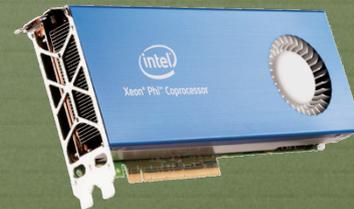
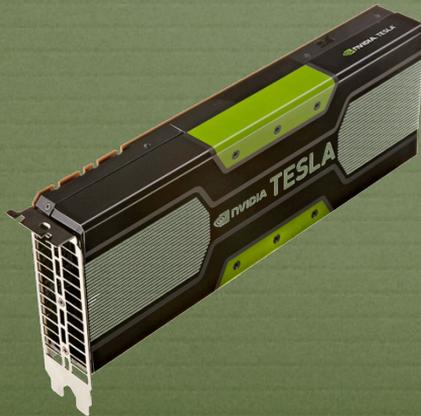


Geant R&D Projects

Ph. Canal
S.Y. Jun, F. Carminati,
K. Genser, D. Elvira



Outline

- ❖ Context
- ❖ Collaborations
 - ❖ Performance Analysis
 - ❖ EM Physics review
 - ❖ Vector Prototype
- ❖ GPU Development
- ❖ Plans

Introduction

- ❖ Future HEP software for HPC/HTC
 - ❖ hardware landscape is rapidly changing, focusing on power efficiency (advent of the many core era)
 - ❖ parallelism is no longer just an option, but must be exploited in every corner of software
 - ❖ maximize data locality and instruction throughput
- ❖ Our vision for HEP detector simulation
 - ❖ to have a massively parallelized particle (track or tracklet level) transportation engine
 - ❖ leverage different architectures (GPU, MIC and etc.)
 - ❖ draw community interests for related efforts

Charge

- ❖ Develop and study the performance of various strategies and algorithms that will enable Geant4 to make efficient use of multiple computational threads
- ❖ Analyze the internal architecture of Geant4
- ❖ Profile and document performance and memory requirements for typical HEP applications
- ❖ Identify components that require re-engineering
- ❖ Begin developing prototypes of the new components

Specific Goals

- ❖ Investigate porting specific portion of Geant4 to GPU and answer the questions:
 - ❖ What is the performance?
 - ❖ What modifications does it imply?
 - ❖ How can it be integrated with general purpose code?

Specific Goals

- ❖ Understand
 - ❖ Geant4 code structure
 - ❖ Coding and Optimization on GPU (Tesla and now Kepler)
 - ❖ How the two can be matched
 - ❖ If the same style of modification benefits CPU code
- ❖ Provide Feedback to global re-engineering effort

How

- ❖ Bottom-up approach
 - ❖ Extract time consuming proportion of the code
 - ❖ Feed prototype with realistic data
 - ❖ Captured from running a full Geant4 example
- ❖ **Experimental software environment: cmsExp**
 - ❖ CMS geometry (GDML) and magnetic field map (2-dim grid of volume based field extracted from CMSSW)
 - ❖ Shooting 100 GeV Pions

Collaborators

Local

- ❖ Philippe Canal
- ❖ Daniel Elvira
- ❖ Soon Yung Jun
- ❖ Jim Kowalkowski
- ❖ Marc Paterno
- ❖ Krzysztof Genser
- ❖ Guilherme Lima

Institutions

- ❖ FNAL
- ❖ RENCI
- ❖ ISI
- ❖ UO (& ANL)
- ❖ CERN
- ❖ SLAC

Collaborations



- ❖ Performance analysis and redesign investigations
 - ❖ *ISI, RENCI, ANL, SLAC, CERN*
 - ❖ Leverage external input/knowledge, alternative point of view, tools.
 - ❖ Optimization opportunity search
 - ❖ Code review
 - ❖ (Performance Analysis) Tools improvement
 - ❖ Plan on shifting focus from legacy code to new code
 - ❖ *Bi-weekly meeting*

Collaborations



- ❖ New Technologies exploration
 - ❖ Join effort on vector prototypes (*CERN*)
 - ❖ Track(let) level parallelism and vectorization
 - ❖ Inserting GPU Prototype in a full(er) fledged prototype
 - ❖ Plan on sharing/reusing code
 - ❖ *Bi-weekly meeting*
 - ❖ GPU Research & Development
 - ❖ Track(let) level parallelism and vectorization
 - ❖ Transportation, Geometry, EM physics (electrons and photons)
 - ❖ Require external driver for full example

ASCR (ISI, RENCIS, ANL)

- ❖ Geant4 Performance Studies
 - ❖ Performance evaluation based on realistic Geant4 application
 - ❖ CMSexp a simplified version of CMS simulation
 - ❖ Alternative track stacker
 - ❖ Performance evaluation of a Geant4 prototype running on GPU
- ❖ Review of the Geant4 electromagnetic physics packages

Optimization of G4

- ❖ Manual application of loop-invariant code motion to the main loop of the Event Manager. This resulted in a 1% performance improvement for real runs and has been already incorporated in version beta 10.0
- ❖ Inlining a specific function in the code calculating cross-section can gain approx. 1.5%
- ❖ Analysis of the CrossSectionDataStore functions and suggested improvements in some of the arithmetic being used
- ❖ Tracing of the calls show that there are "potential" opportunities for the memorization of these calls (using for example a splay tree)
 - ❖ Of the first 2787 calls to this function there are only 387 that have a unique combination of the addresses of its three inputs.

Compiler Choice

- ❖ GEANT4 is usually compiled using GCC
- ❖ 8-core Intel Xeon 5462 2.8GHz 16 GB RAM.
run_cmsExp with 10,000 events
 - ❖ gcc 4.7.3 : 1440.99 seconds
 - ❖ Intel 13.0.1 : 1272.56 seconds
- ❖ Alternative choices of compiler can yield significant performance advantages
- ❖ An autotuning exercise exploring compilation flags may be productive.

