Final Report of the ATLAS Detector Simulation Performance Assessment Group

J. Apostolakis, A. Buckley, A. Dotti, Z. Marshall

March 31, 2010 CERN-LCGAPP-2010-01

Abstract

In this document we discuss the output of the Detector Simulation Performance Assessment Group. We list a number of methods which consume significant CPU time and/or allocate significant memory as first targets for potential improvements which are uncorrelated with the physics content of the simulation. We suggest several approaches for reducing CPU time, by reducing the number of steps or tracks simulated. To maintain physics accuracy for the estimation of second-order quantities, e.g. resolution, the CPU reduction should ideally be achieved while maintaining the important steps and tracks which contribute to the leakage of energy from an absorbing element into an active element (or vice-versa), or between active elements. An investigation is attempting to identify the most productive areas for reducing the number of steps and tracks simulated beyond the existing use of production thresholds.

We have identified a number of existing Geant capabilities which can be exploited in order to reduce the number of simulation steps and, proportionally, the CPU time. A number of these carry an expected low or moderate impact on the physics accuracy of typical setups, and are strong candidates for investigation in the near and medium future. Others carry the risk of greater impact on physics accuracy, and will require greater investigation. We have started to investigate the relationship for electrons between energy deposition, kinetic energy and step size, in order to identify whether a fraction of steps could be eliminated with negligible or low impact on the location or quantity of energy deposition. The initial finding is that opportunities could exist to eliminate about 8% of the steps, responsible for about 1% of energy deposition; there is need to study the impact of this change and to determine the best way of implementing it. Deeper investigations of the relation between these quantities is planned. We consider the allocation and use of memory as an important factor for further performance improvement and a target area for short and medium-term investigation.

Note: The Abstract and Section 4 comprise the Executive Summary

Contents

1 Introduction

The ATLAS Detector Simulation Performance Assessment Group was set up as a joint effort of the ATLAS and SFT Groups in November of 2009.

1.1 Mandate and Membership

The mandate of the group was to identify the parts of the ATLAS detector simulation and of the GEANT4 toolkit which are currently limiting events production. The group was to report on the key elements of the ATLAS and GEANT4 implementations which are the highest consumers of CPU cycles, memory allocations, and memory use. It was to note additional issues which were found to be or could be relevant to performance. It was to make a ranking of the leading elements for improvement, with a first assessment of their potential impact.

The group comprised simulation software experts from both the ATLAS and GEANT4 Collaborations, each of which is familiar with the code involved. The members are Andrea Dotti and John Apostolakis from GEANT4, and Andy Buckley and Zach Marshall from ATLAS.

1.2 Structure of the Simulation and Understanding of the Mandate

The ATLAS simulation software is divided into several steps, shown in Figure 1 [1]. The most time consuming step of the simulation, which limits production for typical physics events, is the step labeled "Simulation." In this step, particles from a physics event generator (e.g. PYTHIA, HERWIG, etc.) are propagated through the ATLAS detector using the GEANT4 toolkit to simulate interactions. Energy deposited in active detector elements is collected by "sensitive detectors."

It is implicit in the timescale and scope of the mandate that the group was meant to provide suggestions to improve the performance, but not to actually implement those suggestions. The mandate pointed to the limiting factor in production of simulated events, which in the case of ATLAS is CPU time per event. It was agreed, therefore, that CPU time per event would be viewed as the most critical performance metric. Other metrics, including total memory consumption, resident set size, memory "churn" (allocation and deallocation per event), cache misses, etc., are viewed as second-order effects that should only be addressed insofar as they affect the CPU time per event.

Minimum bias (minbias) events generated at a center of mass energy of 10 TeV using PYTHIA were used for all performance evaluations. Results were cross-checked against top anti-top production ($t\bar{t}$) events generated using PYTHIA at center of mass energy of 10 TeV. While it is clear that PYTHIA is sub-optimal for top physics studies, for simulation performance evaluations only event activity is relevant. Generally, the top events produced more significant activity in the central calorimetry, as expected. The general conclusions and major

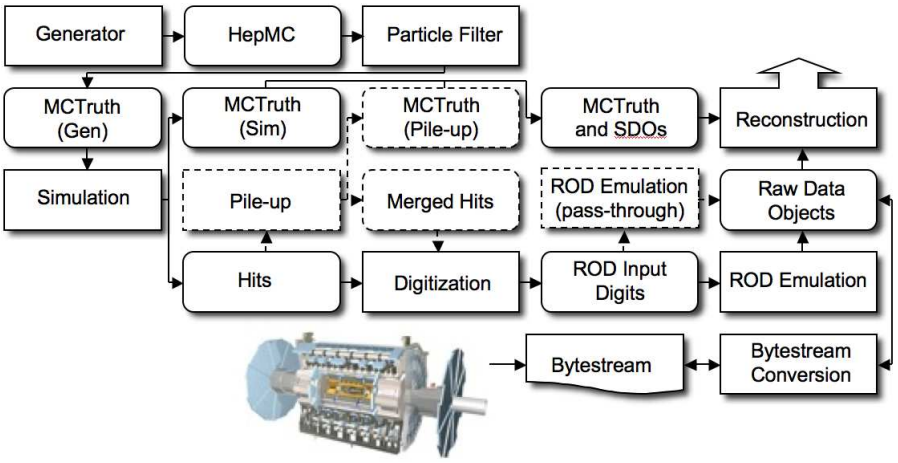

Figure 1: The flow of the ATLAS simulation software, from event generators (top left) through reconstruction (top right). Algorithms are placed in boxes and persistent data objects are placed in rounded boxes. The optional pile-up portion of the chain, used only when events are overlaid, is dashed. Generators are used to produce data in HepMC format. Monte Carlo truth is saved in addition to energy depositions in the detector (hits). This truth is merged into Simulated Data Objects (SDOs) during the digitization. Also, during the digitization stage, Read Out Driver (ROD) electronics are simulated. This paper will mostly address the current production choke-point, which is the Simulation step that relies on the GEANT4 toolkit.

hotspots in both samples, however, remain the same. The top events are representative of typical signal events at the LHC, and the minimum bias of typical backgrounds.

1.3 Methods for improving simulation performance

There are two fundamental ways to improve simulation performance: reducing the simulation time required by removing or simplifying the treatment of "unnecessary" particles, and by improving the simulation code itself. The approaches must be combined in order to achieve the best software performance without sacrificing the accuracy of the simulated detector physics.

The GEANT4 toolkit can be used to describe physics for many different applications. Naturally, the physics most of interest to ATLAS is only a small subset of the physics that can be simulated using the toolkit. Very low energy particles in large blocks of iron shielding, for example, need not be simulated in order to achieve even the most precise physics results. One must be careful in modifying cuts on particles, however, that one does not affect detector response. Section 2 will describe the profiling already done to determine the magnitude of this problem in the current ATLAS simulation and make some suggestions about how to best address this issue.

Profiling is generally sufficient to reveal fundamental problems with code. The solutions are also well-known. In some cases, CPU time can be traded for memory use or disk space. Code optimization may involve both improving code where necessary and, when CPU time is by far the limiting factor, reducing CPU use at the expense of other resources. Section 3 will describe the profiling done to expose problematic methods in the code and suggest some modifications to improve performance.

Section 4 presents a prioritized summary of the key issues with time scale estimates for each.

The Appendix provides some of the recipes for running benchmarking code that were used during these analyses.

This report is also available online [2], where many of the figures have better resolution.

2 Simulation performances: tracking of particles

The time spent in the simulation is proportional to the number of steps (`G4Step`s) that are needed to track a particle. We can thus estimate the simulation time in terms of number of performed `G4Step`s. Table 1 shows the number of `G4Step`s for different regions of the ATLAS detector and for different particle species. The numbers (in units of 10^6) have been obtained with the simulation of 50 $t\bar{t}$ events.

Number of G4Steps Units in 10^6	e^-	e^+	Photon	$\pi^\pm$	Proton	Neutron	Other	Total
Px	2.32	0.24	2.37	0.68	0.14	0.30	0.21	0.3%
Sct	2.42	0.37	6.52	0.82	0.20	0.89	0.25	0.6%
Trt	5.37	1.05	40.20	1.82	0.50	5.46	0.53	2.9%
Sev	49.88	7.35	40.63	0.42	0.35	3.82	0.46	5.4%
Cry	48.92	6.48	49.01	0.34	0.40	7.38	0.41	6.0%
Pre	3.99	0.49	10.41	0.09	0.06	1.73	0.03	0.9%
EMB	79.94	11.76	66.26	0.74	0.56	32.59	0.38	10.2%
EMEC	137.13	22.73	114.97	1.34	1.03	63.34	0.65	18.1%
HCB	10.10	0.96	14.88	0.14	0.18	11.46	0.25	2.0%
HEC	17.49	1.67	23.01	0.23	0.30	20.79	0.58	3.4%
FC1	303.56	31.82	204.14	1.79	1.61	64.26	0.80	32.2%
FC23	90.15	13.18	42.67	0.91	0.83	87.99	0.72	12.5%
FCO	10.23	1.46	3.80	0.10	0.12	1.01	0.03	0.9%
LAr	2.50	0.25	2.97	0.06	0.11	0.54	0.05	0.3%
Mu	31.85	4.34	22.09	0.11	0.23	4.25	0.50	3.4%
Other	4.89	0.86	10.13	0.13	0.11	1.05	0.04	0.9%
Total (%)	42.4%	5.6%	34.6%	0.5%	0.4%	16.2%	0.3%	100%

Table 1: Number of `G4Step`s processed for some ATLAS regions and different particle species. These number have been collected during the simulation of 50 $t\bar{t}$ events.

