The Geant4 Virtual Monte Carlo

2012 J. Phys.: Conf. Ser. 396 022024

(http://iopscience.iop.org/1742-6596/396/2/022024)

Journal of Physics: Conference Series **396** (2012) 022024 doi:10.1088/1742-6596/396/2/022024

The Geant4 Virtual Monte Carlo

I Hřivnáčová

Institut de Physique Nucléaire (IPNO), Université Paris-Sud, CNRS-IN2P3, 91406 Orsay Cedex, France

E-mail: ivana@ipno.in2p3.fr

Abstract. The Virtual Monte Carlo (VMC) [1] provides the abstract interface to the Monte Carlo transport codes: GEANT 3.21 [2], Geant4 [3], and FLUKA [4]. The user VMC based application, independent from the specific Monte Carlo codes, can be then run with all supported simulation programs. VMC has been developed by the ALICE Offline Project and it has drawn attention in other experimental frameworks.

Since its first release in 2002, the implementation of the VMC for Geant4 (Geant4 VMC) has been continuously maintained and developed, driven by the evolution of Geant4 on one side and the requirements from users on the other side. In this paper we report on new features in this tool, we present its development multi-threading version based on the Geant4 MT prototype [5] as well as the time comparisons of equivalent native Geant4 and VMC test applications.

1. Introduction

Geant4 VMC [6] has been previously described in detail in [7] and in the context of ALICE in [8]). In this paper, we briefly recall the concept and the design of the Virtual Monte Carlo, before reporting on the new developments in Geant4 VMC. One section is devoted to the Geant4 VMC multi-threading prototype and, finally, the time comparison of equivalent Geant4 native and VMC test applications are discussed.

2. Virtual Monte Carlo (VMC)

VMC has already been presented in detail in [9] and [10], here we give a brief summary of the main ideas. VMC defines an abstract layer between a detector simulation user code and the Monte Carlo transport code (MC). In this way the user code is independent from any specific MC and can be used with different transport codes, such as GEANT 3.21 [2], Geant4 [3], FLUKA [4], within the same simulation application (Fig. 1).

In VMC, we introduce on one side the interface to the transport MC itself, *TVirtualMC*, and on the other side the interface to the user application, *TVirtualMCApplication*, see Fig. 2. In this way, we decouple the dependence between the user code and the concrete MC. Both of these classes are available together with two other interfaces and a few more utility classes in the *vmc* package in the ROOT framework [11].

The implementation of the *TVirtualMC* interface is provided for two Monte Carlo transport codes, GEANT3 and Geant4 within their VMC packages available from the ROOT site. The implementation for the third Monte Carlo transport code, FLUKA , was discontinued by the FLUKA team in 2010. The *TVirtualMCApplication*, and optionally two other VMC interfaces, have to be implemented in the user application.

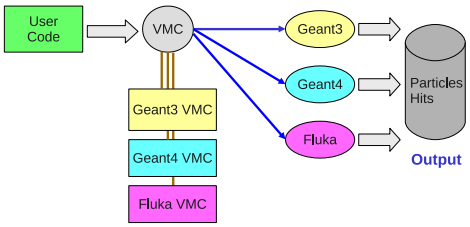

Figure 1. The VMC concept.

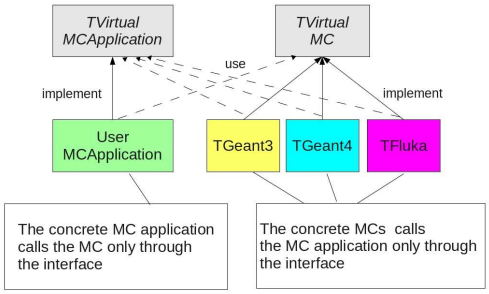

Figure 2. The VMC design.

VMC is now fully integrated with the ROOT geometry package, TGeo [12], and users can easily define their VMC application with TGeo geometry and this way of geometry definition is recommended for new users.

