EvtGen — recent developments and prospects

Fernando Abudinén[1,*], *John* Back[1,**], *Michal* Kreps[1,***], and *Thomas* Latham[1,****]

[1]Department of Physics, University of Warwick, Gibbet Hill Rd,
Coventry CV4 7AL, United Kingdom

Abstract. EvtGen is an event generator specialised for decays of heavy hadrons. Since its early development in the 90's, the generator has been extensively used and has become an essential tool for heavy-flavour physics analyses. Throughout this time, its source code has remained mostly unchanged, except for additions of new decay models. In view of the upcoming boom of multi-threaded processing, we have launched a modernisation campaign with the chief goal of making EvtGen thread safe. We report on the challenges encountered in this endeavour and the milestones reached so far.

1 Introduction

EvtGen [1] is an event generator specialised for the simulation of decays of heavy hadrons. Due to its independence from the experimental environment, EvtGen is used in several computing frameworks in high-energy physics to simulate the decays of heavy hadrons produced in particle collisions and heavy-quark jets. Since its origins within the CLEO and BaBar collaborations, EvtGen has been extensively used to simulate underlying physics processes, and has become an essential tool for heavy-flavour physics analyses.

EvtGen contains currently 139 decay models. Each one corresponds to a separate module implementing the dynamics of a specific decay type. Most decay models use amplitudes to calculate the decay probability while others set the probability directly. Based on the output of the decay model, the probability for each node in a decay tree is used to simulate the entire decay chain including all kinematic correlations.

Among the implemented features, EvtGen maintains a detailed decay table with a large number of explicit decays. When the sum of the branching fractions of known hadronic decays does not add up to unity, the remainder is filled up by generating the appropriate quark configurations and passing them to Pythia 8 [2] for fragmentation. In particular for b-baryon decays, where far fewer branching fractions are currently measured, there is a much greater dependence on Pythia 8 than for other hadrons. There are two further external dependencies in EvtGen: TAUOLA [3–6] is used to simulate the decays of τ leptons, and PHOTOS [7] is used to simulate final-state radiation (FSR).

Since its origins, the EvtGen core framework has been kept almost unchanged. The main developments have been the addition of new decay models provided by various collabora-

*e-mail: fernando.abudinen@cern.ch
**e-mail: j.j.back@warwick.ac.uk
***e-mail: michal.kreps@cern.ch
****e-mail: t.latham@warwick.ac.uk

tions. In view of the current transition of computing frameworks towards multi-threaded processing, we have launched a modernisation campaign with the chief goal of making EvtGen thread safe. As a first step, we focused on identifying and implementing the modifications necessary to enable thread safety. In this process, we have taken the chance to work also on reducing code duplication and unifying the coding style across the decay models.

An important part of this campaign has been the development of a global testing framework, which will be described in Sec. 2. We will then discuss the implemented modifications to make EvtGen thread safe in Sec. 3. Part of the challenge has been the external dependencies on generators that are not yet thread safe. We explore alternatives for them in Secs. 4 and 5. The subsequent section 6 is devoted to discuss possibilities for further improvements identified during the campaign. Section 7 describes a technique we are currently exploring for updating the database of the decay table, followed by a summary and an outlook in Sec. 8.

2 Testing framework

The need to validate the physics output of the simulation after implementing code modifications has prompted us to develop a global testing framework. It consists of a general testing module that is steered with JSON [8] configuration files. Each configuration file specifies a decay tree, the decay model for each node, and a list of validation histograms of physics observables. We choose observables associated with the specific kinematics and topology of the underlying decay processes. Based on the configuration file, the testing module generates events for the specified decay tree, calculates the physics observables for each event and fills the validation histograms, which are then compared with reference distributions produced prior to the modifications.

Figure 1 shows example validation histograms for two decay models. Before the development of the global testing framework, tests in different formats existed for about 40% of the decay models. We migrated the preexisting tests to be consistent with the new testing framework and introduced new configuration files in order to have at least one test for each existing decay model. However, the current list of tests is not exhaustive since some decay models support various configurations that are not yet covered, such as various decay topologies or form factors. Thus, we will continue expanding the list of tests and will request tests to be added for each future new decay model.