hpcviewer: cmsExpMT

Calling Context View Callers View Flat View

Scope

Scope	PAPI_TOT_CYC:Sum (I)	PAPI_TOT_CYC:Sum (E)	PAPI_TOT_...
Experiment Aggregate Metrics	1.73e+12 100 %	1.73e+12 100 %	1.49e...
▼ G4UserWorkerInitialization::StartThread(void*)	1.59e+12 92.3%		1.31e...
▼ loop at G4UserWorkerInitialization.cc: 158	1.59e+12 92.0%		1.31e...
▼ loop at G4UserWorkerInitialization.cc: 267	1.59e+12 92.0%		1.31e...
▼ G4UImanager::ApplyCommand(char const*)	1.59e+12 92.0%		1.31e...
▼ G4Ulcommand::Dolt(G4String)	1.59e+12 92.0%		1.31e...
▼ G4RunMessenger::SetNewValue(G4Ulcommand*, G4String)	1.59e+12 92.0%		1.31e...
▼ G4RunManager::BeamOn(int, char const*, int)	1.59e+12 92.0%		1.31e...
▼ G4WorkerRunManager::DoEventLoop(int, char const*, int)	1.52e+12 88.0%		1.23e...
▼ loop at G4WorkerRunManager.cc: 102	1.51e+12 87.6%		1.22e...
▼ G4WorkerRunManager::ProcessOneEvent(int)	1.51e+12 87.6%		1.22e...
▼ G4EventManager::DoProcessing(G4Event*)	1.51e+12 87.6%	1.58e+09 0.1%	1.22e...
▼ inlined from ThreeVector.icc: 96	1.50e+12 87.1%		1.21e...
▼ loop at ThreeVector.icc: 96	1.50e+12 87.1%		1.21e...
▼ inlined from G4AllocatorPool.hh: 135	1.50e+12 87.1%		1.21e...
▼ loop at G4AllocatorPool.hh: 136	1.50e+12 87.1%		1.21e...
▼ inlined from G4EventManager.cc: 172	1.50e+12 87.1%	5.00e+08 0.0%	1.21e...
▼ G4TrackingManager::ProcessOneTrack(G4Track*)	1.50e+12 86.8%	1.85e+09 0.1%	1.21e...
▼ inlined from G4TrackingManager.cc: 98	1.44e+12 83.3%	1.82e+08 0.0%	1.16e...
▼ loop at G4TrackingManager.cc: 121	1.42e+12 82.2%	6.66e+08 0.0%	1.15e...
▼ G4SteppingManager::Stepping0	1.42e+12 82.1%	2.55e+10 1.5%	1.15e...
▼ inlined from G4SteppingManager.cc: 137	1.36e+12 79.0%	6.69e+09 0.4%	1.09e...
▼ G4SteppingManager::DefinePhysicalStepLength0	6.00e+11 34.8%	2.41e+10 1.4%	4.80e...
▼ inlined from G4SteppingManager2.cc: 162	3.19e+11 18.5%	3.32e+08 0.0%	2.72e...
▼ loop at G4SteppingManager2.cc: 164	3.19e+11 18.5%	1.03e+10 0.6%	2.72e...
▼ inlined from G4VProcess.hh: 474	3.07e+11 17.8%	4.42e+09 0.3%	2.62e...
▼ G4VDiscreteProcess::PostStepGetPhysicalInteractionLeng	2.28e+11 13.2%	6.88e+09 0.4%	2.13e...
▼ G4HadronicProcess::GetMeanFreePath(G4Track const&	2.00e+11 11.6%	6.64e+09 0.4%	1.99e...
▼ G4CrossSectionDataStore::GetCrossSection(G4Dyna	1.94e+11 11.2%	1.62e+10 0.9%	1.94e...
▼ inlined from G4CrossSectionDataStore.cc: 82	1.93e+11 11.2%	8.25e+09 0.5%	1.94e...
▼ loop at G4CrossSectionDataStore.cc: 95	1.85e+11 10.7%	7.46e+09 0.4%	1.90e...
▼ G4CrossSectionDataStore::GetCrossSection	1.77e+11 10.2%	2.94e+10 1.7%	1.79e...
▼ inlined from G4CrossSectionDataStore.cc:	1.75e+11 10.2%	1.65e+10 1.0%	1.77e...
▼ loop at G4CrossSectionDataStore.cc: 14	1.12e+11 6.5%	1.15e+10 0.7%	1.31e...
▼ G4CrossSectionDataStore::GetSo	1.00e+11 5.8%	1.25e+10 0.7%	1.15e...
▼ inlined from G4CrossSectionData	9.76e+10 5.7%	7.65e+09 0.4%	1.13e...
▶ G4HadronCrossSections::G	2.13e+10 1.2%	1.88e+09 0.1%	1.68e...
▶ G4PhotoNuclearCrossSecti	1.74e+10 1.0%	1.64e+10 1.0%	2.56e...

86M of 4079M

Depth of the “hottest” call chain.
 2nd column is the inclusive cost summed across all threads.

Performance Analysis

- ❖ CPU performance analysis of Geant4 and Geant4MT
 - ❖ Effects of different compilers and compiler options
 - ❖ Callpath profiling of a CMS experiment benchmark (execution time, memory performance)
- ❖ Initial conclusions
 - ❖ Deep call chains in integrator do not allow local optimizations (including compiler optimizations)
 - ❖ Bad CPU and memory utilization caused by operating on a single particle at a time in functions at the bottom of deep call paths

hpcviewer: cmsExpMT

Calling Context View Callers View Flat View

↑ ↓ 🔥 f(x) ✓ CSV A+ A-

Scope	PAPI_TOT_CYC:Sum (I)		PAPI_TOT_CYC:Sum (E)	
Experiment Aggregate Metrics	1.73e+12	100 %	1.73e+12	100 %
▶ __ieee754_log	1.51e+11	8.8%	1.51e+11	8.8%
▶ __ieee754_exp	1.18e+11	6.8%	1.18e+11	6.8%
▶ G4ElasticHadrNucleusHE::HadrNucDifferCrSec(int, int, double)	1.68e+11	9.7%	5.73e+10	3.3%
▶ G4Navigator::LocateGlobalPointAndSetup(CLHEP::Hep3Vector const&,&	1.28e+11	7.4%	4.29e+10	2.5%
▶ __ieee754_atan2	3.90e+10	2.3%	3.90e+10	2.3%
▶ G4PhysicsVector::Value(double, unsigned long&) const	6.99e+10	4.1%	3.80e+10	2.2%
▶ G4CrossSectionDataStore::GetCrossSection(G4DynamicParticle const*	1.77e+11	10.2%	2.94e+10	1.7%
▶ G4SteppingManager::Stepping()	1.42e+12	82.1%	2.55e+10	1.5%
▶ G4SteppingManager::DefinePhysicalStepLength()	6.00e+11	34.8%	2.41e+10	1.4%
▶ G4VoxelNavigation::ComputeStep(CLHEP::Hep3Vector const&, CLHEP::	1.27e+11	7.3%	2.39e+10	1.4%
▶ G4Navigator::ComputeStep(CLHEP::Hep3Vector const&, CLHEP::Hep3V	1.85e+11	10.7%	2.25e+10	1.3%
▶ G4hPairProductionModel::ComputeDMicroscopicCrossSection(double,	8.72e+10	5.1%	2.14e+10	1.2%
▶ G4HadronCrossSections::CalcScatteringCrossSections(G4DynamicPart	2.06e+10	1.2%	1.91e+10	1.1%
▶ G4ParticleChange::CheckIt(G4Track const&)	1.78e+10	1.0%	1.78e+10	1.0%
▶ CLHEP::RanecuEngine::flat()	1.72e+10	1.0%	1.72e+10	1.0%
▶ G4CrossSectionDataStore::GetCrossSection(G4DynamicParticle const*	1.94e+11	11.3%	1.67e+10	1.0%
▶ G4PhotoNuclearCrossSection::GetIsoCrossSection(G4DynamicParticle	1.74e+10	1.0%	1.64e+10	1.0%
▶ G4BGGNucleonInelasticXS::CoulombFactor(double, int)	1.55e+10	0.9%	1.53e+10	0.9%
▶ G4Transportation::PostStepDolt(G4Track const&, G4Step const&)	1.57e+11	9.1%	1.43e+10	0.8%
▶ sincos	1.42e+10	0.8%	1.42e+10	0.8%
▶ G4VEmProcess::PostStepGetPhysicalInteractionLength(G4Track const&	4.35e+10	2.5%	1.34e+10	0.8%
▶ G4CrossSectionDataStore::GetIsoCrossSection(G4DynamicParticle con	1.04e+11	6.0%	1.34e+10	0.8%
▶ G4SteppingManager::InvokeAlongStepDoltProcs()	1.47e+11	8.5%	1.31e+10	0.8%

120M of 4079M

HPCToolkit screenshot showing the most expensive procedures in cmsExpMT (GEANT4 10.0.beta, GCC 4.6.3).

