GPU Prototype

ASCR-HEP @Fermilab

Feb. 5, 2014

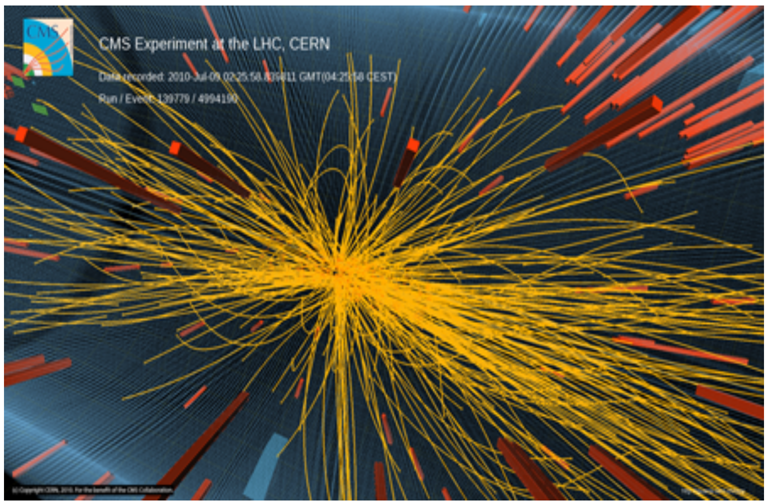
P. Canal, D. Elvira, S.Y. Jun, G. Lima

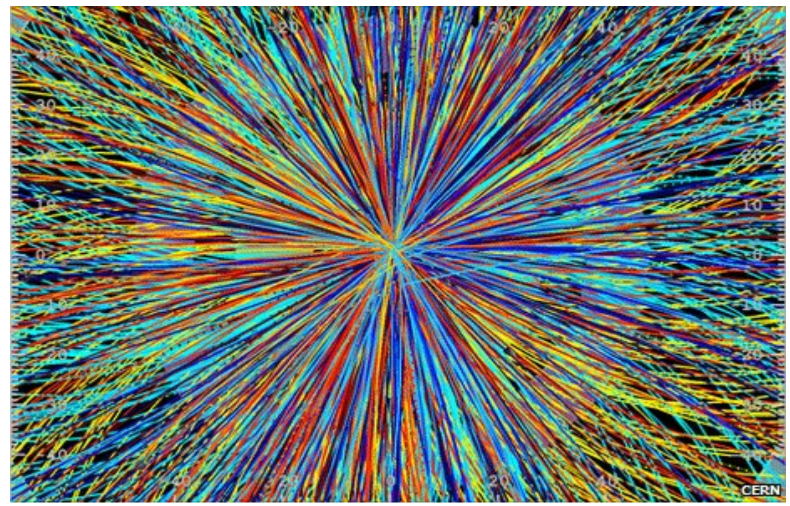
Introduction

- Parallelism was nearly dead in 20 years ago
  - ages of Pentium (clusters rule HTC)
  - Moore's law had been prevailed till 2005
  - efficient memory hierarchy (cache)
- Parallelism is resurrected by a new hardware trend
  - quantum leakage (hitting power-wall)
  - multi-cores, many-cores, SMX
  - not a problem, but new opportunities ?
- How can we explore parallelism in HEP?
  - requires evolution or revolution, and many challenges

