Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds., eds.

A CO-SIMULATION TECHNIQUE FOR EFFICIENT PARTICLE TRACKING USING HYBRID NUMERICAL METHODS WITH APPLICATION IN HIGH ENERGY PHYSICS

Lucio Santi
Rodrigo Castro

Departamento de Computación FCEyN, UBA and ICC-CONICET
Ciudad Universitaria, Pabellón 1
C1428EGA, Buenos Aires, ARGENTINA

ABSTRACT

Particle tracking in physical systems is a well known simulation challenge in many domains. In particular, High Energy Physics (HEP) demand efficient simulations of charged particles moving throughout complex detector geometries in a magnetic field. Quantized State Systems (QSS) is a modern family of hybrid numerical methods that provides attractive performance features for these problems. Its state-of-the-art implementation is the general-purpose QSS Solver toolkit. Meanwhile, Geant4 is the most widely used platform for computational particle physics, embedding vast amounts of physics domain knowledge. Yet, Geant4 relies rigidly on classic discrete time numerical methods. In this work we present a robust co-simulation technique to apply QSS in the simulation of HEP experiments, thus leveraging the best of both toolkits. We obtained speedups of up to three times in synthetic, yet representative scenarios, and a competitive performance in a difficult benchmark modeled after the Compact Muon Solenoid (CMS) particle detector at CERN.

1 INTRODUCTION AND MOTIVATION

The tracking of particle trajectories in physical systems is a well known simulation problem with application in many domains such as fluid dynamics, crowd systems, 3D rendering and particle physics, to name a few. Almost every well established domain of knowledge (and even every particular problem) counts with a palette of simulation software developed throughout decades that tend to embed significant amounts of valuable domain-specific knowledge. Innovating in such tools and algorithms can be a risky task, as legacy code tend to be rigid and difficult to scale up.

In this work we focus on the field of High Energy Physics (HEP) applications. HEP particle simulations deal with the tracking of particles affected by physics processes in complex detector geometries (adjacent 3D volumes of different shapes and materials). In the HEP domain the accuracy and performance of the algorithms for particle tracking are of great interest as they can significantly impact the requirements of computing resources and their associated cost. In this context, our motivation is twofold: to provide means for improving on the performance of particle tracking algorithms, and to extend valuable pre-existing code in an elegant and robust way. We achieve these goals by means of a co-simulation strategy.

HEP simulations are widely based on the Geant4 simulation toolkit (Allison et al. 2016). It uses classical numerical methods that rely on time discretization (Cellier and Kofman 2006), in particular variations of the Runge-Kutta family (RK) of solvers (Butcher 1987). On the other hand, Quantized State System (QSS) methods (Cellier and Kofman 2006, Kofman and Junco 2001) discretize the state variables instead of slicing time, and solve ordinary differential equations (ODEs) using discrete–event approximations of continuous models. QSS is a family of hybrid numerical methods that combine continuous with discrete-event dynamics to approximate continuous systems. A feature of QSS that stands as very attractive in the context of HEP simulations is that these methods handle discontinuities (such as volume crossings) very efficiently (Kofman

2004) by means of a computationally cheap procedure (solving a zero-crossing polynomial function, seen as a discrete–event).

In previous works we verified the potential of QSS to offer speedups in particle tracking (Santi et al. 2017) comparing for this purpose Geant4 against QSS Solver (Fernández and Kofman 2014), a standalone simulation toolkit that provides state-of-the art implementations of QSS methods. We have also prototyped a proof-of-concept communication between Geant4 and QSS Solver that allowed connecting Geant4 to arbitrary external integrators but focusing on the QSS family (Santi et al. 2018). In this work we extend and improve this interface by providing a fully fledged co-simulation scheme for QSS Solver including thorough performance optimizations. We also devote special attention to a real world HEP application: the Compact Muon Solenoid (CMS) particle detector in the Large Hadron Collider at CERN, for which simulation has taken approximately 85% of the total CPU time since the start-up in 2009 through May 2016, with a total of about 40% for Geant4. Assuming an estimated cost of 0.9 US dollar cents per CPU core hour (Elvira 2017), the annual simulation cost of CMS is in the range of 3.5 to 6.2 million US dollars (each 1% reduction in simulation times would yield savings from 50 to 80 thousand US dollars per year).

