Geant4 – Towards major release 10

G Cosmo[1]
On behalf of the Geant4 Collaboration

E-mail: gabriele.cosmo@cern.ch

Abstract. The Geant4 simulation toolkit has reached maturity in the middle of the previous decade, providing a wide variety of established features coherently aggregated in a software product, which has become the standard for detector simulation in HEP and is used in a variety of other application domains.
We review the most recent capabilities introduced in the kernel, highlighting those, which are being prepared for the next major release (version 10.0) that is scheduled for the end of 2013.
A significant new feature contained in this release will be the integration of multi-threading processing, aiming at targeting efficient use of modern many-cores system architectures and minimization of the memory footprint for exploiting event-level parallelism.
We discuss its design features and impact on the existing API and user-interface of Geant4. Revisions are made to balance the need for preserving backwards compatibility and to consolidate and improve the interfaces; taking into account requirements from the multi-threaded extensions and from the evolution of the data processing models of the LHC experiments.

1. Introduction
The new release of Geant4 [1] planned for 2013 is going to be the first major release of the simulation toolkit since 2007. Release 10.0 will incorporate many changes in almost all areas, included an evolved API for allowing efficient execution of simulation programs on modern parallel architectures, exploiting multi-threading with parallelisation at event level. Particular care has been taken in designing the system, in order to allow for preserving backwards compatibility with user's code traditionally running in sequential mode. We report on the main features, which characterizes multi-threading for event-level parallelism by illustrating the main design principles and choices made, together with the measurements performed on different hardware architectures for performance and scalability over the number of worker threads.
We are also going to review a selection of important features, which will be provided with the new release, not covering developments in the physics modules, which in any case represent a relevant part of the features characterising the new release [2] [3].

2. Multi-threading
The first public release of a Geant4-MT prototype was made in 2011, based on a study initiated in 2009, where different event-parallelism techniques (multi-processing with or without copy-on-write)

[1] CERN, PH Department.

have been studied and compared [4]. The first prototype has served as proof of principle for the introduction of multi-threading in the Geant4 code and for identifying the subset of C++ classes to review for thread-safety, performing first-level testing on selected benchmarks. A new version of the prototype was delivered in 2012, based on release 9.5 of Geant4, establishing the basis for the integration of the new features to the main code base. Significant re-engineering of the prototype took place in order to incorporate it into the main Geant4 development line, which took place in early 2013 and resulting in the 10.0-*beta* release, deployed last June.

2.1. Design principles

As a Monte Carlo simulation toolkit, Geant4 profits from improved throughput via parallelism derived from independence among modeled events and their computation. Geant4 release 10 allows the generation of events in parallel, by defining a number of threads available in the system as independent worker units, which get their internal state initialized by a master thread, including the initial random seed for their internal random generator. Each worker thread will then compete for the generation of the next event and perform the computation independently (figure 1)

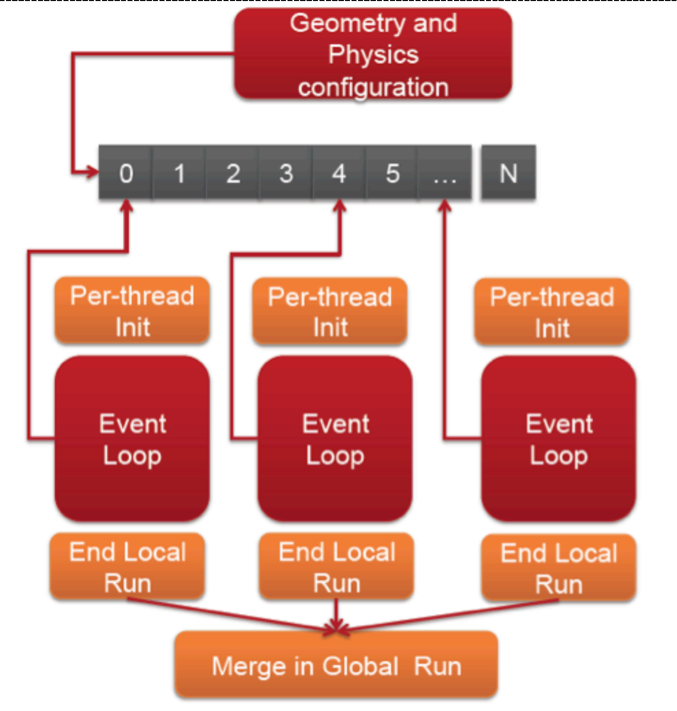


Figure 1. Each worker thread performs independently, by accessing to shared data (geometry description, physics data and configurations, etc,); a thread gets initialised in its internal state by a master thread, which provides per-event initial random seeds prepared in a queue; threads will then compete for the next event to be processed, filling their internal structures for hits/digits; final results can be finally merged at the end of the run.