The various subdetector categories in Table 1 are:

- Px : Pixels and pixel services
- Sct : Semi-conductor Tracker and support structure
- Trt : Transition radiation tracker and support structure
- Sev : General ID services, beampipe, beam luminosity monitor, and beam conditions monitor
- Cryo : associated with the cryostat
- Pre : presamplers in front of the barrel and endcap calorimeters
- EMB : anything in the electromagnetic barrel calorimeter (including EMB-specific support)
- EMEC : anything in the electromagnetic endcap calorimeter (including EMEC-specific support)
- HCB : Tile (including tile-specific support)
- HEC : anything in the hadronic endcap calorimeter (including HEC-specific support)
- FC1 : forward calorimeter (FCAL), EM module
- FC23 : forward calorimeter (FCAL), hadronic modules
- FCO : Other parts of the FCAL
- LAr : anything not included in those volumes above, but still a part of the liquid argon system
- Mu : All muon systems
- Other : anything else

A very large fraction of `G4Step`s (80%) are needed to simulate the electromagnetic showers in the liquid argon (LAr) calorimeters (barrel, endcap electromagnetic and hadronic, and forward calorimeters). The simulation optimization should concentrate on the simulation of electrons in LAr calorimeters.

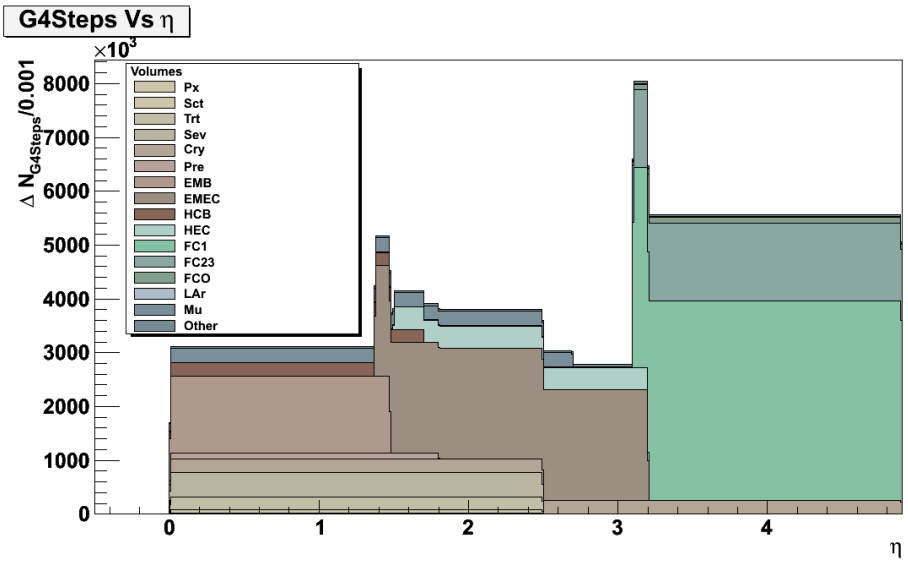

Figure 2: Number of `G4Step`s performed in the considered ATLAS volumes as a function of pseudo-rapidity.

Figure 2 shows the number of `G4Step`s performed in the different detectors, spread evenly over the detectors' coverage in pseudo-rapidity. This plot shows that a large fraction of the simulation is performed in the very forward region (see FC1, FC23 and FCO rows of Table 1). This region (η >3.2) is responsible for 46% of the steps. It is likely that the distribution of steps within the detectors is not uniform in pseudo-rapidity, and so the distribution may be even more heavily weighted towards the forward region than this picture indicates.

GEANT4 provides a set of built-in tools to improve the performance of any simulation. These potentially allow a reduction in the time spent in the simulation at the price of a decrease of the detail of the simulation. Is it up to the developer of the application to study the effect of these simplifications on the quality of the predictions and identify a reasonable trade-off between speed and accuracy.

2.1 Production threshold

The first possibility is to tune the production threshold. Production thresholds are discussed in Chapter 5.4 of the GEANT4 User's Guide for Application Developers [3]. These cuts, applied by default to the δ-electron production and bremsstrahlung electromagnetic processes, prevent the production of a secondary below threshold. The threshold values, which are identical for electrons, positrons, and photons, for the ATLAS LAr calorimeters are show in Table 2.

When defining a production threshold it is important to verify the effect of

Subdetector	Range Cut	Absorber Thickness	Liquid Argon Thickness
EM Barrel	100 µm	1.13-1.53 mm	200 µm
EM Endcap	100 µm	1.7-2.2 mm	200 µm
Hadronic Endcap	1 mm	25-50 mm	8.5 mm
Forward Calorimeter	30 µm	Varies	269-508 µm

Table 2: Range cuts, absorber thickness, and liquid argon thickness for each of the ATLAS LAr calorimeters. The volume thicknesses are provided to give a sense of the fundamental scale of the geometry of the detectors.

this in the simulation quality. Figure 3 [4] shows the variation of the response (left) and resolution (right) of the electromagnetic (EM) barrel calorimeter. These plots show a bias of less than 1% in response and an increase of 2% of the resolution varying the production threshold range cut from 10 µm to 1 mm.

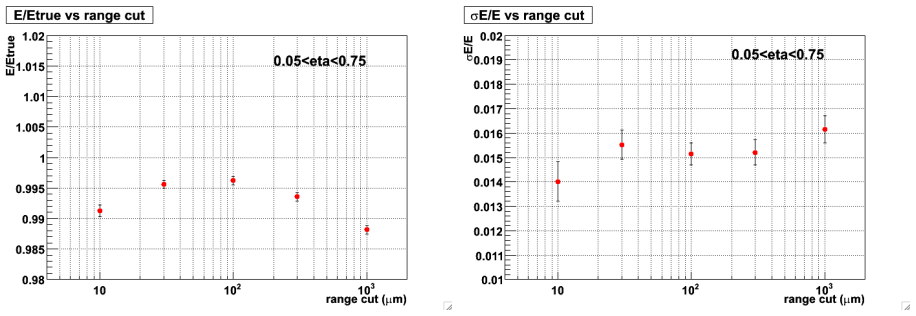

Figure 3: Left, variation in average deposited energy (response) and, right, variation in energy resolution with production threshold cut. Simulation of the EM barrel calorimeter with the QGSP_BERT physics list.

The values of these cuts should be optimized studying the variation of CPU time as a function of the cut value. As an example, the left plot of Figure 4 shows the correlation between the response to electrons as a function of the CPU time (in relative units) obtained with a simplified Pb-LAr calorimeter. In the right plot different physics lists and GEANT4 versions are compared in terms of CPU performance as a function of the production cuts. Red points show the results for GEANT4 9.3 with the default EM physics list. The ∼35% faster EMV option has been studied by ATLAS and results showed a poor agreement with test-beam data.

In this simplified setup a change in the production threshold from 100 µm to 1 mm would decrease CPU time spent in the simulation by ∼15%, at the price of a reduction of collected visible energy well below 1%.

A review of the production threshold could bring a gain a substantial benefit in terms of CPU time, while keeping a good quality of physics performance.

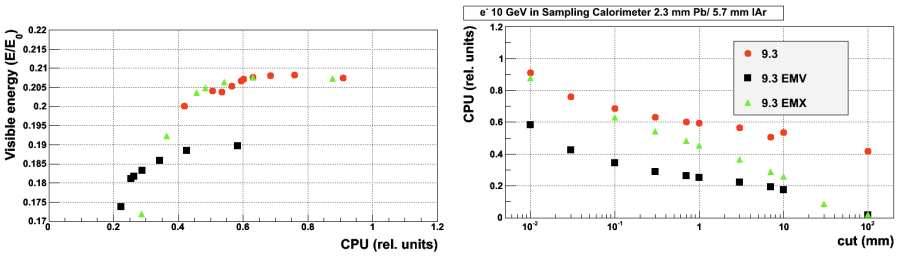

Figure 4: Simulation of 10 GeV electrons in sampling calorimeter 2.3 mm Pb / 5.7 mm LAr. Left: Bias on the energy response as a function of CPU timing. Right: CPU timing as a function of the production threshold cut. Different physics lists and GEANT4 version are shown.

At the moment ATLAS strategy is to use a single cut for electrons and photons. However it is also possible to set a cut value separately for electrons and photons. Table 3 (the Opt0 rows) summarizes the CPU performance and the deposited energy (in active media) for different cut values. The results have been obtained on a simplified ATLAS barrel EM calorimeter setup (primary is a 10 GeV electron). Results are normalized to ATLAS defaults.

