Geant4 Computing Performance Benchmarking and Monitoring

Geant4 Computing Performance Benchmarking and Monitoring

Andrea Dotti^1, V. Daniel Elvira^2, Gunter Folger^3, Krzysztof Genser^2, Soon Yung Jun^2, James B. Kowalkowski^2 and Marc Paterno^2

^{1}SLAC National Accelerator Laboratory, 2575 Sand Hill Rd, Menlo Park, CA, 94025, USA
^{2}Fermilab^0, P.O. Box 500, Batavia, IL, 60510, USA
^{3}CERN, PH Department, J27210, CH-1211 Geneva, Switzerland

E-mail: `syjun@fnal.gov`

Abstract. Performance evaluation and analysis of large scale computing applications is essential for optimal use of resources. As detector simulation is one of the most compute intensive tasks and Geant4 is the simulation toolkit most widely used in contemporary high energy physics (HEP) experiments, it is important to monitor Geant4 through its development cycle for changes in computing performance and to identify problems and opportunities for code improvements. All Geant4 development and public releases are being profiled with a set of applications that utilize different input event samples, physics parameters, and detector configurations. Results from multiple benchmarking runs are compared to previous public and development reference releases to monitor CPU and memory usage. Observed changes are evaluated and correlated with code modifications. Besides the full summary of call stack and memory footprint, a detailed call graph analysis is available to Geant4 developers for further analysis. The set of software tools used in the performance evaluation procedure, both in sequential and multi-threaded modes, include FAST, IgProf and Open|Speedshop. The scalability of the CPU time and memory performance in multi-threaded application is evaluated by measuring event throughput and memory gain as a function of the number of threads for selected event samples.

1. Introduction

Geant4 [1][2][3] 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 and material science, medicine, biology, security and industrial applications. In high energy physics (HEP), Geant4 simulation applications are indispensable in the design of the detector in the planning phase, in developing and optimizing the particle reconstruction algorithms, and in simulating the signal and background events associated with the actual data analysis during data taking. As the most commonly used detector simulation tool, Geant4 plays a critical role in the simulation of detectors at the existing and future experiments and facilities.

Geant4-based simulations are very complex and are some of the most compute-intensive tasks in experimental HEP. Throughout the LHC Run-I, more than half of WLCG (Worldwide LHC Computing Grid) [4] has been dedicated to the simulation of about 10^{10} high energy events (equivalent of 10^{12} sec CPU time). Similar simulation computing needs are expected in the

^{0} Operated by Fermi Research Alliance, LLC under Contract No. De-AC02-07CH11359 with the United States Department of Energy.

LHC Run-II. Therefore, it is important to monitor Geant4 throughout its development cycle for changes in computing performance to assure optimal use of computing resources and to promote good software practices.

The testing and quality assurance working group of the Geant4 collaboration conducts computing performance evaluation of Geant4 applications for all Geant4 development and public releases. The group roles are: monitoring of Geant4 software for any expected and unexpected changes in the computing performance, identification of problems and opportunities for code improvement and optimization, and communication of the results and findings to the appropriate Geant4 working group leaders and developers. In this paper, the procedure of Geant4 computing performance benchmarking and profiling is briefly described and selected results are presented.

2. Profiling Tools and Software

Geant4 applications incorporate a large body of physics knowledge from modern high energy and nuclear physics, from theory and from experiments. The applications tend to be large in both required compute time and memory space, mainly because of the complex geometries through which particles will be propagated. This complexity has made it challenging to find tools that can profile these applications systematically, producing results that are accurate and reproducible. A good tool must provide a robust mechanism for profile data collection over long time periods to accumulate sufficient statistics for generating a faithful performance profile of the application. The tool must also have minimal impact on the application with regards to overall run-time (for practical purposes) and changing the overall performance (not reflecting reality). The Geant4 developers must also be able to easily use the tool. In general, a good sampling profiler have been found to meet the many of these needs.