The Geant4 VMC package provides the implementation of the Virtual Monte Carlo for Geant4. It is available from the ROOT site as well as from the ROOT SVN server.

The package releases follow the releases of Geant4 which often require a migration of the user code. A new tag can be also triggered by new developments or a new ROOT version. In general, the tagged Geant4 VMC version can be used with the ROOT version with which it was tested and with higher ones, and the Geant4 version with which it was tested, including its patches, but not with higher Geant4 versions. The patch versions include fixes applied to the base versions and are usually maintained for the last two versions, and so for the last two versions of Geant4 too.

3. Geant4 VMC: New Features and Developments

New developments in Geant4 VMC are mostly driven by the evolution of Geant4 on one side and the requirements from users on the other side. They are presented in two subsections: non physics developments and developments related to user physics selections. Several fixes in the tool, thanks to user feedback, contributed to its robustness and future stability.

3.1. Non Physics Developments

The new class, *TG4FieldParameters*, is introduced to allow a user customization of the Geant4 magnetic field integrator and the precision parameters. It has been inspired by Geant4 extended/field examples. Its associated messenger class, *TG4FieldParametersMessenger*, defines a set of Geant4 interactive commands which can be used directly from the user configuration macro written in the ROOT framework. An example of how to use the new commands is provided in the *g4config.in* macro in the VMC example E02. All available commands are also documented on the Geant4 VMC Web site [6].

A new utility class, *TG4ParticlesChecker*, allows to compare particle properties defined in ROOT and Geant4. This class was introduced when some discrepancies in ROOT particles definitions were reported by users in order to be able to verify the consistency of particle definitions in a general way. Its associated messenger class, *TG4ParticlesCheckerMessenger*, defines a set of commands that can be used to select a particle, a particle property to be checked and, the precision of the checked values.

In the standard VMC application a user action is called at each step. Typically a current volume identifier is compared with a sensitive volume and, only if they match, the user action is performed. A possibility for the user to select sensitive volumes in the initialization phase was implemented on user request. In this case, the *TVirtualMCApplication::Stepping()* function is

called only when a track is located in a selected sensitive volume in a way similar to the native Geant4 application. This feature can speed up a user application, especially when the accounting of MC information happens in a small number of volumes in a rather complex geometry.

Among the other new features we would like to mention the implementation of new *TVirtualMC* functions for drawing tracks from ROOT: *SetCollectTracks(..)*, *IsCollectTracks()* and a new, faster implementation of the *TVirtualMC* functions: *VolId(..)*, *VolName(..)*, *GetMediumID()* which are intensively used in most VMC applications. Passing the random number seed from the ROOT random number generator, *TRandom*, to the Geant4 random number engine, *CLHEP::HepRandom* was also enabled in response to users requests.

3.2. Physics Selection

The physics list selection in Geant4 VMC has changed significantly since the CHEP 2007 paper [7]. The default Geant4 VMC physics list has been completely removed. Almost all pre-defined physics lists were removed in the 2.4 version in December 2007. The remaining part, dealing with optical physics processes could only be removed recently, in the version 2.13 from December 2011, after having adopted the Geant4 VMC messenger class for a user customization of optical physics in Geant4 9.5.

The selection of the physics list is done in a user configuration macro via a Geant4 reference physics list name. The existence of the selected physics list is then verified using the *G4PhysListFactory* class and this class is also used for the physics list instantiation. In this way Geant4 VMC does not need to maintain the information on the existing physics lists itself and is more flexible with Geant4 updates. The mapping of Geant4 physics processes to the VMC process codes (the enumeration named constants defined in VMC for each physics process type) however, requires an update with each Geant4 release. In order to automate this process a new test for processing all available Geant4 physics lists has been developed.

