High Energy EM Particle Transportation on Coprocessors

FNAL/PDS/Geant4 R&D
P. Canal, D. Elvira, S.Y. Jun, G. Lima

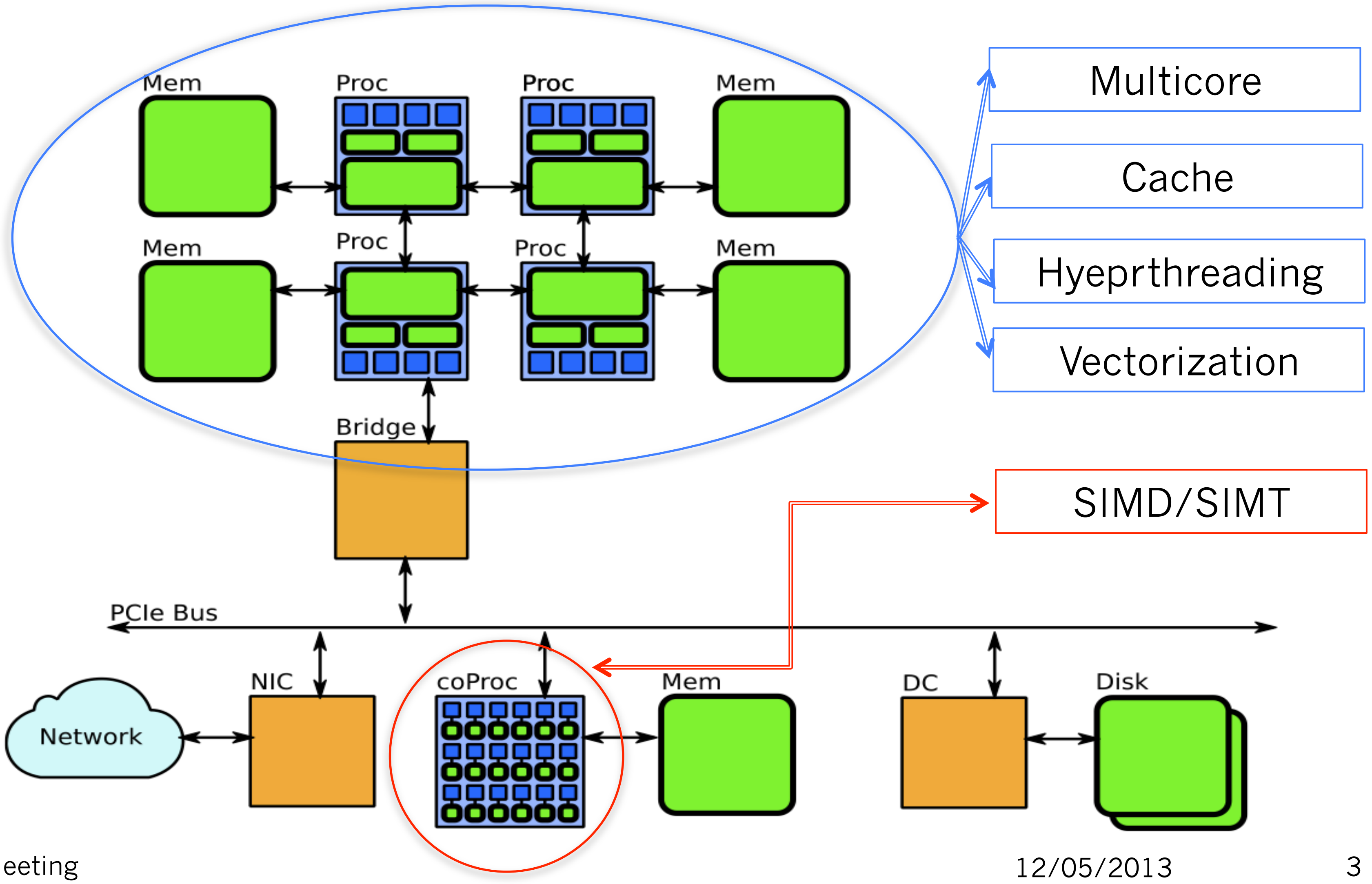
Physics and Detector Simulation Meeting
December 5, 2013