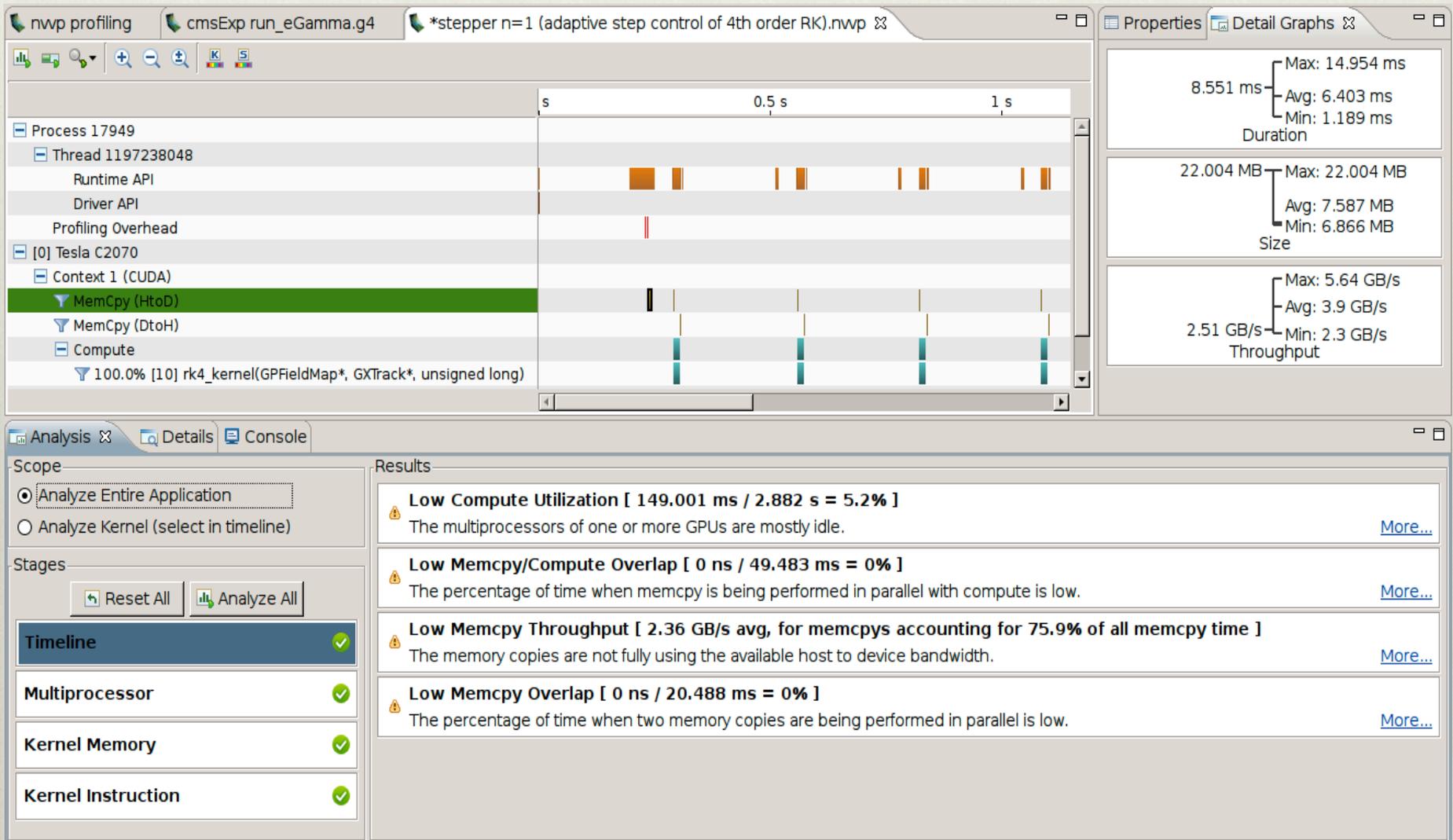
Note that the IEEE transcendental functions are called from many sites each. The other routines have few callers.

Memory Hierarchy

- ❖ In general, the instruction cache miss rates are found to be reasonable and do not constitute a bottleneck. There are a few sections of the code that exhibit significantly higher rates, but these routines represent a miniscule part of the total time.
- ❖ Data cache miss rates are, in general, low enough to not constitute a hot spot.
- ❖ The “Cross Sections” and “Isotope” classes have loops that do table lookups with higher miss rates. In aggregate, these routines make a non-negligible contribution to execution time.

Overall Analysis

- ❖ The general result is that when compiled correctly, cmsExpMT with geant4_mt_proto.9.5.p01 has no significant computational hot spots and cache usage is efficient.
- ❖ “Hot spot” is being used in the sense of “a small section of code that makes a large contribution to the cost of execution”.
- ❖ Due to its object-oriented design, GEANT4’s costs are diffused across a broad set of classes and deep call chains. There are hot functional areas, however, such as computing the physical interaction lengths as part of the process of stepping along tracks.



Performance profile of the GPU implementation of the 4th-order Runge-Kutta electromagnetic field integrator.

GPU Performance

- ❖ GPU performance analysis and tuning of the RK4 integrator
- ❖ Potential for exploiting greater concurrency through multiple streams and better overlap of memory transfers and computation
- ❖ Work in progress to generate and autotune portions of the kernel implementations
- ❖ One parameter to play with is the number of register used limit the number of warps you can have.
- ❖ Results from Nvidia's Insight are a bit cumbersome. Looking at Tau as an alternative.
- ❖ Working on analysis scripts based on the measurements and looking for ways to more accurately associate performance information with source code in the presence of aggressive inlining.

Electromagnetic Code Review

- ❖ Scope and Initial Plans
 - ❖ Review of performance aspects of a subset of ElectroMagnetic (EM) and closely related classes of Geant4 code with the initial goal to assess if the code is written in a computationally optimal way and to see if it could be improved, keeping in mind however code
 - ❖ correctness, performance, maintainability and adaptability
 - ❖ Multi-Threading aspects,
 - ❖ potential issues related to parallelization and/or migration to GPU
 - ❖ issues or potential improvements related to future migration to C++11
 - ❖ The review should initially concentrate on the most costly classes and functions
 - ❖ After the initial phase it may be needed or useful to expand the scope of the review to other related areas or aspects of the code.

Team

- ❖ Participants: John Apostolakis/CERN, Andrea Dotti/SLAC, Krzysztof Genser/FNAL, Soon Yung Jun/FNAL, Boyana Norris/ANL/now at Univ. of Oregon
- ❖ Team members backgrounds and experiences cover various aspects of Geant4 and Computer Science
 - ❖ Geant4 itself
 - ❖ C++, source code analysis/transformation, performance tools, performance analysis, optimization
 - ❖ Profiling/Benchmarking
 - ❖ MultiThreading/GPU/Parallel code
- ❖ Mix of High Energy Physics and Computer Science backgrounds allows for interdisciplinary knowledge exchange and feedback also related to enhancement of code tuning and analysis tools
 - ❖ Thanks to the support of US DOE for a High Energy Physics (HEP)-Advanced Scientific Computing Research (ASCR) team

Areas covered so far

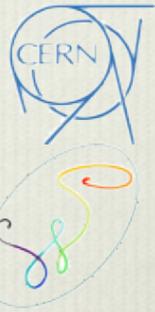
- ❖ Created a list of functions using a significant amount of CPU
- ❖ Initially concentrated on commonly used classes and especially the data structures they contain
 - ❖ G4PhysicsVector, esp. (Compute)Value and underlying classes
 - ❖ G4Physics2DVector
- ❖ Started looking at G4VEmProcess, one of the main classes
- ❖ We have settled on using SimplifiedCalo as the executable which performance we analyze to study the effects of the transformations we undertake;
 - ❖ it allows us to concentrate on the EM code related processes and to minimize other effects

Current Findings and Plans

- ❖ Changing underlying data structures may have an impact bigger than the fraction of the CPU taken by the functions using them
- ❖ Using Standard Library algorithms and compiler supplied functions should help simplify and optimize the code.
- ❖ Plan to present detailed findings and plans at the collaboration meeting to receive feedback and then review the remaining main classes.
- ❖ Also expect to learn more about code analysis and tuning tools and help to improve them.

