EPJ Web of Conferences **214**, 02004 (2019) https://doi.org/10.1051/epjconf/201921402004
CHEP 2018

Adopting new technologies in the LHCb Gauss simulation framework

Dominik Müller[1,*] on behalf of the LHCb collaboration

[1]CERN, Geneva, Switzerland

Abstract. The increase in luminosity foreseen in the future years of operation of the Large Hadron Collider (LHC) creates new challenges in computing efficiency for all participating experiment. For Run 3 of the LHC, the LHCb collaboration needs to simulate about two orders of magnitude more Monte Carlo events to exploit the increased luminosity and trigger rate. Therefore, the LHCb simulation framework (Gauss) will go through a significant renovation, mostly driven by the upgraded core software framework (Gaudi) and the availability of a multithreaded version of Geant4. The upgraded Gaudi framework replaces single-threaded processing by a multithreaded approach, allowing concurrent execution of tasks with a single event as well as multiple events in parallel. A major task of the required overhaul of Gauss is the implementation of a new interface to the multithreaded version of Geant4.

1 Introduction — The Gauss framework

Monte Carlo samples are a crucial tool to understand and interpret the data recorded by the experiments in High Energy Physics (HEP). They allow analysers to extract physics information from recorded data by providing a means to separate detector effects and reconstruction efficiencies from genuine physical phenomena. This usually happens in a two stage process. First, the collisions are simulated by dedicated generators to provide behaviour as close as possible to nature. The resulting particles of the generation step are then passed through a simulation of the detector response. However, a detailed simulation of the detector response, which behaves similar to nature and tracks particles traversing the various detector volumes leaving energy deposits, is very CPU intensive and accounts for large fractions of the overall CPU usage of HEP experiments. During Run 2 of the Large Hadron Collider (LHC), the LHCb collaboration used approximately 90% of the available offline computing resources for the creation of Monte Carlo samples. Despite the considerable investment of resources, recent measurements hinting at tensions with respect to the predictions of the Standard Model of elementary particle physics have systematic uncertainties that are dominated by the insufficient amount of simulated data [1]. With the changes to the data-taking schemes being implemented for Run 3 [2], two orders of magnitude more Monte Carlo events need to be simulated to exploit the increased luminosity and trigger rate. While multiple fast simulation options are under development which are certainly necessary to achieve this goal, it remains mandatory to produce fully simulated events as efficiently as possible and to make use of modern CPU architectures. Therefore, this paper summarises the ongoing efforts to modernise Gauss, the LHCb simulation framework.

*e-mail: dominik.muller@cern.ch

1.1 The Run 1 and 2 framework and its shortcomings

The LHCb simulation framework Gauss [3] is built on top of Gaudi [4, 5]. It is responsible for steering of the different steps in the generation of an event using interfaces to external generators and the Geant4 toolkit [6].

In the most commonly used procedure to simulate events in LHCb, pp collisions are generated with the Pythia 8.2 [8, 9] generator using a specific LHCb configuration similar to the one described in Ref. [10]. Decays of particles are described by EvtGen [11] in which final-state radiation is generated with Photos [12]. The implementation of the interaction of the generated particles with the detector, and its response, uses the Geant4 toolkit. The event generation and simulation with Geant4 are executed in the same job.

The simulation of a typical event in Run 1 or Run 2 conditions takes around one minute, depending on the simulated signal decay, LHC run period and the employed CPU hardware. Of this time, well above 95% is spent in the simulation of the detector response using Geant4. The increase in luminosity recorded by LHCb drives the need to upgrade the framework in two ways: a higher luminosity implies more signal candidates being recorded, directly implying the need for larger Monte Carlo samples and the fact that most of the luminosity increase is achieved by increasing the average number collisions per bunch crossing from just above one to five. As all collisions within a simulated bunch crossing are treated as a single event, the time required for the simulation of one event will increase substantially. To meet these challenges and to be able to generate a useful amount of fully simulated events, all available computing resources need to be utilized as efficiently as possible.

The current Gauss framework was written over 15 years ago and processes events in a single thread. This limits its efficiency on machines with modern CPUs that usually have a high core count: due to the relatively high memory-consumption of a simulation application and the poor memory scaling when launching individual processes, it might not be possible to fill all available CPU cores. An example for this is the usage of LHCb's software trigger farm during LHC downtimes for Monte Carlo production where a full utilisation of all CPU cores fails due to the limited memory. Therefore, the majority of work on an LHCb simulation framework for Run 3 focuses on the migration to a multi-threaded framework utilising a multi-threaded event loop. This event loop is steered by Gaudi which interfaces to multi-threaded Geant4. The work on this framework is the topic of the remainder of this paper.

