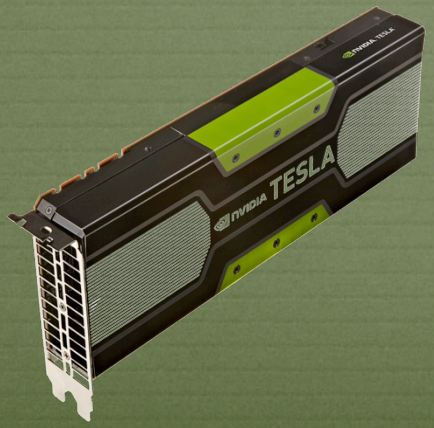
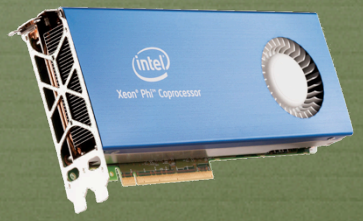
Geant R&D Projects

Ph. Canal
S.Y. Jun,
Guilherme Lima





Outline

- Context
- Collaborations
  - Performance Analysis
  - *EM Physics review*
  - Vector Prototype
- 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) fledge 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, RENCI, 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

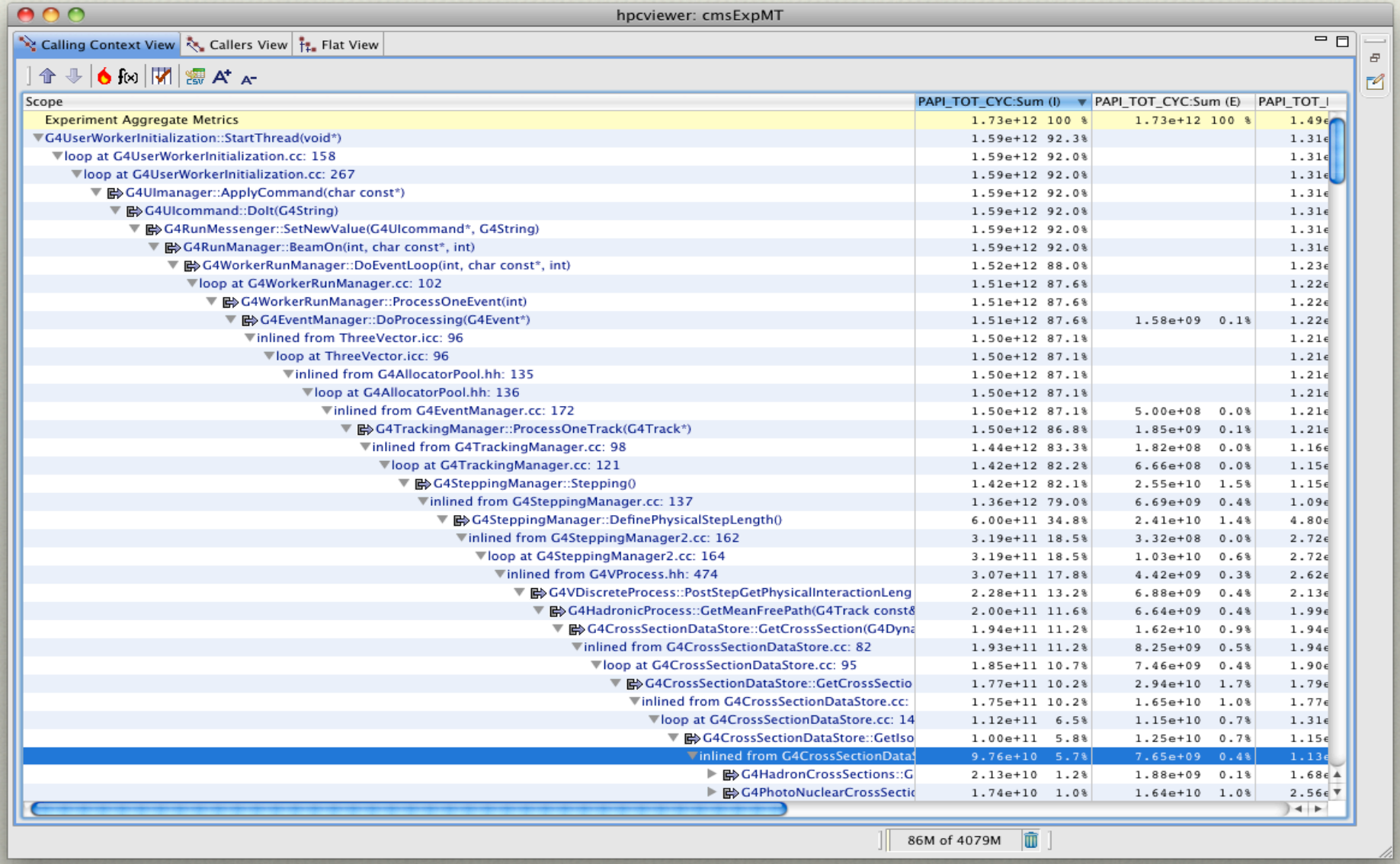
- Manual application of loop-invariant code motion to the main loop of the Event Manager. This resulted in a 1% performance improvement for real runs and has been already incorporated in version beta 10.0
- Inlining a specific function in the code calculating cross-section can gain approx. 1.5%
- Analysis 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.