This paper is organized as follows: Section 2 briefly introduces the QSS theory and the essential concepts of the Geant4 simulation toolkit, and then summarizes our previous contributions on the field. Section 3 gives a detailed explanation of our co-simulation technique, starting with a contextual summary of other related co-simulation strategies. A performance comparison in two complementary scenarios between standard Geant4 and our approach is then shown in Section 4. Finally, Section 5 contains a summary, conclusions and comments on our work in progress.

2 BACKGROUND INFORMATION

2.1 The Quantized State System (QSS) Numerical Integration Methods

QSS are numerical methods that apply a state quantization approach to solve systems of ordinary differential equations (ODEs) in the form of Eq. (1), where $\mathbf{x}(t)$ is the *state vector* and $\mathbf{u}(t)$ is the *input vector* representing independent variables for which no derivatives are present in the system.

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{1}$$

$$\dot{\mathbf{x}}(t) = f(\mathbf{q}(t), \mathbf{u}(t)) \tag{2}$$

Traditional ODE solvers (e.g. the widely adopted Runge-Kutta family (Butcher 1987)) make use of *time slicing*: given the current and past values of state variables and their derivatives, the solver estimates the next value of the state x_{k+1} one "time step" Δt into the future (i.e., at $t_{k+1} = t_k + \Delta t$) where Δt applies to all state variables and represents the means to control the approximation accuracy.

QSS solvers operate differently: they make use of *state space quantization*. Rather than slicing the time axis every Δt units, QSS discretizes the state variable axis. QSS calculates the first time instant t_{k+1} into the future at which the state variable differs from its current value by one "quantum level" ΔQ, i.e., when $x_{k+1} = x_k \pm \Delta Q$. The system described by Eq. (1) is thus approximated by the quantized system shown in Eq. (2), where $\mathbf{q}(t)$ is the *quantized state vector* resulting from the quantization of the state variables $x_i(t)$. In the first-order QSS method (QSS1) each $q_i(t)$ follows a piecewise constant trajectory that is updated by a (hysteretic) *quantization function* when the difference between $q_i(t)$ and $x_i(t)$ reaches the *quantum* $\Delta Q_i = \max(\Delta\text{QRel} \cdot |q_i|, \Delta\text{QMin})$, derived from the precision demanded by the user by means of a relative and a minimum quanta, ΔQRel and ΔQMin. In QSS1, $\mathbf{q}(t)$ follow piecewise constant trajectories, which implies that $\mathbf{x}(t)$ follow piecewise linear trajectories. Along the same principle, higher-order QSS methods generalize this behavior: in QSSn, $\mathbf{x}(t)$ follow piecewise n-th degree polynomial trajectories and $\mathbf{q}(t)$ follow piecewise $(n-1)$-th degree polynomial trajectories (Kofman and Junco 2001).

QSS was thoroughly studied in different application domains, comparing these methods against discrete-time solvers with both fixed and adaptive step size control. For instance, in (Fernández and Kofman

2014) QSS is compared with DASSL and Runge-Kutta solvers, while (Bergero et al. 2016) studies the performance of advection-diffusion-reaction models comparing QSS against DASSL, Radau and DOPRI methods. In the context of HEP applications, QSS exhibits several attractive features such as *inherent asynchronicity* (each state variable updates its value independently, at self clocked time instants), *dense trajectory output* (supported by the piecewise polynomial approximations of the trajectory) and *efficient handling of discontinuities* (they are modeled by zero-crossing functions involving the QSS polynomials, thus being computationally cheap to solve).