Besides the validation after code modifications, the global testing framework has helped us to uncover and fix issues within existing decay models that were not tested before. As a further step, we set up a strategy for the execution of the tests and the comparison of the validation histograms. When modifications are committed, the testing framework will identify which decay models are affected based on the modified files. It will then carry out the tests only for the affected decay models. If the modifications involve files concerning the EvtGen core framework, then all tests will be carried out.

For the comparison of validation histograms, our current strategy is to use the histograms produced with the master branch as reference. For any feature branch, the histograms used as reference will be the ones corresponding to the best common ancestor commit between the master and the feature branch, that is the so-called `merge base`.

3 Making EvtGen thread safe

Making EvtGen thread safe requires to overcome internal and external limitations. The internal limitations are associated with the structure of the core framework, while the external limitations are associated with third-party software dependencies. We have identified the

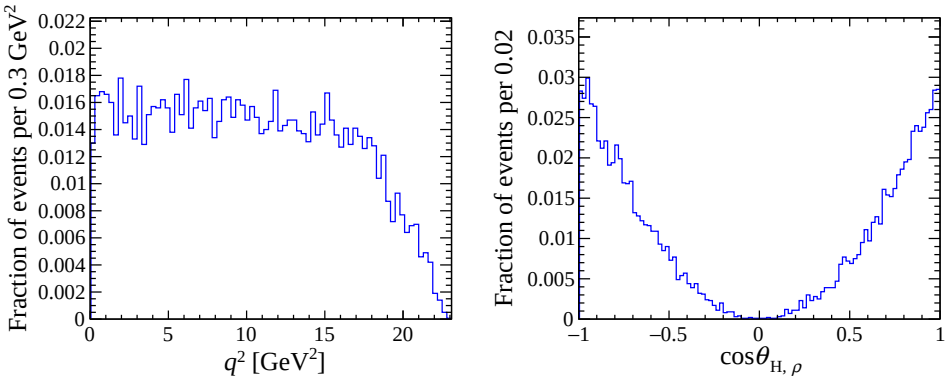

Figure 1. Examples of validation histograms. Left: distribution of dimuon invariant-mass squared $m(\mu^+\mu^-)^2 = q^2$ (in natural units $c = 1$) for $B^+ \to K^+\mu^+\mu^-$ decays simulated with the `BTOSLLBALL` decay model. Right: distribution of helicity angle $\cos\theta_{\mathrm{H},\rho}$ for $B^+ \to \overline{D}{}^0\rho^+(\to \pi^+\pi^0)$ decays simulated with the `SVS` decay model. The helicity angle $\cos\theta_{\mathrm{H},\rho}$ corresponds to the angle between the B^+ and the π^+ momenta in the ρ^+ frame.

changes necessary to overcome the limitations and implemented a preliminary set of solutions.

The internal limitations are mainly associated with global static instances of the random number generator, the particle properties and the decay table. Static objects are thread safe only at initialisation. They are not thread safe if they mutate during execution as multiple threads can modify them simultaneously irrespective of each other. We have tackled this issue by, where possible, converting static objects into static constant objects. For example, cached values of particle properties. Where it was not possible to make static objects constant, they have instead been converted into static thread-local objects, such that there is effectively one global object per thread. The random number generator and decay table are examples of objects that have been treated in this way. Regarding the random number generator, this treatment also guarantees that the results are reproducible irrespective of the number of threads, provided that the seeding is based on the event number.

The external limitations are associated with the TAUOLA and PHOTOS generators which are currently not thread safe. While the authors of these generators are exploring ways to implement thread safety, we have temporarily overcome the limitations by serialising the calls for these generators using the `C++` mutex functionality. In this way, each time a thread calls up one of these generators, the associated objects are locked for all other threads until the operation is concluded.

This preliminary set of solutions enables thread safety, passing successfully the tests for all decay models. We have further studied the performance in a multi-threaded environment. Figure 2 shows the speedup reached by the simulation with and without simulation of FSR using PHOTOS. In both cases some speedup is achieved, especially when switching off the FSR simulation. However, there is only very little speedup when switching on the FSR simulation.