Scope	WALLCLOCK (us).[0,0] (E)	WALLCLOCK (us).[0,0] (I)
Experiment Aggregate Metrics	1.29e+09 100 %	1.29e+09 100 %
main		1.29e+09 100 %
131: G4UImanager::ApplyCommand(char const*)		1.28e+09 99.3%
422: G4UIcommand::DoIt(G4String)		1.28e+09 99.3%
210: G4UIcontrolMessenger::SetNewValue(G4UIcommand*, G4String)		1.28e+09 99.3%
277: G4UImanager::ExecuteMacroFile(char const*)		1.28e+09 99.3%
234: G4UIbatch::SessionStart()		1.28e+09 99.3%
215: G4UIbatch::ExecCommand(G4String const&)		1.28e+09 99.3%
170: G4UImanager::ApplyCommand(char const*)		1.28e+09 99.3%
422: G4UIcommand::DoIt(G4String)		1.28e+09 99.3%
210: G4RunMessenger::SetNewValue(G4UIcommand*, G4String)		1.28e+09 99.3%
287: G4RunManager::BeamOn(int, char const*, int)		1.28e+09 99.3%
155: G4RunManager::DoEventLoop(int, char const*, int)		1.26e+09 97.1%
237: G4RunManager::ProcessOneEvent(int)		1.26e+09 97.1%
264: G4EventManager::DoProcessing(G4Event*)	9.34e+05 0.1%	1.26e+09 97.1%
185: G4TrackingManager::ProcessOneTrack(G4Track*)	2.98e+06 0.2%	1.25e+09 96.6%
125: G4SteppingManager::Stepping()	5.61e+06 0.4%	1.17e+09 90.8%
180: G4VProcess::AlongStepGPIL(G4Track const&, double, double, double&, G4GPILSelection*)	2.27e+05 0.0%	5.64e+08 43.6%
458: G4Transportation::AlongStepGetPhysicalInteractionLength(G4Track const&, double, double, double&, G4GPILSelection*)	9.47e+06 0.7%	5.20e+08 40.2%
321: G4PropagatorInField::ComputeStep(G4FieldTrack&, double, double&, G4VPhysicalVolume*)	1.00e+07 0.8%	3.45e+08 26.7%
286: G4ChordFinder::AdvanceChordLimited(G4FieldTrack&, double, double, CLHEP::Hep3Vector, double)	2.51e+06 0.2%	2.29e+08 17.7%
202: G4MagInt_Driver::AccurateAdvance(G4FieldTrack&, double, double, double)	2.40e+06 0.2%	1.17e+08 9.1%
185: G4ChordFinder::FindNextChord(G4FieldTrack const&, double, G4FieldTrack&, double&, double, double*, CLHEP::He	4.28e+06 0.3%	9.35e+07 7.2%
323: G4MultiLevelLocator::EstimateIntersectionPoint(G4FieldTrack const&, G4FieldTrack const&, CLHEP::Hep3Vector const	2.29e+06 0.2%	1.02e+08 7.9%
210: G4ChordFinder::ApproxCurvePointV(G4FieldTrack const&, G4FieldTrack const&, CLHEP::Hep3Vector const&, double	1.02e+06 0.1%	6.01e+07 4.6%
298: G4VIntersectionLocator::IntersectChord(CLHEP::Hep3Vector const&, CLHEP::Hep3Vector const&, double&, double	8.82e+05 0.1%	1.52e+07 1.2%

HPCToolkit screenshot illustrating the deep call chains in the integrator.