One goal

- Research and develop a massively parallelized HEP simulation engine





LHC Run-I — Events over Grid, Tier-X

HighLumi-LHC + Pileups — Challenges for HEP computing

- New programming models for HEP-HTC/HPC

Two-dimensional Game

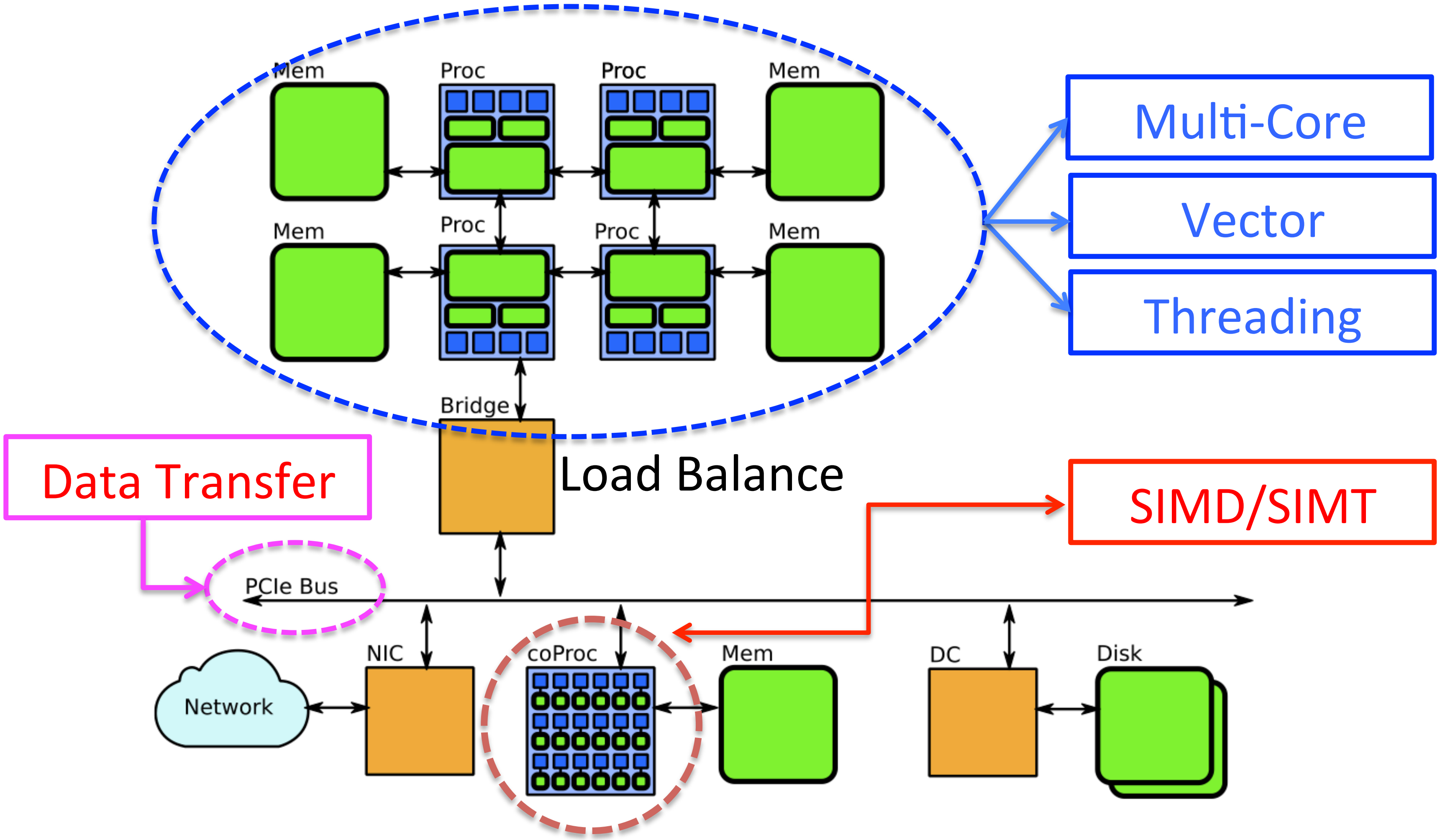
- Two-fold problem

	maximize	minimize
memory	locality	latency
instruction	throughput	divergence

- Strategies: events → tasks
  - track-level parallelism by R. Brun (vector)
  - decompose and regroup in a pipeline (ILP)