As expected, increasing the cut value reduces the time spent for simulation at a price of less energy deposited in the active material. For example increasing the cut from 0.1 mm (current choice) to 0.7 mm reduces the simulation time by 12%, and the energy deposited in LAr is reduced by almost half a percent. Choosing independently the cut value for electron and photons (e.g increasing only the cut for photons to 0.7 mm) reduces the simulation time by 4% while keeping the difference in response within 0.4%.

Increasing the cut values is, at the moment, the most promising strategy to increase CPU performance while keeping a (relatively) small bias on the deposited energy.

2.2 Other cut strategies

ATLAS is using a cut for neutrons (tracking is abandoned after 150 ns), and neutrinos are not tracked. These allowed for a significant speed up of the simulation. Other strategies to decrease the simulation time provided by GEANT4 are discussed here.

2.2.1 Cut per region

A different threshold cut can be set for each defined `G4Region`. At the moment `G4Regions` in ATLAS are defined at the detector level. Increasing the

E.m. Option	Cut $e^{\pm}$ (mm)	Cut γ (mm)	CPU perf.	$\Delta E_{\rm dep}$	Notes
Opt0	0.1	0.01	+6%	+0.52%	
Opt0	0.1	0.1	—	—	ATLAS: Reference EM Barrel and End-Cap
Opt0	0.1	0.7	-4%	-0.39%	
Opt0	0.7	0.7	-12%	-0.45%	
Opt2	0.7	0.7	-30%	-1.55%	Apply Cut: EMX (see Section 2.2.2

Table 3: CPU performance and deposited energy relative difference as a function of different cuts. Results are normalized to ATLAS defaults.

granularity of `G4Region`s would allow optimization of each part of the detector independently and reduce the simulation time in the uninteresting regions (using high cut values in dead-material).

Review of the cut values for the different regions is probably the easiest and most effective strategy to reduce CPU requirements.

Some time ago, a "Range Monitor" was developed for ATLAS. The code is still available (in ∼disimone/public/RangeMonitor/ on CERN AFS), although it would need to be ported to a newer release of Athena. The monitor runs at simulation time (e.g. during simulation of $t\bar{t}$ events), collecting information about all particles created in a particular volume. For that volume, it tests how many of the particles escape, and based on that information proposes range cuts for each volume. The volumes with similar range cuts can then be collected into a single `G4Region` and the range cuts in that region can be modified during a typical simulation run. This tool may be helpful in evaluating and assigning appropriate range cuts, particularly to dead materials.

2.2.2 Electromagnetic option EMX

The production threshold cut can be turned on for any process. The electromagnetic physics option 2 turns on a threshold cut for all electromagnetic processes [5]. The EMX physics lists use this option (caveat: up to GEANT4 version 9.1, EMX also used the sub-cut option, described in Section 2.2.3). This option gives an important increase in performances, at the price of introducing a relatively large bias in the visible energy.

Figure 4 shows that EMX is about 20% faster than standard electromagnetic-physics (with a cut value of 1 mm) . The energy difference is of about 2%. One may be able to decrease the cut value and apply the EMX option in order to decrease both the CPU time and the difference in deposited energy.

2.2.3 Sub-cut option

Soft electromagnetic particles play a different role depending on the volume in which they are produced. Secondaries produced in the active material have a higher impact on calorimeter simulation than secondaries produced in the absorber. Ideally, one would like to have a high range cut value for secondaries produced in the absorber material that are completely absorbed before reaching any active material, while having a much lower range cut for particles that are either produced in the active material or near the boundaries absorber/active material. GEANT4 allows this through the sub-cut option.

Before GEANT4 version 9.2, the EMX electromagnetic option included this sub-cut by default. Since version 9.2, however, this option has been turned off by default, because it caused some issues when used in combination with parallel navigation. This option may provide some benefits in ATLAS.

Figure 5 shows the comparison in a simplified ATLAS HEC calorimeter of different physics lists as a function of the range cut. Concentrating on GEANT4 version 9.1.p02 with EMX (sub-cut option activated), deposited energy and energy resolution are very stable as a function of the range cut.

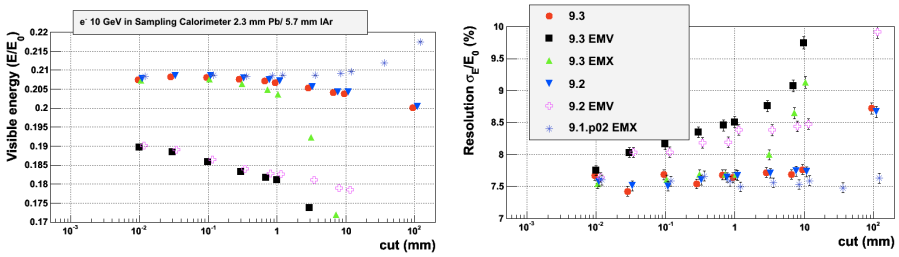

Figure 5: Simulation of 30 GeV electrons in a simplified ATLAS HEC. Left: visible energy as a function of range cut. Right: resolution as a function of range cut. Different versions of GEANT4 and different physics lists are shown.

2.3 Low energy secondaries

GEANT4 also provides tracking cuts, discussed in paragraph 5.4.6 of the GEANT4 User's guide for Application Developers [3]. In order to evaluate the potential impact of these cuts, one can estimate the role of the low energy particles.

From Table 1 one can see that the largest fraction of the steps is taken tracking electrons (42%). The LAr calorimeters are clearly the dominant detectors in terms of number of steps (three quarters of all steps). We thus concentrate on energy deposited by electrons in LAr in this section.

Electrons undergo electromagnetic processes developing electromagnetic showers; basically all their kinetic energy is released in the calorimeters. A tracking cut would prevent further tracking of the electron and all kinetic energy would

be deposited locally. In this sense a tracking cut "disturbs" the development of electromagnetic showers. If an electron is killed in the absorber, all its kinetic energy is "lost" and the value of the measured energy decreases. On the other hand, an electron being killed in the active material (LAr) increases the response of the calorimeter. The two effects will compete and it is difficult to predict the final effect without dedicated studies. However, one can try to estimate the order of magnitude of this "displacement."

The left plot of Figure 6 shows the electron spectra in terms of kinetic energy for the LAr barrel calorimeter. A single electron contributes to this distribution several times, once for each `G4Step`, until it finally reaches zero kinetic energy.

From this distribution we can compute the fraction of `G4Step`s that would be cut by a tracking cut. Since the simulation time spent in one detector is proportional to the number of `G4Step`s, this fraction is a rough measurement of the fraction of CPU needed to track low energy particles.

We calculate the fraction of energy deposited by these cut `G4Step`s, obtaining a rough estimate of the effect, in terms of deposited energy, of a tracking cut.

Figure 6 illustrates the results for a tracking cut of about 44 keV. The cumulative distribution of the number of `G4Step`s as a function of their kinetic energy is shown in the middle plot; about 8% of the `G4Step`s have a kinetic energy below 44 keV. Finally the third plot shows the distribution of the deposited energy at each `G4Step` as a function of the particle kinetic energy at the beginning of the step. The area at the left of the tracking cut is the energy deposited by killed tracks and is less than 2% of the total deposited energy.

Results, for different values of the tracking cut and for the dominant volumes, are summarized in Table 4. The use of such tracking cut could give an improvement in the CPU performance. Initial trials have been inconclusive as to the value of such a tracking cut in improving performance. Further investigation is planned. In addition such a strategy requires a detailed validation on the ATLAS setup to evaluate the effect on calorimetric observables.

2.4 Conclusions

In this section we have briefly outlined several strategies that could lead to a possible speed-up of the simulation of the ATLAS detector. Additional investigation of these strategies should be carried out, and a validation of physics results should be performed. We concentrated on tools already available in GEANT4 that could reduce the computing time with essentially no further software development.

Given that a large fraction of the simulation time is spent tracking EM particles in calorimeters (mainly LAr EM barrel, endcap, hadronic endcap and forward), a possible list of prioritized recommendations is as follows:

- We can clearly see that the majority of the `G4Step`s are tracked in the forward ($|\eta| > 3$) region. Documenting the necessary physics simulation accuracy in forward region and seeking a compromise between peformance

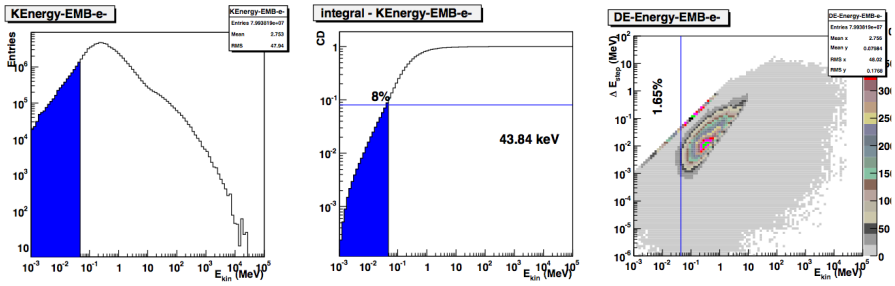

Figure 6: `G4Step` distribution as a function of the particle kinetic energy (E_{kin}) at the beginning of each `G4Step` (left), the cumulative distribution of this quantity is shown in the middle. The third plot shows the two-dimensional distribution of the deposited energy (E_{dep}) in each `G4Step` as a function of the track kinetic energy. The effect of a possible tracking cut at 44 keV is shown (see text for explanation).

and accuracy should be investigated with priority. Given that 47.5% percent of steps in $t\bar{t}$ events are made in the FCAL, the potential for CPU savings is the greatest for this detector and could reach 33% of CPU time.