ASCR Collaboration

- ❖ Confirmation of our previous analysis
- ❖ Build a collaboration and dialogue
 - ❖ Learn each other's language domain
- ❖ Improve existing tools to better fit our needs
 - ❖ Different scale, complexity and focus that they were used too
- ❖ On going efforts/reviews
 - ❖ EM Physics, cross section calculations, alternative track stacker, deep call chains.



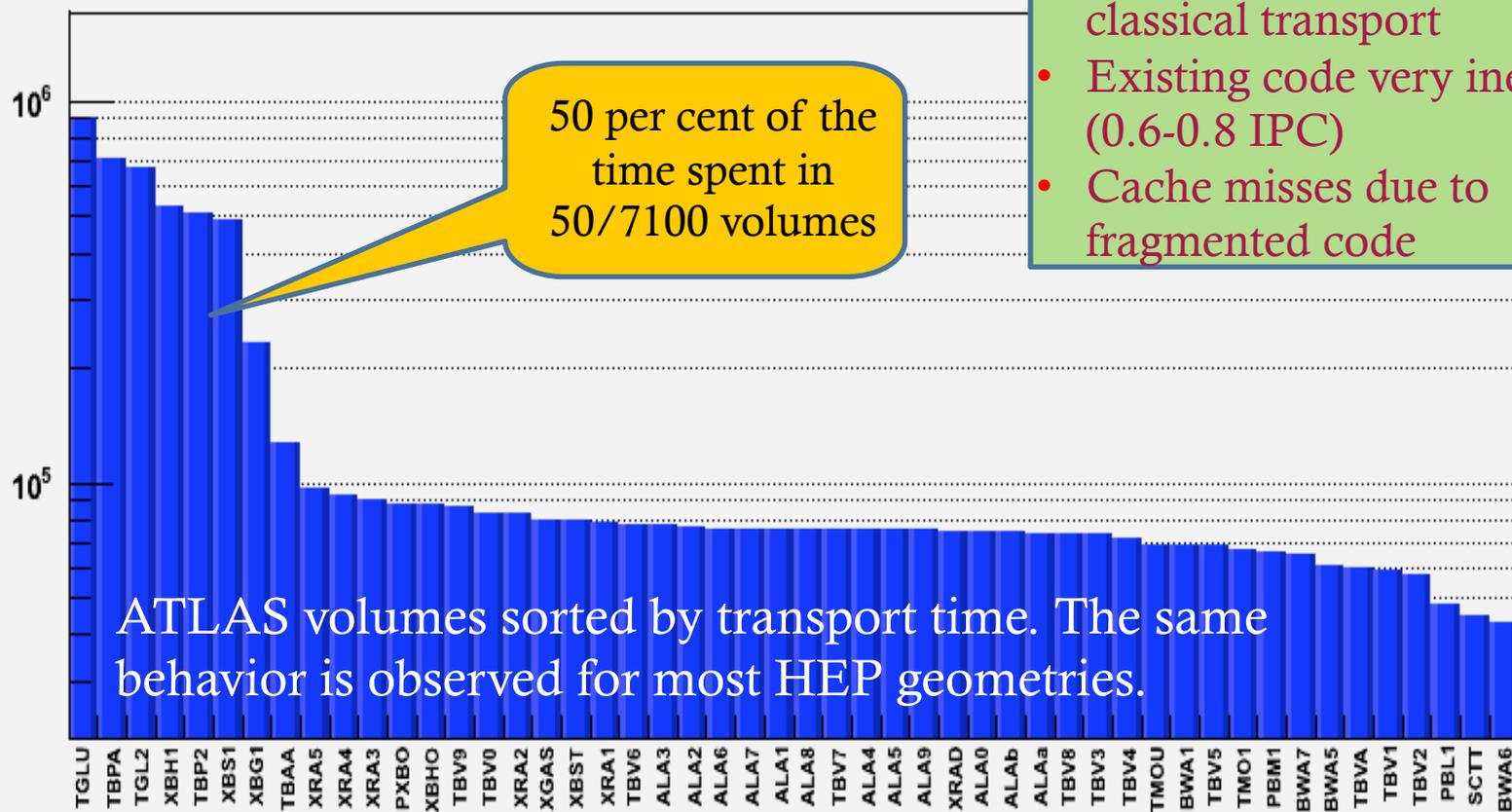
Geant Vector Prototype

- ❖ Grand strategy
- ❖ Explore from a performance perspective, no constraints from existing code
- ❖ Expose the parallelism at all levels, from coarse granularity to micro-parallelism at the algorithm level
- ❖ Integrate from the beginning slow and fast simulation in order to optimise both in the same framework
- ❖ Explore if-and-how existing physics code (GEANT4) can be optimised in this framework Improvements (in geometry for instance) and techniques are expected to feed back into reconstruction



HEP transport is mostly local!