Regarding practical tools, *QSS Solver* (Fernández and Kofman 2014) is an open-source standalone software that provides optimized C implementations of different QSS methods as well as other traditional algorithms (e.g., Dormand-Prince method). Equations are expressed in μ-Modelica (Bergero et al. 2012), a subset of the more general Modelica language (Fritzson 2015).

2.2 The Geant4 Simulation Toolkit

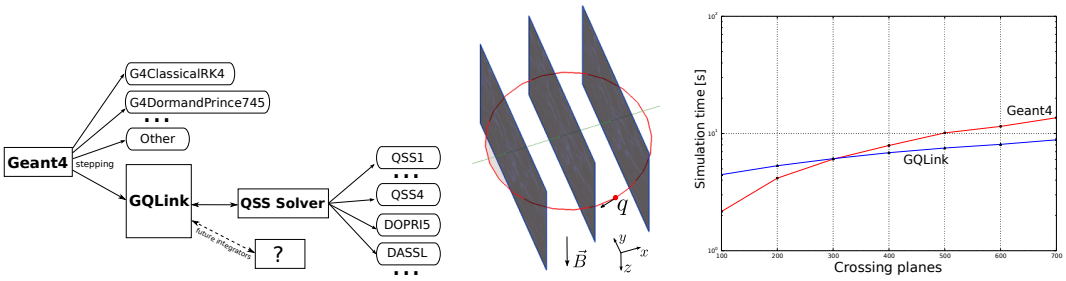
Geant4 is the most widely used simulation toolkit in contemporary HEP experiments. A Geant4 simulation is typically encompassed within a *run*, which is a sequence of *events*, the basic unit of simulation. At the beginning of a run, the detector *geometry* (a collection of physical volumes organized in a tree structure) is already assembled and cannot be changed. Each event is, in turn, made of one or more *tracks*. A track is a snapshot of a particle, containing physical quantities through which the particle is, indeed, tracked along the detector as its trajectory is simulated. Upon starting a new event, *primary tracks* are generated and pushed into a stack. Tracks are then popped up one at a time and simulated. Physical interactions might generate new, *secondary tracks*, which are also pushed into the stack and simulated serially (particles do not interact with each other). An event finishes when the stack becomes empty. Each track can be thought as a sequence of *steps* advancing the particle for a given distance. This step length can be limited by the *physics processes* that model particle interactions with matter. Processes are assembled into a *pyhsics list* covering all combinations of incident particle type, energy, and target material. Geant4 provides ready-to-use reference physics list for HEP. The recommended one, `FTFP_BERT` (Dotti et al. 2011), contains all standard electromagnetic processes (e.g., ionisation and multiple scattering) and uses Bertini-style cascade for hadron-nucleus interactions with an incident hadron energy below 5 GeV and the Fritiof model for high energies. For synthetic experiments these processes can be turned off.

Stepping in Geant4 is done by the *steppers*, which are custom implementations of several well-known Runge-Kutta-based solvers. The default stepper is the fourth-order accurate Runge-Kutta, RK4. A step might end before covering its length due to reaching a volume boundary. The computation of these boundary crossings is done through other custom, iterative algorithms that will be briefly discussed in Section 3.2 (see (Santi et al. 2017) for a more detailed and comprehensive explanation of the stepping algorithm).

2.3 Previous Work

In (Santi et al. 2017) the performance of QSS Solver was studied in the context of a minimal HEP-like setup consisting of a charged particle under a uniform magnetic field describing a circular 2D motion. The particle crosses equidistant parallel planes as shown in Figure 1b. We found configurations with 200 planes and a track length of 100 m where QSS Solver is 6x faster than Geant4 with better accuracy, when the physics interactions are turned off. We continued this work by designing, implementing and testing a proof-of-concept version of GQLink (Santi et al. 2018), conceived as an abstract interface that allows connecting Geant4 to arbitrary external integrators, in particular the QSS family as implemented by QSS Solver, as shown in Figure 1a. We found that GQLink (using QSS2) outperformed Geant4 in the aforementioned setup depicted in Figure 1b, with a 35% speedup for 700 crossing planes (Figure 1c). Also, we analyzed a realistic HEP application featuring a full CMS detector geometry. We verified the statistical consistency of the simulations and found that GQLink was about 17% slower than Geant4. In

this work we shall extend and improve GQLink by providing a co-simulation scheme for QSS Solver, including performance optimizations for this strategy.



