (a simple detector description)
  - EM physics (electrons and photons)
  - concurrent CUDA kernels
- Consideration for GPU applications
  - reduce branches (avoid thread-level divergences)
  - reuse data (efficient memory transactions, latencies)
  - pRNG, floating-point, multiple streams and etc.

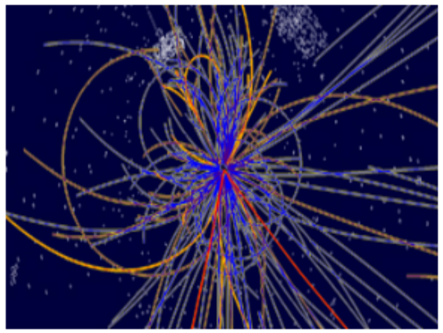
## Overview of key components

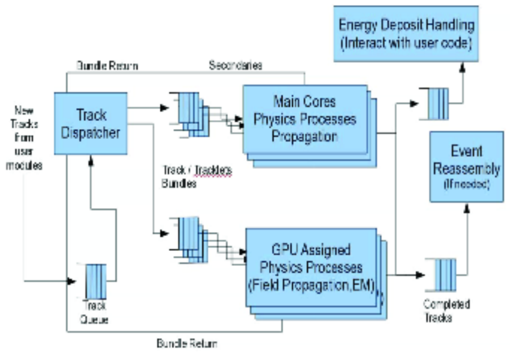







Detector and Magnetic Field
EM Physics and pRNG
Navigation and Transportation
Primary/Secondary Particles
GPU Engine (CUDA C/C++)
Track Dispatcher




Energy Deposit Handling (Interact with user code)
Bundle Return
Secondaries
New Tracks from user modules
Track Dispatcher
Main Cores Physics Processes Propagation
Track / Tracklets Bundles
Event Reassembly (If needed)
GPU Assigned Physics Processes (Field Propagation,EM)
Track Queue
Completed Tracks
Bundle Return

# Overview of GPU Kernels

- Asynchronous data transfer (tracks from a dispatcher)
- Other input data (one time allocation on global memory)
  - random states (MTwister) for each thread
  - detector geometry and a magnetic field map
  - physics tables (x-secs, brem, ionization tables, and etc.)
  - containers for secondary tracks/temporary stacks
- Stepping/tracking (split) kernels
  - GPIL-kernel
  - sorting tracks by the physics process
  - DoIt –kernel
- Also separate kernels for electrons and photons

# Performance

- Hardware (host + device)

| | Host (CPU) | Device (GPU) |
|---|---|---|
| M2090 | AMD Opteron™ 6134<br>32 cores @ 2.4 GHz | Nvidia M2090 (Fermi)<br>512 cores @ 1.3 GHz |
| K20 | Intel® Xeon® E5-2620<br>24 cores @ 2.0 GHz | Nvidia K20 (Kepler)<br>2496 cores @ 0.7GHz |

- Performance measurement
  - (4096x32) tracks
  - Gain = Time(1 CPU core)/Time(total GPU cores)
    Time=(data transfer + kernel execution)
  - default <<< Block, Thread >>> organization
    M2090<<<32,128>>> and K20<<<26,198>>>

## Particle Transportation

- Transport a particle for a proposed step length in a magnetic field (volume based CMS B-field map)
  - photon kernel: linear navigator
  - electron: propagation in a magnetic field
- Arithmetic intensity of the adaptive step control
  - occupancy/off-chip memory operand is low
  - data transfers between host and device >> kernel time
- A full chain of transportation requires geometry
  - geometry intersect and other decision trees
  - add intensity, but also introduce kernel divergence and memory operands (require optimization for SIMT)

## Performance - Transportation

- Decompose transportation by the particle type
  - separate kernels is ~30% faster for $\gamma$:e- = 0.2:0.8 mixture
- Performance of numerical algorithms for the equation of motion of a charged particle in a magnetic field

| GPU Type | Algorithm | CPU[ms] | GPU[ms] | Kernel[ms] | CPU/GPU | CPU/Kernel |
|---|---|---|---|---|---|---|
| M2090 | Classical RK4 | 106.9 | 9.7 | 2.6 | 10.9 | 41.0 |
| | RK-Felhberg | 119.3 | 9.9 | 2.8 | 12.0 | 42.3 |
| | Nystrom RK4 | 39.4 | 7.9 | 0.8 | 5.0 | 51.8 |
| K20 | Classical RK4 | 78.6 | 4.5 | 1.7 | 17.5 | 47.4 |
| | RK Felhberg | 87.9 | 4.4 | 1.6 | 19.8 | 55.2 |
| | Nystrom RK4 | 30.9 | 3.5 | 0.7 | 8.6 | 46.9 |

## Geometry

- A set of geometry classes to support EM physics and the particle transportation
  - material (element, material and Sandia table)
  - solids (box, tubs and etc.) and logical/physical vol.
  - Navigator, multilevel locator
- A simple, but realistic detector is constructed on CPU and re-mapped on GPU global memory
- Create a navigator per thread on GPU and reuse it (locating the global position is expensive)

