Regular Performance Monitoring

Tools, Process, and Results

Daniel Elvira, Krzysztof Genser, Soon Yung Jun
Fermilab

Geant4 Collaboration Meeting
September 10-14, 2012
Chartres, France

Outline

- Performance Monitoring and Protocol
- Tools, Applications and Procedure
- Profiling Results
- Performance Related Discussion Threads
- Future Improvements
- Conclusions

Geant4 Performance Monitoring

- Monitoring performance changes for new releases
- Benchmarking and profiling of Geant4 applications
  - CPU time per event
  - CPU usage and memory footprint for Geant4 applications
- Looking for opportunities for performance improvements
  - hot spots: disproportionately high amount of processing time
  - bottlenecks: resource inefficiencies and unnecessary delays
  - reduce time and memory for a serial programming

Benchmarking and Profiling Protocol

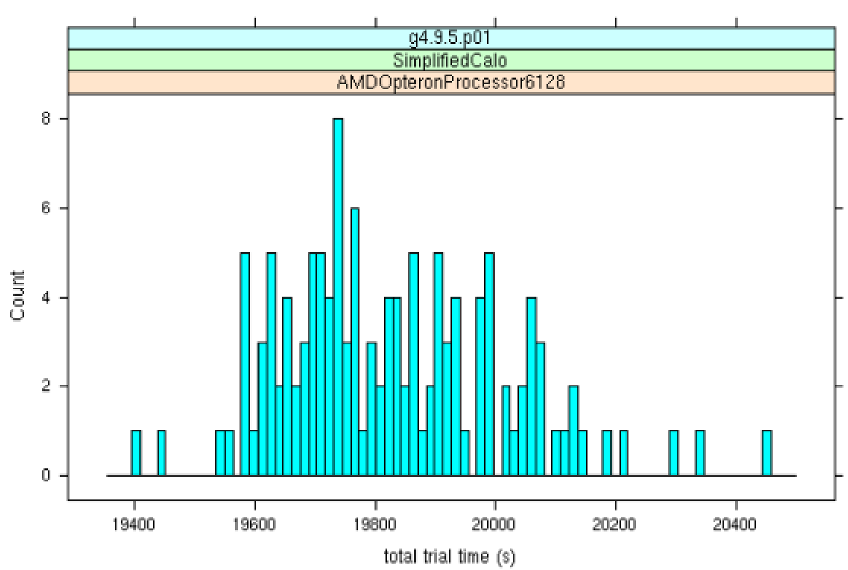
- Proposed by G4CP task force in late January, 2012
  http://oink.fnal.gov/perfanalysis/g4p/admin/task.html
- Major activities for 2012
  - adapted an existing infrastructure developed by K. Genser
  - expanded it based on discussions with the Test and QA team (A. Dotti, G. Cosmo, G. Folger, D. Elvira, K, Genser, S.Y. Jun)
  - added a memory profiling tool (igprof) and trend/summary plots
  - migrated to a new hardware platform (AMD 6x32 cores, SL5/6)
- Executed the task for 10 new releases starting from 9.5
  - results are summarized at
    http://oink.fnal.gov/perfanalysis/g4p/index.html