2 The upgraded framework

The major change in the architecture of the Gauss framework is the creation of an LHCb-independent core framework, called Gaussino, on which Gauss is built. This is illustrated in Figure 1: Gaussino is build directly on Gaudi, providing interfaces to Pythia and Geant4. Hence, it implements the core functionality and steers the aforementioned multi-threaded event loop. This allows the incremental migration of functionality and quick prototyping of new features in Gaussino with a minimal configuration. Gauss for Run 3 then depends on Gaussino and incorporates any LHCb specific additions and features. In the following, the work undertaken to migrate the core functionality of the current Gauss framework to Gaussino are detailed. Gaussino employs Gaudi's task based parallelism in which the event loop is separated into multiple algorithms which perform various action on the data of an event by either producing, transforming or consuming it. Generated events are passed between different algorithms in the HepMC3 format. Each algorithm is written following the idea of functional programming and only has constant internal state to ensure thread-safety by design. Additionally, an algorithm has to declare its data dependencies (which are immutable) and what output data it provides. This information is then used to schedule these tasks to run once their

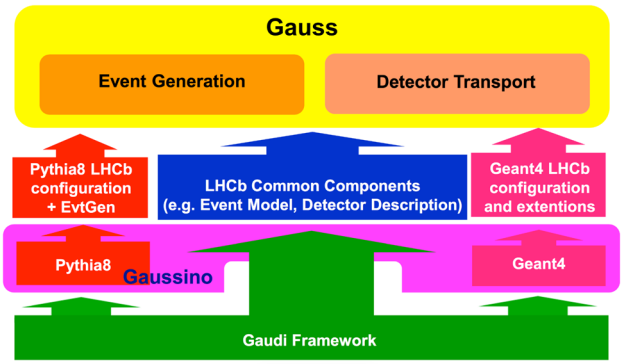

Figure 1: Illustration of the dependencies of Gauss after the upgrade.

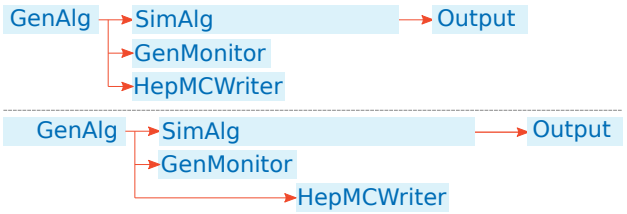

Figure 2: Illustration of the different algorithms and the data flow in Gaussino: a generation algorithm interfaces to the MC event generator and produces a generated event. This generated event can then be transformed by the algorithm running detector simulation. Additionally, multiple consumers such as monitoring algorithms and a dedicated writer for HepMC3 events may run concurrently.

dependencies are satisfied. Therefore, the framework allows parallelization within individual events as well as the processing of multiple events in parallel and an illustration for Gaussino is given in Figure 2. The remainder of this section discusses the major changes made in order to migrate to a multi-threaded framework.

2.1 Handling of random numbers

It is highly beneficial if the simulated events are reproducible and the same exact events can be generated again. With an appropriately chosen random seed for every production, this further guarantees that statistically non-overlapping samples are created. In the current Gauss framework, a global singleton instance of a random number engine is used which is seeded to the event-number and run-number at the beginning of each event. This approach is clearly not applicable anymore as multiple events would access the random engine concurrently. Due to the dependence on the scheduling of the various tasks, which can be influenced by the overall occupancy of the machine, the smallest predictable unit in the framework are the individual tasks, i.e. the execution of a single algorithm. Therefore, random engines in Gaussino are explicitly handled by the algorithm and created on the stack at the beginning of each execution of an algorithm. To insure independent random sequences in case multiple instances of an algorithm are used the seed is set to the event-number, run-number and the unique name of

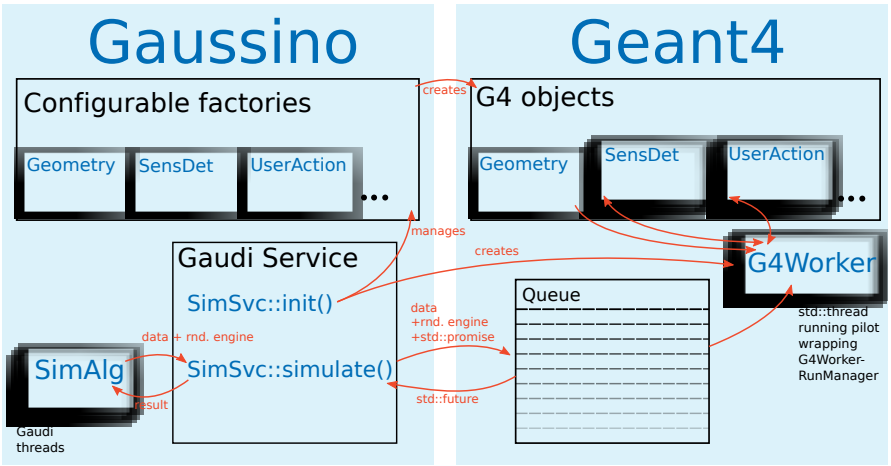

Figure 3: Illustration of the multi-threaded, flexible interface to Geant4 as implemented in Gaussino. Gaudi configurable objects are used to create corresponding Geant4 objects. These objects are then handled by Geant4. Geant4 worker run managers are executed in separate threads and receive their workload via a FIFO queue, allowing to flexible assign workloads from Gaudi to Geant4 threads.

the instance of the algorithm. The random engine obtained this way is then passed to any external library used by the algorithm to perform the event generation or simulation. As each execution of an algorithm has its own random engine which is only accessible during this single execution, events are reproducible independent of the scheduling order.