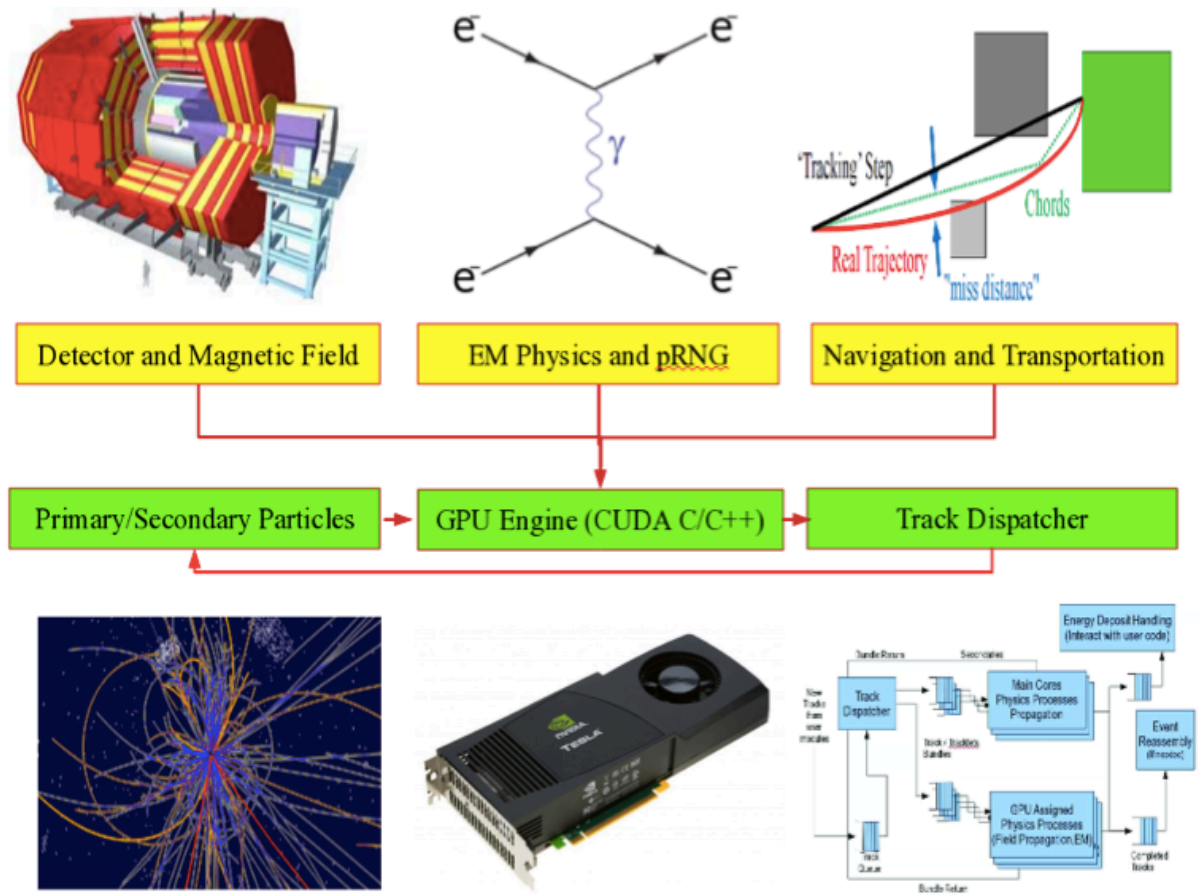
Concurrent Programming Model

- Host (CPU) + Coprocessors (GPU, MIC)



GPU Prototype: Three Core Components

- Geometry
  - detector
  - B-field
  - transport
- Physics
  - cross section
  - final state
- Scheduler
  - task stealing
  - load balance