The set of software tools used in the current Geant4 performance evaluation procedure include FAST [5] and IgProf [6] for sequential applications, and Open|Speedshop [7] for applications that include multi-threading. TAU [8] and HPCToolkit [9] are also used for internal code reviews and other performance studies.

Three applications suitable for performance analysis and benchmarking are built using each new Geant4 release:

- SimplifiedCalo was developed to emphasize computing performance aspects of physics processes and models. It employs a sampling calorimeter with a simple cylindrical geometry, with repeating concentric layers of an absorber material and an active volume.
- cmsExp was developed for an extended measure of performance in a typical, complex HEP detector. It uses the Geometry Description Markup Language (GDML) interface to read a simplified CMS [10] detector geometry and a magnetic field map extracted from the CMS software suite.
- cmsExpMT is the multi-threaded version of cmsExp.

The 4.9.2 version of GCC is currently used with the default Geant4 optimization level ($-O2$). R [11] and ROOT [12] are used as statistical and visualization tools for post analysis and for presentation and summarizing results.

3. Profiling Platform

Performance benchmarking and profiling requires a stable set of 'quiet' machines free from uncertainties due to hardware fluctuations, changes in software and system configurations, and interruptions due to network activity. This dedicated set of machines must capable of providing sufficient computing resources (CPU and memory) to collect significant sampling data for all relevant input parameter combinations and deliver results within the required time constraints.

The Fermilab Wilson cluster provides the primary hardware platform used for profiling and benchmarking Geant4 releases. It is also used to run relevant parts of the test suite. The cluster worker nodes are equipped with four 8-core 6128HE 2.0GHz AMD Opteron processors. The dedicated head node provides NFS access to the Geant4 disks for long-term storage. The head node is also used for software builds, batch job submission, and for final analysis of the performance data. Standard release profiling requires 2500 CPU-hours on this cluster. A detailed description of the Wilson cluster can be found at the link [14]. In addition, a standalone 32-core AMD server and an Intel Xeon Phi server node are available for benchmarking multi-threaded Geant4 applications.

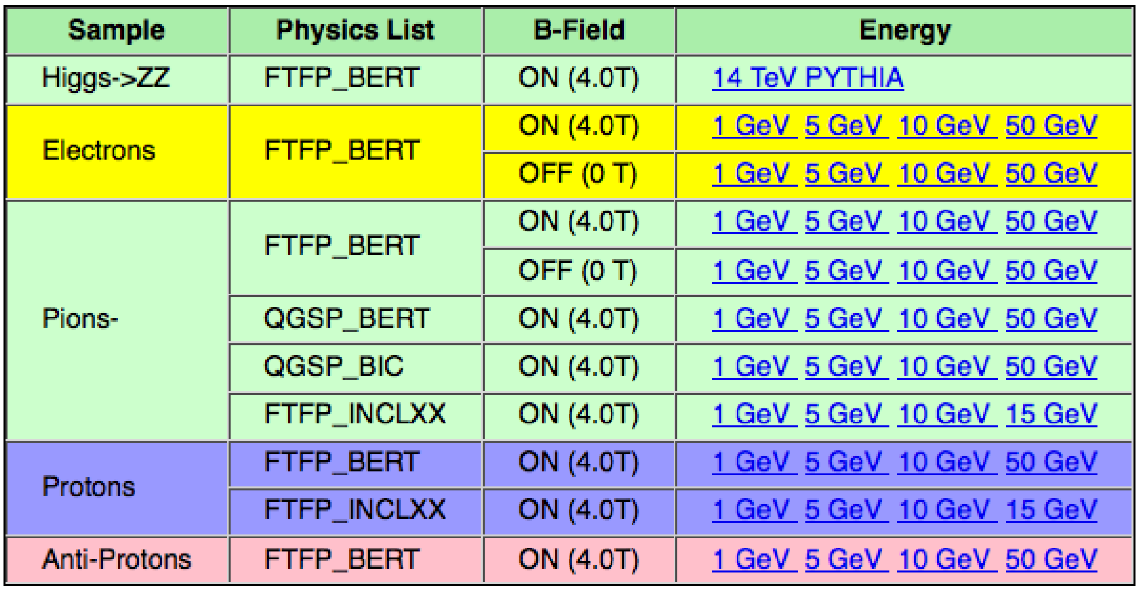
To minimize performance measurement uncertainty, a reference release (typically the previous release) is re-compiled and re-profiled along with a new release of Geant4. The NUMA controls are used to lock Geant4 processes to specific cores to eliminate performance effects that can occur from any core-to-core process migration that might take place. The system activity is collected during performance runs using the standard Linux System Activity Data Collector (sadc) and processed with sar. Performance measurements are reproducible, with uncertainties due to hardware and system fluctuations, usually within $\pm 0.5\%$.