## EM Physics

- Processes and models implemented

| Primary | Process | Model | Secondaries | Survivor |
|---|---|---|---|---|
| $e^-$ | Bremsstrahlung | SeltzerBerger | $\gamma$ | $e^-$ |
| | Ionization | MollerBhabhaModel | $e^-$ | $e^-$ |
| | Multiple Scattering | UrbanMscModel95 | – | $e^-$ |
| $\gamma$ | Compton Scattering | KleinNishinaCompton | $e^-$ | $\gamma$ |
| | Photo Electric Effect | PEEffectFluoModel | $e^-$ | – |
| | Gamma Conversion | BetheHeitlerModel | $e^-e^+$ | – |

- Use look-up tables for lambda and other parameters for energy loss and sampling
- Secondary particles are stored atomically on GPU, and may be transported to CPU or rescheduled for the next tracking cycle on GPU

## Global Memory

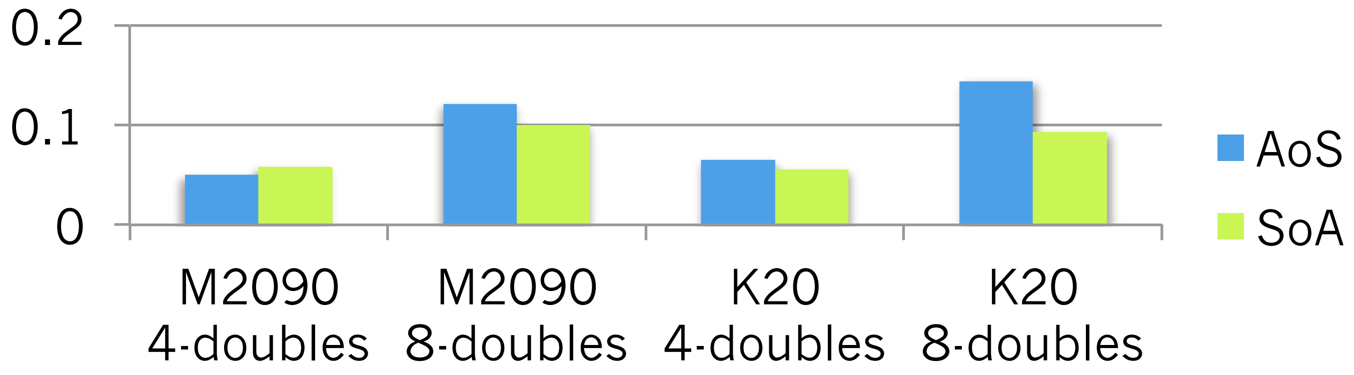
- EM physics processes and models require frequent data access from/to global memory
  - input: material information, physics tables
  - output: secondary particles (N=0,1,2 per step) and hits
- Memory transaction (atomic add) for 100K secondaries

| NVIDIA M2090 <<<32,128>>> | GPU [ms] | CPU [ms] |
|---|---|---|
| Pre-allocated fixed memory | 1.5 | 39.5 |
| Dynamic allocation per thread | 49.8 | 59.1 |
| Dynamic allocation per block | 79.0 | 59.0 |

- Strategies for secondary particles, hits and etc.
  - any dynamic memory allocation is very expensive
  - use pre-allocated memory (a fixed size stack on GPU)

# Data Structure

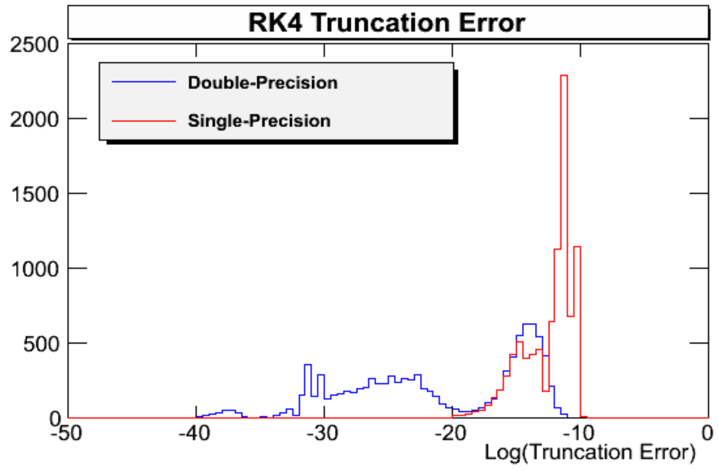
- Coalesced global memory access
  - align memory address for efficient data access
- Array of Struct (AoS) vs. Struct of Array (SoA)
  - a simple test of loading data (4-doubles, 8-doubles) and writing back to the global memory (65K accesses)