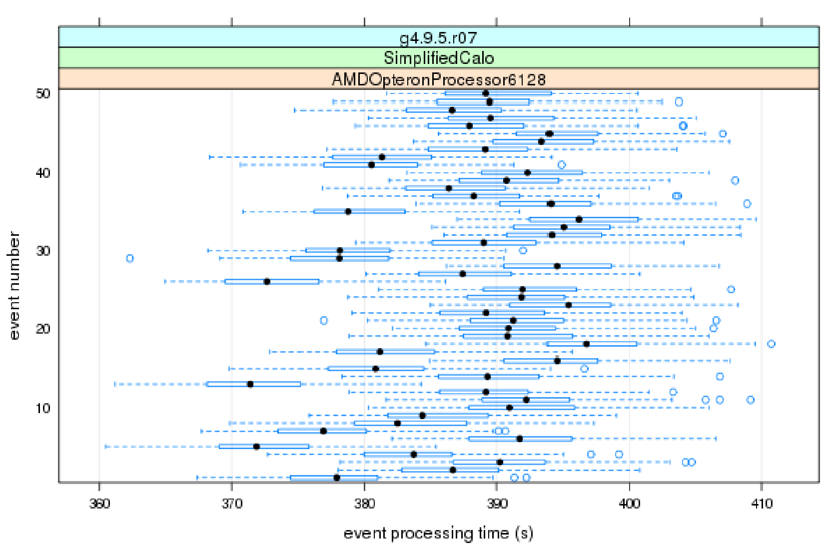
Review of Tools: FAST

- FAST (Flexible Analysis and Storage Toolkits)
  - designed to help improve primarily the CPU performance
  - applicable to single threaded programs written in C/C++
  - elements to collect data, statistical analysis and display
- Major components
  - SimpleProfiler: a sampling profiler for collecting measurements with a default frequency of 100Hz
  - ProfGraph: a graphical analysis tool for call stack data
- Documentation at https://cdcvs.fnal.gov/redmine/projects/fast (also see talks by K. Genser at the 2011 Collaboration Meeting)

FAST/SimpleProfiler

- SimpleProfiler is a sampling profiler
  - more runs increases accuracy for a given set of test
  - more events increases the number of call stacks collected





- With low time overhead (~1%) and few % uncertainty

CPU Profiling/SimpleProfiler

- Call stack: a stack data structure that stores information about the active functions
- Observables
  - leaf count: number of times observed as the last entry in the call stack, proportional to the amount of time executing the function
  - path count: number of times observed anywhere in the stack, proportional to the amount of time executing the function plus all the functions that it calls (inclusive time)
  - total count: number of times appeared in the call stack (recursive function calls, not related to the time spent)

Review of Tools: IgProf

- IgProf (Ignomimous Profiler)
  - measuring and analyzing application memory and performance
  - fast, light weight (15-250% in the run time overhead)
  - no changes to the application or the build process
  - ready for multi-thread
- Major components
  - igprof: memory/performance/heap profiling
  - igprof-analysis: ascii or web-navigable reports
- Documentation: http://igprof.sourceforge.net/index.html
  (also see the talk by M. Kelsey at the 2011 Collaboration Meeting)

Memory Profiling/igprof

- IgProf reports memory allocations at any given instance
  - MEM_TOTAL: the total amount of memory allocated by any function, a snapshot of poor memory locality
  - MEM_LIVE: memory that has not been freed in heap, a snapshot of a possible memory leak
  - MEM_MAX: the largest single allocation by any function
- For heap profiling, small application code modification is needed to obtain live memory allocations in the heap

```
#include <dlfcn.h>

void (*dump_)(const char *);

if (void *sym = dlsym(0,"dump")) dump_ = __extension__ (void(*)(const char*)) sym;
else dump_=0;
...
if ( dump_ && evt->GetEventID() % n_dump_at == 0 ) {
     sprintf(outfile,"|gzip -9c > IgProf.%d.gz",evt->GetEventID()+1);
     dump_(outfile);
}
```

Profiled Applications

- Geant4 applications used for performance monitoring
  - SimplifiedCalo (sampling calorimeter/uniform magnetic field)
  - cmsExp (complicated geometry/magnetic field)
- Performance depends on
  - physics processes
  - geometry
  - magnetic field, I/O and etc.
- Given that performance depends on details of specific application, we encourage each developer to use profiling tools before finalizing their code

Profiled Samples and Physics Lists

- Single particle and general physics processes

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
	LHEP	ON (4.0T)	1 GeV 5 GeV 10 GeV 50 GeV
Protons	FTFP_BERT	ON (4.0T)	1 GeV 5 GeV 10 GeV 50 GeV
Anti-Protons	FTFP_BERT	ON (4.0T)	1 GeV 5 GeV 10 GeV 50 GeV

