Fermilab

U.S. DEPARTMENT OF ENERGY | Office of Science

High Energy Electromagnetic Particle Transportation on the GPU

P. Canal, D. Elvira, S.Y. Jun, G. Lima (Fermilab)

J. Apostolakis (CERN)

A. Mametjanov (Argonne National Laboratory, US)

Annual Concurrency Forum Meeting, April 2-3, 2014, CERN

Overview

- Introduction
- GPU Prototype
- Physics Validation
- Performance
- Challenges and New Strategies
- Incorporating into the Vector Prototype
- Plans

Introduction

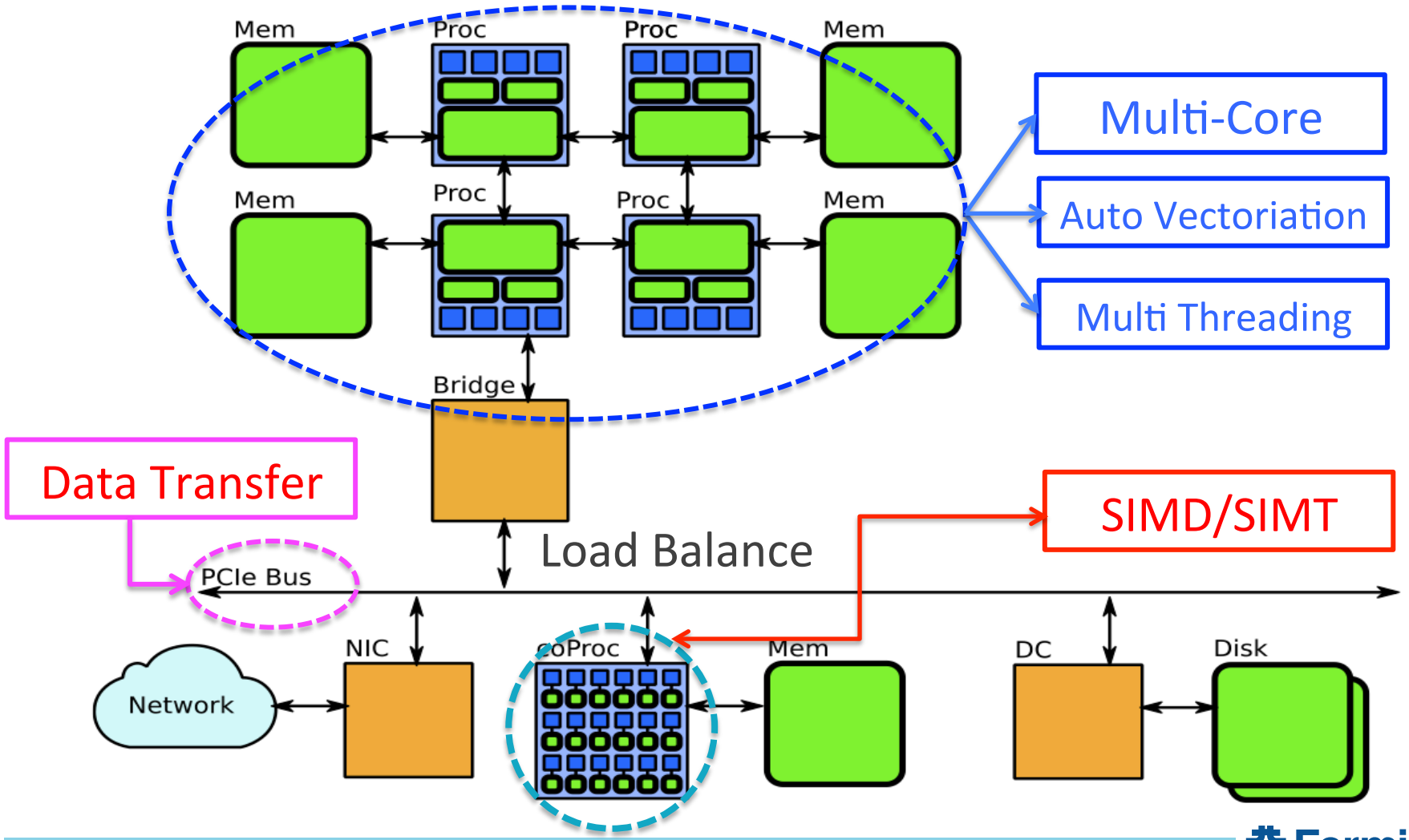
- Goals
  - R&D a massively parallelized HEP simulation engine
  - explore new HPC programing models for HEP
- Two-fold problem: data locality and instruction throughput

