HEP-ASCR R&D Effort Towards Geant4 Re-engineering

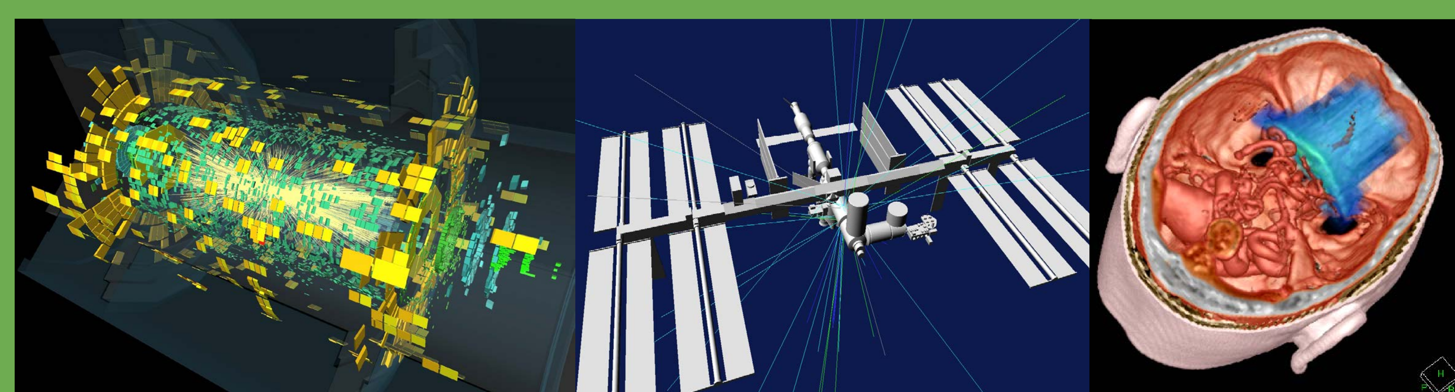
Fermilab · Argonne National Laboratory · RENCI · U.S. Department of Energy · SLAC National Accelerator Laboratory · USC Viterbi School of Engineering

Boyana Norris, Paul Ruth, Robert Fowler, Pedro Diniz, Robert Lucas, Makoto Asai, Andrea Dotti, Richard Mount, Philippe Canal, Daniel Elvira, Soon Yung Jun, James B. Kowalkowski, Marc F. Paterno, Robert M. Roser

Geant4

Geant4

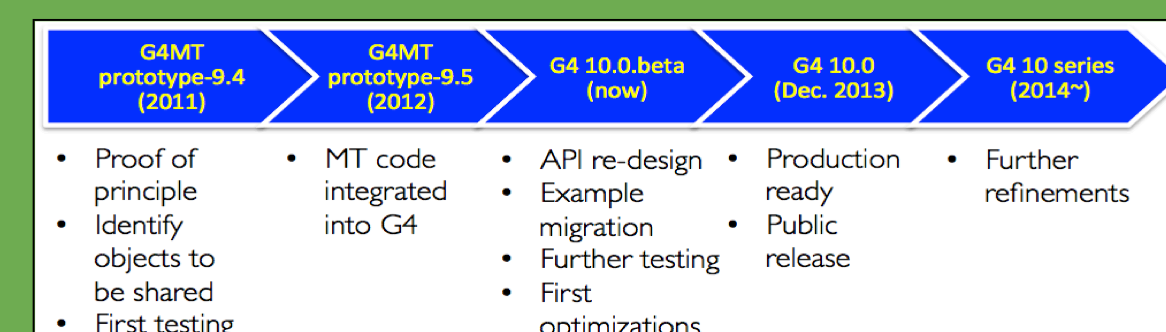
Geant4 is a toolkit for the simulation of particles passing through and interacting with matter. Its areas of application include high energy, nuclear and accelerator physics, as well as studies in space science, material science, medicine and biology, and also in security and industrial applications. In high energy physics, the Geant4 toolkit fulfills a critical need for the simulation of detectors at the LHC and at other existing and future experiments and facilities.