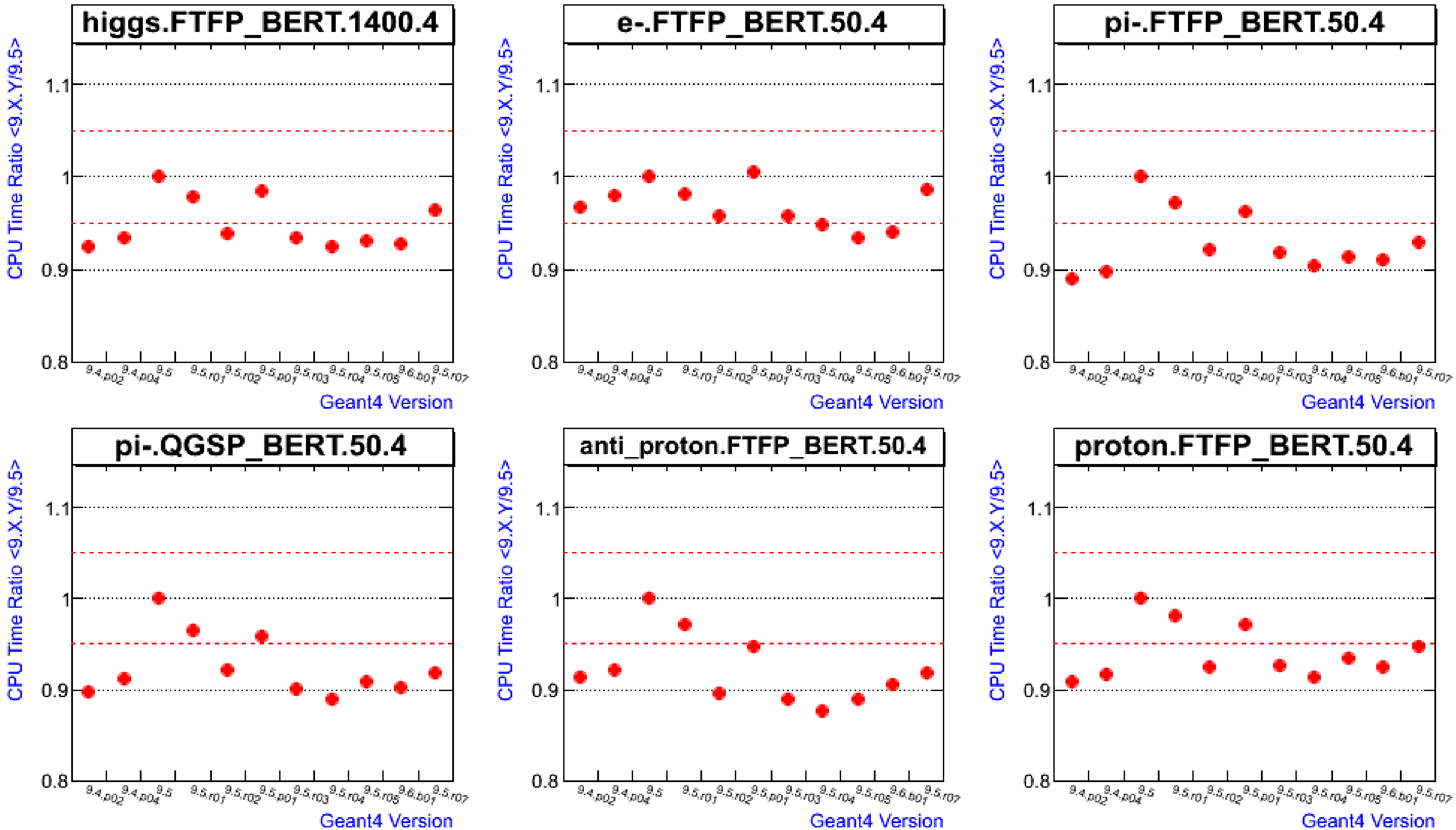
- Physics lists and beam energies are selected to probe typical HEP applications

Profiling Procedure

- Build Geant4 and applications
  - C/CXX_FLAGS: -O3 -g -fno-omit-frame-pointer
  - GEANT4_USE_SYSTEM_CLHEP=ON
  - GEANT4_USE_GDML=ON
- PBS batch jobs for CPU and memory profiling
  - igprof: 37 consecutive individual jobs on a selected node
  - profrun (FAST/SimpleProfiler): 128x1 + 32x36 = 1280 jobs
- Analysis
  - Igprof-analysis
  - statistical analysis with R-scripts and root
  - publishing results on the web