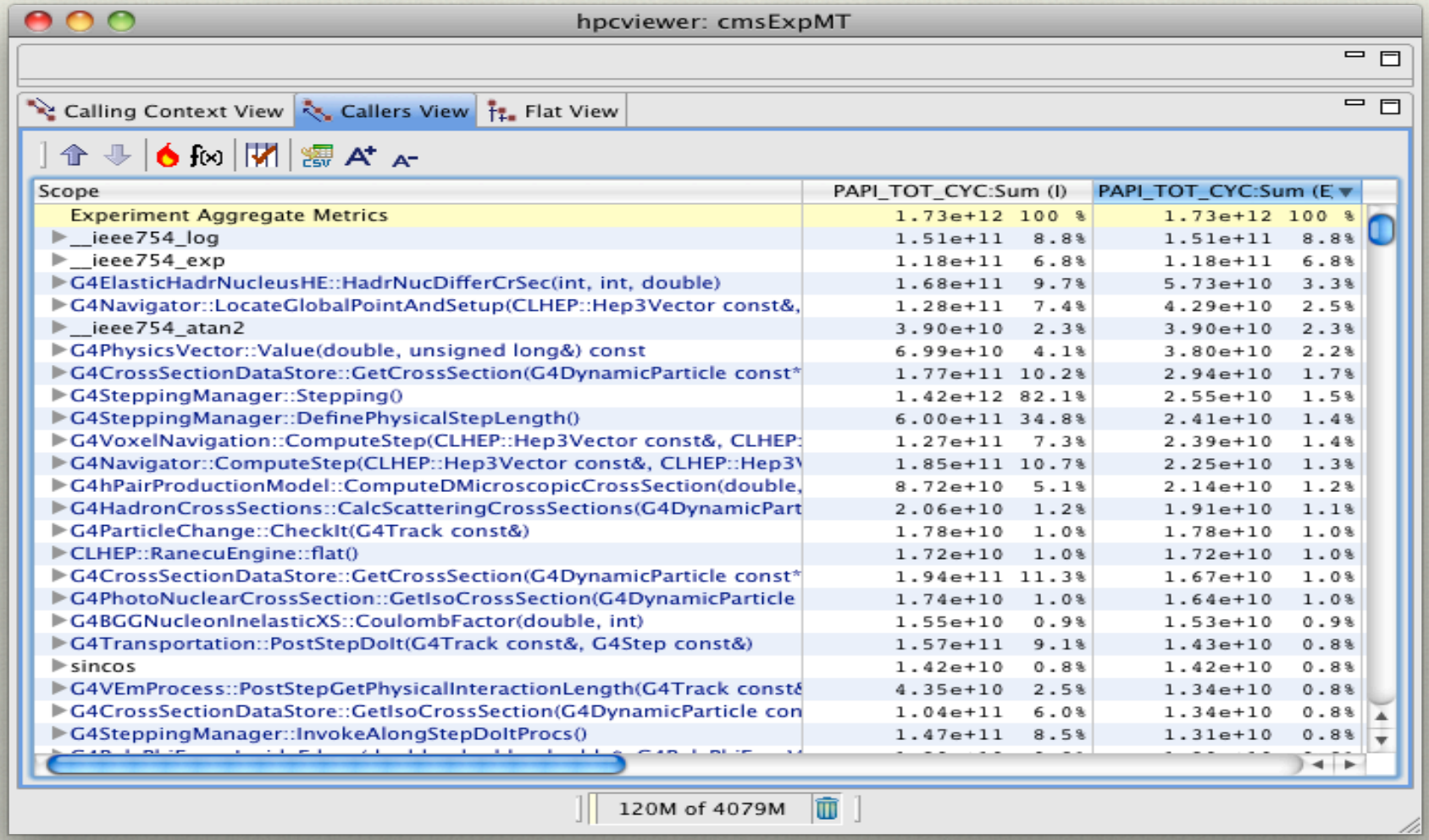


Depth of the “hottest” call chain.

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



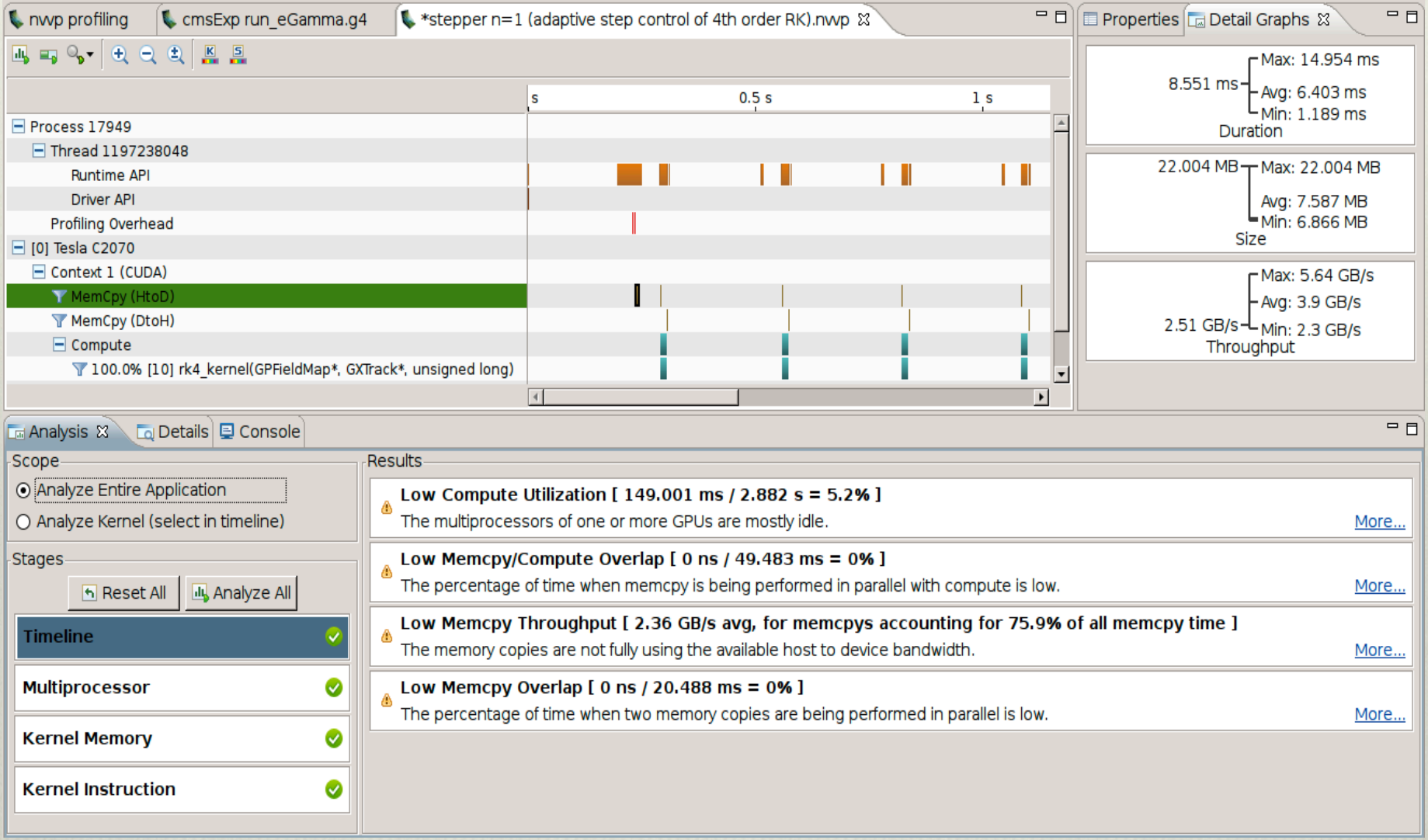
HPCToolkit screenshot showing the most expensive procedures in cmsExpMT (GEANT4 10.0.beta, GCC 4.6.3).
Note that the IEEE transcendental functions are called from many sites each. The other routines have few callers.

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 Nvida's Insight are a bit cumbersome. Looking atTau 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 minimizes 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.

Speedup Cross-Sections calc

- Finding which materials and isotopes info to cache.
  - seems that the best ones to track for hadronic processes would be Iron (Fe) or lead (Pb).
  - Materials do not change over the course of the simulation so it is better (more complex but more speed-up) to focus on materials rather than on isotopes.
- Next question is “which material to cache” and what will the savings really be.
- From a publication Andrea had access to, the preliminary functions we've recorded have similar properties, hence providing ample hope that it will be able to cache and/or interpolate the data for many of the more frequently occurring materials.
- Planning on using splay-tree and combine interpolation with calculation and caching.