(a) GQLink high-level architecture (b) Experimental setup (c) Performance of proof-of-concept GQLink

Figure 1: Preliminary comparison between Geant4 and a proof-of-concept version of GQLink.

3 A CO-SIMULATION SCHEME BETWEEN GEANT4 AND QSS METHODS

GQLink is our proposed interface for co-simulation between Geant4 and a QSS simulation engine such as the QSS Solver tool. GQLink orchestrates robustly and transparently the interaction between the latter and aspects such as geometry definition and physics processes, controlled by Geant4. It is then regarded as a co-simulation scheme where Geant4 drives the simulation and relies upon a QSS engine to drive particle motions. A key advantage of this approach is that each tool is responsible for a well-defined subset of tasks (namely, physics definition, evaluation and geometry navigation in Geant4, and particle transport through ODE solving in QSS Solver). This follows the separation-of-concerns principle in software engineering (De Win et al. 2002). GQLink then fully takes over the responsibility of particle tracking and step computation, relieving Geant4 of those tasks.

Several co-simulation strategies exist, such as the modern Functional Mockup Interface (Blochwitz et al. 2011). In a typical scenario, each FMI co-simulated entity solves independently a given subsystem for a lapse of time, exchanging data later at certain communication points. This approach is not well suited for our setting as the set of sub-models each FMI entity is responsible for are disjoint. Similarly, the co-simulation software MpCCI (Wirth et al. 2017) provides algorithms and interfaces to couple different tools so as to solve collaboratively the simulation of disjoint models. A technique for distributed discrete event co-simulation with application to the automotive industry, termed Complex Control Systems Simulation (CCSS), was presented in (Munawar et al. 2013). This approach is parallel by design and defines an improved synchronization mechanism derived from conservative synchronization mechanisms, which currently is beyond the scope of our problem. Another interesting co-simulation tool is the SDLPS simulator (Fonseca 2013), which allows for co-simulation of models described by the SDL language (http://sdl-forum.org/SDL). As we discussed, GQLink leverages the physics domain knowledge already embedded into Geant4; it would require a significant effort to develop new, custom SDL-based models to properly simulate e.g. the particle-matter physics interactions. Finally, a closely related concept is simulation model interoperability, where distributed models simulate with their own clocks and exchange messages in real time (see e.g. (Moallemi et al. 2011)). This approach appears as inadequate in our case as our co-simulating entities must be exactly aware of their shared time advance at all instants.

3.1 High-Level Design

GQLink-enabled simulations operate in two phases: *initialization*, where the QSS engine is bootstrapped following the user's requirements, and *simulation*, in which particles are successively transported along the detector until no secondaries are produced.

Initialization Phase The initialization phase consists in choosing an appropriate model for the problem at hand (currently, the equations of motion for charged particles in a magnetic field is the only one available) and instantiating a GQLink model object that exposes an API for the interaction between Geant4 and the QSS Solver engine. The engine is started by first loading into RAM the pre-compiled shared library for the user-requested QSS method (e.g., QSS2) and later invoking the usual initialization routines provided by QSS Solver. Once this is done, the user-requested accuracy parameters (ΔQRel and ΔQMin) are applied through the GQLink API. Figure 2 illustrates the initialization stage.