CPU Time Ratio 9.X.Y/9.5

- Relative CPU time change by version compared with 9.5



- SimplifiedCalo: e.g. selected events samples out of 37

Leaf Counts for Top Functions

- First 20 functions: SimplifiedCalo(left) and cmsExp(right)

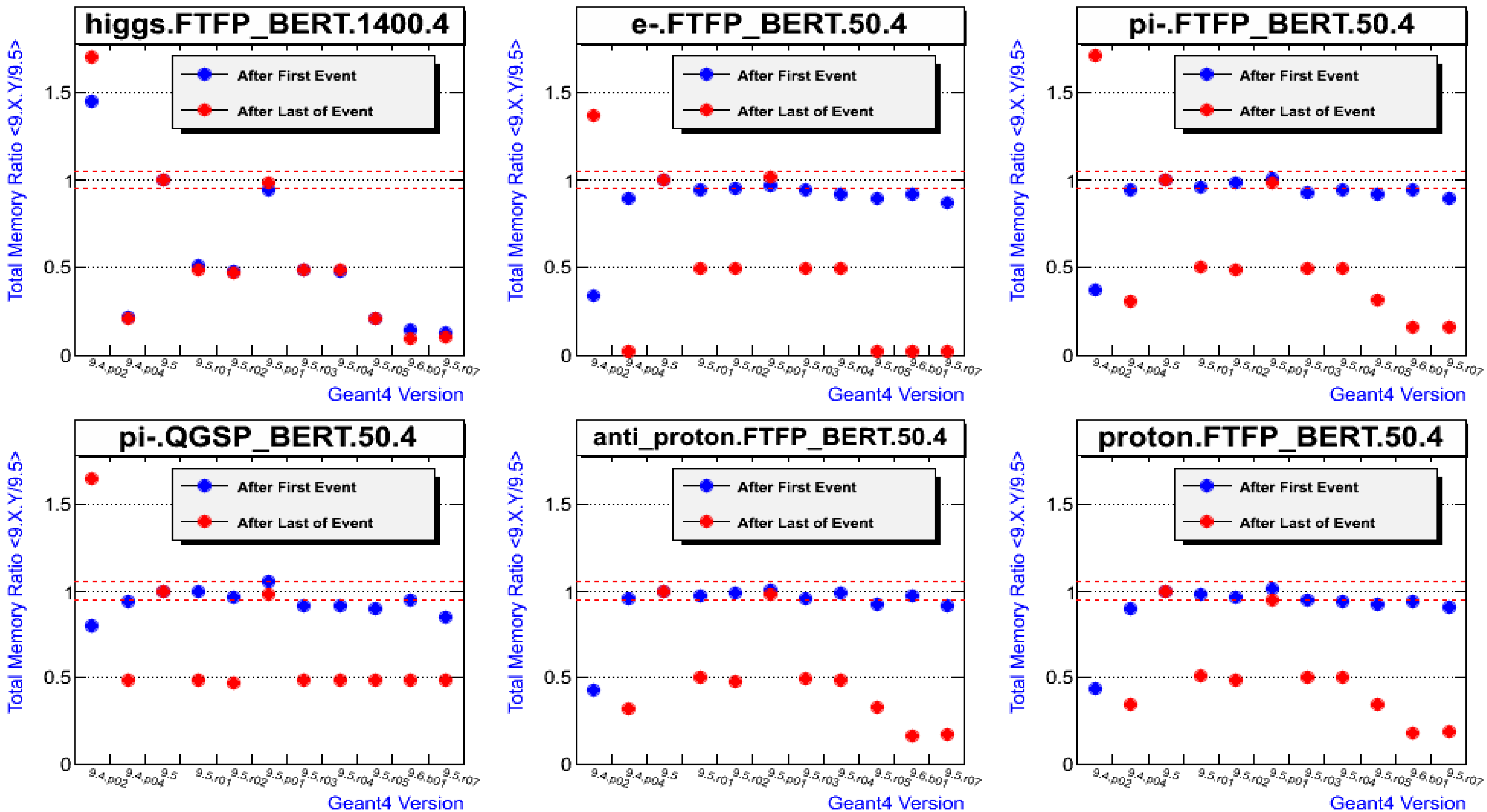
Name	short
16	__ieee754_log
1	CLHEP::MTwistEngine::flat
15	__ieee754_exp
4	G4Mag_UsualEqRhs::EvaluateRhsGivenB
6	G4PhysicsVector::ComputeValue
8	G4SteppingManager::DefinePhysicalStepLength
9	G4SteppingManager::InvokePSDIP
19	sin
11	G4Transportation::AlongStepGetPhysicalInteractionL
18	cos
2	G4ClassicalRK4::DumbStepper
7	G4PropagatorInField::ComputeStep
13	G4VEmProcess::PostStepGetPhysicalInteractionLeng
12	G4UniversalFluctuation::SampleFluctuations
14	G4VEnergyLossProcess::PostStepGetPhysicalInterac
5	G4Navigator::ComputeStep
10	G4SteppingManager::Stepping
3	G4HadronCrossSections::CalcScatteringCrossSectio