4. Performance Monitoring

Performance monitoring is an integral part of performance evaluation which requires quantitative measurements of changes in utilization of computing resources such as CPU consumption and memory footprint. Since Geant4 consists of large sets of modules such as of e.g. tracking kernels, physics processes and models, geometry shapes and materials, it is a challenge to detect all expected or unexpected changes in performance. All development and public releases are being profiled with a set of applications that utilize different input event samples, physics parameters, and detector configurations. Results from multiple benchmarking runs are compared to previous public and development reference releases to monitor changes in CPU and memory usage. Figure 1 shows the current list of samples used in regular profiling. Each specific sample is profiled multiple times with a given number of events to collect statistically significant data. For an example, the physics sample containing 50 events of $H \rightarrow ZZ$ (Z to all channels) is profiled in 128 runs distributed over available batch nodes while each single particle sample is profiled 32 times using 2000 (10, 50 GeV), 4000 (5 GeV), 10000 (1 GeV) events with energy of the input particle given in parenthesis. The number of events is chosen to keep the run time of each sample similar within each particle group. The standard deviation (σ) of the measurements is evaluated for each sample and σ/(the mean CPU time) is required to be less than 0.5%.

Since one of the goals of the performance monitoring is to understand the changes in computing performance, it is important to be able to conveniently inspect the behavior of the new releases and to be able to compare it to that of the reference and previous releases on a timely basis. The evolution of two most basic, but important performance metrics, i.e. the average CPU time per event and the total memory footprint are shown in Figure 2 and Figure 3 for all releases made during 2014. Summary of performance evaluation for each release is available within 48 hours. Unexpected changes in the CPU time greater than one percent are considered serious and are reported to the relevant Geant4 working groups. Additional profiling data are examined in such cases. The total number of geometrical steps and the total number of secondary tracks produced are recorded to help decide whether performance changes are due to modifications related to physics or geometry. Many other observables are available on the benchmarking and profiling web page [13].

Since the release of Geant4 version 10 (December, 2013), a multi-threaded version of Geant4 (Geant4-MT) is available enabling the event-level parallelism. The primary goal of the design of Geant4-MT was to reduce the total memory usage by sharing common data (such as physics data and geometry), while maintaining the linearity of the event throughput versus the number of threads. The scalability of the CPU time and memory performance in multi-threaded application is evaluated by measuring event throughput (events per sec) and memory reduction as a function



Sample	Physics List	B-Field	Energy
Higgs->ZZ	FTFP_BERT	ON (4.0T)	14 TeV PYTHIA
Electrons	FTFP_BERT	ON (4.0T)	1 GeV 5 GeV 10 GeV 50 GeV
		OFF (0 T)	1 GeV 5 GeV 10 GeV 50 GeV
Pions-	FTFP_BERT	ON (4.0T)	1 GeV 5 GeV 10 GeV 50 GeV
		OFF (0 T)	1 GeV 5 GeV 10 GeV 50 GeV
	QGSP_BERT	ON (4.0T)	1 GeV 5 GeV 10 GeV 50 GeV
	QGSP_BIC	ON (4.0T)	1 GeV 5 GeV 10 GeV 50 GeV
	FTFP_INCLXX	ON (4.0T)	1 GeV 5 GeV 10 GeV 15 GeV
Protons	FTFP_BERT	ON (4.0T)	1 GeV 5 GeV 10 GeV 50 GeV
	FTFP_INCLXX	ON (4.0T)	1 GeV 5 GeV 10 GeV 15 GeV
Anti-Protons	FTFP_BERT	ON (4.0T)	1 GeV 5 GeV 10 GeV 50 GeV