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.)
  - demonstrate a highly parallelized realistic HEP detector simulation on a hybrid computing platform

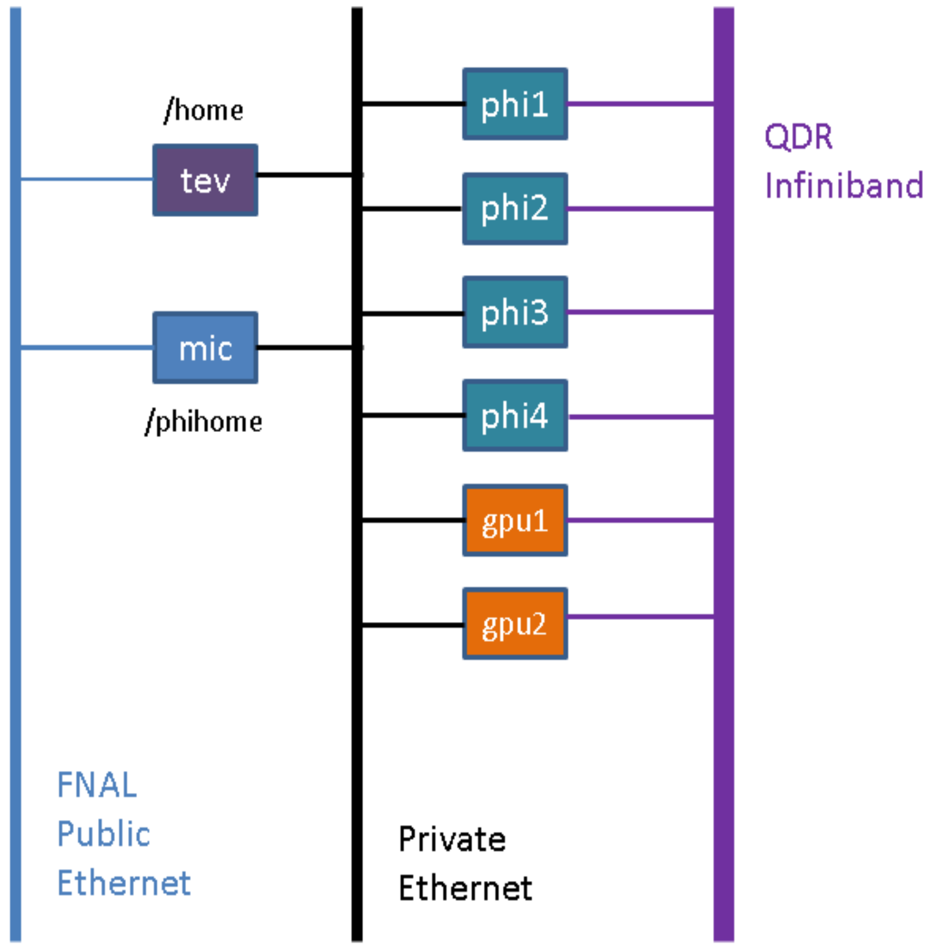
Concurrent Computing Model

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

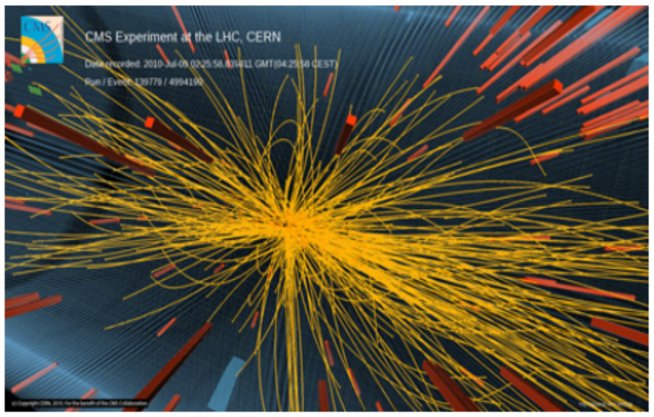


Phi Cluster on Wilson

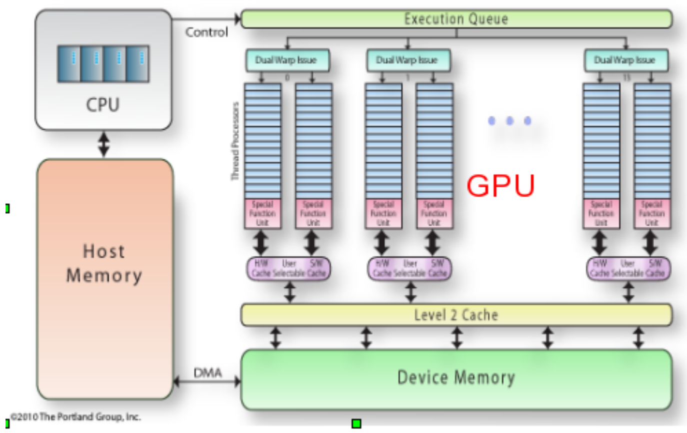
- 6 Intel Xeon E2620
  - 12 cores @ 2GHz
  - 32 GB each phix and gpux
- 4x4 Intel Xeon Phi 5110P
  - 60 cores @ 1.053 GHz
  - 8GB
- 2 Nvidia K20 Kepler GPU
  - 2496 cores @ 0.7 GHz
  - 5GB
- Geant4R&D on cluck
  - AMD Opertron (12 cores)
  - Fermi M2090 (512 cores)



Detector Simulation in Coprocessor


CMS Experiment at the LHC, CERN
Run / Event: 139779 / 4994190