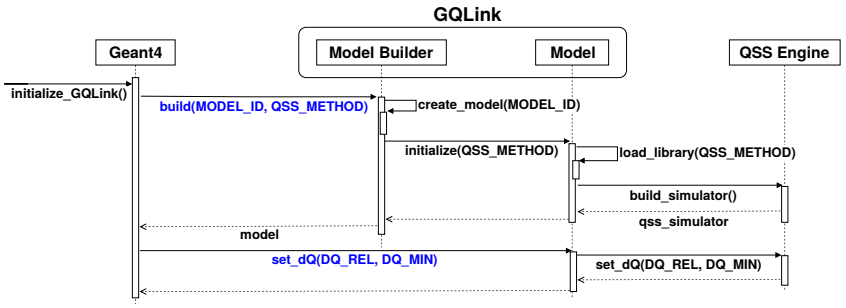


Figure 2: Initialization stage of GQLink simulations.

Simulation Phase After the initialization phase GQLink is ready to run. This is done after other Geant4 bootstrapping tasks. The core of the simulation stage is the new method `GQLink_ComputeStep`, which is the entry point to the QSS integration routines. The standard Geant4 method to propagate particles in a magnetic field, `ComputeStep`, was adapted to call this new method instead of Geant4's default stepping algorithms. Before taking the first step, the QSS simulator is configured by the setting the particle charge, its mass, and its initial position and velocity. Given the position, the initial value of the magnetic field is also retrieved by calling back a Geant4 routine through the GQLink interface. This process, implemented in the `GQLink_reset` routine, is performed every time a secondary particle is about to be tracked.

The first action of GQLink's stepping algorithm is to assemble a step data object to gather relevant information about the step needed by both the QSS engine and Geant4. Initially, it provides the Geant4-defined step length and other track parameters such as current momentum and traversed curve length. Before taking the step through the main QSS integration loop, it is essential to manually reinitialize the velocity state variables, as the underlying physics processes evaluated by Geant4 might have changed the direction of the particle. This is done in the `GQLink_reinit` routine, which implements an optimized reinitialization of state variables (this is typically done in the discrete-event handlers of QSS Solver models).

The main QSS integration loop implemented in the QSS Solver engine was modified so as to take the step data, update it accordingly as it iterates and eventually stop to handle control back to Geant4 if e.g. the current length traversed reaches the aforementioned step length or if a volume boundary was crossed. Each of the iterations defines a *substep* along which the particle position can be approximated by the QSS dense output. A key optimization with respect to the proof-of-concept GQLink presented in Section 2.3 consisted in packing together a (user-configurable) number n of substeps before calling back the Geant4 geometry routines to find out if a volume boundary was crossed. In order to achieve this, after each iteration, the new substep is saved into a data structure that holds the substep information (e.g. the coefficients of the QSS polynomials for every state variable). The substep packing strategy is a core aspect in the efficient intersection finding routines exposed by GQLink. This will be further developed in Section 3.2.

Once n substeps are packed, a *substep block* is assembled and a *checkpoint* is reached: the integration procedure is temporarily suspended in order to check if a volume boundary was crossed. This is achieved by following the same call pattern as in standard Geant4 simulations, using the GQLink interface as proxy: first, the method `LocateGlobalPointWithinVolume` is called so as to notify the geometry navigator that the particle has moved to a new position inside the current volume. Then, `IntersectChord` computes an initial estimation of an intersection by means of a linear segment between the endpoints of the substep

block, which is later refined through an iterative procedure in `EstimateIntersectionPoint`. This last function was considerably improved as a consequence of the substep packing strategy; we offer a cheaper particle transport derived by the evaluation of the substep polynomials instead of running the iterative, adaptive stepping algorithm implemented in Geant4's `AccurateAdvance`. The key routine behind this strategy is `GQLink_advance_constrained`, which is exposed by the GQLink API with the same signature as `AccurateAdvance`: given a start time t and a distance d, the goal is to compute the position of the particle after taking a step of length d, constrained to the current substep block. This routine might be called several times as `EstimateIntersectionPoint` is executed.Figure 3 summarizes the step computation flow, focusing on the co-simulation interactions.