Although further improvements in the EvtGen core framework are possible, the external dependencies remain the limiting factor to exploit the capabilities of multi-threading. We therefore explore the possibility to use alternatives for the simulation of τ decays and FSR.

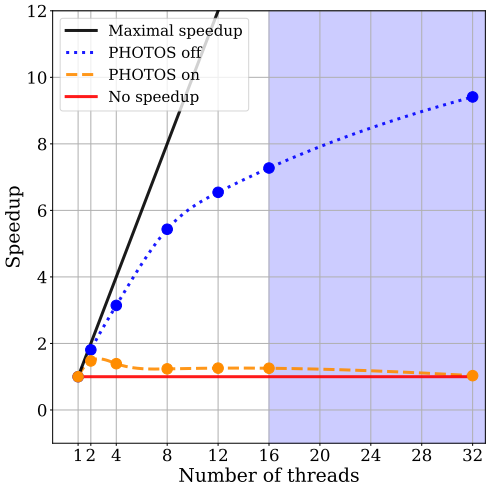

Figure 2. Speedup versus number of threads after implementing thread safety. The black solid curve shows the ideal case with linear maximal speedup, while the red solid horizontal curve shows no speedup. The dotted blue curve shows the large increase in code execution speed when FSR is not enabled (PHOTOS off), while the dashed orange curve shows the limited speedup when FSR is enabled (PHOTOS on). The blue shaded area corresponds to the hyperthreading regime.

4 Simulation of τ decays

Decays of τ leptons are simulated through an interface with TAUOLA. Besides the fact that TAUOLA is currently not thread safe, the present interface does not propagate information about the spin state of the τ lepton, which is essential for physics analyses that are sensitive to the τ polarisation.

As an alternative, we are exploring the possibility to simulate τ decays with spin-state propagation using the Helicity-Matrix-Element (HME) module inside Pythia 8. We have finalised a first version of an interface between EvtGen and the Pythia HME module and tested the output of the simulation. A particular challenge has been the spin-basis conversion between the two generators. We have implemented a preliminary version of the basis conversion that has provided the expected results in specific cases. However, further development and validation is needed for general use. An implementation of a general spin-basis conversion would be also useful for propagation of spin-state information with TAUOLA.

Figure 3 shows a comparison of the physics output for $B^+ \to \tau^+(\to \pi^+\bar{\nu}_\tau)\nu_\tau$ and $B^+ \to \tau^+(\to \mu^+\nu_\mu\bar{\nu}_\tau)\nu_\tau$ decays obtained using the Pythia HME module and TAUOLA. For these two cases, the output of the two simulations agree within expectations.

5 Simulation of final-state radiation

Most of the simulation produced with EvtGen is generated with FSR. The only currently available option for simulating FSR photon emission for charged particles is to use PHOTOS for the whole decay tree or for particular decay nodes. When FSR is enabled, the decay is passed to PHOTOS. The decay is then retrieved to add the emitted photons and update the momenta of the particles inside the decay. The propagation of the information between