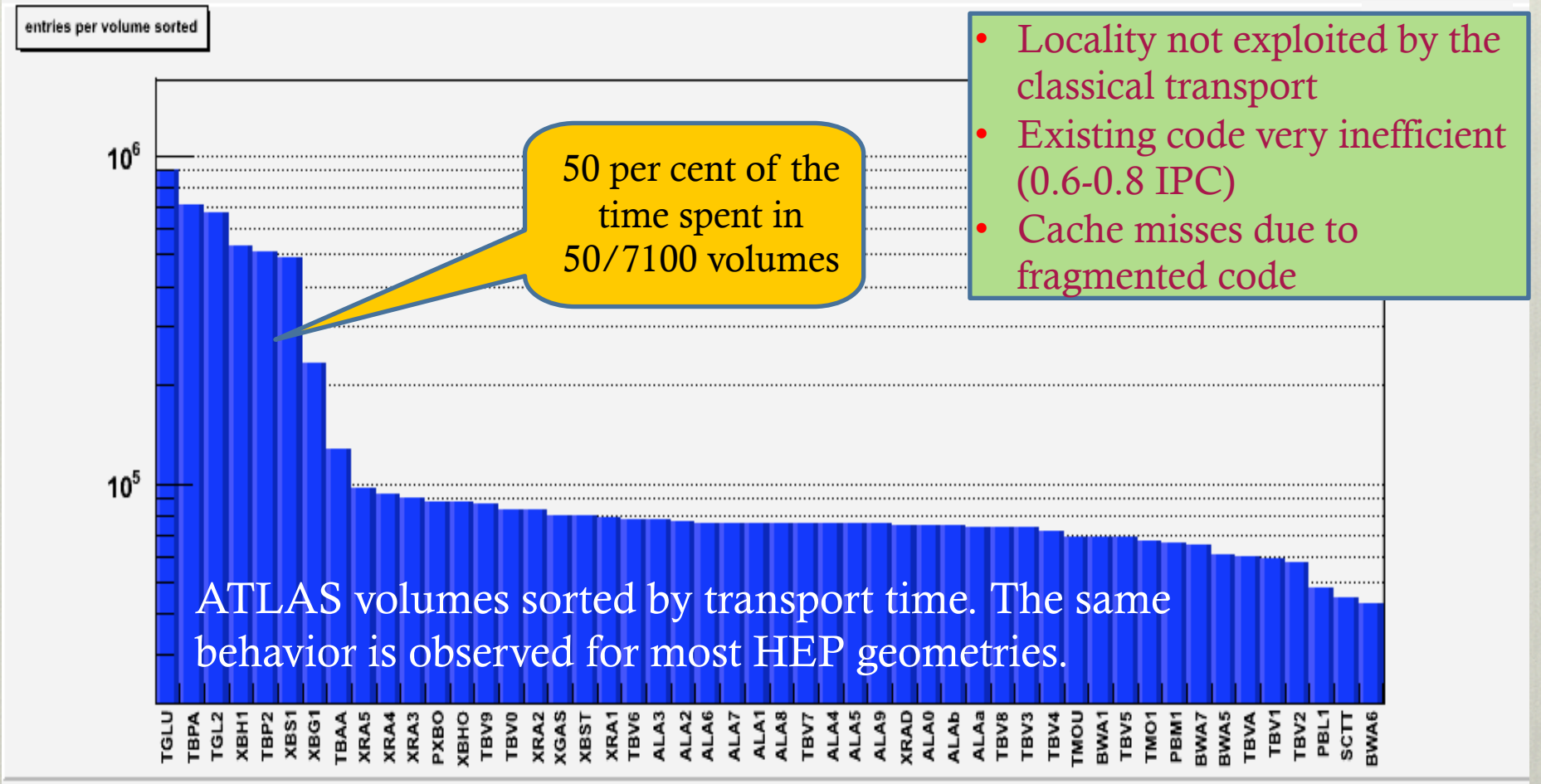
ASCR Collaboration

- 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

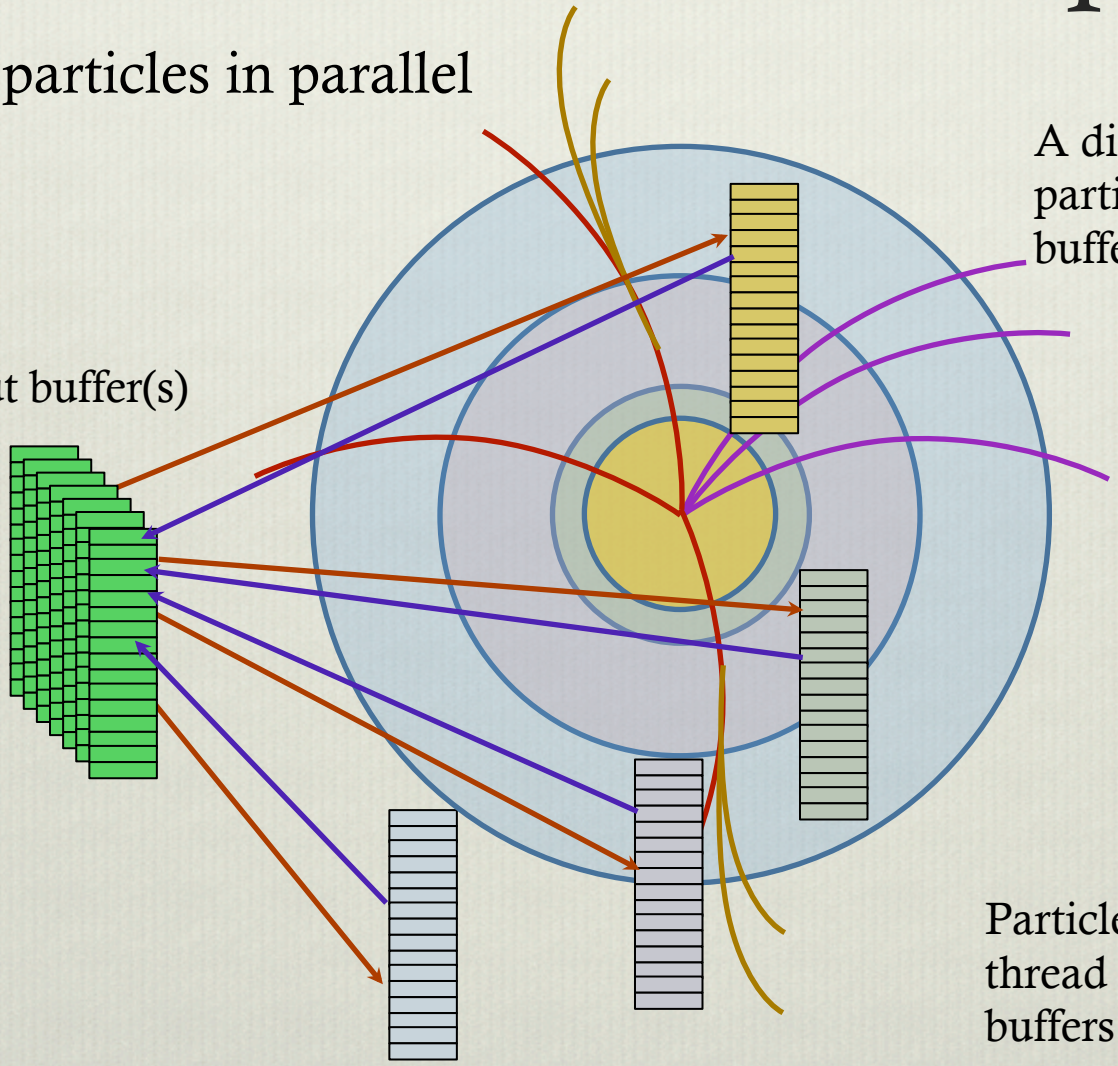
HEP transport is mostly local !