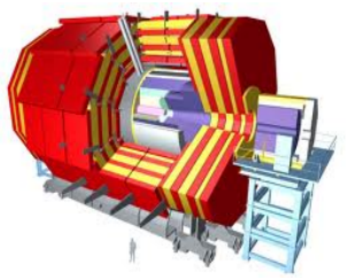
CUDA/OpenCL/OpenAcc

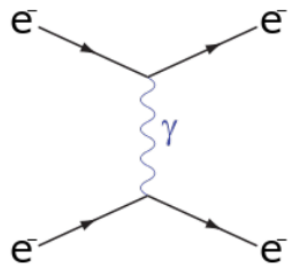


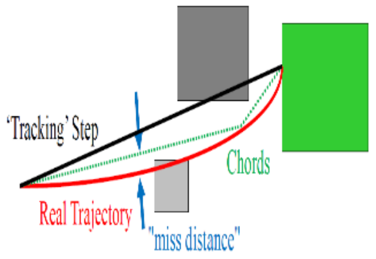
- GPU (CUDA) applications
  - require maximum SIMT in conjunction with TLP
  - a good example of hybrid HPC
  - many opportunities for challenging development in algorithms and efficient memory managements
- MIC for parallel computing
  - generic C/C++
  - OpenMP/TBB/CilkPlus/OpenCL
  - Vectorization/SIMD
  - load balancing/work stealing

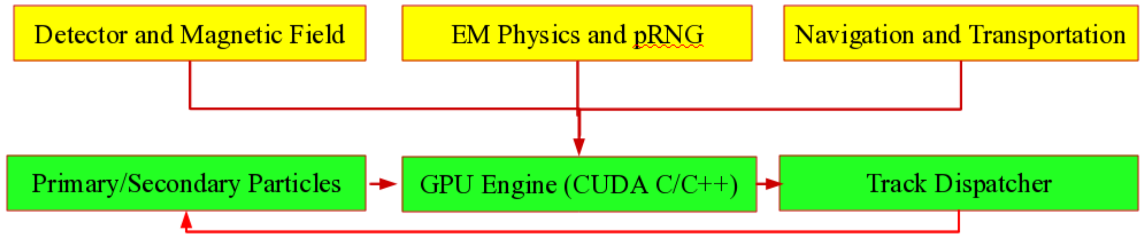
Goal and Key Components

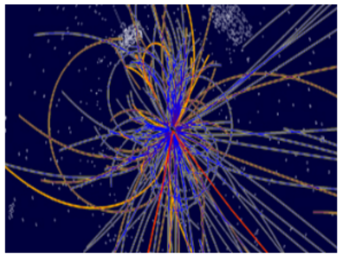
- Develop a massively parallelized EM particle transportation engine for many-core architects
- Geometry, physics, transportation, task management