Name	short
20	__ieee754_log
9	G4HadronCrossSections::CalcScatteringCrossS
3	cmsExpMagneticField::GetVolumeBaseBfield
8	G4ElectroNuclearCrossSection::GetIsoCrossSe
6	G4CrossSectionDataStore::GetCrossSection
1	CLHEP::MTwistEngine::flat
14	G4PolyconeSide::DistanceAway
7	G4CrossSectionDataStore::GetIsoCrossSection
19	__ieee754_exp
10	G4Mag_UsualEqRhs::EvaluateRhsGivenB
13	G4PhysicsVector::ComputeValue
5	G4ClassicalRK4::DumbStepper
16	G4SteppingManager::DefinePhysicalStepLengt
2	cmsExpMagneticField::GetFieldValue
12	G4PhotoNuclearCrossSection::GetIsoCrossSec
15	G4PropagatorInField::ComputeStep
21	sin
17	G4UniversalFluctuation::SampleFluctuations
11	G4Navigator::LocateGlobalPointAndSetup
18	G4VEmProcess::PostStepGetPhysicalInteractio

- CPU utilization is quite different for applications
- One can use ProfGraph to further analyze data

Total Memory Ratio 9.X.Y/9.5

- Relative memory change by version compared with 9.5



- SimplifiedCalo: e. g. selected event samples out of 37

Cumulative Memory Usage

- Cumulative allocations by a function and its callees

IgProf_higgs.FTFP_BERT.1400.4_MEM_TOTAL_51 g4p_9.5.p01_SimplifiedCalo_02

Back to profiles index

Counter: MEM_TOTAL, first 1000 entries

Sorted by cumulatative cost

(Sort by self cost)

Rank	Total %	Cumulative	Calls	Symbol name
1	100.00	43,962,464,690	231,564,978	<spontaneous>
4	100.00	43,962,461,380	231,564,971	main
3	100.00	43,962,461,380	231,564,971	__libc_start_main
2	100.00	43,962,461,380	231,564,971	_start
5	99.92	43,926,926,754	230,936,087	G4UImanager::ApplyCommand(char const*)
6	99.92	43,926,926,447	230,936,079	G4UIcommand::DoIt(G4String)

- Total allocation in bytes and the number of allocations

Total Memory used by Function

- Memory allocations within a function

Counter: MEM_TOTAL

Sorted by self cost

Rank	% total	Counts: to / from this	Counts: Total	Calls: to / from this	Calls: Total	Paths: Including child / parent	Paths: Total	Symbol name
	75.12	33,026,428,616	33,579,266,746	124,144,767	134,580,961	4	4	G4Quasmon::HadronizeQuasmon(G4QN
[24]	**75.12**	**32,424,188,544**	**602,240,072**	**122,818,896**	**124,144,767**	**4**	**4**	**G4QNucleus::InitCandidateVector(**
	1.37	602,240,072	907,922,312	1,325,871	2,524,639	4	8	std::vector<G4QCandidate*, std::

