Gaussino - a Gaudi-Based Core Simulation Framework

B. G. Siddi, D. Müller, on behalf of the LHCb collaboration.

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. To cope with these challenges and in preparation for the third running period of the LHC, the LHCb collaboration currently overhauls its software framework to better utilise modern computing architectures. This effort includes the LHCb simulation framework (Gauss). In this article, we present Gaussino, an LHCb-independent simulation framework which forms the basis for LHCb’s future simulation framework which incorporates the reimplemented or modernised core features of Gauss. It is built on Gaudi’s functional framework making use of multiple threads. Event generation is interfaced to external generators with an example implementation of a multi-threaded Pythia8 interface being included. The detector simulation is handled by the multithreaded version of Geant4 with an interface allowing for the parallel execution of multiple events at the same time as well as for parallelism within a single event.

I. INTRODUCTION

Monte Carlo samples are a crucial tool to understand and interpret the data recorded by the experiments in High Energy Physics (HEP). Particles are produced by specialized generators and 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. While multiple fast simulation options are under development which are certainly available computing resources need to be used 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. Therefore, one of the main efforts on the LHCb Simulation framework in preparation for the Run3 focuses on the migration to a multi-threaded framework utilising a multi-threaded event loop. This event loop is driven by GAUDI with interfaces to multi-threaded GEANT4. The work on this upgraded framework is discussed in the following sections.

II. THE RUN1 AND RUN2 FRAMEWORK

The LHCb simulation framework GAUSS is built on top of GAUDI [3][4]. In the most commonly used procedure to simulate events in LHCb, GAUSS interfaces the PYTHIA 8.2 generator [5] mainly with EVTGEN [6] to handle the decays of heavy particles and to PHOTOS [7] to add final state radiation. Moreover, more external generators can be used. The implementation of the interaction of the generated particles with the detector, and its response, uses the GEANT4 toolkit on which GAUSS simulation is based [8]. The event generation and simulation with GEANT4 are executed in the same job. The simulation of a typical Run1 and Run2 event takes around a minute depending on the simulated decay, LHC run period and the employed CPU hardware. Most of the time is spent in the simulation of the detector response using GEANT4 (about 95%). With the increase of luminosity recorded by LHCb, a larger number of Monte Carlo samples will be needed and all available computing resources need to be used 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. Therefore, one of the main efforts on the LHCb Simulation framework in preparation for the Run3 focuses on the migration to a multi-threaded framework utilising a multi-threaded event loop. This event loop is driven by GAUDI with interfaces to multi-threaded GEANT4. The work on this upgraded framework is discussed in the following sections.

A. Random Numbers

The main characteristics of the simulated events is the reproducibility and the possibility to generate the same exact event. In the current GAUSS framework this is achieved using a global singleton instance of a random engine seeded
with the event-number and the run-number at the beginning of each event. To ensure reproducibility, 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 exists, the used seed is set to the event-number, run-number and the unique name of the instance of the algorithm.

B. Multi-threaded generation phase

Generator interfaces to external generators are found in a variety of different programming languages making a general solution to multi-threading difficult to find. In order to achieve at least thread-safety, the access to the external event generators can be locked. This means that the methods can be accessed by only one thread at a time, while other threads will be blocked until the method is unlocked by the first thread. Thus, synchronization has a penalty in performance, due to the underlying logic of synchronized access. For the event generation, GAUSSINO provides two interfaces to PYTHIA 8.2: a single-threaded interface locking access to the instance of PYTHIA 8.2 and a multi-threaded interface that manages thread-local PYTHIA 8.2 instances which are created on-the-fly when a thread queries the interface for the first time. Figure 3 shows the memory and throughput scaling for these two generator interfaces. In this context just after about ten threads there in no more gain in terms of maximal throughput due to custom multi-threading interface to the single-threaded generator.

C. GEANT4 interface

The GEANT4 interface uses customised run managers where each worker run manager is manually configured in its own thread. A GAUDI algorithm places the generated event to be simulated into a FIFO queue and sleeps until the result of the detector simulation is available. This allows for a flexible assignment of the workload between GAUDD and GEANT4 threads with the possibility of having one GAUDI event being split up and processed by multiple GEANT4 threads, minimising the time required to complete an individual event. For the configuration, GAUDI objects act as factories that can create the required objects and register them with GEANT4. An illustration of the new interface is shown in Figure 4. After the simulation of the detector the information is converted into MParticles, an output of the event model storing the Monte Carlo truth. In the upgraded framework this is redesign to use a wrapping to the event model internally that can convert to a
Hyperthreading

Fig. 5. Memory and throughput scaling for the generation of $D^{0}+\rightarrow K^{0}\pi^{+}$ with $D^{0}\rightarrow K\pi^{+}$ from minimum bias events with beam conditions as found in the 2016 data-taking period in LHCb. Shown are the curves for a shared (P8) and thread-local (P8MT) interface to Pythia 8. All particles known to EvtGen are declared stable in Pythia 8 and decayed by a shared instance of EvtGen. No simulation of a detector is performed [12].

useful output.

Lastly, in the upgraded framework DD4HEP [9] [10] will be used to handle the geometry description and its conversion to GEANT4. Currently DD4HEP is not used in Gaussino as only an elementary support exists.

D. Prototype validation

In order to provide a working prototype of the upgraded framework some additional tools have been ported, such as EvtGen and the LHCb geometry description (the migration to DD4HEP is in progress), and automatic testing using LHCbPR2 [11] (a LHCb performance and regression monitoring tool) has been added. Figure 5 shows the generation of a typical LHCb decay ($D^{+}\rightarrow D^{0}\pi^{+}$ with $D^{0}\rightarrow K\pi^{+}$) using the two PYTHIA interfaces and EvtGen single-threaded locking interface to handle the decays. As can be seen, the LHCb specific configuration and cuts at the generation level do not affect the overall performance. Figure 6 shows the output from LHCbPR2 from the generation and simulation phases in GAUSS integrated in GAUSSINO and from the current GAUSS framework. The curves represent the energy deposited per hit in the VELO detector and the energy of the particle that creates it using the typical LHCb decay mentioned before. The agreement between the current and the upgraded framework is good.

The performances for generation and simulation phases using GAUSS in GAUSSINO are shown in Figure 7. The decay used to test the performance is the same reported above and the generation phase is tested using thread-local and locking interface. Performances show an overall good scaling in throughput and in the memory usage.

IV. CONCLUSIONS

In this paper, we have summarised GAUSSINO, a core simulation framework developed within the LHCb collaboration which allows relies on multi-threading to optimise the benefits of modern CPU architectures. It forms the basis of the LHCbs simulation framework for Run 3 of the LHC but it is itself independent of LHCb and thus can be adopted in other projects. We have compared multiple current GAUSS processes to future GAUSS. The plots show encouraging first results showing that the performances are not affected by LHCb specific tools, moreover good agreement between the current and upgraded simulation framework has been presented.

REFERENCES

Fig. 7. Memory and throughput scaling for the generation and simulation of $D^{*+} \rightarrow D^0 \pi^+$ with $D^0 \rightarrow K \pi^+$ from minimum bias events with beam conditions as found in the 2016 data-taking period in LHCb. Shown are the curves for a shared (P8) and thread-local (P8MT) interface to Pythia 8. All particles known to EvtGen are declared stable in Pythia 8 and decayed by a shared instance of EvtGen. For comparison, events identical to those generated in the same job are loaded from a file (Sim only), skipping the generation phase in the job [12].