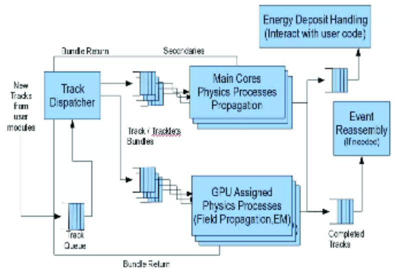






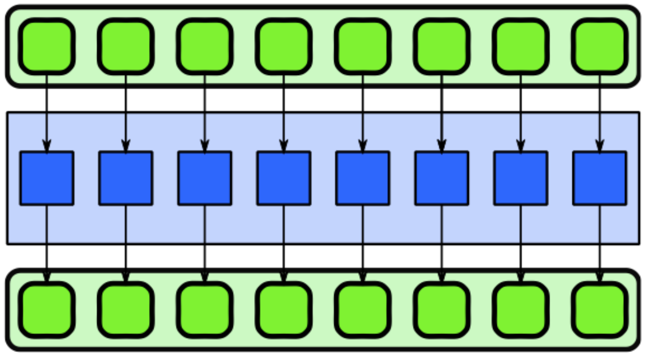

Detector and Magnetic Field
EM Physics and pRNG
Navigation and Transportation
Primary/Secondary Particles
GPU Engine (CUDA C/C++)
Track Dispatcher

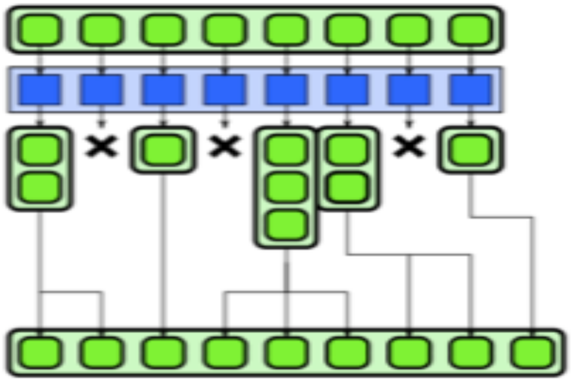



Energy Deposit Handling (Interact with user code)
Bundle Return
Secondaries
New Tracks from user modules
Track Dispatcher
Main Cores Physics Processes Propagation
Track / Tracklets Bundles
Event Reassembly (If needed)
GPU Assigned Physics Processes (Field Propagation,EM)
Track Queue
Completed Tracks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

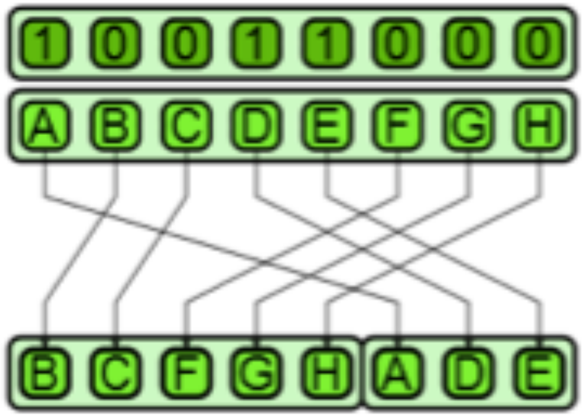
Track Level Parallelism and Split Kernel

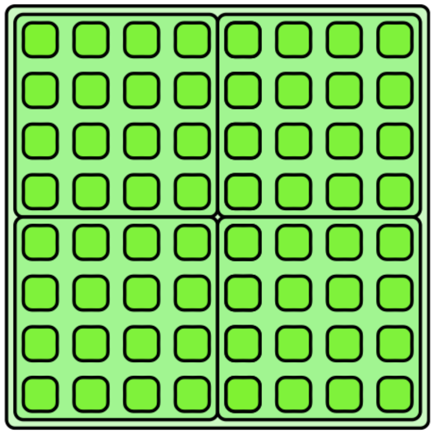
- Stepping: simple map or expand (survivor/secondary)





- GPIL+sorting+DoIt: split and repartition (by physics process, particle type, geometry and etc.)