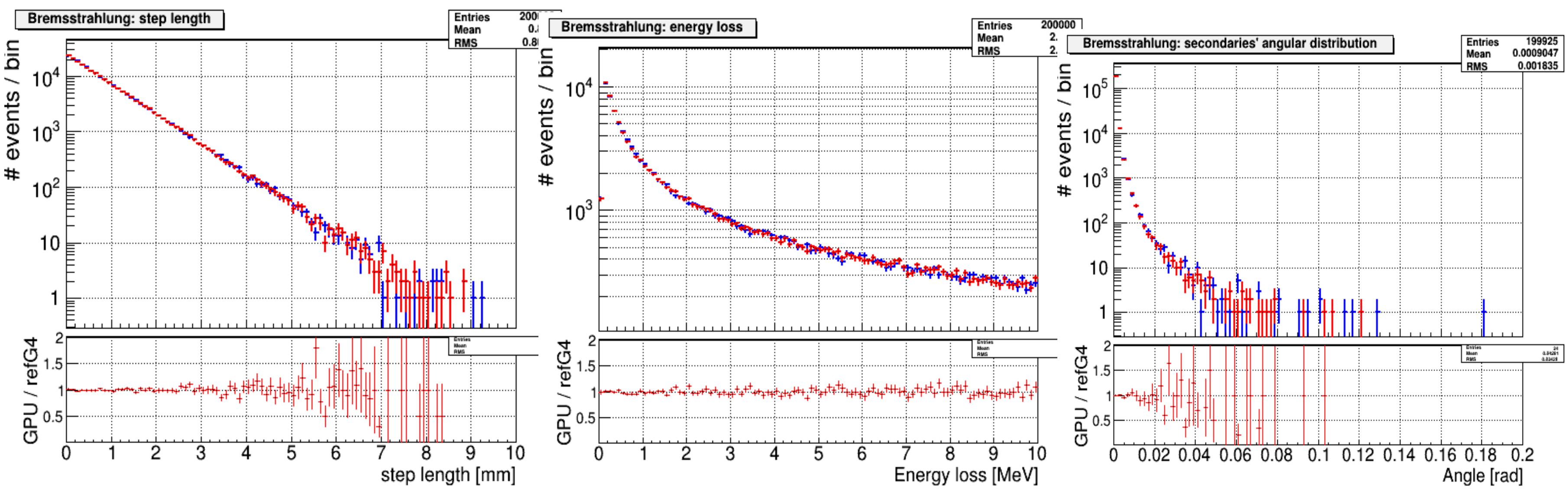


Bottom-up Approach

- Build a detector on CPU and relocated on GPU
- Ported Geant4 standard EM physics onto coprocessors and evaluate performance with a sequence of tasks
  - separated kernels by the particle type (e-, γ)
  - get the interaction length (cross section)
  - sorting (by logical volume) (not implemented)
  - transportation
  - sorting (by the physics process)
  - post stepping actions and final state sampling

Physics Validation

- Compare simulated physics outputs (G. Lima)
  - device code (RED) vs. Geant4 (BLUE)
  - ex. Bremsstrahlung process (1 GeV e-)
  - interaction length, energy loss, angular distribution of secondary photons, etc.



Performance Evaluation: GPU

- Hardware (host + device)

	Host (CPU)	Device (GPU)
M2090	AMD Opertron™ 6134 32 cores @ 2.4 GHz	Nvidia M2090 (Fermi) 512 cores @ 1.3 GHz
K20	Intel® Xeon® E5-2620 24 cores @ 2.0 GHz	Nvidia K20 (Kepler) 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<<<26,192>>>

Performance: Realistic Simulation

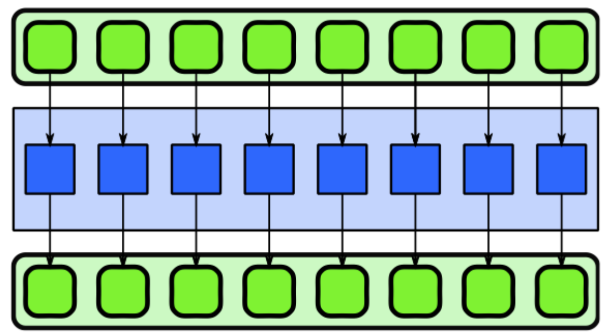
- A simple calorimeter (a.k.a CMS Ecal)
- Tracking for one step: split kernels (GPIL+sorting+DoIt)

	CPU [ms]	GPU [ms]	CPU/GPU
AMD+M2090	748	37.8	19.8
Intel®+K20M	571	30.4	18.7

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

Xeon Phi Performance

- Ported the GPU device codes to MIC
- Offload mode for different magnetic field integrators
  - one step with the CMS magnetic field
  - Parallel loop (map)
    - openMP (omp parallel)
    - TBB (parallel_for)
    - CilkPlus (cilk_for)



- Performance measurement
  - 100K (random) tracks
  - number of MIC threads = 236
  - gain = time (1CPU core)/time (236 MIC cores)

Performance Results

Models	Alogorithm	CPU[ms]	MIC[ms]	CPU/MIC
openMP	Classical RK4	97.0	29.4	3.3
	FK Felhberg	104.9	29.9	3.5
	Nystrom RK4	49.0	23.7	2.1
TBB	Classical RK4	98.6	40.1	2.5
	FK Felhberg	109.2	38.1	2.9
	Nystrom RK4	50.6	29.0	1.8
CilkPlus	Classical RK4	101.2	55.5	2.0
	RK Felhberg	106.4	51.4	2.3
	Nystrom RK4	49.4	49.1	1.1

- All programming models show similar performance results (no optimization for VPU/memory align)

No Free Lunch

- HEP detector simulation (Geant4) is a giant
  - complicated, object oriented
  - designed for efficient memory footprints
  - real-life problems of natures
  - random (ex. acceptance and rejection)
- Coprocessor architectures
  - hard to scale performance for conventional HEP detector simulation - almost impossible (?)
  - fine tuning is critical, but how much (good for sequential computing anyway)

Many Challenges

- Structural programming models for VPU/SMX
  - architecture, compiler, algorithm are all matters
  - purely SIMD/SIMT driven (memory, instruction throughput)
  - generic codes for scalar, vector, coprocessors (platform independent approach - template)