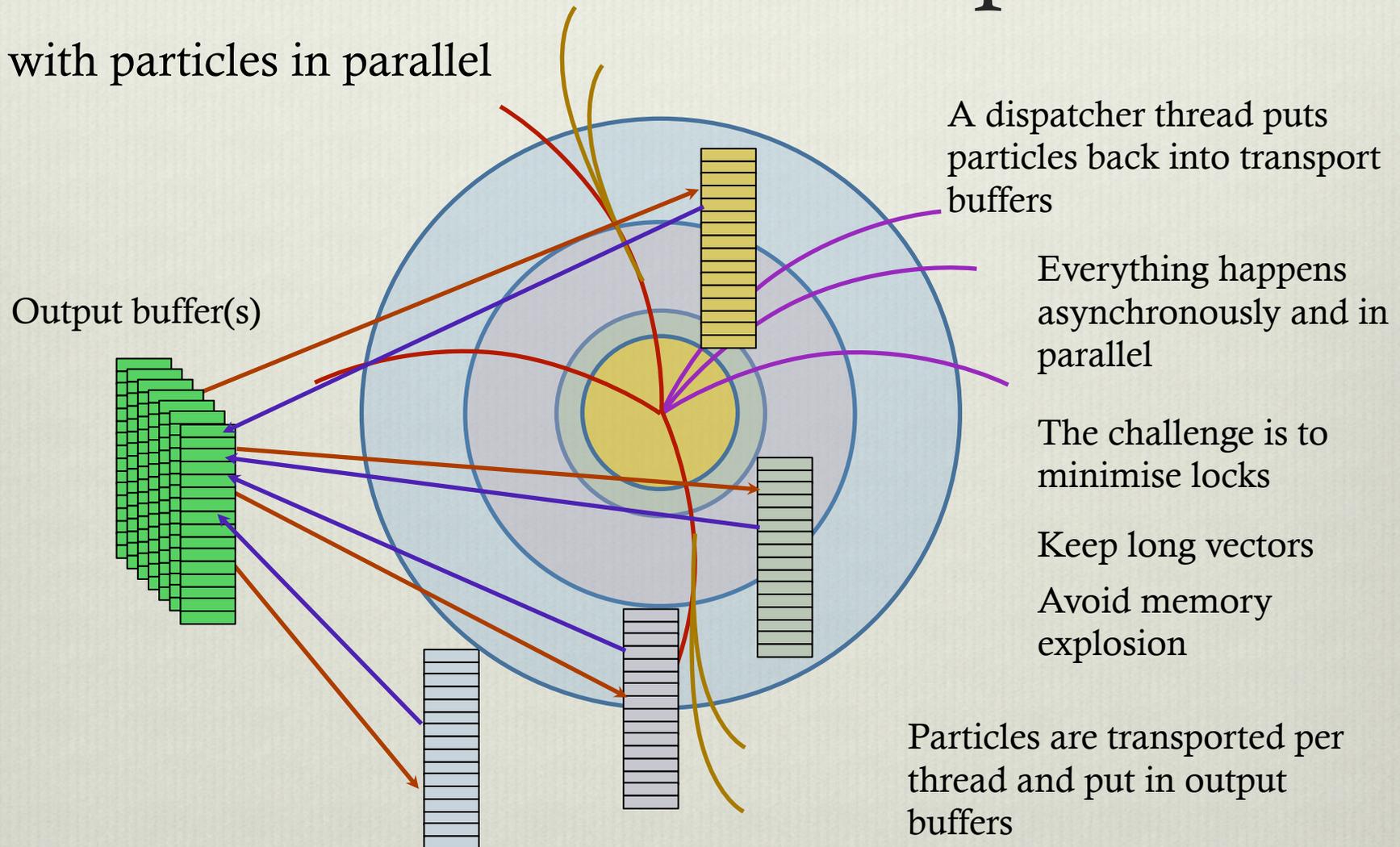
entries per volume sorted



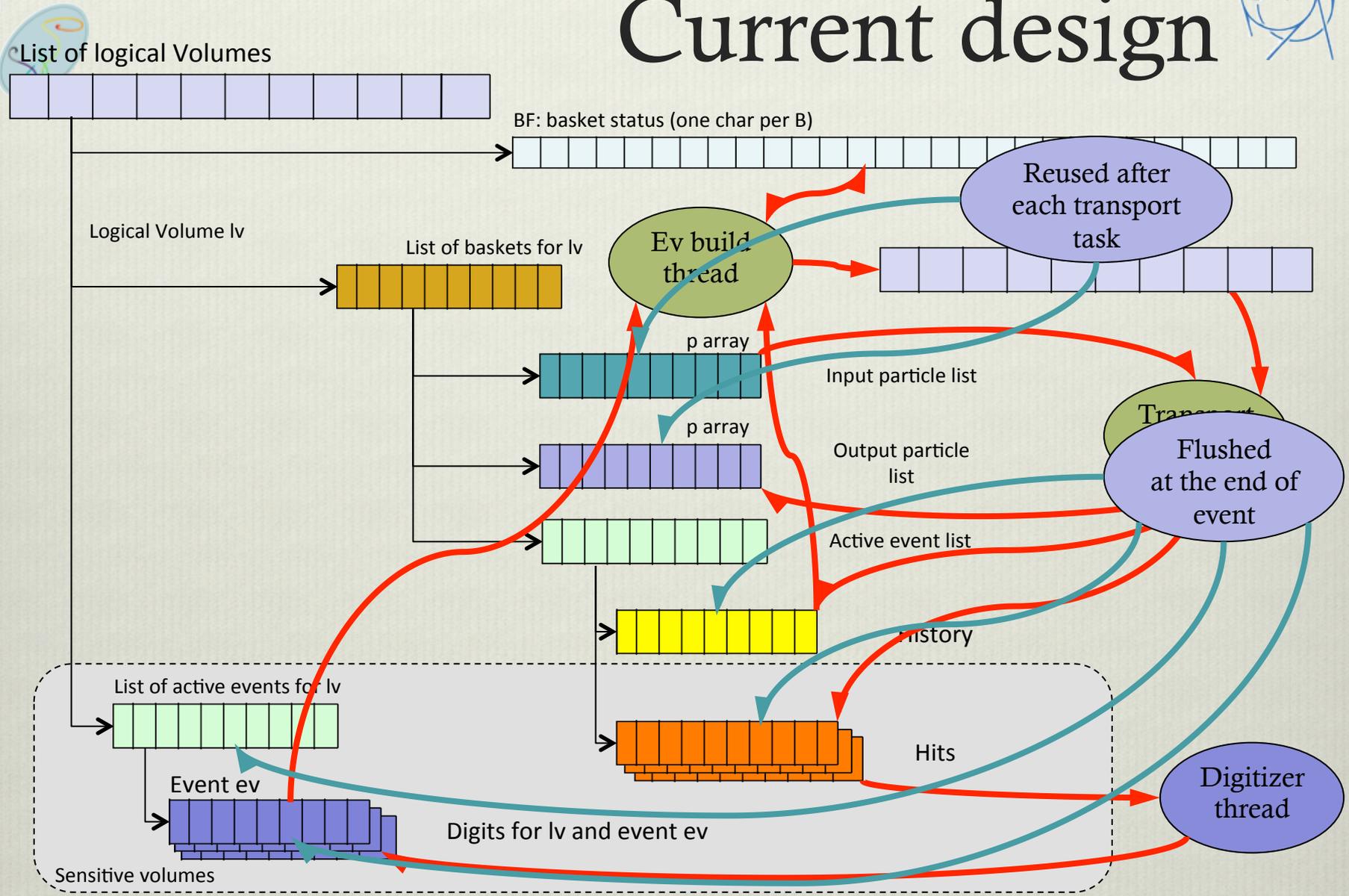


“Basketised” transport

Deal with particles in parallel



Current design



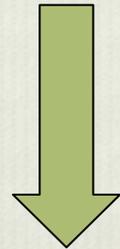


Grand strategy

- This will give better code anyway even for simple architectures
- e.g. ARM CPUs

Simulation job

Create vectors

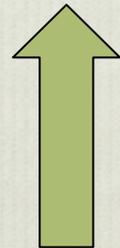


Locality is prepared here

And it is exploited here

- Algorithms will be more appropriate for one or the other of these techniques
- The idea being to expose the maximum amount of parallelism at the lowest possible granularity level

Use vectors



Basic algorithms

And then explore the optimisation opportunities

- The real gain in speed will come at the end from the exploitation of the (G/C)PU hardware
- Vectors, Instruction Pipelining, Instruction Level Parallelism (ILP)



Vector processing: Update on Gains for Geometry Calculations

- ❖ Motivation: How much can geometry navigation gain from vector processing of particles?
- ❖ benefit from SIMD instruction sets (see talk by S. Wenzel 5.6.2013)
- ❖ benefit from instruction cache reuse
- ❖ To address second point, developed a more systematic benchmark scheme to quantify gains from instruction cache reuse (no code changes necessary)
- ❖ For any shape/volume, benchmarker creates automatic test cases (tracks) and probes geometry performances for varying number of particles