- Significant gains may be achieved by reducing the primary particle pseudorapidity acceptance in the simulation. Currently all primaries are simulated out to pseudorapidity of 6.0, resulting in some unnecessary activity in the very forward region.

- Increasing cut values in the FCAL is also a promising strategy to reduce CPU time.

- An optimization of the production range cut, or the use of advanced electromagnetic options could lead to a reduction of the computing time, in some cases, of several percent. The bias introduced by these cuts has to be studied in more detail, first on simplified setups and then on full ATLAS geometry.

- An important fraction of the simulation time is used in tracking low energy particles. These are responsible for a relatively small fraction of the energy deposited in the calorimeters. A tracking cut could reduce the number of steps required in LAr by about 10%, while affecting only 1% of the deposited energy.

Additional CPU time could be saved by the use of more advanced features such as, for example, sub-cut or EMX options. The effect of these options is, at the moment, not completely clear.

Detector	Tracking cut threshold	Fraction of G4Steps (with E_{kin} <Thr)	Fraction of E_{dep} (for steps with E_{kin} <Thr)
EMB	30 keV	5%	0.87%
	44 keV	8%	1.65%
	53 keV	10%	2.27%
	91 keV	20%	5.30%
EMEC	30 keV	5%	0.83%
	44 keV	8%	1.56%
	53 keV	10%	2.13%
	110 keV	20%	6.66%
FC1	17 keV	5%	0.43%
	30 keV	8%	1.06%
	37 keV	10%	1.43%
	76 keV	20%	4.63%
FC23	21 keV	5%	0.50%
	36 keV	8%	1.23%
	44 keV	10%	1.67%
	110 keV	20%	6.00%

Table 4: Fraction of energy deposited by electron with E_{kin} smaller than the given threshold. The fraction of `G4Step`s with E_{kin} smaller than threshold is also displayed.

3 Software performance measurement

Software performance metrics are obtained for the stage of ATLAS simulation involving GEANT4, using the callgrind and Hephaestus tools to count CPU cycles and memory allocation respectively.

Callgrind is accessed as a tool in the valgrind suite [6], instrumenting function call rates through the symbol call tree. Hephaestus [7] is a leak-checker option on the ATLAS Athena framework, designed to monitor calls to `malloc` within the main Athena event loop, but which optionally produces `malloc` call trees usable by callgrind analysis tools such as kcachegrind (N.B. Hephaestus misses calls to `calloc`/`realloc`, but these are not used in the relevant GEANT4 and Athena code). For these tests, several modifications are made to the normal Athena setup:

- Athena's normal use of the tcmalloc memory allocators is suppressed, since tcmalloc does not play well with valgrind
- Hephaestus' usual leak checking scope is reduced from `Gaudi::Algorithm::sysExecute()` to `G4TrackingManager::ProcessOneTrack()`, which still should be sufficient to catch all relevant memory allocation and deallocation.

Monitoring output from G4 simulation runs of pre-generated minimum bias and $t\bar{t}$ events has been used to identify the busiest processing hot-spots in the ATLAS simulation code. The benchmark version of the ATLAS simulation software to be used for these studies is the ATLAS software release 15.6.1, for which the production version of GEANT4 is 4.9.2.patch2.atlas1, a modification of the standard G4.9.2.patch2 to also include:

- A new stepper implementation, `G4AtlasRK4` (turned off by default, including for these studies)
- A new magnetic field, `G4CachedMagneticField` (turned off by default, including for these studies)
- Bug fixes to `G4QElasticCrossSection` and `G4HadronElastic`

For comparison, we will use a new ATLAS patch of G4.9.2.patch2 which is not yet used for any ATLAS production, G4.9.2.patch2.atlas3. In addition to the private patches in the "atlas1" version, "atlas3" includes:

- A new version of `G4TouchableHistory` to reduce per-event "churn" of memory allocation
- A patch to modify EM physics initialisation in `G4ProductionCutTable`. This patch results in range cut changes on the level of 0.5%, and so may affect physics results
- Modification to avoid per-hadron newing of double arrays in the Bertini cascade (in lieu of a more complete Bertini model rewrite)

The "atlas2" patch only differs from atlas1 in that it is built against a different patch version of CLHEP. No tests have been performed with this patch.

More detailed "recipes" for running the ATLAS simulation in these configurations with minimum bias and $t\bar{t}$ events, with callgrind and Hephaeustus instrumentation, may be found in the Appendix.

3.1 Results with Hephaestus

Hephaestus provides two quantities for every method called during the simulation: the inclusive amount of memory allocated by that method and all methods that it calls (*incl*), and the amount of memory allocated by that method alone, including calls to `malloc` and stl functions (*self*). The methods with large self typically indicate hotspots for optimization, and methods with large incl may indicate places were code re-writes or even re-designs may be beneficial. Table 5 shows the methods with largest self in both the atlas1 and atlas3 builds, and Table 6 shows the methods with largest incl in both builds, eliding those which also appear in Table 5. In both tables, the displayed percentages have been scaled from the raw Hephaestus output as shown in kcachegrind to the percentage allocated to the top level `G4SteppingManager::Stepping` function,

	atlas1		atlas3	
method	self	incl	self	incl
malloc	62.11	62.11	47.10	47.10
G4NavigationHistory::G4NavigationHistory	23.69	23.69	29.37	29.37
G4Transportation::PostStepDoIt[a]	23.69	47.37		
G4NucleiModel::generateParticleFate	7.68	24.69	9.36	37.07
LArG4Identifier::LArG4Identifier(LArG4Identifier const&)[b]	5.73	5.73	7.15	7.15
G4ElementaryParticleCollider::generateSCMfinalState	5.42	7.65	6.54	16.27
G4NucleiModel::generateModel	4.68	4.68	5.63	5.63
LArHitMerger::process[b]	4.22	4.22	5.27	5.27
G4NucleonSampler::GetMultiplicityT0			4.63	4.63
G4IntraNucleiCascader::collide	3.60	33.47	4.33	47.56
G4NucleiModel::generateInteractionPartners	3.50	3.50	4.28	4.28
G4ElementaryParticleCollider::collide(G4Inucl--ElementaryParticle*, G4InuclElementaryParticle*, G4CollisionOutput&)	2.73	11.76	3.31	21.25
G4NucleonSampler::GetMultiplicityT1			3.22	3.22
G4NucleiModel::boundaryTransition	1.77	1.77	2.18	2.18
LArBarrelCalculator::Process[b]	1.47	1.47	1.84	1.84
G4LorentzConverter::toTheCenterOfMass			1.66	1.66
G4EquilibriumEvaporator::collide(G4InuclParticle*, G4InuclParticle*)			1.55	2.90

Table 5: Methods with the largest allocation and deallocation per event within the method itself (self) in the atlas1 and atlas3 builds.

namely division by 61.07% for the atlas1 build, and division by 67.01% for the atlas3 build.

The entries marked with [a] indicate the method that changed most dramatically between 15.6.1-4.9.2.p2.a1 and 15.6.1-4.9.2.p2.a3: the `G4Transportation::PostStepDoIt` function. This improvement was obtained by use of better memory management in `G4TouchableHistory` to reduce memory churn. In "atlas1," the memory activity was assigned to `G4Transportation` as self, since `G4TouchableHistory` had an inline constructor. The entries marked with [b] are those which are not present or which have significantly lower values of the metrics for MB events. These are, as expected, concentrated in the LAr methods, since with MB events a reduction in calorimetric activity is expected relative to $t\bar{t}$ events. The entries marked with [c], by contrast, are those which are increased in MB events relative to $t\bar{t}$, in particular the `LArFCALCalculator-Base::identifier` method: this reflects the increased proportion of forward activity in minimum bias events, interacting with the forward calorimeter (FCAL).

From analysis of these Hephaestus profiles, it is clear that four effects dominate the memory allocation and deallocation per event during simulation with Athena in the atlas1 build:

- `G4TouchableHistory` is responsible for 25% of the memory churn
- QGSP_BERT (in particular the Bertini cascade) and its related methods are responsible for 40% of the memory churn
- `G4NavigationHistory`'s constructor is responsible for 25% of the total churn

- `LArG4Identifiers` and related methods are responsible for 10% of the memory churn

The first of these, `G4TouchableHistory`, is completely alleviated by the patches in the atlas3 build. No further action is necessary.