Top-down

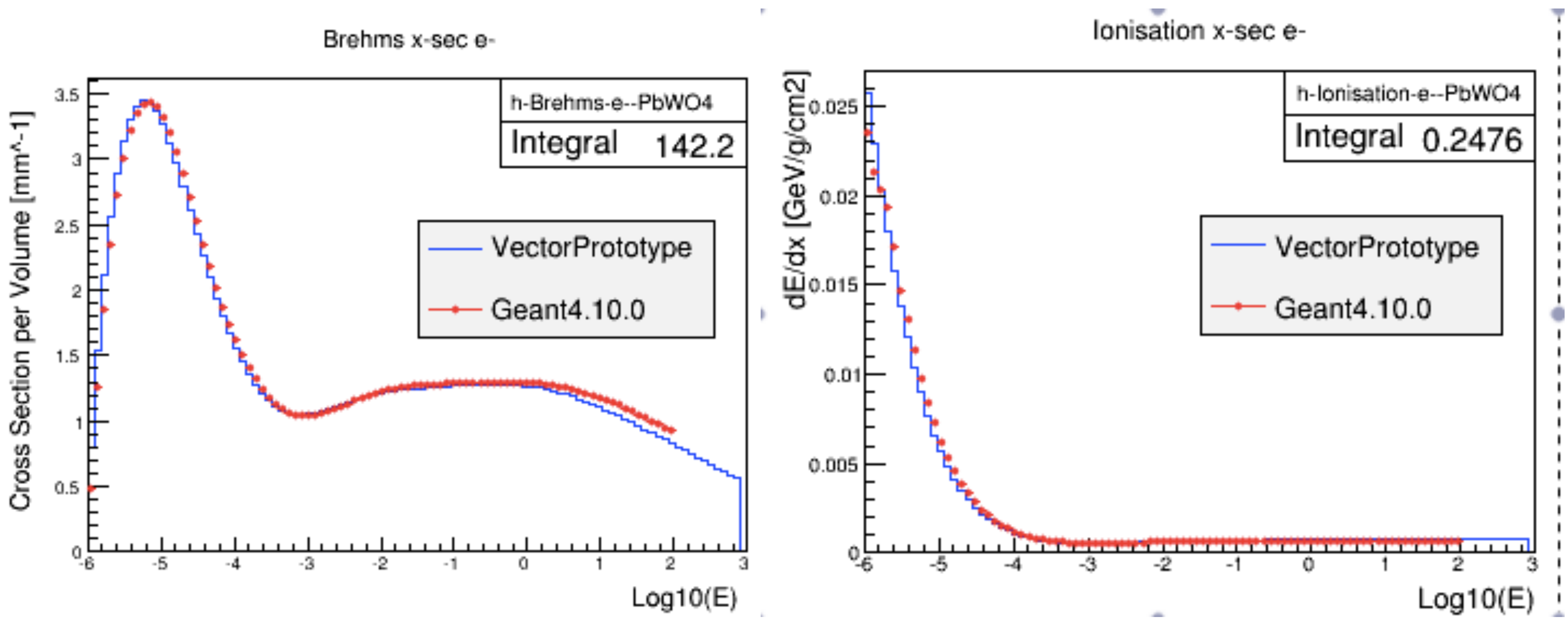
- Build a framework and add fully optimize and vectorized components
- Evaluate performance for each additional component exclusively
  - understanding impacts of additions/substitutions
  - identifying and prioritizing for optimization
- Interface for different frameworks
  - ported Geant4 code (existing CUDAized codes)
  - vectorized codes (geometry, navigation, physics)
  - tabulated physics used in the vector prototype

Snapshot of Tabulated Physics

- Cross section (16 MB)
  – $\sigma(Z,A,E)$ per element for major pid and processes
- Interaction length for a composite material
  – $1/\lambda = N_A \, \rho \, \Sigma \, w_i \, \sigma_i$
- Energy loss: dE/dx per element
  – $dE/dx\ (\text{material}) = \rho \, \Sigma \, (w_i/\rho_i) \, (dE/dx)_i$
- Final state sampling (1.2 GB) and timing (20MB)
  – multiple scattering: angle, length, etc.
  – secondary particles: N(10) samplings per energy bin

Comparison to Geant4

- Tabulated VP physics vs. cmsExp + Geant4 10.0
- Ex: e- cross section (bremsstrahlung) and dE/dx (ionization) for PbWO4