	maximize	minimize
memory	locality	latency
instruction	throughput	divergence

- Strategies:
  - track-level parallelism (originally proposed by R. Brun)
  - SIMD: vectorization (VPU + data locality)
  - SIMT: thread level parallelism (ALU + shared memory)

Concurrent Programming Model

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

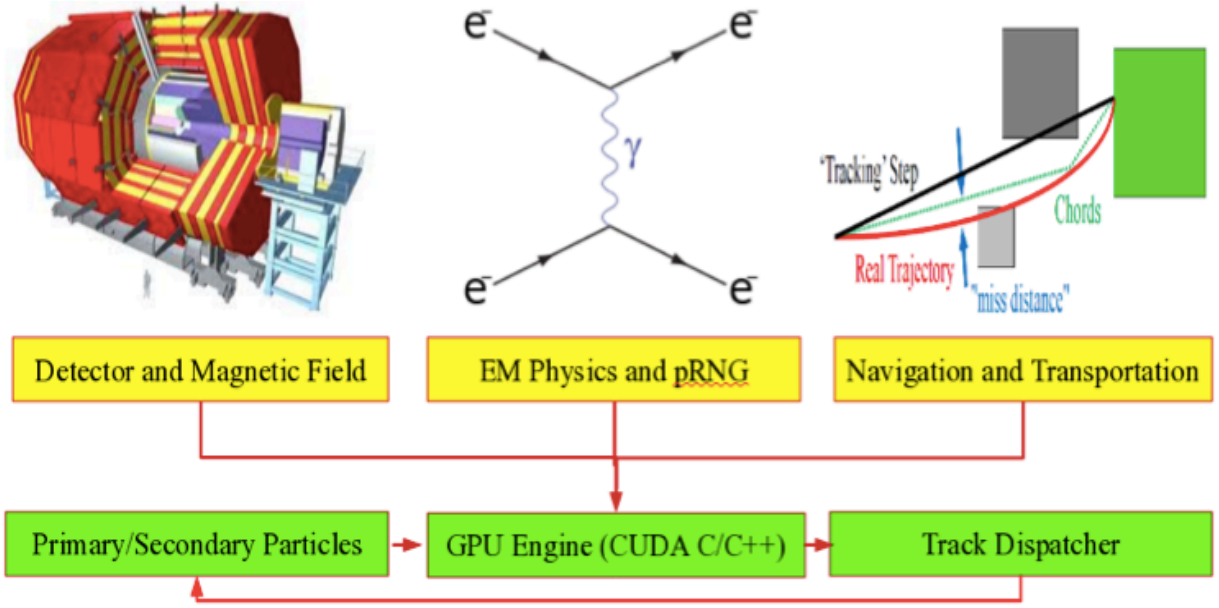


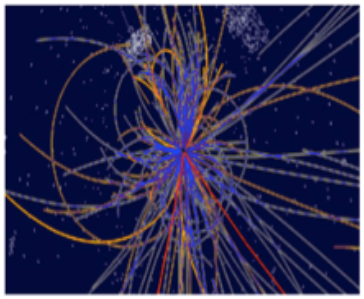
Particle Transport on the GPU

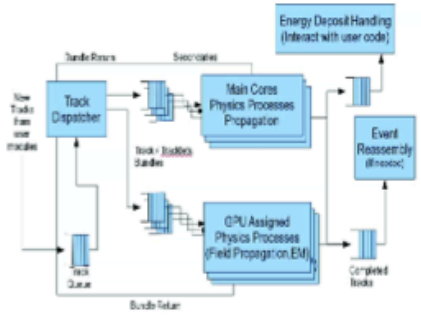
- Initial scope: charged particle transportation in a magnetic field as a demonstrator
  - propagate {tracks} for {given step lengths} through a simple geometry (similar to the CMS electromagnetic calorimeter)
  - lesson learnt (see backup for performance)
    - arithmetic intensity = instructions/(memory load) is too low
    - data transfer is costly (compared to kernel execution time)
- Extension: full EM particle tracking on the GPU
  - implemented physics processes and models for e-/γ
  - nothing is free: additional divergences and memory accesses
- Restructuring the simulation flow
  - separate kernels and regroup tracks for each subtask

GPU Prototype: Three Core Components

- Geometry
  - detector
  - B-field
  - navigator
- EM Physics
  - e-/gamma
  - cross section
  - final state
- GPU Scheduler
  - task stealing
  - load balance