An almost complete rewrite for optimization of the Bertini methods is already underway by the GEANT4 collaboration. This rewrite may be available for testing in the coming months, but it may take some time before it reaches a sufficiently high level of validation for physics production in ATLAS. Nonetheless, it would be extremely beneficial for the ATLAS collaboration to closely follow these developments and assist with testing when a new version is available.

A `G4NavigationHistory` is carried by all tracks, and whenever secondaries are created the navigation history is copied from the primary. It seems unlikely that any dramatic optimization is possible. But indirect optimizations by, for example, reducing the complexity of the geometry tree, may still be possible. In the inner detector, for example, it should not be necessary to embed all modules within a mother. By removing the mothers and relying on the voxelization of GEANT4, it may be possible to reduce memory churn and speed up the simulation. In this example in particular, the number of total steps may also be reduced.

`LArG4Identifiers` identify read-out channels within the liquid argon calorimeter of ATLAS. Far too many energy deposits are made in the liquid argon during a typical event to save each one separately to disk. Instead, the hits are collected during the simulation step into read-out channel collections, each with its own identifier. In principle, once an identifier has been created, it should be kept in memory for the remainder of the event. It is reasonable that some time would be spent in the constructor of the identifiers. However, that the vast majority of that time is spent copying `std::vector`s indicates that the construction of the identifiers has not been optimized or that unnecessary copying is being done. It is recommended, therefore, that this code be examined carefully for opportunities for optimization.

Method	atlas1 incl	self	atlas3 incl	self
G4SteppingManager::Stepping	100	0	100	0
G4SteppingManager::InvokePSDIP	87.43	0	84.36	0
G4SteppingManager::InvokePostStepDoItProcs	87.43	0	84.36	0
G4HadronicProcess::PostStepDoIt	40.00	0	54.90	0
G4CascadeInterface::ApplyYourself	39.00	1.06	53.63	0.49
G4InuclCollider::collide	37.40	1.06	52.30	1.04
G4TouchableHistory::G4TouchableHistory			29.37	0
G4Transportation::PostStepDoIt[a]			29.37	0
G4ElementaryParticleCollider::collide(G4InuclParticle*,G4InuclParticle*)	11.75	0	21.25	0
LArG4SD::ProcessHits[b]	11.15	0	13.91	0
G4ElementaryParticleCollider::generateMultiplicity			8.19	0.31
std::vector<G4InuclElementaryParticle>::_M_insert_aux	6.27	0	7.60	0
std::vector<short>::operator=	5.73	0	7.15	0
st::vector<double>::_M_insert_aux			6.57	0
std::vector<G4CascadeParticle>::_M_insert_aux	4.60	0	5.64	0
std::vector<std::pair¡G4InuclElementaryParticle, double>>::_M_insert_aux	3.39	0	4.15	0
std::vector<int>::push_back	2.68	0	3.18	0
std::vector<G4CascadParticle>::push_back	2.45	0	3.00	0
G4EquilibriumEvaporator::collide	2.42	1.29		
std::vector<std::vector<double>>::push_back	2.34	0	2.81	0
std::vector<double>::push_back	2.23	0	2.88	0
std::vector<double>::vector	1.93	0	2.33	0
G4NucleiModel::boundaryTransition	1.77	0		
std::vector<double>::_M_fill_insert	1.64	0	1.99	0
std::vector<LArG4Identifier>::push_back [b]	1.51	0	1.88	0
LArBarrelCalculator::indentifier [b]	1.47	0	1.84	0
LA4G4::EC::EnergyCalculator::identifier [b]	1.41	0	1.73	0
G4QEnvironment::Fragment [c]	1.21	0.01	1.54	0.01
G4QEnvironment::FSInteraction [c]	1.21	0.01	1.52	0.01
G4QEnvironment::HadronizeQEnvironment [c]	1.19	0	1.51	0
G4Quasmon::Fragment [c]	1.18	0.01	1.48	0.01
G4Quasmon::HadronizeQuasmon [c]	1.16	0.01	1.48	0.01
LArFCALCalculatorBase::identifier [c]	1.15	0	1.45	0

Table 6: Methods with the largest allocation and deallocation per event within the method itself and all methods it calls (incl) in the atlas1 and atlas3 builds.

3.2 Results with Callgrind

We have analyzed the output of valgrind runs (with the callgrind module) to study in which software component most time is spent during the simulation. The metric analyzed in these runs is the "Instruction Fetch" (Ir). It should be noted that this metric is a measurement of how many processor instructions are encountered in each code function and not (directly) the time spent in each portion of the code. However, as there is a strong correlation between the time spent in each section of the code and the number of measured Irs, for this first analysis we decided not to use more sophisticated metrics (CPU cycles per instructions, cache misses). Measurements have been obtained by simulating 50 events of MB and $t\bar{t}$ with the two different GEANT4 patch versions (atlas1 and atlas3).

Figure 7 shows the call graph for the most time consuming function: `G4SteppingManager::Stepping`. The time spent in operations different from this is less that 3% of the entire time. The stepping is responsible for tracking the particles in the geometry, simulating physics processes and manipulating hits.

From Figure 7 we can recognize three "areas": the sub-tree that includes the processing of hits in the LAr (`LArG4SD::ProcessHits`) is responsible for 3.98% of the total CPU time. This is expected due to the very large number of hits in LAr calorimeters. This branch of the call graph will not be discussed anymore in the following: potential improvements in this area require the in-depth review of the code; the review of LAr geometry (see later) should also bring benefits in this area.

3.2.1 Definition of step length

In Figure 7, the right-most branch, responsible for the largest fraction of CPU time (68.86%), shows classes responsible for the definition of the current `G4Step`'s length (`G4SteppingManager::DefinePhysicalStepLength`). This is the most important component of the simulation, from the CPU time point of view, and it is possible to sub-divide this branch into two main components:

- Definition of step length from geometry boundaries: a very large part of the time is spent in transportation process (65.75% of CPU of the entire application). There are two main elements that contribute in this area (Figure 9):
  - The magnetic field access: `G4MagErrorStepper::Stepper` is responsible for 26.47% of the CPU time. This time is spent interrogating directly (2.23%) or indirectly (via `G4ClassicalRK4::DumbStepper`, 20.03%) the magnetic field (`FADS::MagneticFieldMap::GetFieldValue`).
  - The navigation in a single volume defined by the `LArWheelSolid` class. The `LArWheelSolid::DistanceToIn` and `LArWheelSolid::DistanceToOut` methods alone account for about 15% of the simulation time. No other volume is so expensive in these calculation. All

calls to `::DistanceToIn` and `::DistanceToOut` for all other solids account, all together, for ∼2%.

- Definition of step length from physics processes:
  - `G4VEnergyLossProcess::PostStepGetPhysicalInteractionLength` (0.91%)
  - `G4VEmProcess::PostStepGetPhysicalInteractionLength` (1.33%), mainly photo-electron cross section calculations.
  - `G4VDiscreteProcess::PostStepGetPhysicalInteractionLength` (5.87%): the time spent in this method is dominated by the access and calculation of cross sections, through the `G4CrossSectionDataStore` class' methods (about 4%). It is not clear at this moment, given the complexity and variety of this aspect of the simulation, if improvements can be made in this part of the code. Further details on the different components involved in cross section calculation and access can be found in Figure 8.
  - `G4VMultipleScattering::PostStepGetPhysicalInteractionLength` (1.83%): the underlying Urban Model 2 is responsible for 1.31% of the CPU time. A deeper analysis of the Urban Model could show possible improvements.

3.2.2 Physics Processes

The second most significant area in terms of CPU cost of the call graph contains two separate sub-trees: the one responsible for the simulation of the continuous processes (`G4SteppingManager::InvokeAlongStepDoItProcs`) accounts for 6.34% of the total CPU time, a second sub-tree (15.21% of the simulation time) is responsible for the simulation of the discrete physics processes.

Of the continuous processes (Figure 10), the most important contribution comes from the sampling of the energy loss (ionization) fluctuations. `G4UniversalFluctuation::SampleFluctuation` is the most time consuming function; this method requires several calls to random engines and the use of complex mathematical functions (exp, log). Any optimization of this part of the code should thus concentrate on this specific aspect.

The second, and more important, element of the physics simulation is the "discrete" processes simulation (all processes that produce secondaries). The call graph of this part is shown in Figure 11. The contribution from the different processes are summarized in Table 7.

We can observe the following:

- Transportation is the most time consuming process. A further break-down of this component shows that about 1.35% of the time is spent in managing `G4TouchableHistory` objects and 3.02% in `G4Navigator::LocateGlobalPointAndSetup` function. The time spent in these operations is

Name	CPU	Notes
G4Transportation::PostStepDoIt	5.27%	Navigation in the geometry
G4HadronicProcess::PostStepDoIt	3.13%	Contains interface to Bertini (2.60%) and gamma nuclear reaction (0.43%)
G4VMultipleScattering::PostStepDoIt	2.98%	Define final displacement due to MS: Urban Model 2
G4VEmProcess::PostStepDoIt	0.53%	Compton (0.19%) and photo-electric effects (0.21%)
G4UHadronElasticProcess::PosteStepDoIt	0.50%	Elastic scattering of hadrons
G4VEnergyLossProcess::PostStepDoIt	0.38%	Electrons Bremmsstrahlung
G4ParticleChange::UpdateStepForPostStep	0.30%	
G4Track::GetVelocity	0.27%	
TRTTransitionRadiation::PostStepDoIt	0.19%	Simulation of transition radiation in TRT

Table 7: Main contributions to the simulation time of the different discrete processes (PostStepDoIt).

proportional to the complexity of the geometry. In particular, the `LArWheelSolid::Inside` method is responsible of 0.40% of the CPU time, to be compared to the generic `G4Polycone::Inside` method (all other volumes described as polycones) of 0.9%. Any improvement in the geometry should bring benefits also in this part of the simulation.