- Locality not exploited by the classical transport
- Existing code very inefficient (0.6-0.8 IPC)
- Cache misses due to fragmented code

"Basketised" transport

Deal with particles in parallel



A dispatcher thread puts particles back into transport buffers

Everything happens asynchronously and in parallel

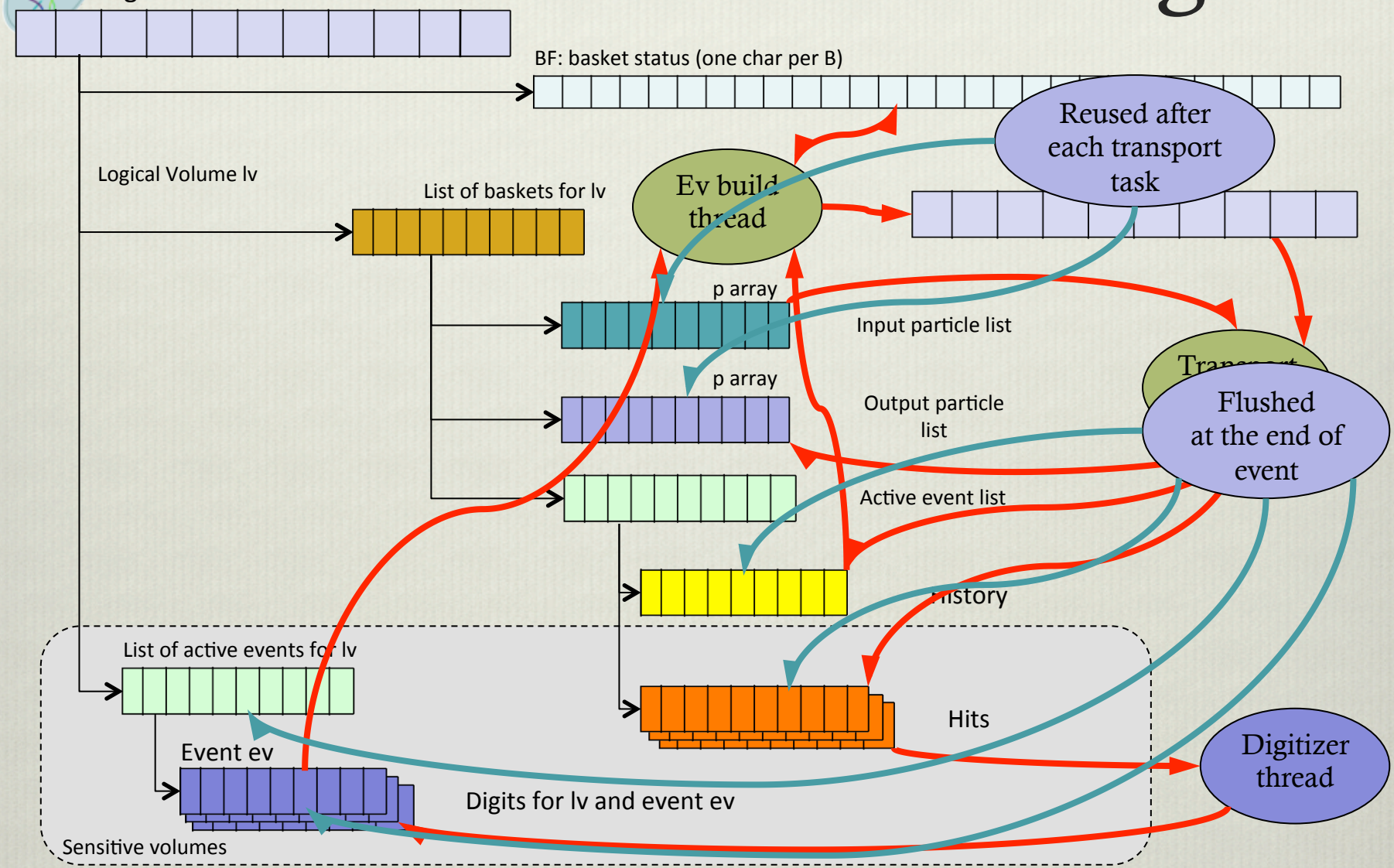
The challenge is to minimise locks

Keep long vectors

Avoid memory explosion