Status

- GPU prototype was implemented with CUDA compatible device codes
  - key components, pRNG, sorting, memory management
  - evaluate performances for a simple CMS ECAL and a magnetic field map as well as sub-components of codes
- GPU device codes are fully ported to MIC platform
  - tested with simple OpenMP, TBB, CilkPlus applications
  - test more examples and understand performance
- Integrate the prototype with the vector prototype
  - coprocessor broker (by Philippe)
  - optimize the device codes (top-down approach)

GPU Performance

- 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 - 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
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	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 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

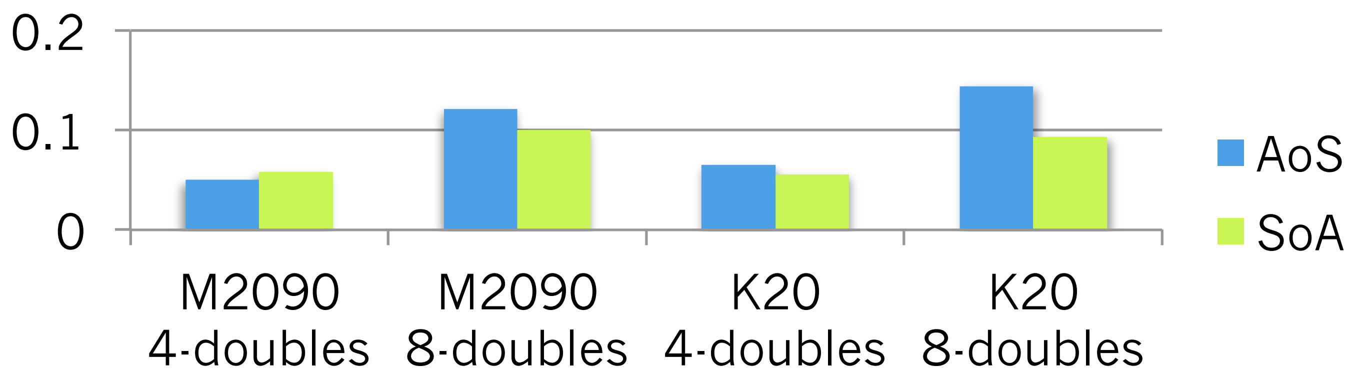
- EM physics processes and models require frequent data access from/to global memory
  - input: material information, physics tables
  - output: secondary particles (N=0,1,2 per step) and hits
- Memory transaction (atomic add) for 100K secondaries

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

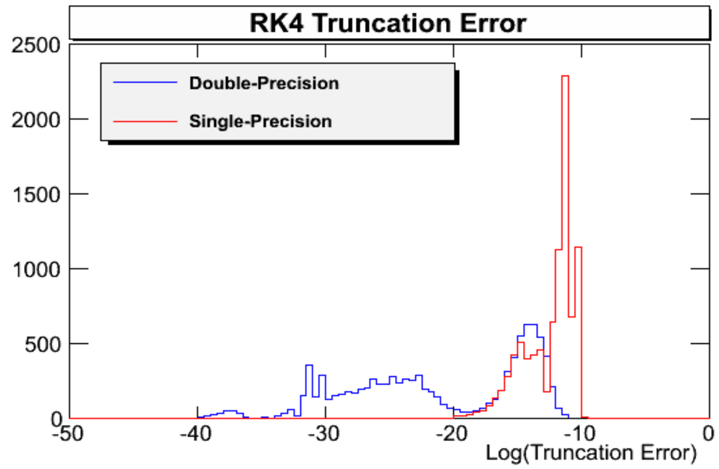
- Coalesced global memory access
  - align memory address for efficient data access
- Array of Struct (AoS) vs. Struct of Array (SoA)
  - a simple test of loading data (4-doubles, 8-doubles) and writing back to the global memory (65K accesses)