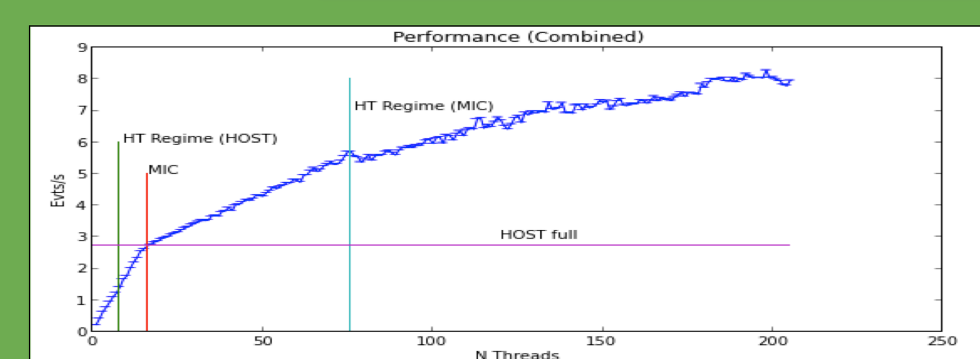


Geant4 is maintained and further developed by an international collaboration, which consists of more than 100 physicists and computer scientists. U.S. involvement in the development of Geant4 has been substantial since its early stages, and has increased with time. Several key Geant4 functionalities, including core framework, hadronic physics and visualization, are lead by the SLAC team, while major contributions in the key areas of hadronic physics and computing performance are made by the Fermilab team.

Geant4 version 10

High energy physics research as we know it today would not be possible without simulations. The massive production of event samples similar to those expected in the real experiment is an integral part of the process to design, build, operate highly complex accelerators and detectors and analyze the physics results. Thus Geant4-based simulation is currently the largest consumer of LHC compute cycles. In recent years, space and medicine have become significant user domains, with applications ranging from instrument and detector response verification to radiation dose and shielding optimization and analysis of biological effects. To address to such an increasing demand of high statistic simulations, the Geant4 Collaboration is releasing a new version of Geant4 (Geant4 version 10.0) in December 2013, which enables the use of multi-core CPUs and coprocessors in multi-threaded mode.

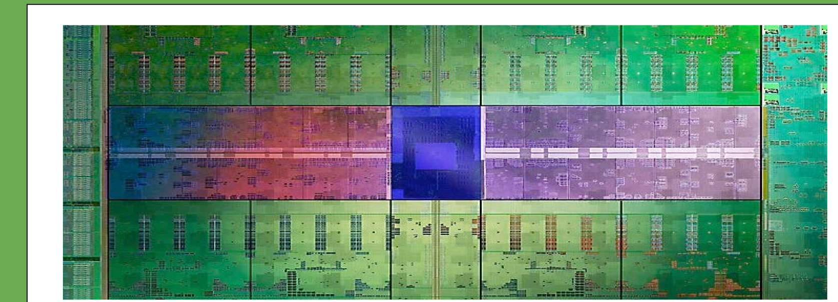




Preliminary scalability test of Geant4 version 10-pre-beta with full CMS detector geometry on 8-core Intel Xeon CPU and one Intel Xeon Phi coprocessor (60 core).
From left to right, CPU, CPU in hyper-thread, coprocessor, and coprocessor in hyper-thread.

HEP-ASCR R&D Effort

The process of re-engineering Geant4 has to be started, targeting on recently emerging new computing hardware such as many-core coprocessors and GPUs. Using these new architectures efficiently requires to develop to leverage massive parallelization, complex memory hierarchy and deep vectorization capability.
FNAL, SLAC, UNC, USC and ANL joined forces to launch an R&D effort to investigate the possible evolution of the software infrastructure and numerical algorithms of the Geant4 toolkit to utilize these emerging technologies.