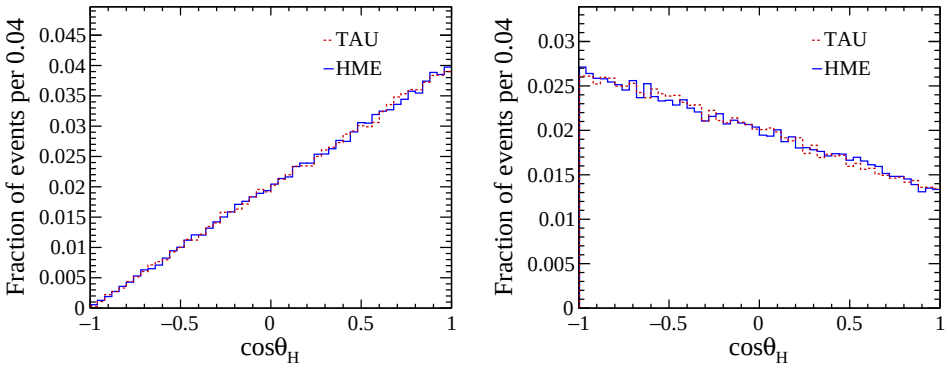

Figure 3. Distributions of helicity angle $\cos\theta_H$ for simulated (left) $B^+ \to \tau^+(\to \pi^+\bar{\nu}_\tau)\nu_\tau$ and (right) $B^+ \to \tau^+(\to \mu^+\nu_\mu\bar{\nu}_\tau)\nu_\tau$ decays. The helicity angle $\cos\theta_H$ corresponds to the angle in the τ frame between the B^+ and the π^+ for the left plot, and between the B^+ and the μ^+ for the right plot. The solid blue (dotted red) curve corresponds to events where the τ decays are simulated with the Pythia HME module (TAUOLA generator).

PHOTOS and EvtGen happens through an interface which converts the EvtGen particle objects into a HepMC event [9] and vice versa. The HepMC event is passed to PHOTOS, which converts it internally into its own objects.

Currently about one third of the CPU time consumed by EvtGen is due to FSR simulation. Based on profiling, we estimate that roughly half of this time is effectively consumed during the conversion of internal objects into HepMC events and vice versa. An adaptation of the interface thus has a large potential to reduce the CPU consumption by avoiding the intermediate conversion into HepMC events. However, the limitation of PHOTOS not being thread safe still remains.

We are exploring the possibility to use the PHOTONS++ module [10] inside the Sherpa generator [11], which is able to simulate soft and hard photon emissions and complies with the thread safety requirement. It includes various configuration parameters, such as the energy cut-off, which need to be tuned depending on the decay topology and kinematics. We have recently started to implement an interface between EvtGen and Sherpa with the aim of transferring decays to the PHOTONS++ module and back. The current strategy is to use the intermediate conversion into HepMC objects for the transfer of information between the two generators. Although it is CPU time consuming, this procedure has been extensively validated. It can thus allow us to prototype expeditiously a preliminary interface and focus on parameter tuning and testing of the physics output. However, we intend to generally avoid the HepMC conversion in the long term.

6 Improving multi-threaded performance

In our current implementation of thread safety (see Sec. 3), the decay table is declared as a thread-local object. However, ideally it should be a constant object to avoid having a replicated instance per thread. The reason why it has not been implemented as such lies deep inside the core framework: the decay table instance incorporates a container of decay models as a data member. Each decay model overrides from the `EvtDecayBase` class a virtual `decay` function, which generates the particle decay at each node. The `decay` function can

mutate the state of the decay model objects and is therefore not constant. Hence the state of the decay table is also mutable.

A possible solution is to modify the `decay` function such that it is constant. Although conceptually straightforward, such a solution requires the modification of every single decay model since the `decay` function is always overridden. Such a far-reaching intervention requires a comprehensive validation and would considerably delay the completion of a first thread-safe release. Thus, we defer the implementation of this solution for future developments.

7 Updating the decay table branching fractions

Another challenge associated with the decay table is the updating of its database. The Particle Data Group (PDG) [12] keeps an up-to-date collection of high-energy physics measurements and averages. However, the amount of provided machine readable information is limited. This situation is expected to improve with a future new application programming interface. However, avoiding ambiguities between branching fraction measurements will still require human intervention, for instance, to avoid double counting of decay modes with intermediate resonances.

We explore the possibility to update the decay table database in a semi-automatic way by generating simulation and comparing the frequencies of the generated decays with the world-average branching fractions. The strategy relies on modifying the branching fractions with the worst discrepancies in an iterative process to minimize a χ^2 quantity defined as the sum of the pulls over all exclusive branching fractions. The pulls correspond to the differences between the branching fraction values in the decay table and the known PDG values, divided by the uncertainties on the PDG values. Inclusive branching fractions are ignored in the χ^2 calculation, but checked for consistency at the end of the iterative procedure.