Figure 1. A set of (total 41) samples for different event types, physics parameters (lists) and configurations used for Geant4 performance benchmarking and profiling.

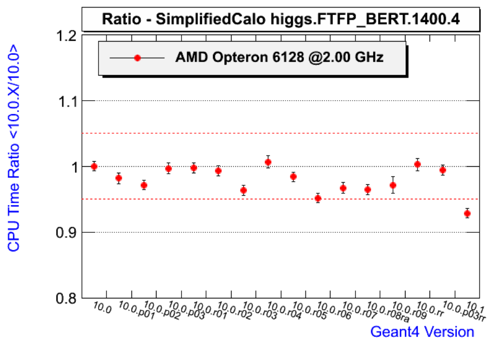


Figure 2. Average CPU time vs. Geant4 version normalized to a reference release.

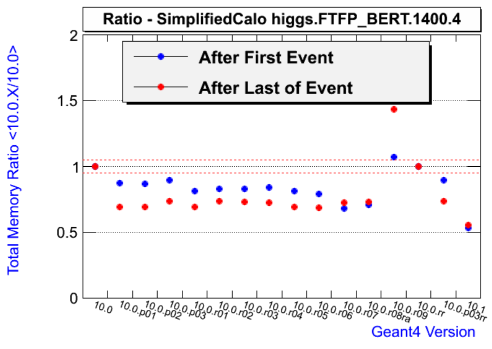


Figure 3. Total memory count vs. Geant4 version normalized to a reference release.

of the number of threads for selected event samples as shown in Figure 4 and Figure 5, respectively. Performance of Geant4-MT is also evaluated on an Intel Xeon Phi coprocessor (Many Integrated Cores (MIC) architecture, ∼60 cores with 4-way hyper-threading and 8 GB memory). The performance shows a good linearity up to the maximum available number of threads ($>$ 200) as tested with the CMS geometry. Performance of Geant4-MT using a single thread is compared to the same application run in sequential mode to estimate the overhead due to multi-threading. For multi-threaded Geant4 applications, and for a given number of threads, the profiling data are collected by the periodic sampling of the program counters (osspcsamp) of Open|Speedshop which gives a low-overhead view of where the (exclusive) time is being spent. A more detailed description of Geant4-MT and its computing performance can be found in [3].

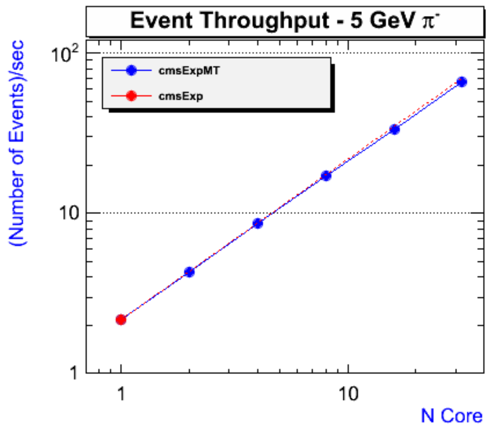


Figure 4. The Event throughput of Geant4-MT as the number of threads on AMD 32 core machines (Opteron, 6128, 2.0 GHz, 4 CPU sockets x 8 cores).