Worker threads are accessing shared data for the geometry description, physics quantities and configurations without violating thread safety. This has been made possible by having carefully identified the sharable objects and, for each of them, having determined which member fields are supposed to be read-only after the worker threads have been spawned. Instances of such objects will be shared among threads; however some of the data fields, which are not invariant during the event-loop, are transformed to thread-private. This is done by defining such objects as *split-classes*, for which the data to be considered as thread-private has been encapsulated in dedicated container classes. These data are then organised in a thread-local storage [5] array by a single dedicated static manager. Such technique allows in addition to considerably reduce the memory footprint required for the spawning of new threads, contributing to improve memory locality.
The system has been designed to make use of the POSIX threads standard, in order to ensure long-term maintainability of the code and allow for the possibility to evolve it to newly emerging

technologies. This facilitates the integration of Geant4 within external frameworks wishing to adopt specific parallelization solutions, based either on MPI or Intel®-TBB (Threading Building Blocks™). The use of locks for mutual exclusion (*mutex*) of threads has been reduced to the minimum and restricted to non-critical areas of the code; this has allowed to keep under control the scalability of the performance in function of the number of threads used in computation, achieving excellent results.

2.2. Porting of user's applications to multi-threading

Special care has been taken in designing the final API for allowing applications to exploit event-level parallelism and assure thread-safety. Backwards compatibility is guaranteed when using the old interfaces for running simulation in sequential mode. The criterion we adopted in designing the new API was that the extra effort required for porting an existing application to multi-threading should be minimal, in any case a small fraction of the total development effort required for developing that application. We expect that for a simple or standalone Geant4 application, the time required to complete a first port should not exceed a few hours effort.

In order to port a Geant4 application to multi-threading, it is required to inspect and potentially adapt a small set of classes under the responsibility of the application developer. In addition to replace the sequential `G4RunManager` with the multi-threading capable one `G4MTRunManager` in the `main` program, the key step is to create separate instances for each thread of the sensitive detectors, i.e. the user's classes including the relevant code for the generation of the hits, or products of the simulation. A new method `CreateSDandField` is now provided; the corresponding part of the detector construction code should be now provided in this method, which will be called by the kernel for each thread initialization. The same method also comes with a generic signature, and can be used in complex applications, like those involving external frameworks as alternative for looping over all the relevant volumes and invoke cloning of the sensitive detectors themselves. Framework and application developers can choose between implementing the `Clone` method for their sensitive detectors or refactoring their `DetectorConstruction` code to implement the relevant `CreateSDandField` method.

A revision of all those classes (*user actions*) where the internal state of the simulation is handled and customized is necessary, in particular if inter-dependencies between these classes are present. A new class `G4VUserActionInitialization` has been introduced for the user to instantiate the user actions classes (either mandatory and optional), and allow for clearly separate those actions which are associated to a worker thread from those associated to a run and therefore handled by `G4UserRunAction`.

At the end of the application run, Geant4 can automatically apply reductions of the produced results, if the default built-in scoring and hits recording capabilities are used; histograms can also be automatically merged when using the built-in analysis module [6]. It is however a choice left to the application developer the way to handle the output, either collecting it into separate files (one per thread), or interleaved as parts of one or more output files. Depending on the application complexity, the handling of the output stream must be carefully evaluated in order to reduce the overhead coming from I/O operations.

2.3. Physics validation and performance measurements

Since the first day the code base was migrated to multi-threading, the overall physics and CPU performance have been regularly monitored, by comparing the sequential version of the code to a multi-threaded equivalent. We refer here to the most recent measurements made to date, based on the development code base of September 2013. The applications chosen are in the High Energy Physics domain, in order to exercise reasonably complex physics configurations and realistic geometrical setups: a *Simplified Calorimeter* test suite, allowing for studying a big variety of primary species on a