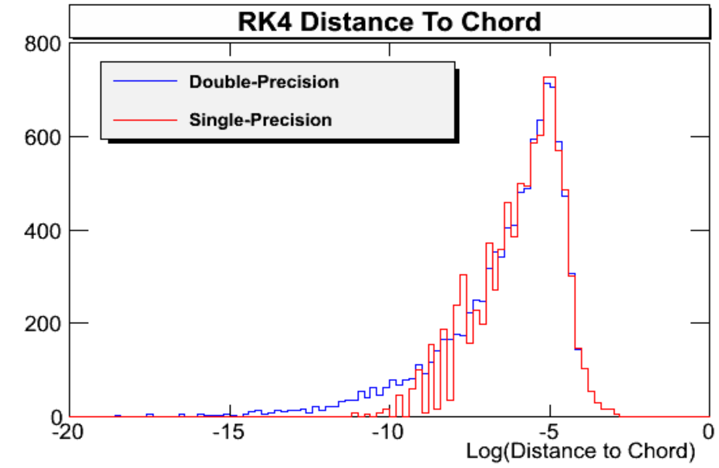


  - CPU: really depends in the size of data and architecture

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: Realistic Simulation

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

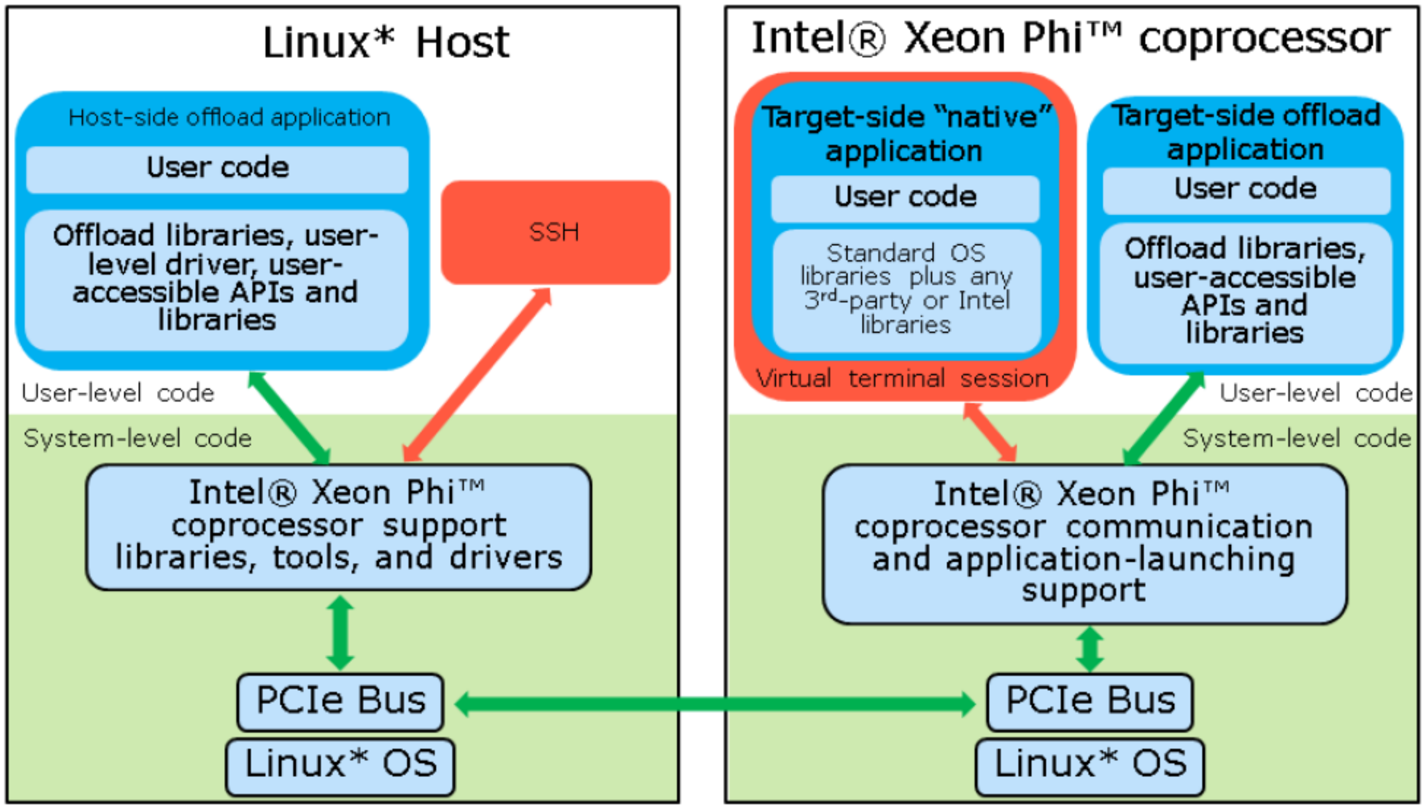
	CPU [ms]	GPU [ms]	CPU/GPU
AMD+M2090	748	37.8 (62.6)*	19.8 (11.9)*
Intel®+K20M	571	30.4 (81.9)*	18.7 (7.0)*