CUDAzing and Vectorizing Physics

- Standard EM physics processes/models
  - start with Bremsstrahlung (e-) and Compton(γ)
  - compatible with Geant4, the vector prototype and the coprocessor prototype (GPU/MIC)
  - evaluate performance and validate quality of physics
- Abstraction and implement the rest of e/γ physics

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

Multi-dimensional Considerations

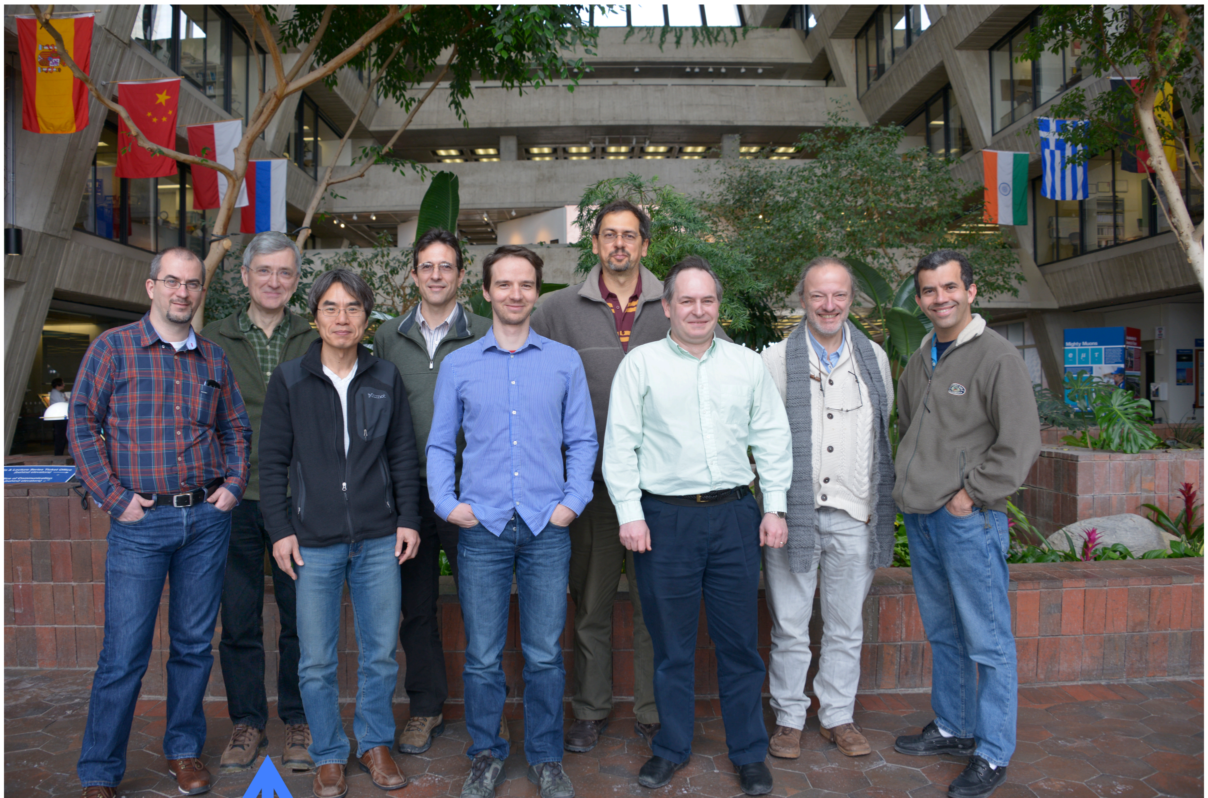
- Global memory access
- Data structure
- Floating point consideration
- Random number generation
- Understanding performance
- Efficient sorting
- Multiple streams and concurrent kernels
- Validation

Plan

- Integration with the current vector prototype
  - demonstrate a working example with the connector
  - adopt/develop vectorized components (geometry, transport, physics)
- Redesign the prototype optimally for SIMT/SIMD
  - minimize branches (granulize tasks)
  - maximize locality (reuse data)
  - efficient data structure, algorithms and kernel managers for leveraging parallelism/vectorization
- Early considerations for hybrid computing models
  - OpenCL, TBB, etc.

More on Philippe's Talk

- Connector to the vector prototype
- 2014 milestones from the January VP-GPU meeting



↑ *Krzysztof took this picture!*