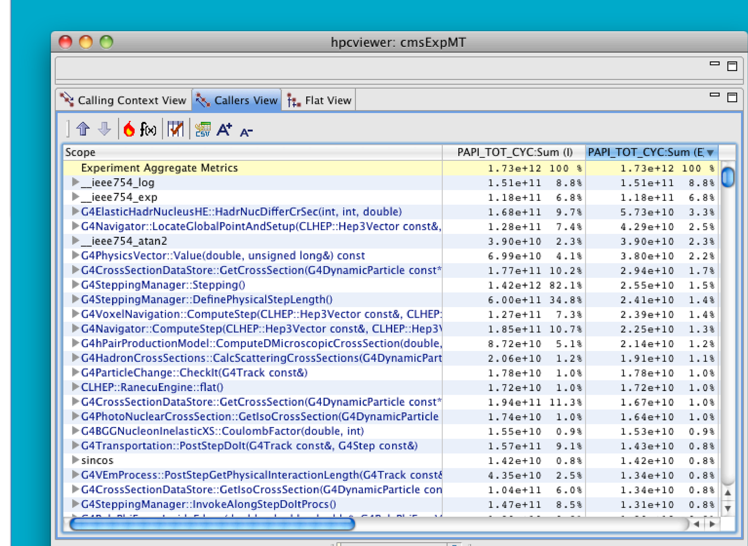


Nvidia Kepler Architecture

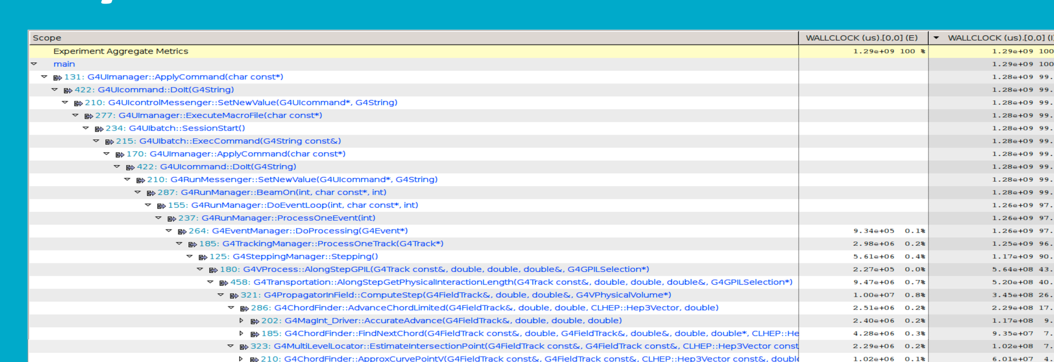
1) Performance analysis of the current Geant4 toolkit and its typical applications,
2) Reorganization of loops in the algorithms to make better use of vectorization
3) Prototyping GPU-based code
4) Studying automated code transformation for GPUs.

Investigations

Performance Analysis



HPCToolkit screenshot showing the most expensive procedures in cmsExpMT (GEANT4 10.0.beta, GCC 4.6.3). Note that the IEEE transcendentals are called from many sites each. The other routines have few callers.



HPCToolkit screenshot illustrating the deep call chains in the integrator.