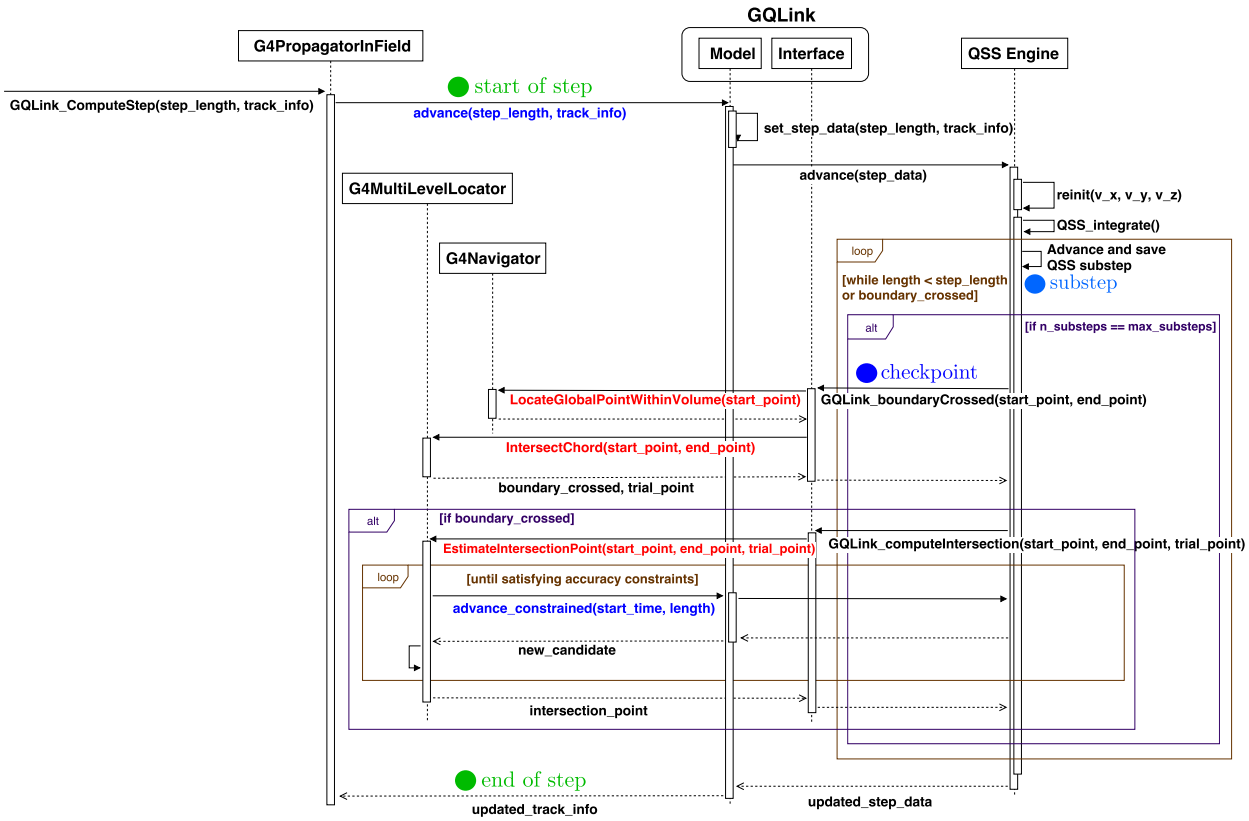


Figure 3: Simulation stage in GQLink with step endpoint, substep and checkpoint markers.