- The simulation of multiple scattering (Urban Model 2) requires several calls to random engines and mathematical functions, this process takes about 3% of CPU time.
- More details on the simulation of hadronic interactions is shown in Figure 12. From these figure one can see that a (small) amount of time is spent in manipulating the standard library containers of final state particles. This, together with results obtained from Hephaestus runs suggest that a review of the use of standard containers in the Bertini code would be beneficial.
- The method `G4Track::GetVelocity` is responsible for 0.77% of the CPU time of the entire application.
- The (relatively rare) gamma-nuclear reaction process takes approximately the same CPU time as the elastic scattering of hadrons (0.50%).

3.2.3 Conclusions on callgrind analysis

From the callgrind profiling, several hot spots are apparent, classified as high or low priority.

High Priority:

- Significant time is devoted to evaluation of the magnetic field (26%). In particular the time is spent in accessing the value of the field. These results

have been obtained using the ATLAS patch number 3 that already includes potential improvements in this area. Including one or both methods for reducing the number of times the field is accessed (`G4AtlasRK4` and/or `StepperDispatcher`) will offer an improvement already measured to be ∼15%.

- Significant time is spent in geometry related methods (distance to in, distance to out) in the EMEC wheel (15% of the CPU time in $t\bar{t}$ events). The EMEC is implemented as a single, large custom solid. Potentially considerable additional time is spent in energy collection methods because of this design.
  - Undertaking a careful estimate of the speedup achieved by rewriting this volume and the costs would take at most 2.5 person-months.
  - A first rough estimate of the time required to rewrite the geometry is 6 person-months for development and around 6 person-months for validation.
  - Currently there is a single solid in the GeoModel description, which is translated into one (or few) large solids in GEANT4, each of which is composed of a large number of surfaces.
  - Additional code is required to identify within which "readout channel" a particle is located, and whether it is in lead or argon. By using a geometry with finer granularity, this additional code would be greatly simplified.
- Given the importance of the Bertini cascade for the ATLAS simulation, it is important to improve the code relative to this area. An area of improvement to pay particular attention to is the usage of standard library containers – however, no clear hot spot has been identified. The ongoing re-design of Bertini should bring benefits in this area.

Low Priority:

- Multiple Scattering (Urban Model 2) takes about 4% of the CPU time, a deeper analysis of the code should be foreseen to identify possible performance improvements.
- Retrieving cross-sections for the calculation of steps length takes about 5% of the total CPU time. This time is distributed equally among several models and different implementations. It appears challenging to improve the performances in this area.
- Two utility methods are called several times and are responsible for 2% of the CPU time: `G4Track::GetVelocity` and `G4PhysicsVector::GetValue`. Another 2% comes from the use of log and exp functions.
- Additional developments available in GEANT4 version 9.3 offer small additional improvements - tests should be planned to evaluate their benefit.

3.3 Suggestions for the Reduction of Memory

There are several possibilities for reducing the total memory (VMEM) required by the simulation software, many of which will also reduce the resident set size (RSS) and potentially provide a CPU improvement. There are already indications from benchmarking performance that cache misses have a significant effect on simulation speed, and, therefore, by shrinking the simulation's memory footprint it may be possible to considerably improve overall performance.

In the liquid argon barrel and endcap, high voltage field maps are loaded into memory at the beginning of the run and are used to adjust the hit collection during the run. In the barrel, floats are used for the high voltage values. In the endcap, doubles are used for the high voltage values. By changing to floats everywhere, 13 MB of memory can be saved.

Oracle provides "light" and "full" versions of libraries to LCG. By switching to the "light" versions during simulation, 40 MB of memory can be saved. These libraries are not typically used during the run, so RSS would not be reduced noticeably.

GEANT4 divides large volumes with many daughters in order to speed particle transportation. By optimizing this division in the muon system and liquid argon barrel calorimeter, it is possible to save 80 MB of memory in the most current geometries without any noticeable affect on CPU.

It should be possible to release conditions information once it has been used to build the detector geometry and initialize all sensitive detectors. This release is already underway, but eventually should save 16 MB of memory.

Several libraries are already available which change the manner in which memory is allocated. By allocating in different size blocks, or by keeping a "pool" of memory for quick allocation, it is frequently possible to change the memory consumption and CPU performance of a memory-heavy application. The reductions in memory churn described above may reduce the effect of the allocator, but it is recommended that various available allocators are tested, including new versions of malloc, tcmalloc, and tbbmalloc. Preliminary tests indicated a 5-10 MB reduction in total memory consumption with no CPU cost.

When GEANT4 constructs an assembly volume, it creates a volume name that is a mangling of the inputs. This mangling can be rather lengthy. Strings alone, in fact, consume 15MB of memory during the simulation. By modifying this mangling, it may be possible to dramatically reduce the memory consumption of these strings. The sensitive detectors in the muon system, however, rely on these mangled names to determine the location of a step in the geometry tree. If the reliance on mangled names were removed, then the code would be much safer against future developments, and it might be possible to reduce the memory consumed by these long string volume names.

The ATLAS geometry is built first in GeoModel, a geometry format common to the ATLAS reconstruction, digitization, and simulation software. Afterwards it is translated into GEANT4 format, and the GeoModel version can be released from memory. A first iteration of the release was performed last year and provided 50 MB in total memory savings. There were some elements

of the GeoModel geometry, however, which could not be released. The benefit from releasing these additional elements should be evaluated, as it could be significant.

The ATLAS software framework, Athena, loads a great deal into memory during initialization in order to prepare for any type of job that might be run. As a result, many unnecessary libraries and dictionaries are opened in memory. Other libraries and objects could be dropped once the initialization phase has been completed. In particular, all python objects could be dropped, all GeoModel libraries and objects could be dropped, and all transient / persistent converters unnecessary for a simulation job could be dropped. Combined, these represent several 10's of MB of total memory consumed by the simulation, though they are likely not a part of the resident set.

4 Conclusions and Next Steps

Several areas of the ATLAS simulation software have been identified and may be addressed in order to improve the software's performance. After some of these areas have been addressed, it will be necessary to re-evaluate the software's performance, as the performance hot spots may change significantly.

In addition to the data already available, the optimization studies would benefit from running some additional profiling tools. PERFMON2, in particular, is recommended for examining the simulation. It has the advantage of being able to look more realistically at CPU time (where callgrind only looks at instructions, the time required by which might vary by orders of magnitude). It can also provide more subtle performance metrics like cache misses and instructions per cycle, which are critical for understanding code performance and could reveal significant problems in the software.

The most critical problem to address in moving forward is how one can validate the changes that have been made, particularly in cases where the changes are expected to affect detector response. This validation will take a coordinated effort between the detector experts, the simulation experts, and some small group that has the resources to produce sufficiently large physics samples.

4.1 List of Items for Optimization

All time estimates below exclude delays due to release cycles. That is, the time should be read as "working time" and not as total time including the time required to produce a fresh software release and deploy it to the Grid (particularly in the case of ATLAS validation). Where noted, changes can be made in an Athena production cache, easing the task of validation considerably for ATLAS. Most, if not all, of these options can be validated in parallel if sufficient manpower is made available.

No development is necessary for the following options:

CPU Optimization:

- Changing stepper (ATLAS with G4 support) - 15-20% potential CPU gain

  - first using G4AtlasRK4 stepper with G4 9.2
  - 1 week of validation for ATLAS (in a cache already)
  - Next validation G4Nystrom in G4 9.3 (improves on revised G4AtlasRK4 avoiding extra square roots added.)
  - Should be delayed until ATLAS adopts G4 9.3
  - Optionally the caching of the magnetic field via G4CachedMagneticField for additional small CPU gain

- Optimizing primary pseudo-rapidity cut (ATLAS) - 0-20% potential CPU gain
  - 1 week of preparation for ATLAS
  - 2 weeks of validation for ATLAS (cache-ready)

- Neutron energy cut - 5-10% potential CPU gain
  - 1 week of preparation for ATLAS
  - 2 weeks of validation for ATLAS (cache-ready)

Memory "Churn" Optimization:

- G4TouchableHistory (25% of the total churn per event, addressed by patch "atlas3." Potential CPU gain of 2-4%. Already tested by GEANT4 and by ATLAS in atlas3.)
  - 1 week of validation for ATLAS

Memory Optimization (actions to reduce the total VMEM, and thus RSS):

- "Light" Oracle library versions (∼40 MB)
  - Very short technical validation for ATLAS

- Changes to voxelization (∼80 MB) - improvements prepared in consultation with GEANT4/ SFT team (during 2009)
  - 1 week of preparation for ATLAS
  - 1 week of validation for ATLAS (cache-ready)

All Three:

- Effect of allocators was seen to be below 2% on CPU. It could be less of an issue in future, if memory churn is much reduced. However, effects on both CPU and memory should be evaluated for various drop-in allocators (including the newest version of malloc).
  - 1 week of preparation for ATLAS
  - 1 week of performance assessment for ATLAS

  - 2 weeks of validation for ATLAS

Some development is necessary for the following options. It is difficult to estimate the potential gain without a prototype implementation, so we have chosen to mark upper bounds to give some guide.