CPU performance analysis of Geant4 and Geant4MT

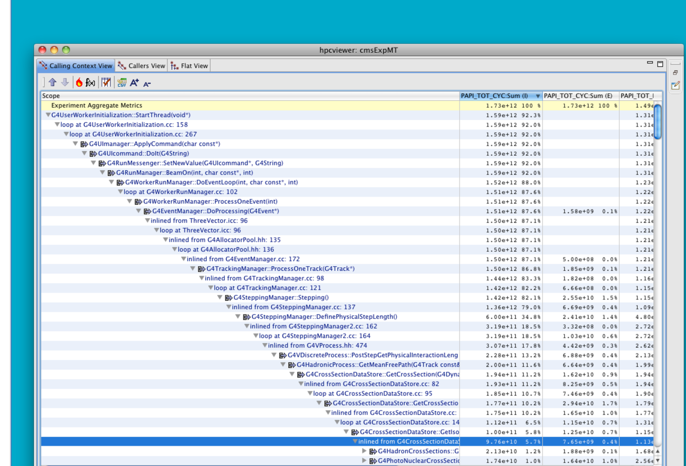
- Effects of different compilers and compiler options
- Callpath profiling of a CMS experiment benchmark (execution time, memory performance)

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

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.



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

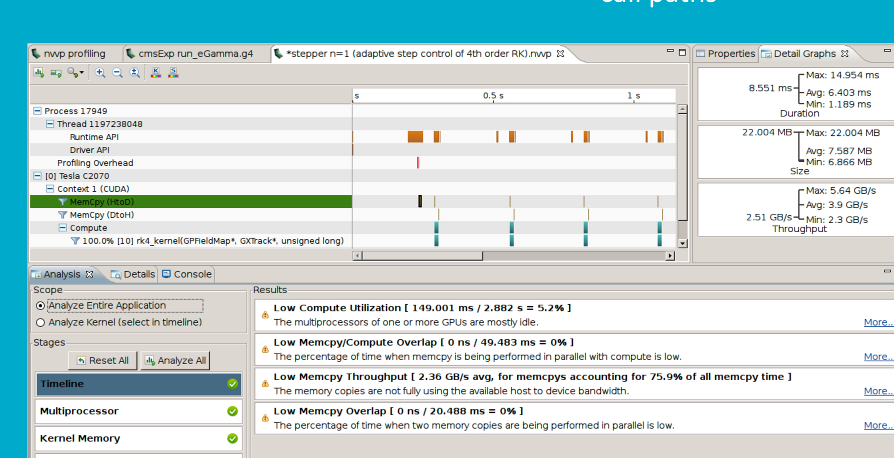
Choice of compiler

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.



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

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

Loops and Data Structure Reorganization

Hotspot analysis (HPCToolkit) revealed:

- Extensive use of Transcendental Functions
- No obvious hot spots
- Irregular threading overhead

Focus on Application Understanding

- Minor Performance Improvements due to manual tuning (1%) that has been already incorporated in version beta 10.0.