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

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

Xeon Phi Performance

- Compilation modes
  - offload (heterogeneous for both host and coprocessor)
  - native (coprocessor = standalone multicore computer)


Linux* Host
Host-side offload application
User code
Offload libraries, user-level driver, user-accessible APIs and libraries
SSH
User-level code
System-level code
Intel® Xeon Phi™ coprocessor support libraries, tools, and drivers
PCIe Bus
Linux* OS
Intel® Xeon Phi™ coprocessor
Target-side "native" application
User code
Standard OS libraries plus any 3rd-party or Intel libraries
Virtual terminal session
Target-side offload application
User code
Offload libraries, user-accessible APIs and libraries
User-level code
System-level code
Intel® Xeon Phi™ coprocessor communication and application-launching support
PCIe Bus
Linux* OS

Xeon Phi Performance

- Offload mode for different numerical algorithms
  - one step with the CMS magnetic field map
  - explicit memory copy model
  - parallel loop (map) with OpenMP (omp parallel), TBB(parallel_for), CilkPlus (cilk_for)
- Performance measurement
  - 100,000 (random) tracks
  - Gain = Time(1 CPU core)/Time(total MIC threads) Time=(data transfer + parallel execution)
  - number of threads = 4 x (N_phicores -1) = 236

Performance Results

- OpenMP

Algorithm	CPU[ms]	MIC [ms]	CPU/MIC	First Event
Classical RK4	97.0	29.4	3.3	0.46
Jameson RK4	99.8	30.0	3.3	0.56
RK Felhberg	104.9	29.9	3.5	0.46
NystromRK4	49.0	23.7	2.1	0.17

- TBB

Algorithm	CPU[ms]	MIC [ms]	CPU/MIC	First Event
Classical RK4	98.6	40.1	2.5	0.035
Jameson RK4	100.4	40.3	2.6	0.036
RK Felhberg	109.2	38.1	2.9	0.040
NystromRK4	50.6	29.0	1.8	0.015

Performance Results

- CilkPlus

Algorithm	CPU[ms]	MIC [ms]	CPU/MIC	First Event
Classical RK4	101.2	55.5	2.0	0.57
Jameson RK4	109.4	54.2	2.2	0.61
RK Felhberg	106.4	51.4	2.3	0.61
NystromRK4	49.4	49.1	1.1	0.22

- Summary
  - all programing models show similar performance results
  - current device codes may be poorly optimized for MIC
    - instruction are not vectorized
    - memory accesses are not aligned