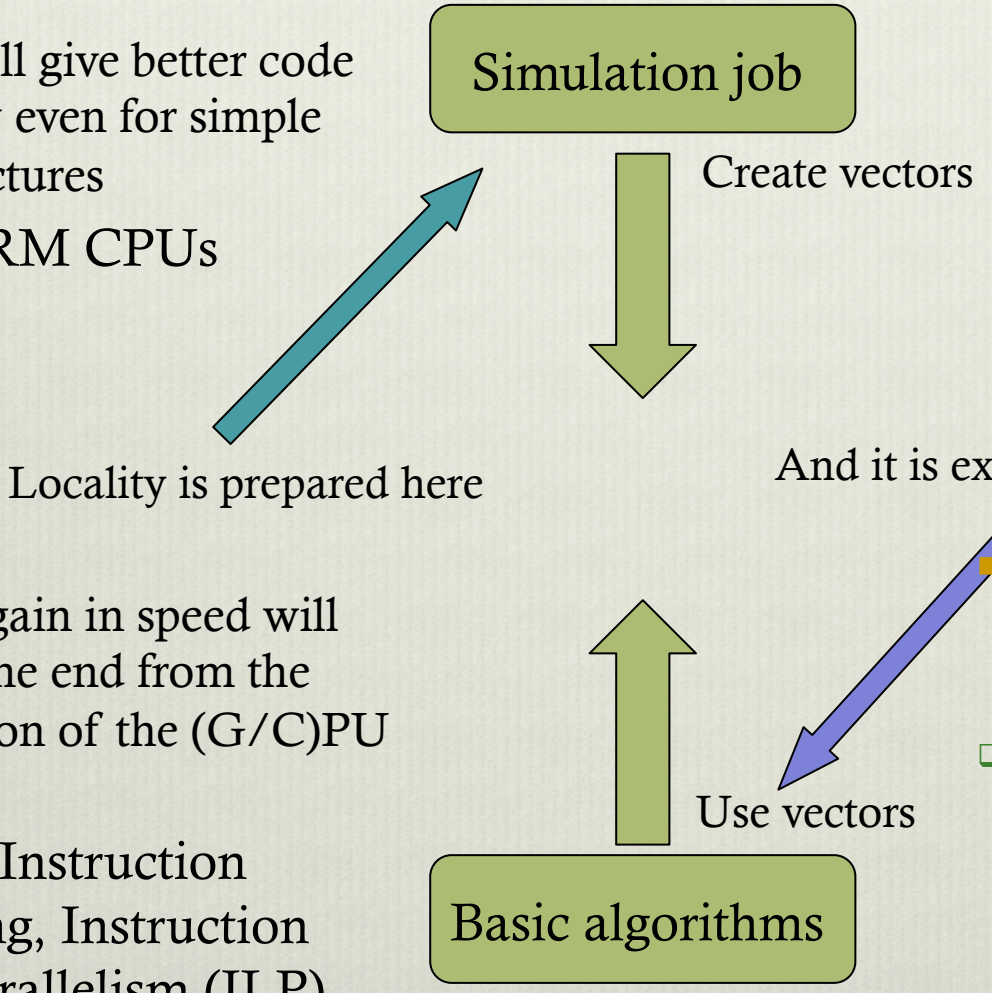
Particles are transported per thread and put in output buffers

Current design



Grand strategy

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



- 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)
- 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

And then explore the optimisation opportunities

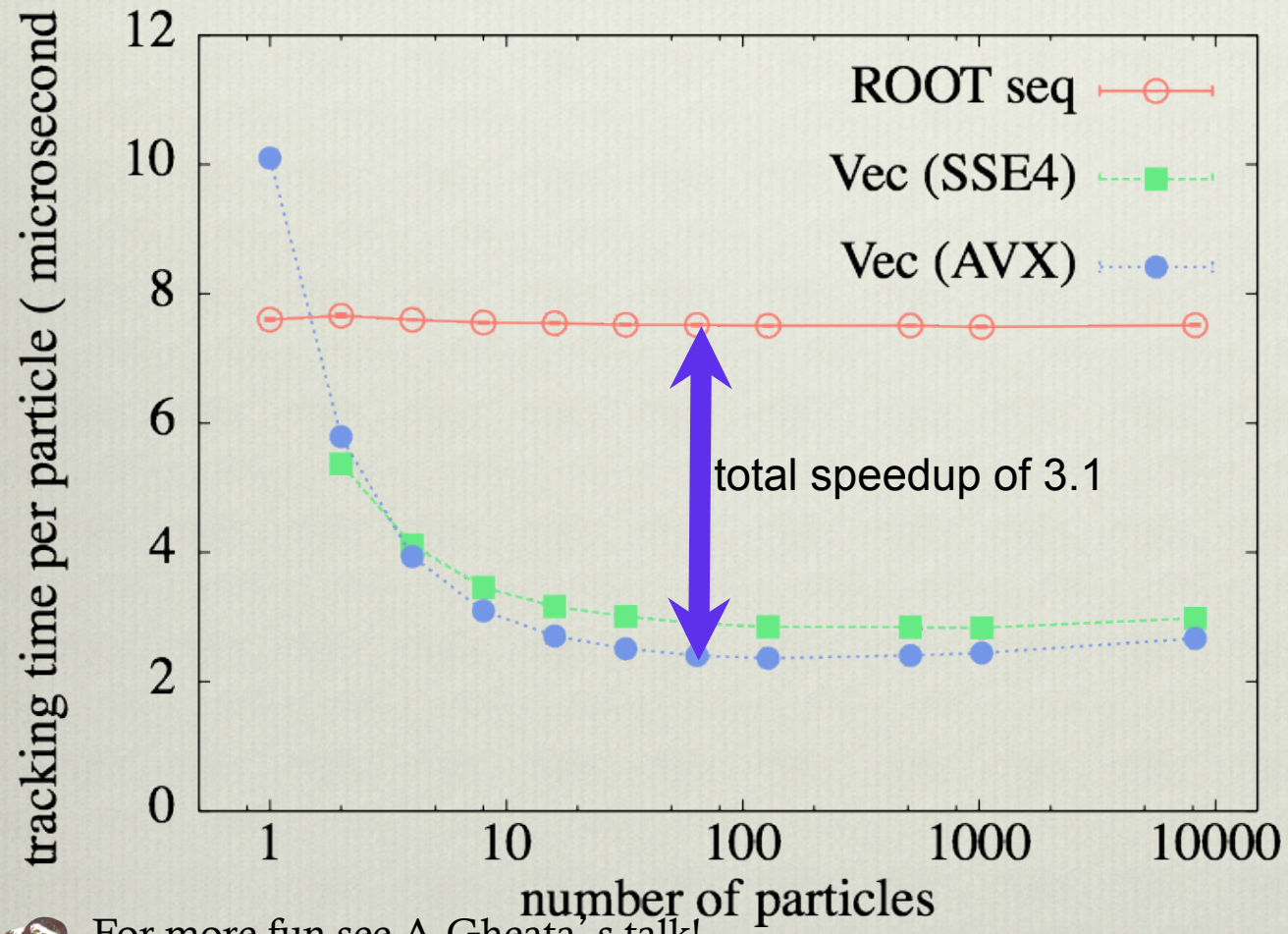
Vector processing: Update on Gains for Geometry Calculations