Callers and Callees

Sort by cumulative cost

Rank	Total %	Self	Calls	Symbol name
24	73.75	32,424,188,544	122,818,896	G4QNucleus::InitCandidateVector(std::vector<G4QCandidate*, std::allocator<G4QCandida
33	8.91	3,914,943,648	14,829,332	G4QEnvironment::InitClustersVector(int, int)
35	4.09	1,797,420,072	10,698,929	G4InuclCollider::collide(G4InuclParticle*, G4InuclParticle*, G4CollisionOutput&)
38	2.74	1,205,823,584	25,734,036	G4NucleiModel::generateModel(int, int)
39	2.07	907,922,312	2,524,639	std::vector<G4QCandidate*, std::allocator<G4QCandidate*> >::_M_insert_aux(__gnu_cxx::
41	1.20	529,385,472	22,057,728	G4NucleiModel::fillPotentials(int, double)
42	1.06	466,092,864	4,855,134	G4Quasmon::CalculateHadronizationProbabilities(double, double, CLHEP::HepLorentzVect
19	0.88	388,067,088	449,423	G4QEnvironment::Fragment()
44	0.87	380,967,464	147,929	G4QEnvironment::CreateQuasmon(G4QContent const&, CLHEP::HepLorentzVector const&, bool
28	0.65	286,200,592	214,222	G4ChiralInvariantPhaseSpace::ApplyYourself(G4HadProjectile const&, G4Nucleus&, G4Had
20	0.44	191,654,496	1,325,140	G4QEnvironment::FSInteraction()
49	0.37	162,692,480	1,016,828	G4QNucleus::EvaporateNucleus(G4QHadron*, std::vector<G4QHadron*, std::allocator<G4QH
31	0.26	115,009,344	128,106	G4QCaptureAtRest::AtRestDoIt(G4Track const&, G4Step const&)

- Looking for a large amount of allocations or number of calls (and their ratio) for a possible problem

Heap Memory Snapshot

- Total bytes in heap and number of the allocations

Counter: MEM_LIVE,

Sorted by self 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()**

Sort by cumulative cost

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<

Discussion Thread I

- There were reports on memory size increase or a possible memory leak using HP physics list (2/10-17,2012). Could the current memory profiling (IgProf) detect/shed light on this type of issue?
  - HP components are not currently profiled, but can be added easily
  - Many contributors (A. Ribon, T. Koi, V. Invantchenko, W. Matysiak, M. Kasztelan, P. Gumplinger, G. Folger, G. Cosmo, D. Wright) observed different (conflicting) results
  - After using Igprof on HP, no direct memory leakage was observed but increase of memory churn was confirmed by A. Ribon (and partly fixed by T. Koi)
  - More systematic analysis of memory profiling results may be necessary

Discussion Thread II

- John Apostolakis pointed out that 15% degradation in CPU performance of the CMS detector simulation between 9.4.p04 and 9.5 was observed. Why did not the G4GP task detect this?
  - The current benchmarking/profiling protocol has been deployed progressively since 9.5 (current reference release)
  - While migrating to the new hardware platform (mid April), we haven't re-profiled the old 9.4 releases yet at that time
  - When 9.4.p04 was released (4/20/2012), we were re-profiling earlier versions of 9.5 series (5 versions up to ref03)
  - After the report, we had profiled 9.4.p04 on the new platform and observed a similar degradation

Future Improvements

- Tools
  - can we add other tools? (vagrind/callgrind, CodeAnalyst, Vtune)
  - are our tools ready for parallelism (igprof validation, TAU)
- Applications
  - should other applications be added?
- Profiled samples and physics lists
  - any other physics list to be added ? HP?
  - are we covering all relevant energy points ?
- Other
  - can we build tools to pin point detailed performance problems?
  - are we ready for multi-threaded Geant4?

Conclusions

- A protocol for benchmarking and profiling has been defined and has been executed
- Can we still improve it?
- Any feedback from users/developers is welcome