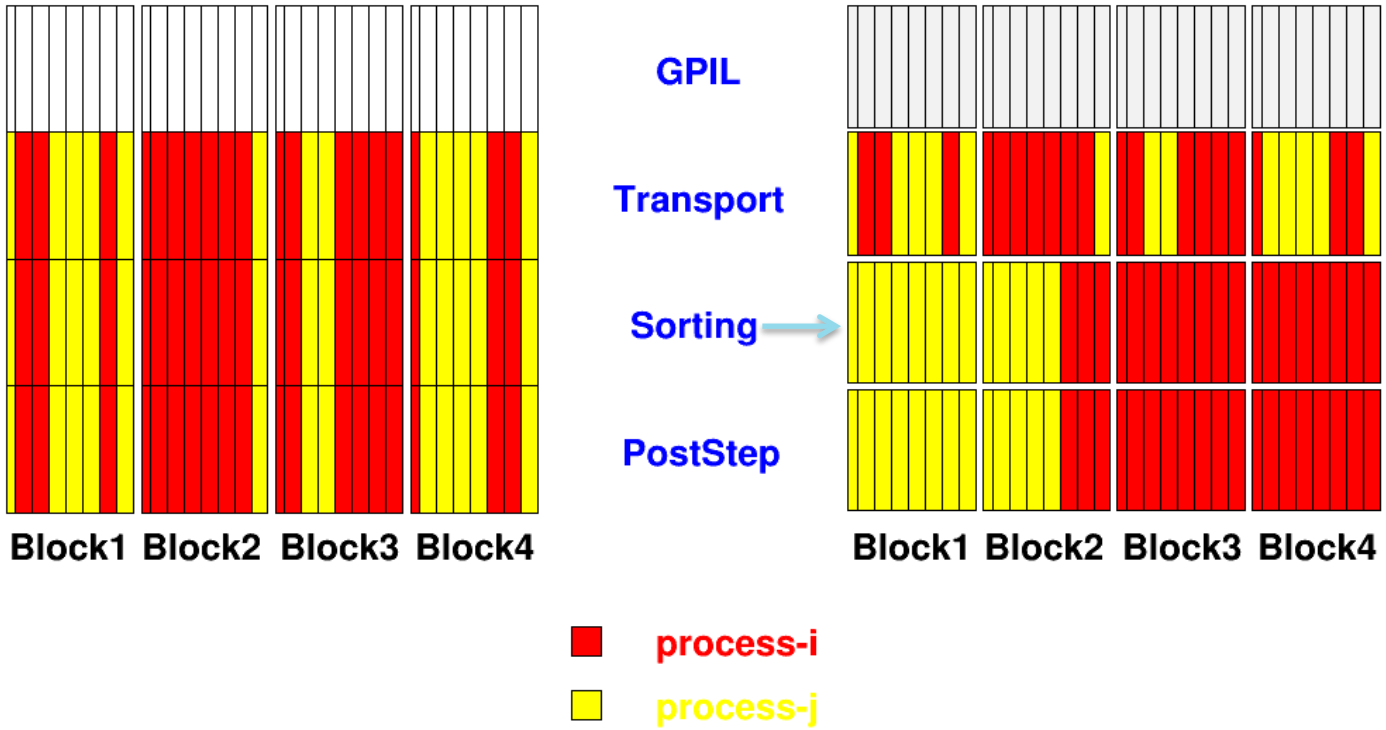
Fermilab

Summary of Simulation Flow

- Initial data transfer to the GPU (global memory)
  - detector (geometry and magnetic field)
  - physics (cross sections and other data used in physics models)
  - random states
- Recurrent data transfers
  - input tracks from an external scheduler
  - output (surviving & secondary particles, hits)
- Concurrent kernel executions