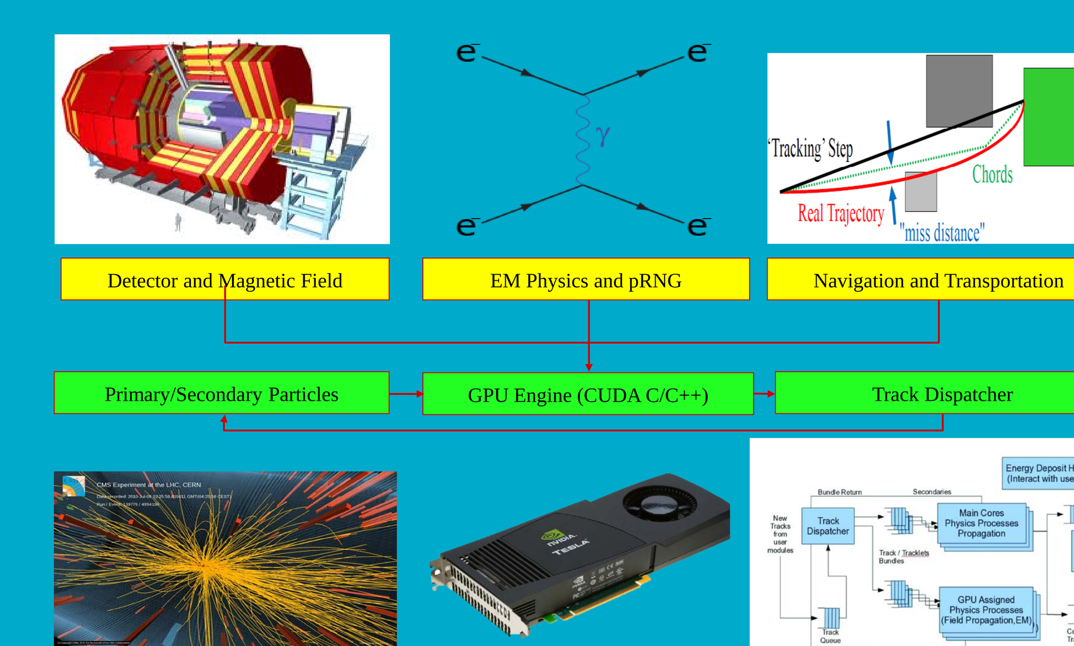
Future Investigations

- Exploit Materials collision Memoization
- Change stacking strategy to exploit coarse- and fine-grain *vectorization*

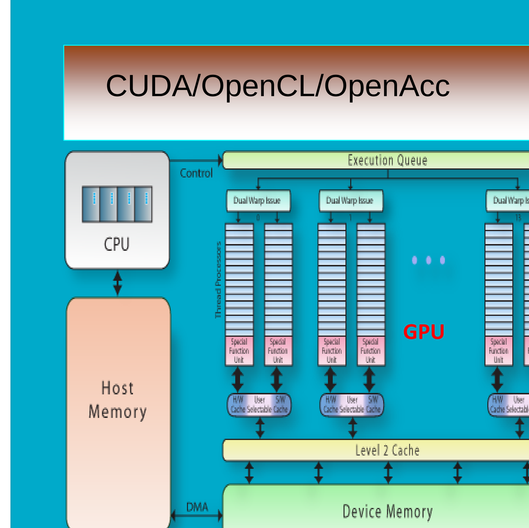
Memory Hierarchy

•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.

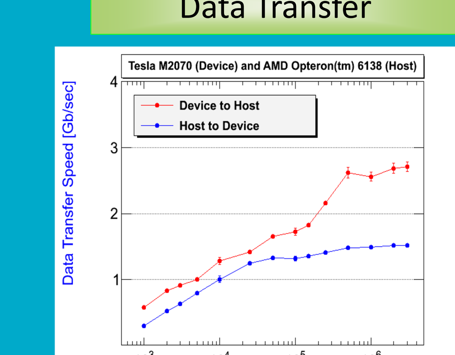
Prototype Running On GPU



- ***Geant4 HEP applications***
  - event-level parallelism
  - highly sequential
  - memory intensive
- **GPU-Vector**
  - track-level parallelism
  - vectorized track dispatche
  - coalesced memory access
  - work balance (CPU-GPU)

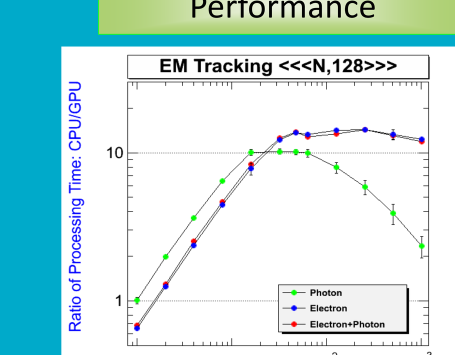


Data Transfer



e- <<<32,128>>>	CPU(ms)	GPU(ms)	Gain
Bremsstralung	2099	104	20
Ionization	558	25	22
Multiple Scattering	1034	185	6
Electron Kernel	751	61	12

Performance



γ <<<32,128>>>	CPU(ms)	GPU(ms)	Gain
Compton Scattering	51	6	8
Photo Electric Effect	70	6	12
Pair Production	50	10	5
Photon Kernel	71	7	10