Besides the reference physics lists Geant4 provides several physics builders which can be added to the physics setup defined in the reference physics list: *G4ExtraPhysics*, *G4OpticalPhysics*, *G4RadioactiveDecayPhysics*, etc. These physics builders are handled in Geant4 VMC with the new *TG4ExtraPhysicsList* class, allowing a user to select these physics builders via a string option which is appended to the reference physics list name with a + character.

In addition to the Geant4 physics processes, Geant4 VMC introduces special processes and maps which are used to support various VMC features: VMC cuts and process controls, step limit per tracking medium, adding user particles to the stack during tracking, etc. These special processes are constructed by their specific physics builders which are then handled all together by the *TG4SpecialPhysicsList* class. Not all special processes are activated by default, some have to be activated explicitly by the user via a string option in their configuration macro.

The Geant4 reference physics list, *TG4ExtraPhysicsList* and *TG4SpecialPhysicsList* are combined in the final physics list class, *TG4ComposedPhysicsList*. The composition of this class and the selection of the physics setup is demonstrated in Fig. 3.

There still remains a possibility to include personal user physics lists which are implemented in the Geant4 framework. An example showing this option is provided in the VMC example E03 in the *Ex03RunConfiguration2* class.

Among the other new features we would like to mention the new *TG4EmModelPhysics* class allowing a user selection of electromagnetic interaction energy loss and fluctuation models and the new *TG4CrossSectionManager* class for inspecting hadronic cross sections. Light anti-ions (anti-deuteron, anti-triton, anti-alpha, anti-He3) were included in the available particle definitions on the request of ALICE and a test processing these particles was added in the VMC example E03.

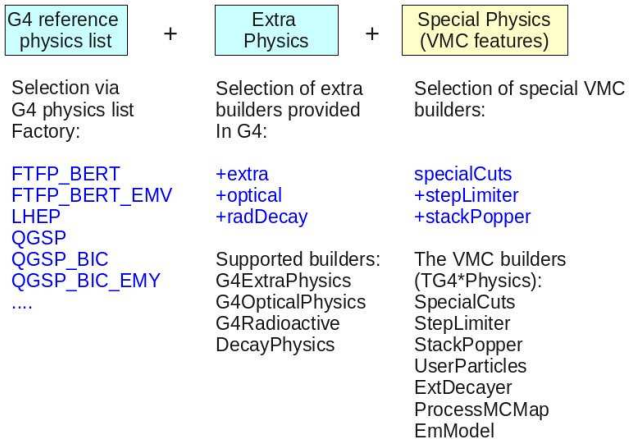

Figure 3. The composition of *TG4ComposedPhysicsList* and the selection of the physics setup.

4. Multi-threading Prototype

4.1. Geant4 VMC MT

The development of the Geant4 VMC Multi-Threading (MT) prototype started in the last quarter of 2011. Naturally, the same approach as used in Geant4 MT [5] is adopted and the main task of the migration to multi-threading processing was the replacement of all singleton objects in Geant4 VMC with singletons per thread. The modifications, as described in the Geant4MT User's Guide, were then applied to the Geant4 VMC classes, mainly in geometry and run categories. Special care was needed in the use of *G4Allocator* in *TG4TrackInformation* which had to be declared thread local.

Changes were also required at the level of the VMC interfaces as both *TVirtualMC* and *TVirtualMCApplication* are defined as singletons. A new function, *TVirtualMC::InitMT(Int_t threadRank)*, was added in *TVirtualMC* for the initialization of MonteCarlo in the thread. These changes have no consequence on single-threaded applications. They are available in the ROOT 5.34/00 release.

New classes for the multi-threading application management, *G4ParRunManager* and *G4ThreadManager*, were developed according to Geant4 MT examples where this code is defined in terms of external functions and variables. As they are not specific to Geant4 VMC, they are now included directly in the Geant4 MT development branch and are just used from Geant4 VMC MT.