  - use separate kernels for each particle type (e-, γ)
  - calculate cross sections and find distance to interaction
  - transport in a realistic HEP magnetic field (CMS)
  - sort by winning physics process (i.e. which occurs first)
  - sample final state and apply user stepping actions

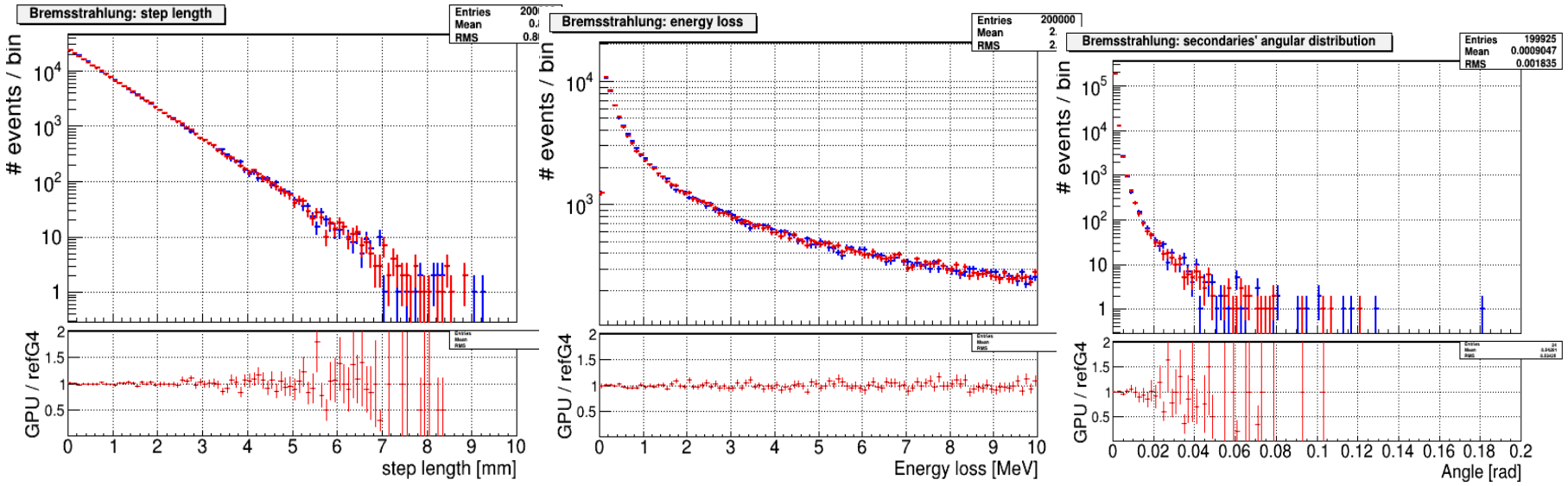
Split Kernel: Example of High Level Restructuring