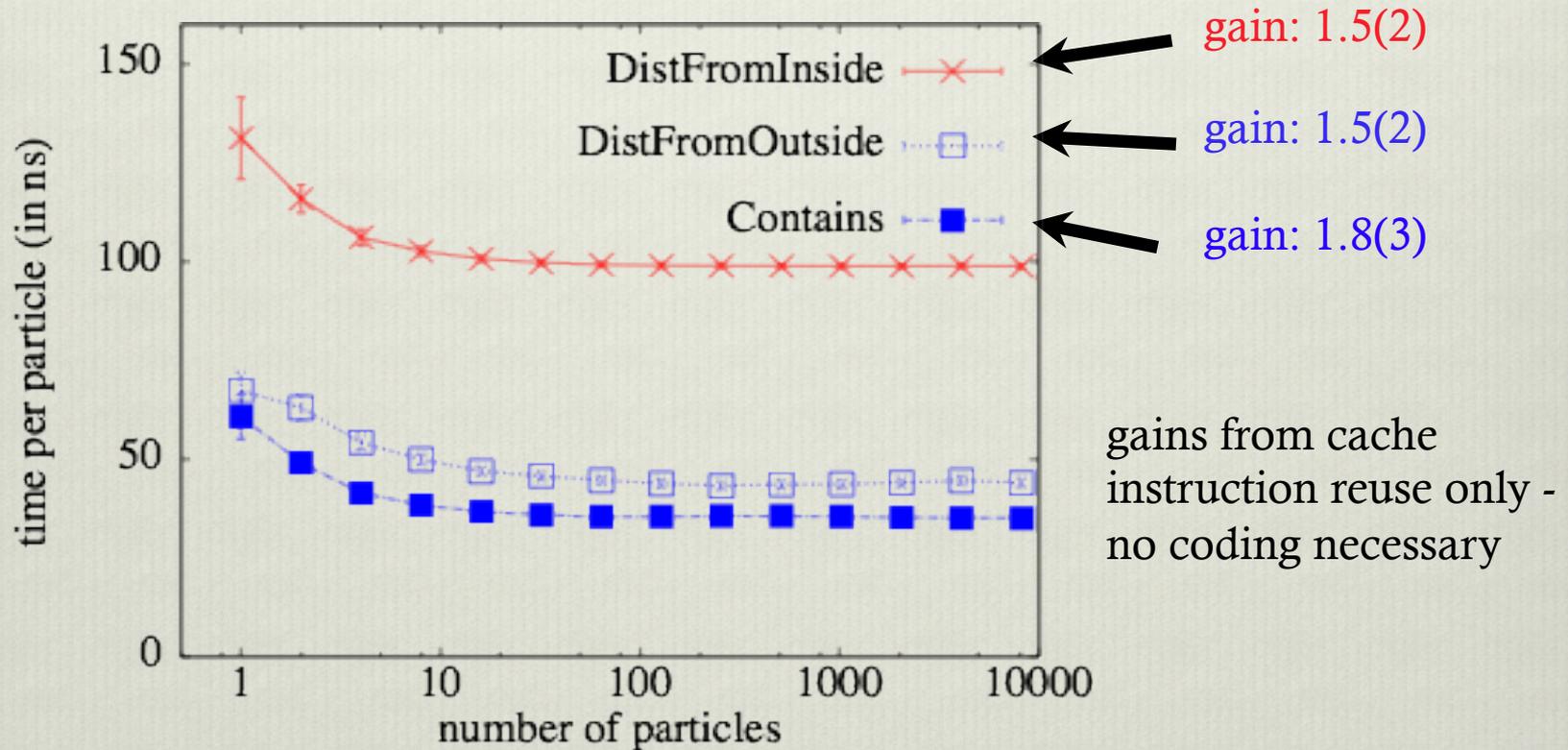
(slide by S. Wenzel)



Vector processing: Update on Gains for Geometry Calculations

- ❖ Result for realistic shape: TGeoPcon (volume 2) from CMS detector (testing Root geometry, compiled with -O3)

preliminary!



(slide by S. Wenzel)



Vector processing: Update on Gains for Geometry Calculations

❖ Overview of max speedup for various shapes

preliminary!

	Box	Polycone	Cone	ConeSeg
Safety	10 (3)	2.4 (3)	2.0 (4)	2.7 (4)
DistFromIn	2.2 (3)	1.5 (2)	1.7 (2)	1.4 (1)
DistFromOut	1.9 (2)	1.5 (2)	1.6 (2)	1.4 (1)
Contains	7 (2)	1.8 (2)	6 (1)	3.1 (4)

- Many different factors (for segments less gain?)
- These factors are trivial gains: more factors from SIMD expected !

(slide by S. Wenzel)



Update on SIMD optimizations: Test of the Vc library

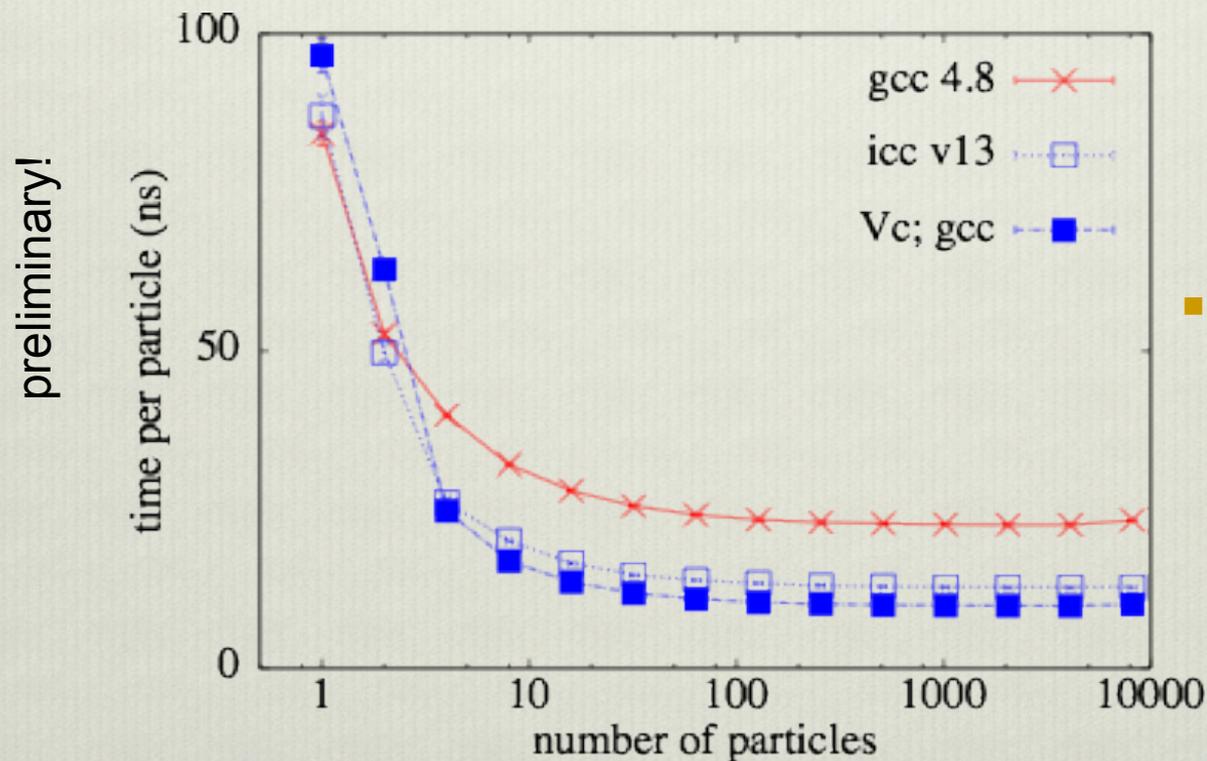
- ❖ In addition to benefit from cache instruction reuse, like to use vector instruction sets (SIMD)
- ❖ First good result obtained for Box geometry, relying so far on compiler autovectorization (additional gains up to factor 4)
(see talk by S. Wenzel 5.6.2013)
- ❖ However: SIMD autovectorization difficult to achieve
- ❖ Alternative: explicit vectorization approach:
 - ❖ intrinsics ?
 - ❖ (gcc) vector extensions ?
 - ❖ Vc library
 - ❖ compiler independent, high level constructs, abstraction of SIMD instruction set without overhead

(slide by S. Wenzel)



Update on SIMD optimizations: Test of the Vc library

- ❖ look at `Box::DistFromOutside` which did not autovectorize previously with gcc
- ❖ rather positive development experience
- ❖ first benchmark result (comparing Vc with autovec (gcc, icc) on AVX)



■ this is encouraging !!