As opposed to a usual VMC application which is run from the ROOT *main* function, the Geant4 VMC MT application starts from its own *main* function linked with all other package libraries (ROOT, Geant4, etc.). This *main* function performs the instantiation of threads via the included *G4ParTop.icc* utility. Both *TGeant4* and *UserMCApplication* objects are then instantiated per thread. The use of Geant4 VMC MT is demonstrated in the VMC example E02 which defines an equivalent setup as ParN02 and ParN02Root in Geant4 MT. Its simulation time on 4 core CPU scales with the number of threads in a way similar to ParN02 or ParN02Root examples.

The modifications towards Geant4 VMC MT were quite straightforward and were applied in a relatively short time. In one aspect, the code in Geant4 VMC was even simplified thanks to the Geant4 logical volume identifiers, available in Geant4 MT only, and which can be reused directly in Geant4 VMC .

The Geant4 VMC MT prototype is based on the Geant4 MT development branch as it uses the classes added on the branch after the first Geant4 MT prototype release in October 2011 (based on Geant4 9.4.p01). It is available in the SVN development branch: *geant4_vmc_mt*. The first tagged version is foreseen after the public release of Geant4 MT based on Geant4 9.5.x.

4.2. Geant4 (VMC) MT ROOT Output

Geant4 VMC operates always with the native ROOT output, that's why the integration of the ROOT output in the multi-threading version of Geant4 VMC was a primary condition for developing this tool.

As Geant4 MT introduces parallelism per event, it is possible to separate the output per thread without the need for merging outputs at the end of a simulation. The ROOT output can then be introduced per thread: each thread opens and writes on its own ROOT file. There is no need for a final merge since the files can be chained directly in the user analysis.

The ROOT output was first implemented and tested with Geant4 MT (without use of VMC). The manager class for handling the ROOT output, analogous to *Ex02RootManager* used in the VMC example E02, was added first to a Geant4 native example N02, and then extended for a multi-threading case in ParN02. This extended ROOT manager class, *RootManagerMT*, and the helper class *RootMutex* are now available in the new example ParN02Root in the Geant4 MT branch. The object of the ROOT *TThread* class type is instantiated at the beginning of the program which makes it possible to release the lock on the ROOT output after the first *TTree::Fill()* in all threads.

The design of classes for the ROOT output in Geant4 VMC MT is demonstrated in Fig. 4. The *TVirtualMCRootManager* interface is implemented separately for single-threading and multi-threading cases (*TMCRootManager* and *TMCRootManagerMT*) with use of the same *TMCRootManagerImpl* helper class. *TMCRootMutex* allows additional locking of the ROOT output from the user application when needed.

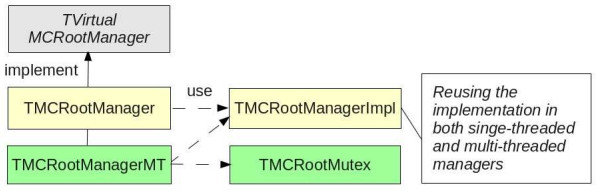

Figure 4. The design of classes for the ROOT output in Geant4 VMC MT.

5. Performance Tests

The Geant4 extended example analysis/A01 was rewritten in VMC in order to evaluate the overhead of the VMC layer compared to a native Geant4 application. This example reflects the real applications in a better way than the existing VMC examples. It defines four sensitive detector types (associated with six volumes) and a geometry including placements with rotations and replicas.

The following modifications were applied to the original example setup in order to be able to define it as a VMC application. The local magnetic field has been changed to a global field with zero value outside the volume with the field, since local fields are not supported in VMC. Then, the use of the *G4PVParameterised* volume in the example geometry has been changed to the use of *G4PVReplica*, since parameterised volumes are not supported in the VGM tool [13] used in Geant4 VMC for geometry conversions between ROOT and Geant4.

The following items briefly summarize the test setup:

- The VMC example was run with the native Geant4 geometry (using the geometry option *"geomGeant4"*).
- FTFP_BERT physics list with 1 mm range cuts was used in all tests.
- The same primary generator periodically changing particles between e^+, μ^+, π^+, K^+ and *proton* with a fixed momentum was used in all tests.
- The Geant4 example was run with both local and global magnetic fields.