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

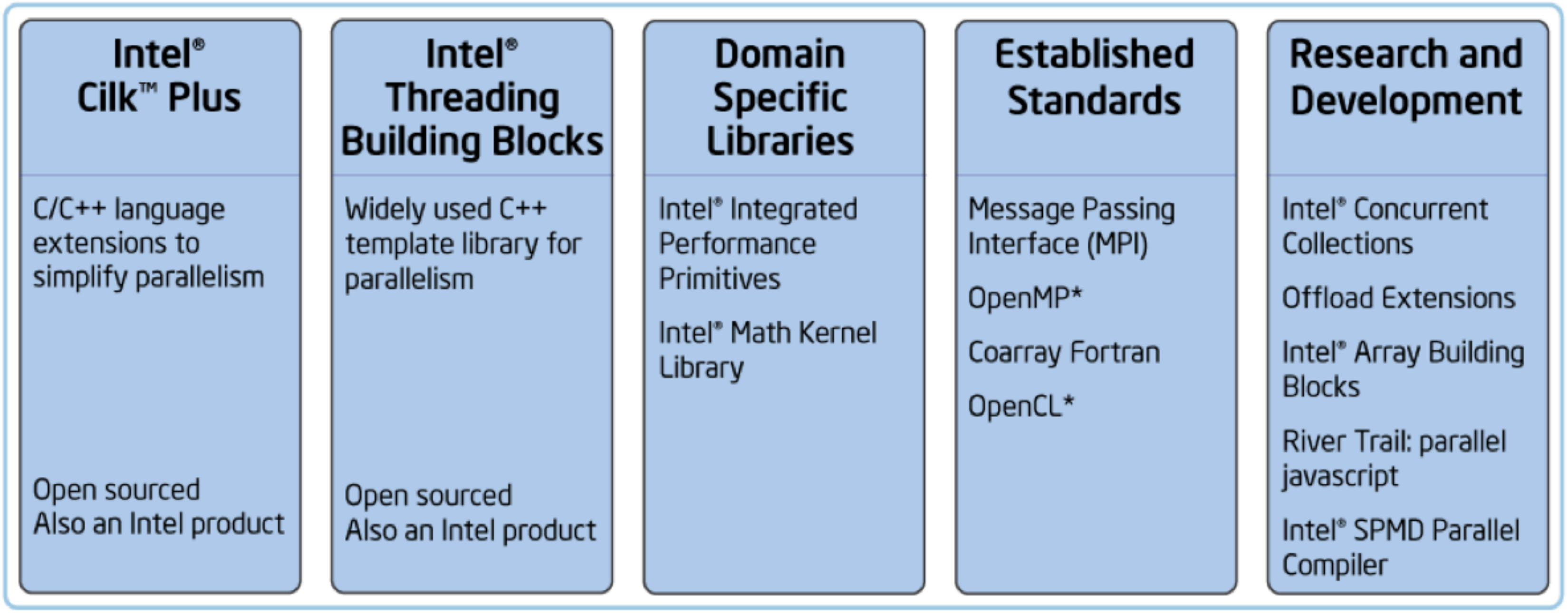
- Implementation
  - send back to CPU particles not handled
  - stage particles in a set of buckets
    - list and type of bucket is customizable, one idea is to buckets based on particle/energy that have a common (sub)set of likely to apply physics.
    - within this baskets the particles are placed 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

Near-term 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
- Explore hybrid models with CPU/Coprocessors

MIC Programming Models

- Intel programing model
  - TBB: C++ template library
  - CilkPlus: C++ language extension
  - choice of high performance parallel programming models



Intel® Cilk™ Plus	Intel® Threading Building Blocks	Domain Specific Libraries	Established Standards	Research and Development
C/C++ language extensions to simplify parallelism	Widely used C++ template library for parallelism	Intel® Integrated Performance Primitives	Message Passing Interface (MPI)	Intel® Concurrent Collections
		Intel® Math Kernel Library	OpenMP*	Offload Extensions
			Coarray Fortran	Intel® Array Building Blocks
			OpenCL*	River Trail: parallel javascript
Open sourced Also an Intel product	Open sourced Also an Intel product			Intel® SPMD Parallel Compiler