(slide by S. Wenzel)



Physics

- ❖ We have selected the major mechanisms
- ❖ Bremsstrahlung, e+ annihilation, Compton, Decay, Delta ray, Elastic hadron, Inelastic hadron, Pair production, Photoelectric, Capture...
- ❖ We have already energy loss
- ❖ For each one of those and for $Z=1-100$ we will tabulate G4 cross sections (say $E=100\text{keV} - 1\text{TeV}$)
- ❖ For each reaction and each energy bin we generate 50 final states with G4
- ❖ When a reaction is selected
 - ❖ Select the set of final states closer in energy
 - ❖ Randomly pick a final state
 - ❖ Scale its CMS energy to the CMS energy of the actual reaction
 - ❖ Random rotate it around φ and roto boost according to the incoming particle



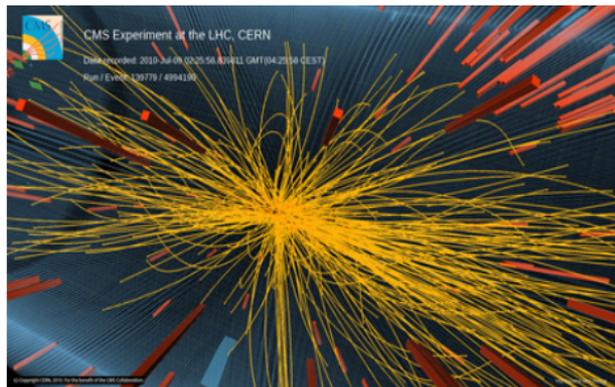


Advantages

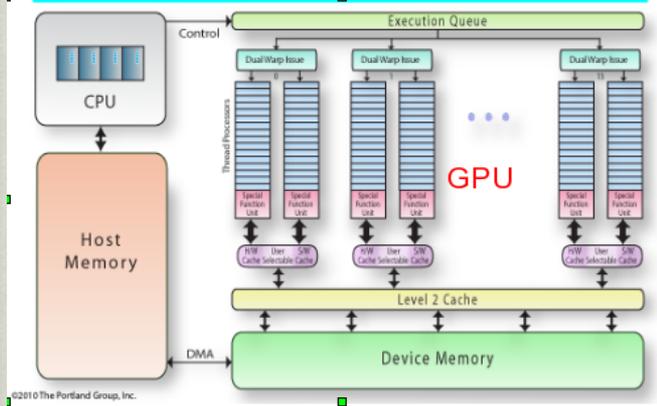
- ❖ This model with tables should be quite appropriate for vectorization.
- ❖ Data locality is optimised (cross-sections/per material/logical volume)
- ❖ This is also a very good model for a fast MC
- ❖ A probably a good alternative to calling G4-like routines for cross-sections and interactions if we increase the number of pre-computed interactions per bin (say from 50 to 200)
- ❖ Of course to be tested



Detector Simulation in GPU as a show-case



CUDA/OpenCL/OpenAcc



- ❖ Geant4 for detect simulation
 - ❖ highly sequential to reduce memory requirement (if-else)
 - ❖ event-level parallelism to take an advantage of using computing 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 arithmetic algorithms and efficient memory managements

Geant4

- ❖ Performance studies shows no ‘hot spot’ but overall non optimal use of current hardware
 - ❖ Relatively low instruction count per cycles
- ❖ New generation of hardware exacerbate the issues
 - ❖ GPU, Intel Xeon Phi
 - ❖ Deeper vector unit,
 - ❖ more (‘small’) cores
- ❖ Requires shift in implementation style



Problem Statement

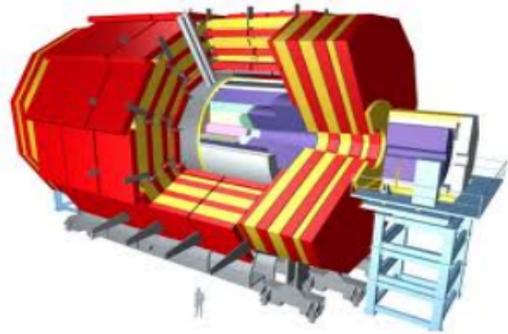
- ❖ Develop a massively parallelized EM particle transportation engine for many-core architectures
- ❖ Key components for a (GPU) prototype
 - ❖ transportation (in a realistic magnetic field)
 - ❖ geometry (a simple detector description)
 - ❖ EM physics (electrons and photons)
 - ❖ concurrent CUDA kernels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

GPU Prototype

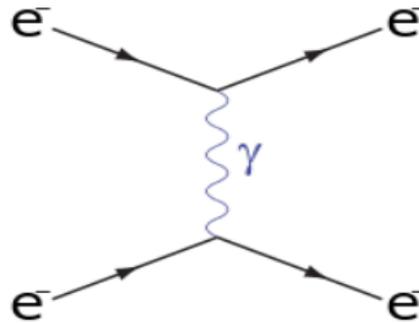
- ❖ Focus on Nvidia and CUDA
 - ❖ Testing cutting edge to understand potential
- ❖ Many parts originally from Geant4
 - ❖ *Particle Transportation*
 - ❖ *Electromagnetic Physics*
 - ❖ *Random Number Generator*
 - ❖ *Geometry Navigator on the GPU*
 - ❖ *Stepping Manager*
- ❖ *Validation Framework*
- ❖ *Kernel scheduling and CPU/GPU communication*



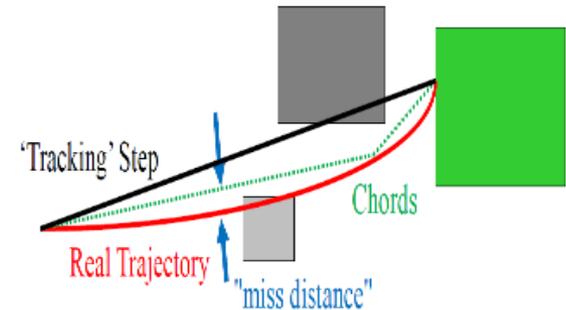
Overview of key components



Detector and Magnetic Field



EM Physics and pRNG

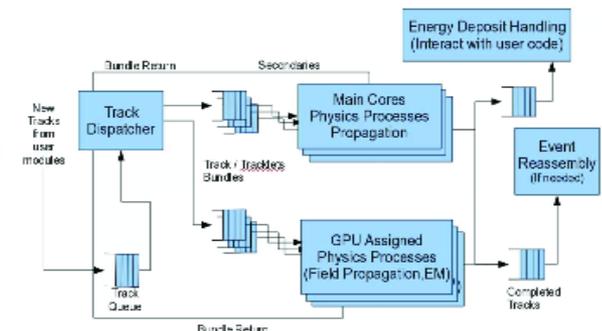
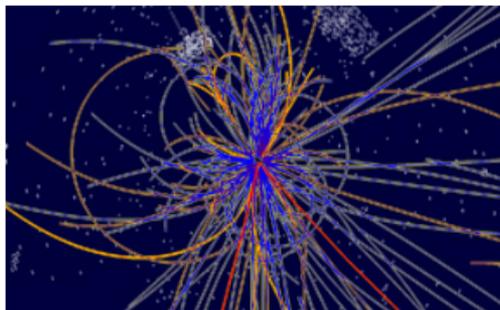


Navigation and Transportation

Primary/Secondary Particles

GPU Engine (CUDA C/C++)

Track Dispatcher



Overview of GPU Kernels

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

Performance

❖ 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
- ❖ $\text{Gain} = \text{Time}(1 \text{ CPU core}) / \text{Time}(\text{total GPU cores})$
 $\text{Time} = (\text{data transfer} + \text{kernel execution})$
- ❖ 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 \gg 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 I

- ❖ 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

- ❖ Decompose transportation by the particle type
 - ❖ separate kernels is $\sim 30\%$ faster for $\gamma : e^- = 0.2:0.8$ mixture