wide energy range [7], and a realistic geometry description of the CMS experiment at LHC [8] (*FullCMS* benchmark) imported from a GDML [9] file.
Physics calculations made with the multi-threaded version are compared and verified to be exactly the same as those obtained in the sequential case. A first test has been made to verify that statistical agreement is reached for relevant physics observables (energy spectrum, longitudinal and transverse shower profile in the calorimeter). A second more stringent test is made comparing the history of the random number engine, to verify that same status is reached at the end of an event generated in sequential mode and in multi-threading mode on a specific thread, given the same initial seed. Perfect event reproducibility could be achieved for most physics configurations, which have been verified.
CPU measurements made on the realistic geometry setup from the CMS experiment [8], with different physics configurations, reveal excellent linearity of the speedup versus the used number of threads, obtained on different systems equipped with either AMD of Intel® chips. The results for one worker thread compared to the same sequential application show a minimal overhead of 1%, while the speedup is linear with efficiencies higher than 94%. The memory saving is excellent; by running Geant4 in multi-threaded mode, an application like the *FullCMS* benchmark with k threads requires about half of the memory needed for running k clones of the sequential version of the same application. The per-thread memory overhead is at the level of 40-80 MB depending on the application.
Good efficiency and linearity of performance versus the number of threads are also confirmed when running on the relatively cheap Exynos 4412 Prime quad-core Cortex-A9 chip.
Finally, a test made of the same realistic geometry setup has been performed on one Intel® Xeon Phi™ co-processor card, featuring 61 cores (4-way hyper-threading) and 16 GB memory, confirming good scalability and memory use up to the maximum allowed of 244 threads (figure 2).

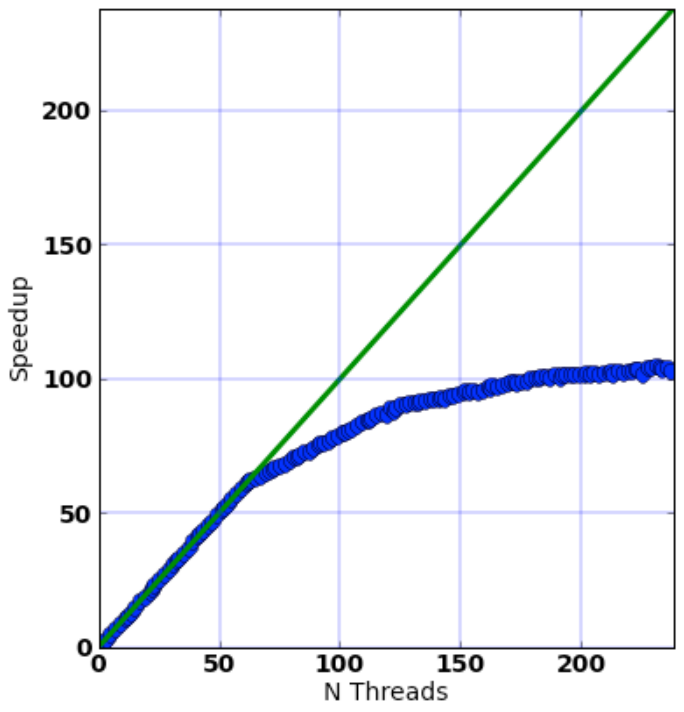


Figure 2. Measurement of the total speedup obtained by running full physics (*FullCMS* benchmark, 50 GeV pions with B-field turned on) on the Intel® Xeon Phi™ (7110P, 1.238 GHz, 61 cores) co-processor, up to the maximum number of threads. Geant4 multi-threaded efficiently uses hyper-threading [10], although no explicit use of the built-in vector units is made.

3. Geometry

The geometry modeler in Geant4 is also evolving in several areas. Key classes have been reviewed to become thread-safe for event-level parallelism; in addition, several new features have been introduced and some obsolete code, which was deprecated in release 9.6, has been removed.

All built-in UI commands for detecting geometrical volumes overlaps have been revisited and re-implemented to make use of the capability of precisely determine location of points on volume's surfaces and inspect for overlaps in the geometrical hierarchy of the placed volumes. These include the possibility to tune the resolution of the test by defining the number of points to generate, or the maximum number of errors to report for a certain volume, or set a tolerance for the extent of the overlap to report, or set limits for depth and number of levels in the geometrical hierarchy.

Navigation in the geometry tree is now adopting a more precise evaluation of the geometrical safety, which is no longer approximated or limited by the voxels' boundaries, as used to happen in the past. This enhancement, coming at zero CPU penalty, will be important in future in particular for electromagnetic processes to better treat physics effects close to volume's boundaries.