- Decompose the GPU kernel by tasks and regroup tracks by
  - particle type (charged vs. neutral) (~30% faster for e-:γ=0.8:0.2)
  - physics process (~2x faster)
  - (logical volume for navigation)



Physics Validation of GPU Physics

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



Performance Evaluation

- Hardware (host + device)

	Host (CPU)	Device (GPU)
M2090	AMD Opteron™ 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 <<< Blocks, Threads >>> organization M2090<<<32,128>>> and K20<<<26,192>>>

Performance: Realistic Simulation

- A simple calorimeter (CMS Ecal) with the CMS b-field map
- Tracking for one 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)

- Performance by each kernel (% of the total application time)
  - random states (MTwister) : 5.8% (one time initialization)
  - physical interaction length and transportation: 47%
  - count_by_process: 2.2%
  - sort_by_process: 2.7%
  - post step actions and writing secondary particles: 42.3%

Performance Issues and Considerations

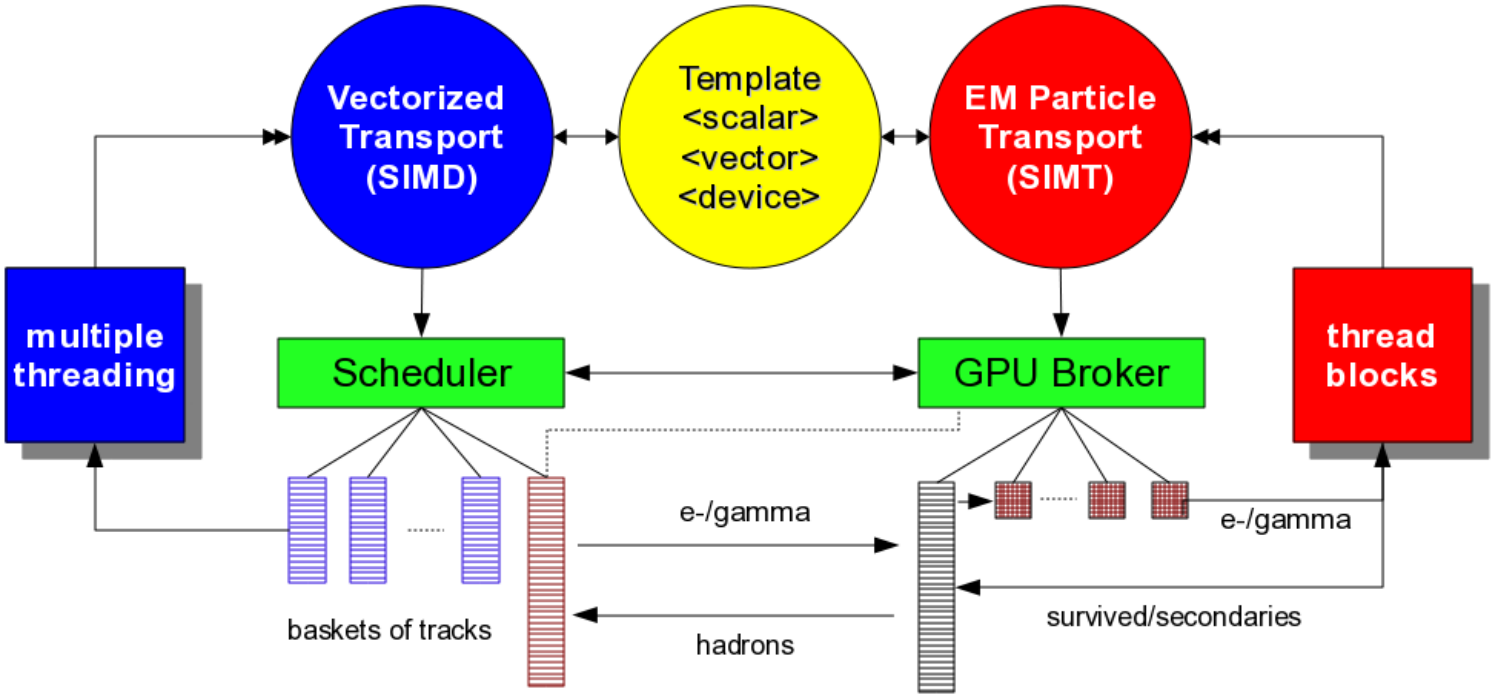
- Observed issues (by the Nvidia profiler)
  - low arithmetic intensity and high branch divergence
  - high memory latencies and low multiprocessor occupancy
- Considerations
  - memory access
    - global: aligned, coalesced and pre-allocated DMA
    - shared: block-based reduction (ex. atomic counter)
    - texture memory (spatial locality, ex. magnetic field map)
  - data structure (AoS vs. SoA)
  - floating point consideration (double vs. float)
  - random number generation (different SIMD pRNGs)
  - efficient sorting (bucket vs. thrust::sort)
  - multiple streams and concurrent kernels