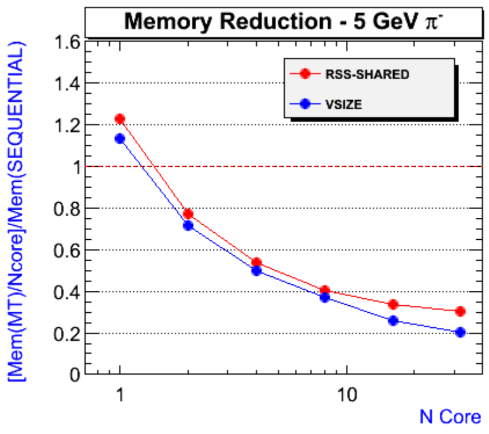


Figure 5. The memory reduction of Geant4-MT as the number of threads on AMD 32 core machines (Opteron, 6128, Total 66 GB memory.

5. Performance Analysis

Performance analysis usually requires specific domain knowledge as well as an excellent understanding of computing architecture and programming language used for the application. Understanding profiling data collected from Geant4 applications is critical for identifying sections of code that cause changes in performance. It also aids in finding functions that consume too much memory or CPU time and are in need of optimization.

Two important metrics provided by FAST (the CPU profiler) are the leaf and path counts for a each encountered function. The leaf count is the number of times a function was observed as the last entry in the call stack and is proportional to the amount of the (exclusive) time that was spent executing the code of that function. The path count is the total number of times the function was observed anywhere in the call stack and is proportional to the amount of the (inclusive) time that was spent executing the code of that function plus the time spent executing all the functions it calls.

In most of the cases, the functions responsible for major change in CPU performance are visible in the function list when sorted by decreasing leaf count ("hot spots"). The same is true for memory profiles when the function list is sorted by the decreasing memory footprint. Samples of CPU profiling results from FAST are shown in Figure 6.

IgProf is used as a memory profiler. It provides snapshots of memory usage data at each performance profiling interrupt (with the default rate of 100 Hz). There are three important modes for measuring and analyzing application memory: the live mode probes the memory that has not been freed on the heap, producing a snapshot of the heap, the maximum mode which traces the largest single allocation by any function, and the maximum total mode which accumulates the total amount of memory allocated by any function, producing a snapshot of poor memory locality. The difference of live memory between N events provides information on a potential memory leakage. An example of memory profiling using IgProf is shown in Figure 7. IgProf provides a web-navigable display of memory profiling information utilizing an underlying sqlite database as shown in Figure 7.

In addition to the full summary of the call stack and memory footprint, a detailed call graph analysis is available to Geant4 developers for further analysis. The profgraph tool within FAST generates a graphical view showing function call relationships centered on a given function. The