A new set of geometrical primitives (solids) is being introduced; it will be possible to use them as alternative to the existing traditional primitives, as part of an optional component extracted from the AIDA [11] Unified Solids Library [12] and included in addition to the current solids modules. The new library is going to provide improved implementations for most traditional shapes, in particular for those (tessellated solid, polycone, polyhedra or Boolean union) where an innovative spatial optimization technique could be applied for the efficient detection of the components candidates for intersection. The new library will include the current state-of-the-art of the implementation, including all the new primitives currently available (figure 3).

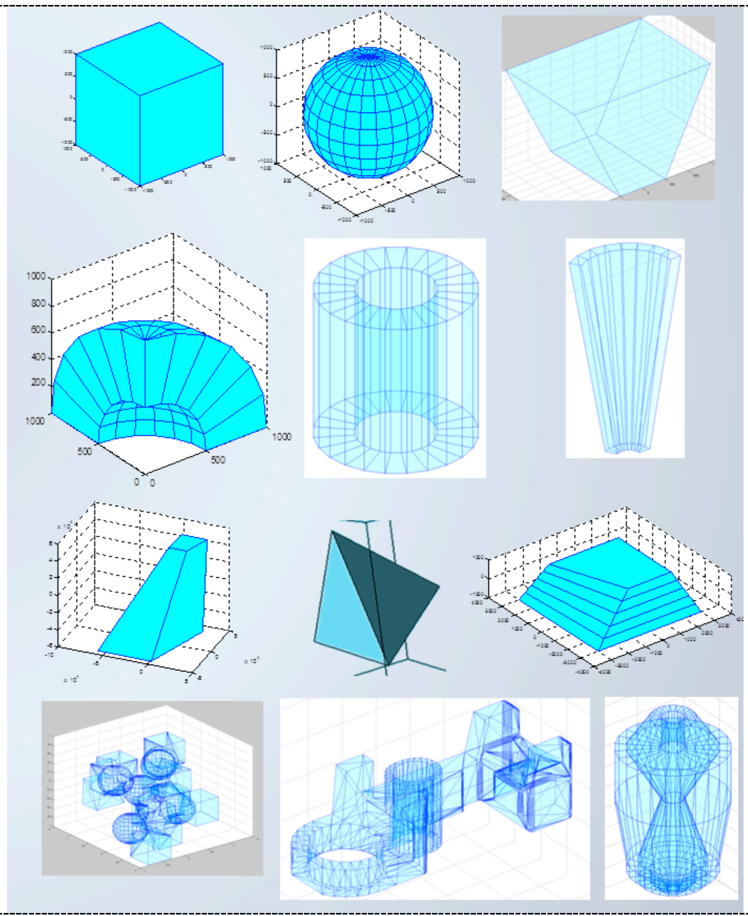


Figure 3. The set of geometrical primitives currently composing the new AIDA Unified Solids Library: box, orb, sphere (+sphere section), tube (+cylindrical section), cone (+conical section), simple, generic & arbitrary trapezoid, tetrahedron, polycone, polyhedra, extruded solid, tessellated solid and new Multi-Union structure.

4. Event biasing

There is a class of applications for which a standard (analogue), simulation is at least inefficient, or even not practicable; this is the class where rare events are the ones of interest.

The estimation of shielding efficiency in radioprotection is one of these examples: an analogue simulation would spend most of the time in transporting particles and interaction products inside the shield, leading to a tiny fraction leaving the shield. In this case, particles leaving the shield are the "rare events". Very thin detectors may be difficult to simulate as well, as only a small fraction of particles may interact inside the volume. Another example is the case of a volume of interest very small compared to the overall structure in which the simulation is to be carried out, like the case of the simulation of single event upset in a small electronic chips inside the large structure of a satellite. In such a case, the events of interest depositing in the chip are rare given they can occur in a fraction of time proportional to the size of the electronic chip volume versus the overall satellite volume.

In such rare events simulation problems, it is possible to boost the simulation efficiency without sacrificing the physics quality provided by a full detailed Monte-Carlo description, by magnifying the occurrence of the rare event [13]. This is the main purpose of event biasing techniques. There are two main classes of event biasing techniques: splitting (killing) and importance sampling. In the first case, tracks are split (killed) as long as they approach (recede from) the phase space region of interest: a typical use case is the shielding problem; the shield volume is artificially divided into slices on the boundaries of which particles are cloned (killed with some probability) if they fly towards (opposite to) the direction of the shield exit; this artificial flux regeneration compensates for the physical absorption while penetrating the volume of the shield. In importance sampling, the underneath physical interaction laws are substituted with biased ones that favour the occurrence of rare events. The thin detector volume case can be treated this way [14]: the natural exponential law is replaced by a truncated exponential one, whose extension is limited to the track path inside the volume, to guarantee that the collision will occur. Many other techniques exist [14]. Whatever technique is adopted, a statistical weight is computed at each step a biasing is applied; this weight is used to counter-balance for the application of the biased simulation when computing the desired quantities.