2.2 A multi-threaded generation phase

While the priority for migrating to a multi-threaded framework is clearly focused on the detector simulation, an implementation of the generation phase utilising more than one core is desirable when considering fast simulation options. However, the generation interfaces to external generators which can be found in a variety of different programming languages. Thus, no general solution to achieve a concurrent generation phase is possible. In order to achieve at least thread-safety, the access to the external event generators can be locked, turning the generation phase effectively single-threaded.

As an example, and to cover LHCb's most common use-case, Gaussino provides two interfaces to Pythia 8.2: a single-threaded interface locking access to the instance of Pythia and a multi-threaded interface that manages thread-local Pythia instances which are created on-the-fly when a thread queries the interface for the first time. We have verified that the thus generated events are fully reproducible.

2.3 The Geant4 interface

The cornerstone of the upgrade efforts for the simulation framework is the migration to a multi-threaded version of Geant4. To this end, the current interface found in Gauss, which heavily employs objects inheriting both from Gaudi and Geant4 objects, is redesigned to separate the different realms. Replacing the current approach, configurable Gaudi objects now act as factories that can create the required objects and register them with Geant4. For some

objects such as sensitive detectors, this is a mandatory change as each GEANT4 worker thread will require its own instance of the objects. Furthermore, this simplifies the ownership model which is now aligned with the description found in the GEANT4 documentation, something that will improve the maintainability in the future. An illustration of the new interface is shown in Figure 3.

The GEANT4 interface uses customised run managers where each worker run manager is manually configured in its own thread separate from the threads running the GAUDI algorithms. Instead, a GAUDI algorithm places the generated event to be simulated via a service into a FIFO queue and sleeps until the result of the detector simulation is available. The GEANT4 worker threads are waiting for input from the queue which also passes the random engine to be used. In the default set-up, a GAUDI algorithm will send the entire generated event into the queue as one unit which is then picked up by one GEANT4 worker. However, configuration options exist which split an event (e.g. every in-time pile-up separately) and enter the parts into the queue individually. While this creates challenges when trying to use the available CPU cores most efficiently when balancing the number of GAUDI and GEANT4 threads working together, it minimises the time required to complete an individual event. This can be the desired setup when processing on opportunistic resources such as LHCb's trigger farm where simulation jobs receive a signal to terminate in case the CPU resources are needed to run the trigger software.

The interface has been implemented in GAUSSINO and early tests with trivial geometries show good scaling with the number of available CPU cores.

3 The upgraded framework

We have presented the developments for LHCb's simulation framework for Run 3 of the Large Hadron Collider. This paper has outlined the design ideas behind the handling of a multi-threaded generation and detector simulation phase in a task-based parallelism approach using functional GAUDI. This core functionality has been implemented in GAUSSINO and tested with trivial geometries.

The current focus of the development is an integration of DD4HEP [13, 14] into GAUSSINO to allow for a detailed handling of complex geometries.

References

[1] R. Aaij et al. (LHCb collaboration), Phys. Rev. Lett. **120**, 171802 (2018), 1708.08856
[2] LHCb collaboration (2014), https://cds.cern.ch/record/1701361
[3] M. Clemencic et al., J. Phys. Conf. Ser. **331**, 032023 (2011)
[4] G. Barrand et al., Comput. Phys. Commun. **140**, 45 (2001)
[5] M. Clemencic, H. Degaudenzi, P. Mato, S. Binet, W. Lavrijsen, C. Leggett, I. Belyaev, J. Phys. Conf. Ser. **219**, 042006 (2010)
[6] J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Dubois et al. (Geant4 collaboration), IEEE Trans.Nucl.Sci. **53**, 270 (2006)
[7] S. Agostinelli et al. (Geant4 collaboration), Nucl. Instrum. Meth. **A506**, 250 (2003)
[8] T. Sjöstrand, S. Mrenna, P. Skands, JHEP **05**, 026 (2006), hep-ph/0603175
[9] T. Sjöstrand, S. Mrenna, P. Skands, Comput. Phys. Commun. **178**, 852 (2008), 0710.3820
[10] I. Belyaev et al., J. Phys. Conf. Ser. **331**, 032047 (2011)
[11] D.J. Lange, Nucl. Instrum. Meth. **A462**, 152 (2001)

[12] P. Golonka, Z. Was, Eur. Phys. J. **C45**, 97 (2006), hep-ph/0506026
[13] M. Frank, F. Gaede, C. Grefe, P. Mato, Journal of Physics: Conference Series **513**, 022010 (2014)
[14] M. Clemencic, A. Karachaliou, Journal of Physics: Conference Series **664**, 072012 (2015)