Challenges and New Strategies

- HEP detector simulation (ex. Geant4) is a giant
  - complicated, object oriented physics simulation
  - designed for efficient memory footprints (data driven)
  - random sampling (ex. acceptance and rejection)
- Coprocessor architectures
  - hard to scale performance for conventional HEP detector simulation (non-deterministic) - almost impossible (?)
  - fine tuning is critical, but restructuring simulation with efficient memory accesses is much more important
- Top-down approach
  - develop fully optimized (cudarized) and vectorized components of geometry and physics
  - Incorporate into a concurrent simulation framework

Incorporating into the Vector prototype

- The vector prototype started at CERN (talk by F. Carminati)
  - scheduler based on p-thread (Andrei Gheata)
  - vectorized geometry (talk by S. Wenzel and J. De Fine Licht)
  - tabulated physics (cross section, final states sampling, etc.)
- Integration to the vector prototype (GPU broker)



GPU Connector to the Vector Prototype

- Goals
  - maximize kernel coherence
  - adapt to GPU ‘ideal’ bucket size (very different from CPU)
  - process only tracks GPU can handle efficiently
  - differences(s) in data layout
- Implementation
  - stages particles in a set of (customizable) buckets
    - particle/energy that have a common subset of physics models
    - keep order provided by the main scheduler
  - delays the start of a kernel/task until it has enough data or has not received any new data in a while
  - starts uploads after each basket processing to maximize overlap (even before the bucket is full)

Plans

- Fully Integrate the GPU prototype with the vector prototype
  - demonstrate a working example with the GPU connector
  - adopt/develop vectorized components (geometry, transport, physics) and evaluate performance
- Redesign the GPU prototype optimally for SIMD/SIMT
  - minimize branches (granulize tasks)
  - maximize spatial locality (reuse data)
  - efficient data structure, algorithms and kernel managers for leveraging TLP/vectorization
- Code abstraction
  - template algorithms for scalar, vector and device
  - early considerations for hybrid/parallel computing models