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

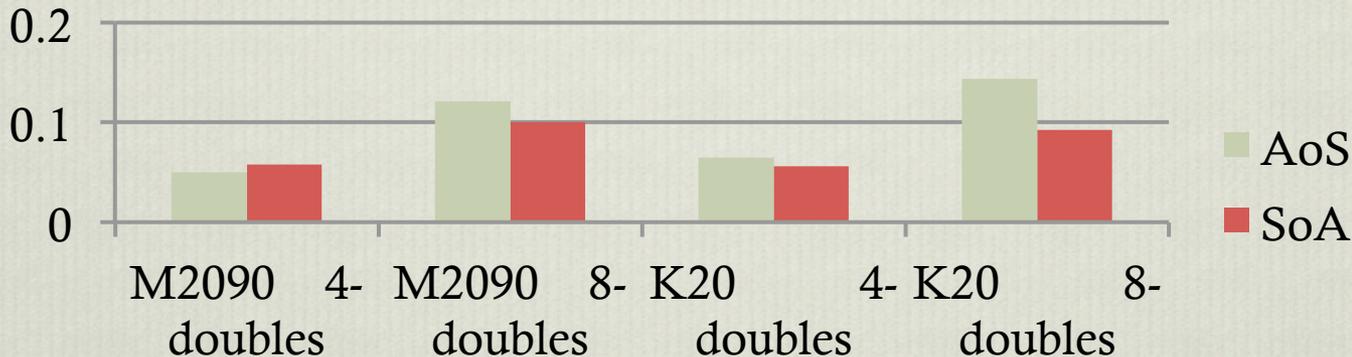
- ❖ EM physics processes and models require frequent data access from/to global memory
 - ❖ input: material information, physics tables
 - ❖ output: secondary particles (N=0,1,2 per step) and hits
- ❖ Memory transaction (atomic add) for 100K secondaries

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

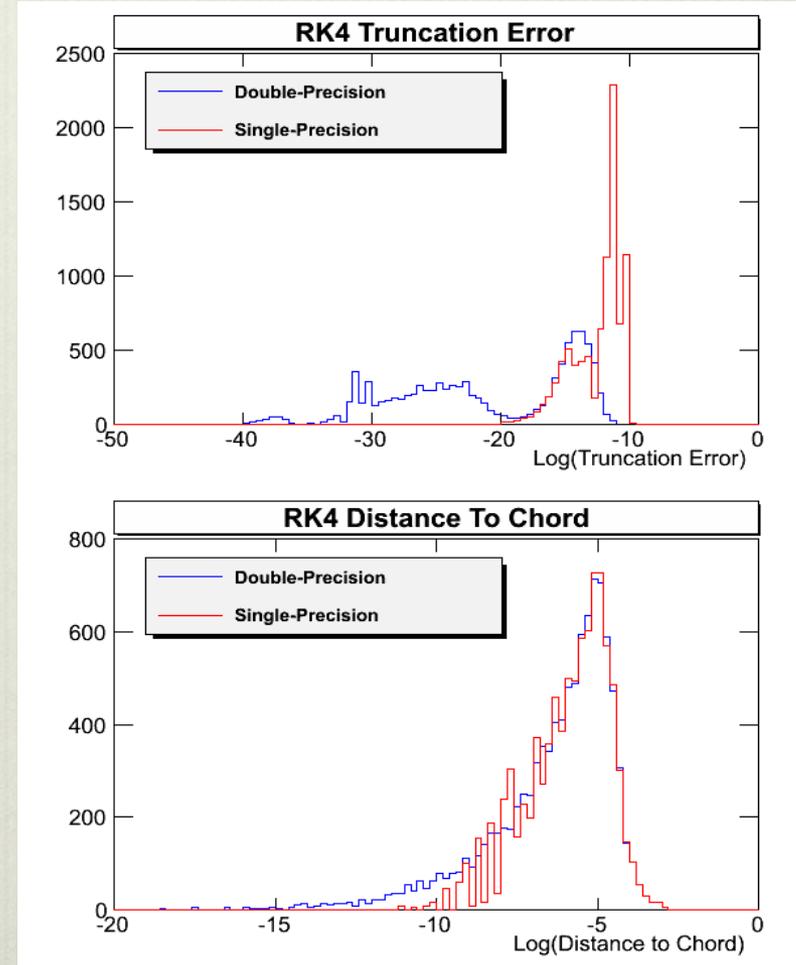
- ❖ Coalesced global memory access
 - ❖ align memory address for efficient data access
- ❖ Array of Struct (AoS) vs. Struct of Array (SoA)
 - ❖ a simple test of loading data (4-doubles, 8-doubles) and writing back to the global memory (65K accesses)



- ❖ 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

❖ 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
- ❖ $\text{Gain} = \text{Time}(1 \text{ CPU core}) / \text{Time}(\text{total GPU cores})$
 $\text{Time} = (\text{data transfer} + \text{kernel execution})$
- ❖ 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)

Host + Device	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`
- ❖ Multiple streams and concurrent kernels
- ❖ Validation
 - ❖ device codes vs. identical host codes (executed on CPU)
 - ❖ host codes vs. back-ported CPU codes

Geant VP – GPU Connector

❖ 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

Geant VP – GPU Connector

- ❖ 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

Outlook

- ❖ Early benchmarks showed GPU was half the cost of a single CPU to purchase *and* to operate for the same workload.

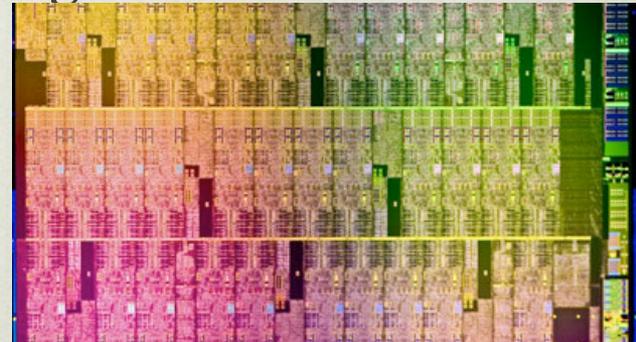
However we need to review this result as price per CPU core is seemingly dropping faster than GPU while keeping up in performance increase.

- ❖ Many collaborative projects on-going.

- ❖ Regular bi-weekly meetings, progress reports.

- ❖ Future

- ❖ Try out K20 and Intel Xeon Phi.
- ❖ Apply lessons from prototypes, reviews and performance analysis to recommend and implement (significant) improvements in Geant4.



Plans

- ❖ Continue integration with the vector prototype
 - ❖ discuss common plan at *Geant4 Workshop*
 - ❖ target to have a clear performance evaluation in 2014
 - ❖ share components (geometry, physics, transport, etc. – to be discussed)
- ❖ Redesign the GPU prototype optimally for SIMT/SIMD
 - ❖ minimize branches, maximize locality (instruction and memory)
 - ❖ data structure and algorithms for parallelism/vectorization
 - ❖ Generalize the GPU prototype for hybrid computing models (MIC, TBB, OpenCL)
- ❖ Extend validation framework
- ❖ Update cost-benefit analysis
- ❖ Leverage ASCR efforts on both the GPU and VP prototypes

Backup slides



Vector Prototype Conclusions

- ❖ Improving throughput for simulation requires rethinking the transport
- ❖ Better use of locality and improvement in the low level optimizations (caching, pipelining, vectorization)
- ❖ The blackboard exercise is moving into a fully functional prototype
- ❖ Most aspects of the new model understood, still many ideas to test and benchmark



Milestones

- ❖ Feb, 2013: Concurrency Annual Meeting @ FNAL
- ❖ Mar, 2013: Ramp-up collaboration with ASCR
- ❖ Summer
 - ❖ EM Physics code review
 - ❖ Working GPU/Vector prototype chain
- ❖ Early 2014:
 - ❖ semi-realistic benchmarks of Vector and GPU prototypes using simple geometry and small sets of physics processes.