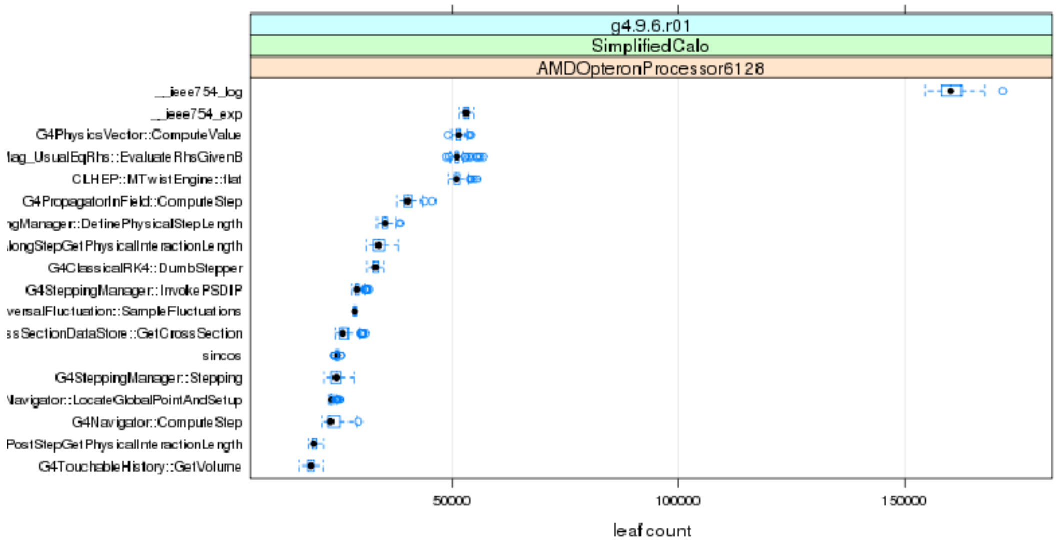
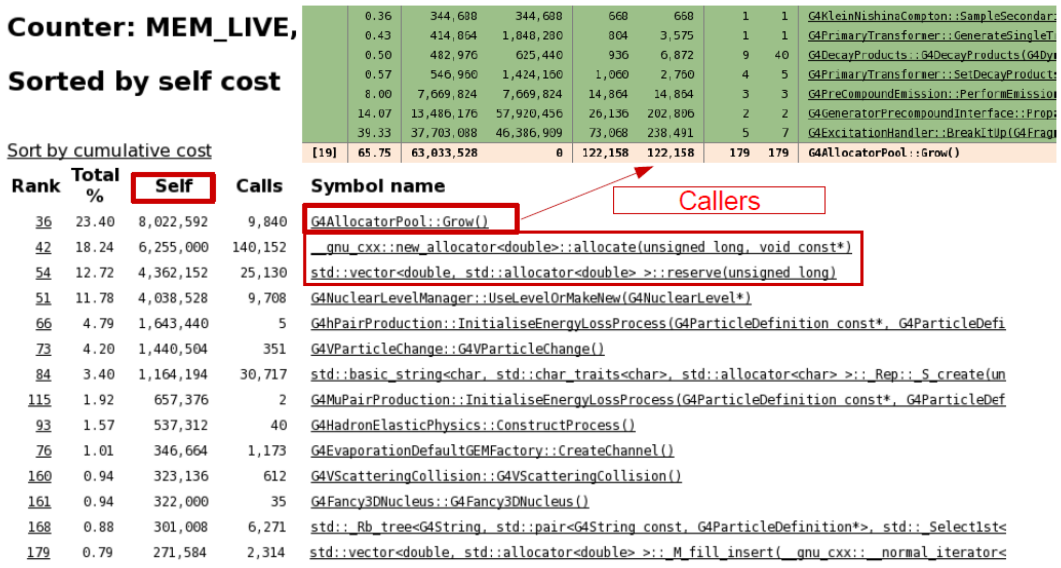


Figure 6. An example plot of CPU profiling using FAST: the list of top functions by the leaf count, which is proportional to the exclusive time that was spent executing the code of the given function.


Counter: MEM_LIVE,

Sorted by self cost

Sort by cumulative cost

	0.36	344,688	344,688	668	668	1	1	G4KleinNishinaCompton::SampleSecondar
	0.43	414,864	1,848,280	804	3,575	1	1	G4PrimaryTransformer::GenerateSingleT
	0.50	482,976	625,440	936	6,872	9	40	G4DecayProducts::G4DecayProducts(G4Dy
	0.57	546,960	1,424,160	1,060	2,760	4	5	G4PrimaryTransformer::SetDecayProduct
	8.00	7,669,824	7,669,824	14,864	14,864	3	3	G4PreCompoundEmission::PerformEmissio
	14.07	13,486,176	57,920,456	26,136	202,806	2	2	G4GeneratorPrecompoundInterface::Prop
	39.33	37,703,088	46,386,909	73,068	238,491	5	7	G4ExcitationHandler::BreakItUp(G4Frag
[19]	**65.75**	**63,033,528**	**0**	**122,158**	**122,158**	**179**	**179**	**G4AllocatorPool::Grow()**