- 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)

Gains from microparallelism & SIMD

- Time of processing/navigating N particles (P repetitions) using scalar algorithm (ROOT) versus vector version



- excellent speedup for SSE4 version
- some further gain with AVX
- already gain considerably for small N
- there is an optimal point of operation (performance degradation for large N)
- Gain due to fewer instructions.

For more fun see A.Gheata's talk!

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)

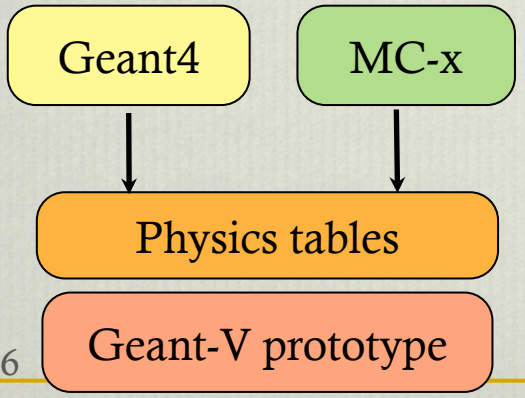
(see talk by S. Wenzel 5.6.2013)

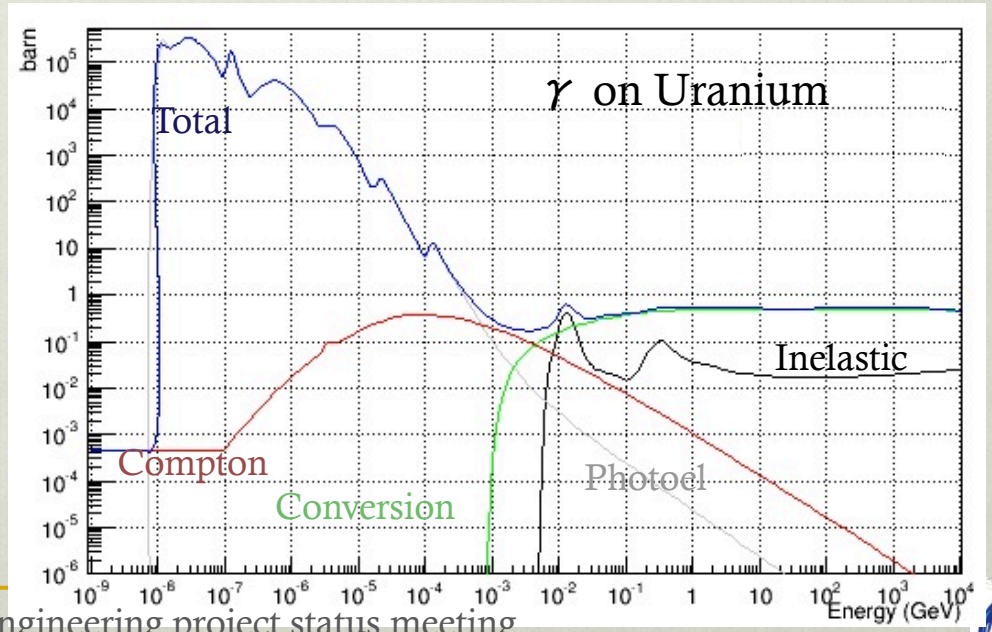
- However: SIMD autovectorization difficult to achieve
- Alternative: explicit vectorization approach:
  - intrinsics ?
  - (gcc) vector extensions ?
  - Vc library
    - compiler independent, high level constructs, abstraction of SIMD instruction set without overhead

(slide by S. Wenzel)

Physics

- A lightweight physics for realistic shower development
  - Select the major mechanisms
    - Bremsstrahlung, e+ annihilation, Compton, Decay, Delta ray, Elastic hadron, Inelastic hadron, Pair production, Photoelectric, Capture + dE/dx & MS
  - Tabulate all x-secs (100 bins -> 90MB)
  - Generate (10-50) final states (300kB per final state & element)
- It will not be good Geant4, but but it could be the seed of a fast simulation option
- Independent from the MonteCarlo that actually generates the tables