Figure 4 shows the branching fraction pulls for known D_s^+ decay modes before and after updating the database of the decay table. The value of χ^2 over the number of degrees of freedom was initially $963/72$ and after the iterative procedure it was $58/72$. This example shows that the procedure can efficiently improve the agreement with the world's best knowledge. Hence we plan to test it with particles having larger numbers of decay modes.

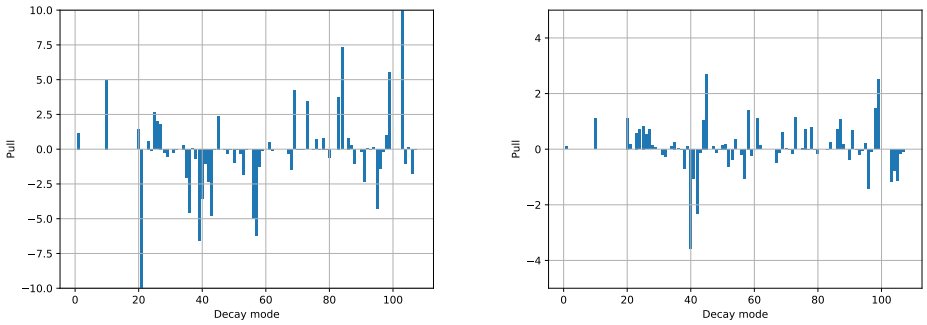

Figure 4. Pulls of branching fractions for known D_s^+ decay modes (left) before and (right) after updating the decay table database as described in the text. Note the different scales on the y-axis.

8 Summary and outlook

EvtGen is an event generator specialised for decays of heavy-flavour hadrons and is an essential tool for physics analyses studying such processes. Since its early developments, its core framework has been kept almost unchanged. The current transition of experimental frameworks into multi-threaded processing has prompted an EvtGen modernisation campaign with the goal of implementing thread safety. A first step in this regard has been the implementation of a global testing framework to ensure that the physics output remains invariant after the modifications.

We have identified internal and external challenges to be overcome in order to implement thread safety. The internal challenges are associated with the structure of the core framework, which is based on global static objects that can mutate during program execution. Our preliminary solution has made those objects either constant or thread local. The external challenges are associated with dependencies on third-party generators that are used to simulate τ-decays (TAUOLA) and final-state radiation (PHOTOS) which are not yet thread safe. We have temporarily serialised their use by deploying the `C++ mutex` functionality. However, these external dependencies remain the limiting factor. We are therefore exploring alternatives for the simulation of τ-decays and final-state radiation.

We have found and implemented a set of preliminary solutions that are necessary to enable thread safety. The modernisation campaign has allowed us to identify further possibilities for improvements, which imply code redesign and general modifications, offering opportunities for future developments.

9 Acknowledgements

We thank Heather Ratcliffe and Christopher Brady at the University of Warwick for helping us identify and implement the modifications needed to make EvtGen thread safe. We also thank the Monash-Warwick Alliance in Particle Physics (MWAPP) for its support.

References

[1] D. J. Lange, NIM **A462**, 152 (2001)
[2] T. Sjöstrand, S. Mrenna, and P. Skands, CPC **178** 852 (2008), arXiv:0710.3820
[3] S. Jadach, J.H. Kuhn, and Z. Was, CPC **64** 275 (1990)
[4] M. Jezabek *et al*, CPC **70** 69 (1992)
[5] S. Jadach *et al*, CPC **76** 361 (1993)
[6] M. Chrzaszcz *et al*, CPC **232** 220 (2018), arXiv:1609.04617.
[7] N. Davidson, T. Przedzinski, and Z. Was, CPC **199** 86 (2016), arXiv:1011.0937
[8] T. Bray, IETF RFC **8259** (2017), rfc:8259
[9] M. Dobbs and J. B. Hansen, CPC **134** 41 (2001), ATL-SOFT-2000-001
[10] M. Schonherr *et al*, JHEP **12** 018 (2008), arXiv:0810.5071
[11] T. Gleisberg *et al*, JHEP **02** 056 (2004), arXiv:0311263
[12] R. L. Workman *et al* [PDG], PTEP **2002** 083C01 (2002), Review of particle physics