Callers

Rank	Total %	Self	Calls	Symbol name
36	23.40	8,022,592	9,840	G4AllocatorPool::Grow()
42	18.24	6,255,000	140,152	__gnu_cxx::new_allocator<double>::allocate(unsigned long, void const*)
54	12.72	4,362,152	25,130	std::vector<double, std::allocator<double> >::reserve(unsigned long)
51	11.78	4,038,528	9,708	G4NuclearLevelManager::UseLevelOrMakeNew(G4NuclearLevel*)
66	4.79	1,643,440	5	G4hPairProduction::InitialiseEnergyLossProcess(G4ParticleDefinition const*, G4ParticleDefi
73	4.20	1,440,504	351	G4VParticleChange::G4VParticleChange()
84	3.40	1,164,194	30,717	std::basic_string<char, std::char_traits<char>, std::allocator<char> >::_Rep::_S_create(un
115	1.92	657,376	2	G4MuPairProduction::InitialiseEnergyLossProcess(G4ParticleDefinition const*, G4ParticleDef
93	1.57	537,312	40	G4HadronElasticPhysics::ConstructProcess()
76	1.01	346,664	1,173	G4EvaporationDefaultGEMFactory::CreateChannel()
160	0.94	323,136	612	G4VScatteringCollision::G4VScatteringCollision()
161	0.94	322,000	35	G4Fancy3DNucleus::G4Fancy3DNucleus()
168	0.88	301,008	6,271	std::_Rb_tree<G4String, std::pair<G4String const, G4ParticleDefinition*>, std::_Select1st<
179	0.79	271,584	2,314	std::vector<double, std::allocator<double> >::_M_fill_insert(__gnu_cxx::__normal_iterator<

Figure 7. An example of memory profiling information using IgProf: the live memory sorted by the self cost. Each function in the table is web-navigable for its callers or callees

tool allows to adjust how many levels of the call stack are displayed and to filter out infrequently used paths. Figure 8 shows an example of a call graph visualized using GRAPHVIZ.

6. Code Reviews

When needs arise, computational aspects of a subset of Geant4 classes are inspected and analyzed in order to identify opportunities for improvements in performance as well as in code practices and design. A particular module of Geant4 source codes is selected and inspected both visually