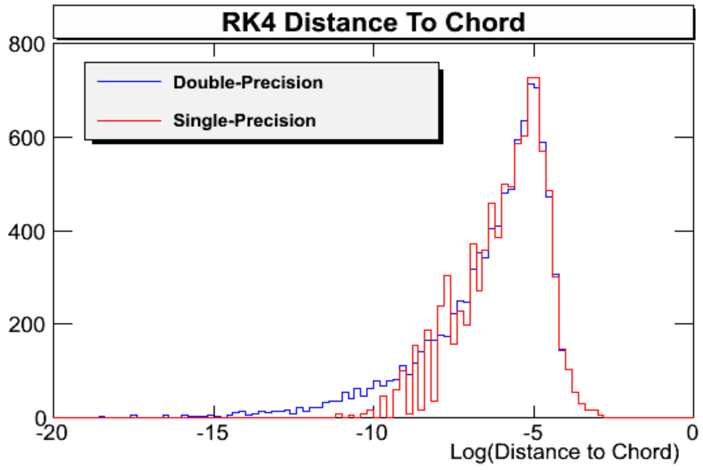


  - CPU: really depends in the size of data and architecture

# Floating-point Consideration

- Cost for double-precision
  - memory throughput (x2)
  - possible registers spilling
  - cycles for arithmetic instructions (x2/x3 in M2090/K20)
  - performance in classical RK4: float/double = 2.24 (M2090)
  - not negotiable for precision and accuracy
- Possibilities for single-precision
  - input physics tables
  - B-field map (texture)
  - local coordination





# Random Number Generators

- SIMD random number engine in each thread
- CUDA pRNG library (CURAND)
  - xor-family (XORWOW)
  - L'Ecuyer's multiple recursive generator (MRG32k3a)
  - Mersenne Twister (MTGP32, 32bit, period 2^11213)
- Performance: (64 blocks x 256 threads)
  - two kernels (initialize states, generation) for efficiency

| CURAND pRNG | Init States [ms] | 10K RNG [ms] |
|---|---|---|
| XORWOW | 4.12 | 7.92 |
| MRG32k3a | 5.02 | 21.88 |
| MTG32 | 0.69 | 31.94 |

# Performance

- Hardware (host + device)

| | Host (CPU) | Device (GPU) |
|---|---|---|
| M2090 | AMD Opteron™ 6134<br>32 cores @ 2.4 GHz | Nvidia M2090 (Fermi)<br>512 cores @ 1.3 GHz |
| K20 | Intel® Xeon® E5-2620<br>24 cores @ 2.0 GHz | Nvidia K20 (Kepler)<br>2496 cores @ 0.7GHz |

- Performance measurement
  - (4096x32) tracks
  - Gain = Time(1 CPU core)/Time(total GPU cores)
    Time=(data transfer + kernel execution)
  - default <<< Block, Thread >>> organization
    M2090<<<32,128>>> and K20<<<26,198>>>

# Performance: Realistic Simulation

- A simple calorimeter (a.k.a CMS Ecal)
- Tracking for 1-step: split kernels (GPIL+sorting+DoIt)

| | CPU [ms] | GPU [ms] | CPU/GPU |
|---|---|---|---|
| AMD+M2090 | 748 | 37.8 (62.6)* | 19.8 (11.9)* |
| Intel®+K20M | 571 | 30.4 (81.9)* | 18.7 (7.0)* |

()* GPU time using one kernel (sequential stepping)

- Optimization strategies
  - kernel basis (high-level restructuring)
  - component basis (low-level improvement by profilers)

# Other Considerations

- Understanding performance of sub-components
  - profiled each physics process/model
  - identified divergent instructions (inefficient sampling for parallel execution, do-while, ...)
  - unit tests for algorithms and data structure
- Efficient sorting without using thrust::sort (bucket-based sorting)
- Multiple streams and concurrent kernels
- Validation
  - device codes vs. identical host codes (executed on CPU)
  - host codes vs. back-ported CPU codes

## GPU Connector to an External Scheduler

- Vector Prototype (presentation by Federico) can serve as the track buckets provider to the GPU prototype
- GPU connector is an interface to the Vector Prototype
- Challenges
  - different geometry implementation – need to translate location and history information back and forth
  - difference in data layout
  - only a subset of particle can be handled
  - (ideal) bucket size very different from CPU
  - try to maximize kernel coherence

# GPU Connector to the Vector Prototype

- Implementation
  - send back to CPU particles not handled
  - stage particles in a set of buckets
    - list and type of bucket is customizable, one idea is to buckets based on particle/energy that have a common (sub)set of likely to apply physics.
    - within this baskets the particles are placed in order/group given by the VP
  - delay the start of a kernel/task until it has enough data or has not received any new data in a while
  - to maximize overlap uploads are started for a task after handling a CPU basket

# Future Plan

- Continue integration with the vector prototype
  - demonstrate a working example with the connector
  - share components (geometry, physics, transport and data structure)
- Redesign the prototype optimally for SIMT/SIMD
  - minimize branches (granulize tasks)
  - maximize locality (instruction and memory)
  - efficient data structure, algorithms and kernel managers for leveraging parallelism/vectorization
- Consideration for hybrid computing models
  - MIC (TBB), OpenCL and etc.