CPU Optimization:

- QGSP_BERT (rewrite from translated fortran, which may be available for testing soon) - <20% improvement
  - Several months development for GEANT4
  - Several months validation for GEANT4
  - ∼6 months validation for ATLAS, including testbeam validation
- Revising implementation of Liquid Argon Endcap (currently a big single volume) - <10% improvement
  - 6 months development for ATLAS
  - 6 months validation for ATLAS
- Killing low energy electrons on creation in particular regions or materials - <20% improvement
  - 4 weeks of development and testing for GEANT4
  - 2-4 months of development, pre-validation and testing for ATLAS
  - 2-4 weeks of validation for ATLAS

In particular for the potential reimplementation of the specialised solid for the Liquid Argon Endcap, an effort involving all stakeholders to understand the extent of the work required and the potential benefits, risks and limitations should be undertaken.

Memory "Churn" Reduction:

- LArG4Identifier (responsible for 10% in atlas1, expect optimisation of at least half of that)
  - 2 weeks of development for ATLAS
  - 2 weeks of validation for ATLAS (technical and physics)
- QGSP_BERT (responsible for 40% of the churn per event in atlas1 - before improvement to geometry module) This should be revisited after the Bertini rewrite available for testing later this year
- G4NavigationHistory (constructor) - (responsible for 25% of the churn per event in atlas1) Unclear what actions should be taken. Some revision of ATLAS geometry should be considered (particularly in the inner detector). This should be considered a low priority.

Memory Reduction (actions to reduce the total VMEM, and thus RSS):

- Liquid argon field map in the endcap changing double to float in two places (13MB)
  - 1 week of development for ATLAS
  - 1 week of validation for ATLAS
- Conditions release (16MB)
  - 2 weeks of development for ATLAS (some already complete)
  - 1 week of technical validation for ATLAS
- Further GeoModel release (<50MB, but uncertain exactly how much)
  - 2 weeks of discussion and development for ATLAS
  - 1 week of technical validation for ATLAS
- String mangling in assemblies (10-15MB)
  - 1 week of development for GEANT4
  - 2-4 weeks of development for ATLAS (muon sensitive detectors would need to be modified)
  - 2-4 weeks of validation for ATLAS
- Releasing additional libraries / objects once they are unneeded (uncertain results)
  - Several months of development for ATLAS
  - Short validation for ATLAS

Other Necessary Optimization:

- GEANT4's transportation must improve its treatment of stuck and abandon tracks. These tracks can cause significant problems in a production system that automatically re-tries failed jobs many times. In order to ensure that the physics results are not incorrect, the events currently must be abandoned. Several ideas already exist for improving the treatment, but they must be implemented.
  - 2 weeks of development for GEANT4
  - 2 weeks of testing for GEANT4
  - 1 week of testing for ATLAS
  - 1 week of validation for ATLAS

References

[1] K. A. et al., *ATLAS Simulation Infrastructure*, ATL-SOFT-INT-2010-002 (2010) .

[2] *This report on the web*, http://docs.google.com/View?id=dqzmfc8_19g4v8vcdk.

[3] *Geant4 User's Guide for Application Developers*, http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/.

[4] A. R. et al., *First Report of the Simulation Optimization Task Force*, ATL-SOFT-PUB-2008-002 (2008) .

[5] *Physics Lists of Electromagnetic Physics Working Group*, http://geant4.cern.ch/collaboration/working_groups/electromagnetic/physlist.shtml.

[6] *Valgrind*, Http://valgrind.org/.

[7] *Hephaestus, a Memory Tracker for Athena*, Https://twiki.cern.ch/twiki/bin/view/atlas/usinghephaestus.

Appendices

Profiling "Recipes"

In this appendix we collect some recipes for profiling ATLAS simulation with hephaestus and callgrind, as explored in Section 3. For minimum bias events, we use a modified version of the
`test_ATLAS_Template.py` job option from the ATLAS simulation `G4AtlasApps` package. For $t\bar{t}$ events, a modified version of the `jobOptions.G4Atlas_Sim.py` job option is used, with a small private sample of $t\bar{t}$ generator-level events used as the input via the `athenaCommonFlags.PoolEvgenInput` variable. In all cases, the G4 hits file output was disabled via a call of
`athenaCommonFlags.PoolEvgenOutput.set_Off()`, since the results of the run are not of interest when profiling, and disabling this avoids potential filesystem permission/quota problems. In all cases for this study, the number of events was explicitly set via `theApp.EvtMax = 50`.

The run environment is set up as usual with an ATLAS CMT configuration, with the
`PREPEND_PROJECT_AREA` requirements macro and `prependProject` tag used as needed to use the non-standard builds of AtlasSimulation with the different G4 builds. The run scripts listed below show the explicit CMT configurations used.

Hephaestus profiling

The job options files are first modified to enable Hephaestus instrumentation, adding the following at the top of the file:

```
## Hephaestus for G4
import Hephaestus.MemoryTracker as memtrack
memtrack.configure(memtrack.PROFILE)
memtrack.trace('')  # resets
memtrack.trace('G4TrackingManager::ProcessOneTrack')
```

The following scripts are used to start the job for minimum bias:

```
#!/usr/local/bin/bash

source setup.sh -tag=15.6.1,32,noTest,prependProject,AtlasSimulation

RTTNUM=101201
export USETCMALLOC=false
`which python` `which athena.py` --stdcmalloc --leak-check-execute \
  -c "RTTRunNumber=$RTTNUM" test_ATLAS_Template.py
```

and for $t\bar{t}$:

```
#!/usr/local/bin/bash

source setup.sh -tag=15.6.1,32,noTest,prependProject,runtime,AtlasSimulation

export USETCMALLOC=false
‘which python‘ ‘which athena.py‘ --stdcmalloc --leak-check-execute \
  jobOptions.G4Atlas_Sim.py
```

Runs of these scripts typically take 24-48 hours to finish, hence they are usually run under ‘nohup’, with output logging to file. The output, in addition to the standard Athena outputs and auxiliary files, consists of two files: hephaestus.prof and hephaestus.symb. These are two zip archives containing the profiling information in a intermediate format: a large amount of disk space is required, several GB for each run in these cases. The utility /afs/cern.ch/user/w/wlav/public/hep_prof.exe is then called from the directory containing the hephaestus output files to process these files and generate a summary file, output_heph.cgc, that can be read by kcachegrind. This generation may take a long time, due to the several-GB size of the hephaestus input files, but the output is typically only a few MB.

Callgrind profiling

The job options files are first modified to add callgrind instrumentation, adding the following before the import of the `G4AtlasApps.SimKernel`:

```
## For callgrind instrumentation
from AthenaCommon.AppMgr import ServiceMgr
from Valkyrie.JobOptCfg import ValgrindSvc
ServiceMgr += ValgrindSvc(OutputLevel = DEBUG, ProfiledAlgs = [])
```

The following scripts are used to start the job for minimum bias:

```
#!/usr/local/bin/bash

source setup.sh -tag=15.6.1,32,noTest,AtlasSimulation

export USETCMALLOC=false
valgrind --tool=callgrind --trace-children=yes --instr-atstart=no \
  --num-callers=8 --dump-every-bb=20000000000 ‘which athena.py‘ \
  --stdcmalloc -c "RTTRunNumber=101201" test_ATLAS_Template.py
```

and for $t\bar{t}$:

```
#!/usr/local/bin/bash

source setup.sh -tag=15.6.1,32,noTest,AtlasSimulation

export USETCMALLOC=false
```

```
valgrind --tool=callgrind --trace-children=yes --instr-atstart=no \
  --num-callers=8 --dump-every-bb=20000000000  `which athena.py` \
  --stdcmalloc jobOptions.G4Atlas_Sim.py
```

After starting a run of one of these scripts, Athena will stall until some auxiliary processes are killed. To identify these, call "ps aux — grep callgrind" and examine the output. The "dumpRunNumber" processes, and the related sed and grep processes need to be killed with the -9 signal. There will typically be two of each of these... of course there is no need to kill the "grep callgrind" process! Once the processes have been removed, execution will proceed as normal.