Backup

EM Physics

- Processes and models implemented

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^+	–

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

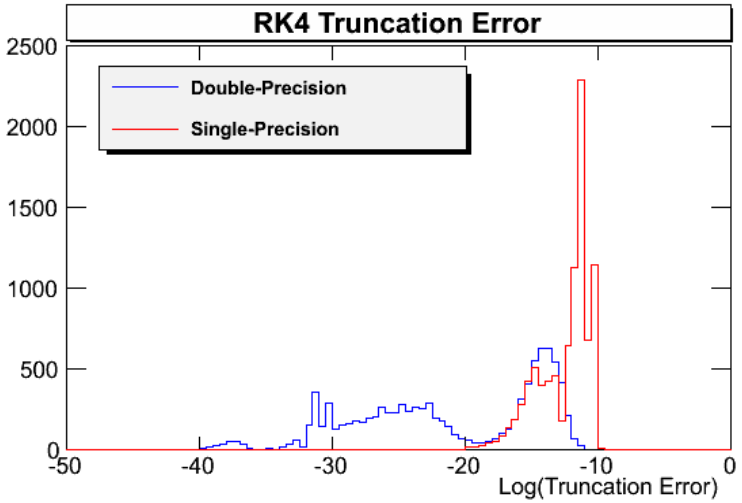
Performance – Transportation

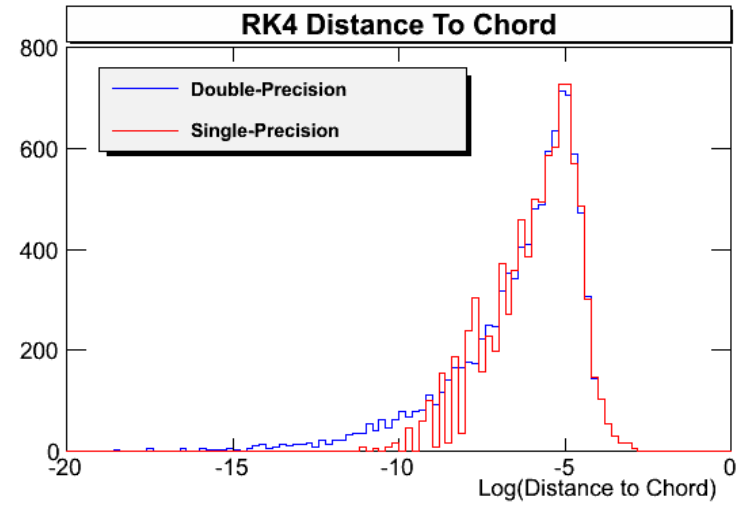
- Decompose transportation by the particle type
  - separate kernels is ~30% faster for γ:e- = 0.2:0.8 mixture
- Performance of numerical algorithms for the equation of motion of a charged particle in a magnetic field

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

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





Fermilab

Global Memory

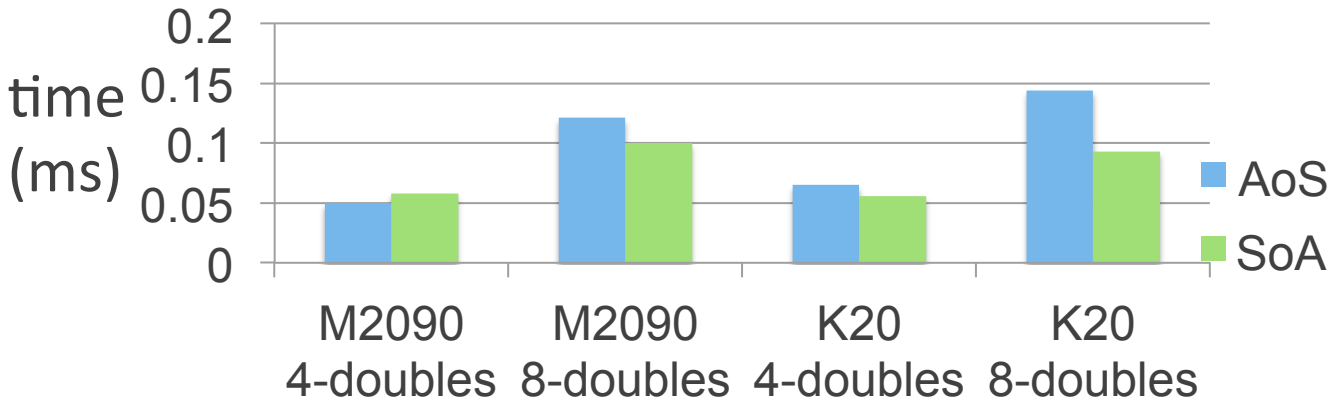
- EM physics processes and models require frequent data access from/to global memory
  - input: material information, physics tables
  - output: secondary particles (N=0,1,2 per step) and hits
- Memory transaction (atomic add) for 100K secondaries

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

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