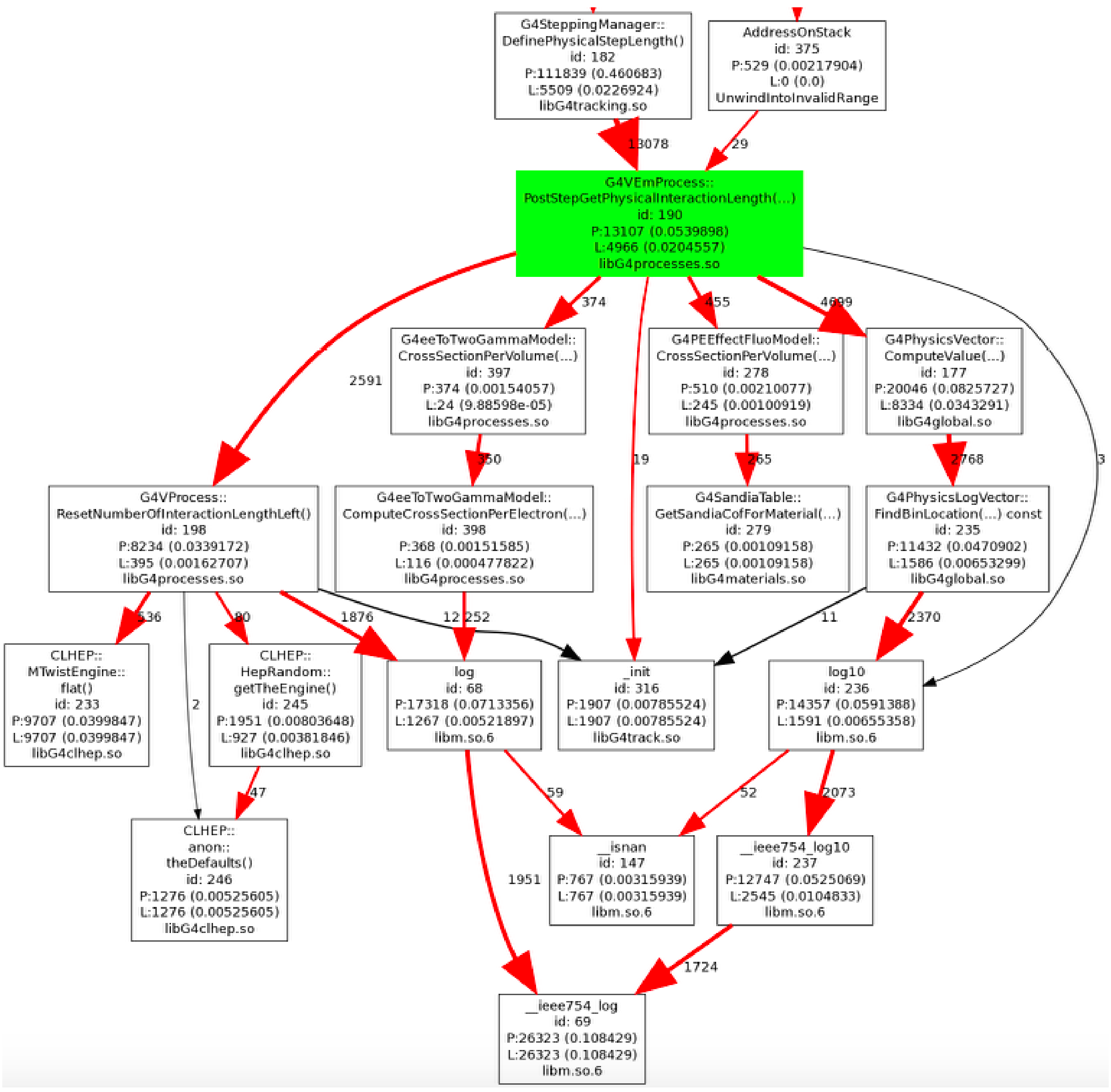


Figure 8. A section of call graph using PROFGRAPH of FAST.

and using a set of performance tools. The code reviewers provide not only general comments on the class structure and design or coding practices, but also deliver code-specific observations. Code reviews are usually performed in close collaboration with the code authors or maintainers who are informed of important findings with a significant computational impact as soon as they are discovered.

7. Summary

In this article, we described the Geant4 computing performance benchmarking and monitoring procedure, and presented examples of performance results and analysis.

References

[1] Allison J et al. 2006 “Geant4 Developments and Applications”, IEEE Transactions on Nuclear Science 53 No. 1 270-278.
[2] Agostinelli S et al. 2003 “Geant4 - A Simulation Toolkit”, Nuclear Instruments and Methods in Physics Research A 506 250-303
[3] Ahn S et al. 2014 “Geant4-MT: bringing multi-threading into Geant4 production”, Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, 04213
[4] Worldwide LHC Computing Grid, http://wlcg.web.cern.ch
[5] The FAST project, https://cdcvs.fnal.gov/redmine/projects/fast
[6] The Ignominous Profiler http://igprof.org/index.html
[7] Open|Speedshop http://www.openspeedshop.org/wp
[8] TAU (Tuning and Analysis Utilities) http://www.cs.uoregon.edu/Research/tau/home.php
[9] HPCToolkit http://hpctoolkit.org/index.html
[10] The CMS (Compact Muon Solenoid) experiment at CERN. http://cms.web.cern.ch
[11] The R Project for Statistical Computing http://www.r-project.org
[12] ROOT: An Object-Oriented Data Analysis Framework, Linux Journal, Issue 51, July 1998, ROOT – A C++ framework for petabyte data storage, statistical analysis and visualization, Computer Physics Communications; Anniversary Issue; Volume 180, Issue 12, December 2009, Pages 2499-2512. https://root.cern.ch
[13] Geant4 Profiling and Benchmarking, https://g4cpt.fnal.gov/perfanalysis/g4p/index.html
[14] The Wilson Cluster at Fermilab, http://tev.fnal.gov/hardware.shtml