Geant4 already provides a number of biasing techniques splitting-based with geometry importance (Bremsstrahlung splitting in electromagnetic or importance sampling based on general particle source biasing [15], or in the advanced technique of the reverse Monte Carlo [16]), but we're missing importance sampling techniques in which the physics processes are biased, either in their interaction probability, or in their final state generation (energy/angular biasing). Such techniques includes the force collision [14], the exponential transform [14], [17] for what concerns the biasing of interaction laws, and techniques like DXTRAN [14] (forcing of scattering inside some solid angle), which involve final state biasing.

A new development is going to provide interface classes to allow for the implementation of virtually any kind of biasing techniques. The approach considers that biasing is applied in some volumes, user selected, outside of which the analogue tracking goes, and inside which a large freedom is to be given to the user to apply techniques the most suited to the application needs.

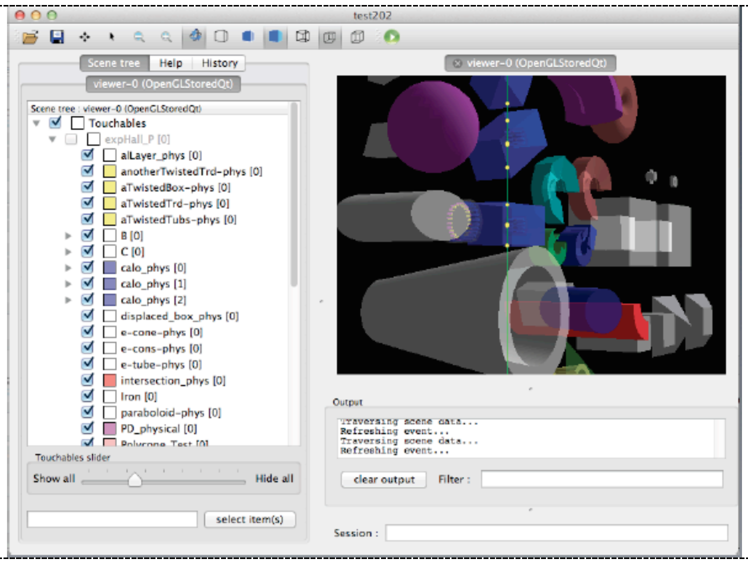
The scheme has been prototyped for now, and will be delivered with release 10.0.

5. Visualization

Geant4 release 10.0 sees two key improvements in visualization: in addition to the support for multithreading it is also added the ability to produce OpenGL EPS graphics in the absence of a hardware graphics card.

Multithreading poses a challenge for Geant4 for those visualization drivers [18] that incorporate a direct (rather than file-mediated) connection from Geant4 to some other technology. The most notable case of such connection are in Geant4's flagship visualization drivers, the OpenGL family (including the direct mode, OGLX), the display-list mode, OGLS, the OpenInventor-wrapped mode, OIX and the Qt-wrapped mode, OGLQt (figure 4). Other examples include the socket-connected forms of the

DAWN [19] and VRML drivers. The use case we are supporting is one of a single graphics window that shows events from any of the worker threads (we considered also the case of having a separate graphics window for each worker thread, but found this to be not of no interest to users aside from as some sort of multi-threading demo). The challenge is that while we must create a single graphics driver in a main thread, the events we want to represent are, by default, only present in the worker threads. While it may be possible to eventually drive some systems, such as OpenGL, by starting the visualization from the main thread and then contributing additional graphics objects from the worker threads, such "context switching" as OpenGL calls it involves a great deal of subtlety. Any solution developed at that driver-specific level would have to be separately replicated, with different technology, for the other direct drivers (DAWN, VRML). We have instead opted to use an "event keeping" strategy, with events copied from worker threads to the master thread, so that drawing of the event can always be handled at the single master thread. As of this writing, this master thread solution is in place, while additional work is under way for release 10.0 or 10.1 to resolve a limitation of the current solution. In the current solution, no events can be drawn until all of the worker threads complete their function. For cases in which event simulation takes much less time than visualization, this solution is fine, but for some other cases, such as visualization of LHC's complex events, event simulation is slow compared to visualization, and we would rather not have to wait for all simulation to cease before we begin visualizing results. To this end, we are attempting to move all visualization to a dedicated "visualization management thread". This thread will start when visualization is first instantiated, doing the initial geometry drawing, then loop, watching the master thread's saved event queue, drawing events as fast as possible when they become available on that queue.



```
Traversing scene data...
Refreshing event...
Traversing scene data...
Refreshing event...
```

Figure 4. OpenGL schene rendered in Qt-wrapped mode (OGLQt), the recommended visualization driver, set as the default.