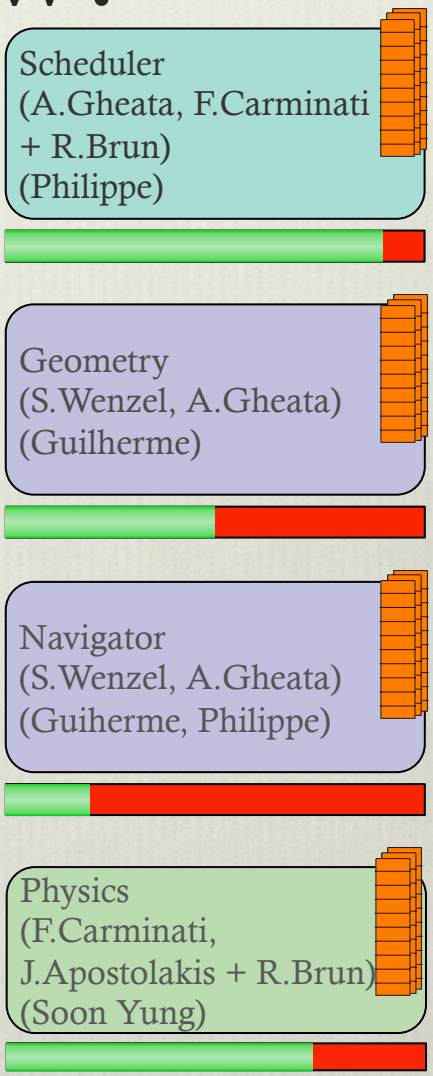


Advantages

- 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

Where are we now?

- Scheduler
  - The new version, hopefully improved of the scheduler has been committed and we are testing it
- Geometry
  - The proof or principle that we can achieve large speedups (3-5+) is there (see A.Gheata's talk), however a lot of work lays ahead
- Navigator
  - "Percolating" vectors through the navigator is a difficult business. We have a simplified navigator that achieves that (S.Wenzel), but more work is needed here
- Physics
  - Can generate x-secs and final states and sample them, but there are still many points to be clarified with Geant4 experts



Targets

- By the end of the year we will "glue" the different pieces together
  - And hopefully demonstrate the speedup potential of MT, locality and SIMD
- Measure the evolution of the memory footprint and the performance of the code at least in terms of hardware counters
- Absolute performance measurements will be harder
  - Difficult compare apples to apples
  - Probably we need to develop dedicated benchmarks
- Compare physics performance with full MC's
- For the moment we use Xeon architecture for the SIMD, but we intend to extend to GPU and to Xeon PHI
- We are working closely with Geant4 for the physics tables
- Once the prototyping phase over, we will have to sit down with the stakeholders and decide how to proceed from there

Plans

- Accelerate integration with the vector prototype
  - discussed common plan at ***Geant4 Workshop,*** target to have a clear performance evaluation in 2014
  - share components (geometry, physics, transport, etc.) and ***expertise***.
- 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
- Workshop in February with ASCR.

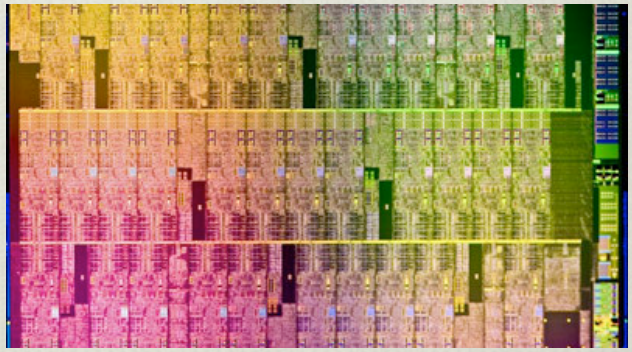
Backup slides

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.

- Common target:
  - a proof-of-principle high performance prototype in one year.
  - reasonable em physics performance (to be further defined) and should implement sampling of hadronic interactions.
- The work to be done has been broken down in four major areas:
- a) Scheduler
  - FNAL will adopt the scheduler developed by CERN and start contributing to it. FNAL will work on it with AG to make it co-processor friendly. All work related to the vector prototype will go into the current repository. We will have to build into the Scheduler the possibility to have particles of the same type contiguous. We will work to converge to a common track data structure to avoid conversions between different track formats.
- b) Navigator
  - We have two different navigators. In the short term the existing G4-like navigator/descriptions will be kept for the GPU code and make it survives a bit longer by (re)using the VMC code to implement the translation layer. This can be achieved using TG4RootNavigator from G4 VMC which can provide G4 nav. functionality on top of ROOT geometry. The long term objective is to develop a vectorized navigator and related detector construction (logical and physical volume, etc) that would both learn from the existing implementation and what we now know/learn about writing vectorized code. This code should be able to run both on CPU's and GPU's. Further discussion is needed between the stakeholders (G4, Root and the Vector/GPU prototype plus more to come in the future) on this geometry description (logical volumes, physical volume types, assemblies, alignment nodes and so on) and on a performant navigator capable of handling it. A plan to get there should be elaborated in common.
- c) Basic geometry
  - Work on Unified Solids should be leveraged optimising Usolid for SIMD to cope with both simple cases (a vector of rays encountering a single solid) and more complex ons (one or more rays encountering a list of solids). Additional work should be done on validation and of course interfacing to Navigator.
- d) Physics
  - FNAL takes our x-section tables + all the physics they need. Both a) and d) require convergence on a set of data structure. Fca will work with Philippe on this.
- e) Performance Measure,
  - compare the same physics with G4 code and make relative measurements switching on and off vectorisation. When enabling the coprocessor, we might need to limit the example to a subset of the geometrical shape and/or a subset of physics processes.
- f) Physics validation
  - Set of standard benchmarks. It would be nice to have real experiments as a baseline and to simulate them with G4 and with the prototype, but this max be more ambitious. Even if it is much too early to have real physics validation, we should prepare few test harnesses to start measuring regression on the physics results.

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.