Since these runs are extremely slow – typically 1-2 weeks for 50 minimum bias events, and 2-4 weeks for the 50 $t\bar{t}$ event runs (informally, the G4.9.2.p2.a3 and G4.9.3 runs were noticeably faster than those forG4.9.2.p2.a1) – it is essential that they be run under nohup with output logging to file. When finished (or even before completion, if desired), kcachegrind may be run on the large number of callgrind.* output files: all these files will be needed for a complete analysis – during the run, callgrind will rotate the file names to avoid generating one massive file for the active process, so all the e.g. callgrind.12345.* files are required as well as callgrind.12345 itself. These files will typically be several hundred MB to 1 GB in total size, per run.

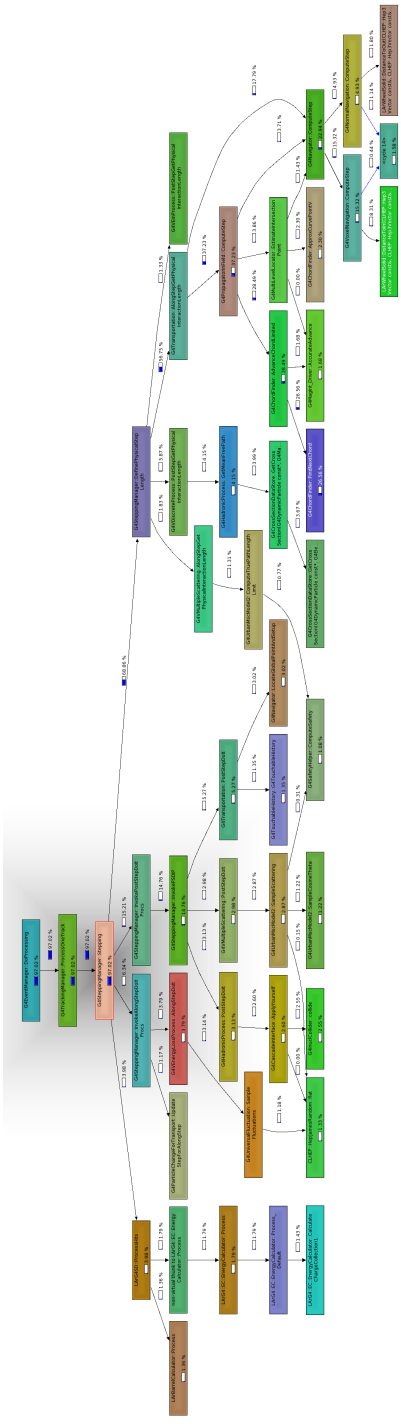

Figure 7: Call graph (and costs) of the simulation of 50 $t\bar{t}$ events (atlas3 patch).

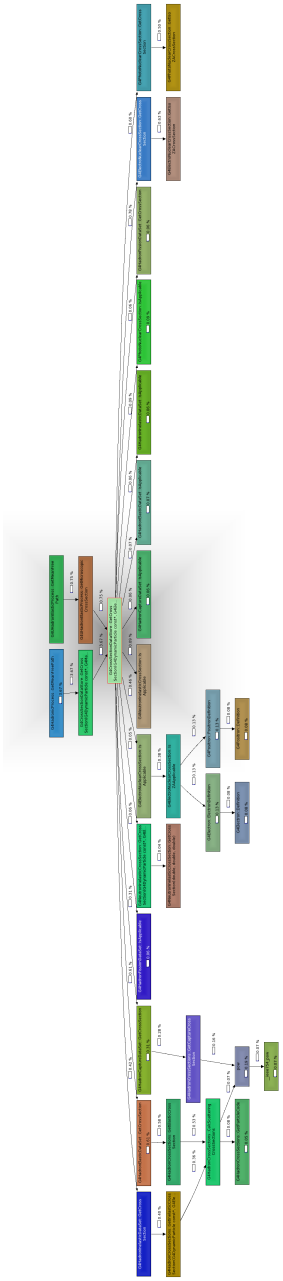

Figure 8: Different contributions of the time spent in the `G4CrossSection-DataStore::GetCrossSection` method (∼4.5% of CPU time).

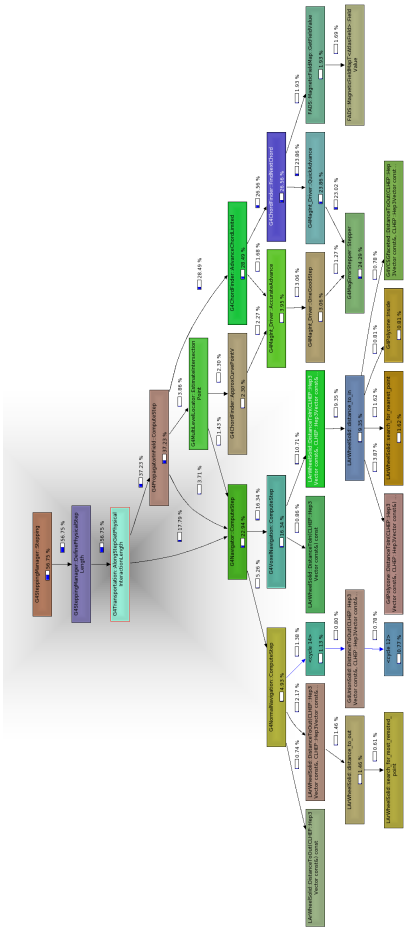

Figure 9: Call graph for the `G4Transportation::AlongStepGetPhysicalInteractionLength` method (56.75% CPU time).

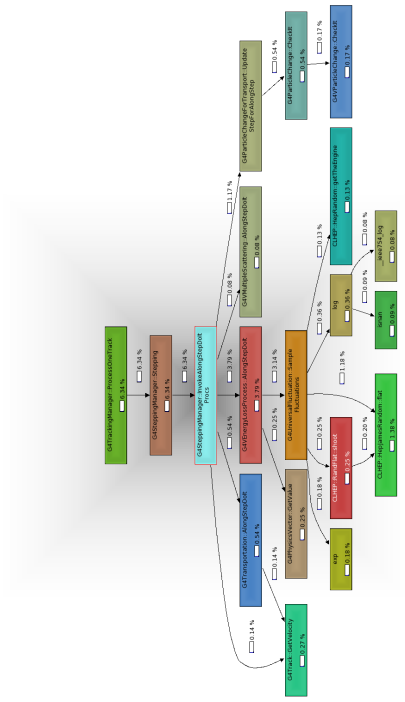

Figure 10: Call graph for the simulation of continuous processes (6.34% CPU time).

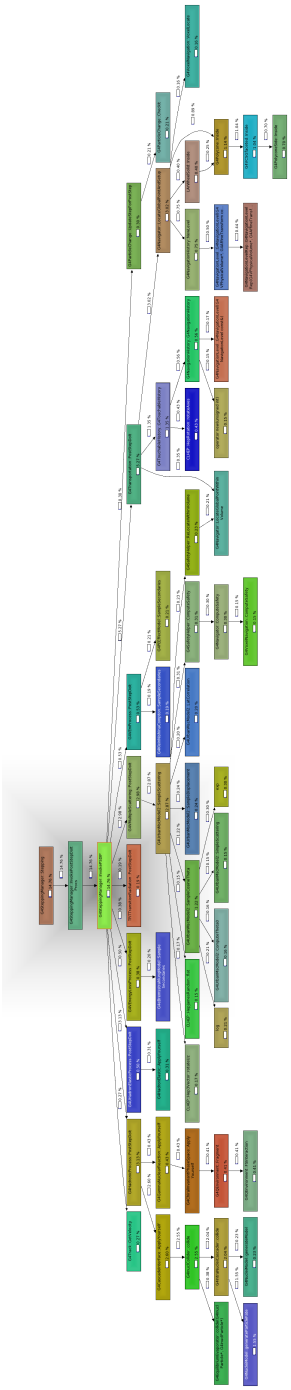

Figure 11: Call graph for the simulation of discrete processes (∼15% CPU time).

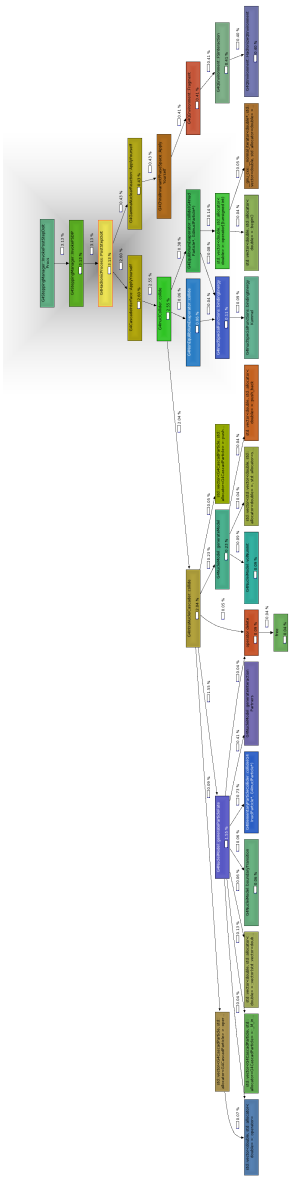

Figure 12: Further details on the simulation of hadronic interactions (∼3% of CPU time).