OpenGL graphics in the absence of a GL graphics card has been a goal of Geant4 visualization for several years as our OpenGL drivers have moved from being used mainly for live display to also become our most popular vector graphics file generator (thanks to the "print to EPS" function that is available from GL cards and accessed from a Geant4's visualization command). EPS graphics has been available in Geant4 since the first years of the toolkit, through the DAWN visualization system, but the GL extension was the first to provide such graphics directly from Geant4, rather than through an external graphics application. A limitation was that our GL solution relied on hardware, the OpenGL graphics card that is often not present on cluster, cloud or grid servers. Geant4 Release 10.0 adds a hardware-free alternative, thanks to the XVFB library (X11 Virtual Frame Buffer) [20].
Developments are also on going to allow for visualization of a Geant4 application through a web browser [21].

6. Conclusions

Geant4 Release 10.0, the next major release of the Geant4 simulation toolkit is going to be a big step forward in the evolution of the Geant4 software and the capabilities any user will have chance to experience. A key feature will be the possibility to exploit event-level parallelism thanks to the multi-threading extensions now built-in. Measurements performed on different hardware architectures capable of hyper-threading or not, have revealed excellent linearity of throughput speedup of the order of 94% and reduced memory footprint per-thread, with good results also when running in hyper-threading regime. Many new features and enhancements are affecting nearly all areas, included the physics, which is not covered in this paper.

7. References

[1] Agostinelli S et al., 2003 *Nuclear Instruments and Methods in Physics Research* **A506**, 250-303; Allison J et al 2006 *IEEE Transactions on Nuclear Science* **53 No. 1** 270-278

[2] Ivantchenko V et al., 2013 *Geant4 Electromagnetic Physics for LHC Upgrade*, Proceedings CHEP-2013, Amsterdam

[3] Pokorski W et al., 2013 *Recent Developments in the Geant4 Hadronic Framework*, Proceedings CHEP-2013, Amsterdam

[4] Dong X et al., 2010 *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, **volume 6272**, p.287-303; Dong X et al 2012 *J. Phys.: Conf. Ser.* **396** 052029

[5] Crowl L, 2006 *Thread Local Storage*, http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n1966.html

[6] Hrivnacova I et al., 2013 *Integration of g4tools in Geant4*, Proceedings CHEP-2013, Amsterdam

[7] Dotti A et al., 2011 *IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*, p.2128-2134

[8] CMS Collaboration, 2008 *Journal of Instrumentation*, **3**, S08004, 187

[9] Chytracek R et al., 2006 *IEEE Transactions on Nuclear Science*, **Vol. 53**, Issue: 5, Part 2, 2892-2896

[10] Leng T et al., 2002 *A Study of Hyper-Threading in High-Performance Computing Clusters*, Dell Power Solutions

[11] AIDA project web site: http://cern.ch/aida/

[12] Gayer M et al., 2012 *Journal of Physics: Conference Series* **396**, 052029

[13] Rubin R et al., 2009 *Rare Event Simulation using Monte Carlo Methods*, John Wiler & Sons Ltd., ISBN 978-0-470-77269-0

[14] MCNP – A General Monte Carlo N-Particle Transport Code, Version 5: http://mcnp.lanl.gov

[15] General Particle Source: http://reat.space.qinetiq.com/gps/

[16] Lei F et al., 2010 *Nuclear Instruments and Methods in Physics Research* **A621**, Issues 1–3, p.247–257

[17] Mendenhall M H et al., 2011 *A Probability-conserving cross-section biasing mechanism for variance reduction in Monte Carlo particle transport calculation*, arXiv:1109.0910

[18] Allison J et al., 2013 *International Journal of Modeling, Simulation, and Scientific Computing*, **4**, Suppl. 1 1340001

[19] Tanaka S et al., 1997 *DAWN for Geant4 visualization*, Proceedings CHEP-97, Berlin

[20] XVFB – Virtual Frame Buffer X Server for X Version 11: http://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml

[21] Garnier L et al., 2013 *Geant4 application in a Web browser*, Proceedings CHEP-2013, Amsterdam