- The VMC example was then run without saving data, with the storage of hits only, with the storage of the particles stack only, and finally with the storage of both.
- The time taken for a run of 100 events with 100 primary particles each was measured.

The test was performed with Geant4 9.5.p01, ROOT 5.32.03 and the Geant4 VMC development version (SVN trunk revision 607) on the Intel Core i7 processor with Fedora Core 14 and gcc 4.5.1.

Table 1. Performance test results.

	Time[s]	Time/ Ref. I	Time/ Ref. II
`G4 local field`	97.93	0.81	0.70
`G4 global field (Ref. I)`	121.43	**1.00**	0.87
`VMC (Ref. II)`	139.04	**1.15**	1.00
`VMC storing hits`	141.24	1.16	1.02
`VMC storing stack`	178.37	1.47	1.28
`VMC storing all`	179.08	1.47	1.29

The measured times are summarized in table 1. The total time of one run in seconds is presented in the first column, the relative value of this time with respect to the time of the native Geant4 application with a global magnetic field (marked as Ref. I) in the second column and the relative value of this time with respect to the time of the VMC application without storing data (marked as Ref. II) in the third column.

The VMC example without storing data (Ref. II) and the Geant4 example with a global magnetic field (Ref. I) define an equivalent experiment setup and so comparing times of these tests can be used to evaluate the overhead of the VMC layer as compared to a native Geant4 application. In the example tested this overhead is around 15 %. Storing hits adds only 2 % simulation time as the hits transient data representation is identical with their persistent data representation (ROOT framework). However, when storing particles, where the transient data representation (Geant4 framework) is different from their persistent one (ROOT framework), an additional 28 % simulation time is observed. It should be noted that the example is using the *Ex03MCStack* with no optimizations which stores all secondary particles. In the end, the full VMC application is 47 % slower than the original Geant4 application, but the greater part of this sluggishness is caused by the added file output (not present in the Geant4 version) and not by the VMC interface itself.

6. Conclusions

Geant4 VMC has been in production for already ten years. In this paper we gave an overview of new features of the tool, we have reported on the work concerning the integration with the Geant4 multi-threading prototype and, evaluated a time overhead that the use of an additional VMC layer adds to the simulation time of a native Geant4 application.

Acknowledgments

The author would like to acknowledge X. Dong from *Northeastern University* for his help with the Geant4 MT prototype and P. Canal from *Fermilab* for his help with ROOT IO in a multi-threading framework.

References

[1] http://root.cern.ch/drupal/content/vmc
[2] Brun R et al 1985 *GEANT3 User Guide* (CERN Data Handling Division, DD/EE/84-1)
[3] Agostinelli S et al 2003 *Nucl. Instrum, and Methods* **A506** 250-303
[4] Fasso A et al 2001 *Proc. of the MonteCarlo 2000 Conference* (Lisbon, Springer Verlag Berlin) 159-164 and 955-960.
[5] http://geant4.web.cern.ch/geant4/support/download_MT_proto.shtml
[6] http://root.cern.ch/drupal/content/geant4-vmc
[7] Hřivnáčová I 2008 *J. Phys: Conf. Series* **119** 032025
[8] Hřivnáčová I et al 2011 *J. Phys: Conf. Series* **331** 032016
[9] Hřivnáčová I et al 2003 *Proc. of Computing in High Energy and Nuclear Physics* (La Jolla) pp THJT006
[10] Carminati et al 2004 *Proc. of Computing in High Energy and Nuclear Physics* (Interlaken) pp 433
[11] http://root.cern.ch
[12] Brun R, Gheata A and Gheata M 2003 *Proc. of Computing in High Energy and Nuclear Physics* (La Jolla) pp THMT001
[13] Hřivnáčová I 2008 *J. Phys: Conf. Series* **119** 042016