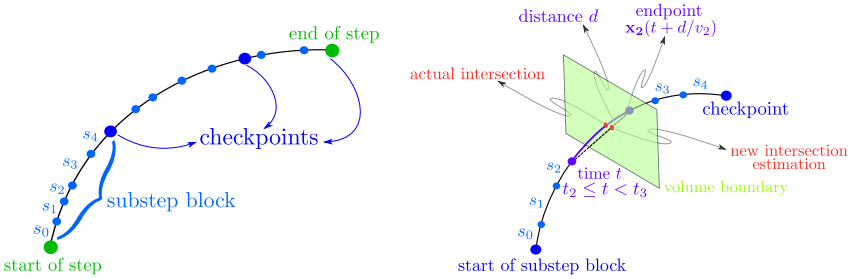
After the step is taken, the updated step data object is used by GQLink to update Geant4's particle tracking data structures. This is the last action in GQLink's stepping algorithm before handling control back to Geant4.

3.2 Substep Packing

The substep packing strategy in GQLink allows for an efficient computation of intersection points by means of the dense output feature of QSS methods. Moreover, it also decreases the overhead introduced by subsequent calls of Geant4 geometry routines since there is no need to do any checks before assembling a complete substep block. Despite this, the larger the number n of substeps packed, the longer the running time of the GQLink routines called by Geant4 during an intersection computation –indeed, the asymptotic running time of `GQLink_advance_constrained` is $O(n)$. In consequence, there is a trade-off between the overhead of the Geant4 geometry routines and that of GQLink routines. Empirically, we found that a value of n below 5 and above 2 provides a good balance between both.

Figure 4a illustrates how a GQLink step is sliced into several substep blocks, each one of them finishing in a checkpoint where Geant4 will be queried for finding possible boundary crossings. For each $i = 0, \ldots, n-1$, the *i-th substep* s_i in the current substep block is the tuple $\langle t_i, v_i, l_i, \mathbf{x}_i \rangle$, where t_i is the start time of the substep, v_i the velocity of the particle along it, l_i the traversed length upon completing s_i and $\mathbf{x}_i : [t_i, t_{i+1}) \to \mathbb{R}^3$ the vector function that approximates the particle trajectory along the substep by means of the polynomial functions provided by QSS. Clearly, we have $t_0 < t_1 < \cdots < t_{n-1}$ and $l_0 < l_1 < \cdots < l_{n-1}$.

When Geant4 detects a boundary crossing, it successively refines the intersection point estimation through an iterative algorithm that starts, in the GQLink setup, with the endpoints of the substep block and an initial estimation given by the linear segment joining them. In each iteration, `EstimateIntersectionPoint` moves the particle a distance d computed as a fraction of the curve length between the endpoints. Then, it determines the volume in which this new endpoint lies. If e.g. it lies in the new volume, a new intersection estimation is computed by means of the linear segment joining this endpoint and the one used as starting point. This process is repeated until the intersection accuracy constraints are met. When GQLink is enabled, each particle movement is efficiently computed by `GQLink_advance_constrained` using the substep information, as depicted in Figure 4b. First, given the start time t of the particle, every substep s_j such that $t \geq t_{j+1}$ has to be skipped. Let s_i be the first substep such that $t < t_{i+1}$. We now have to determine whether the distance d can be completely covered inside s_i. If so, the endpoint is thus given by $\mathbf{x}_i(t + d/v_i)$. Otherwise, we need to keep moving through substeps $s_{i+1}, s_{i+2}, \ldots$, checking their lengths $l_{i+1}, l_{i+2}, \ldots$ and stopping once we find the final substep s_k that can cover length d. The endpoint is computed as $\mathbf{x}_k(t_k + d'/v_k)$, where d' is the remaining distance after subtracting $(t_{i+1} - t) \cdot v_i, l_{i+1}, \ldots, l_{k-1}$ from d.



(a) Substep blocks and checkpoints (b) Example of *GQLink_advance_constrained*

Figure 4: Sketch of the substep packing strategy with $n = 5$.

4 SIMULATION EXPERIMENTS

In this Section we shall present a performance comparison between GQLink (with QSS2 as the chosen integrator) and Geant4 (with its default fourth-order Runge-Kutta stepper, RK4) in two different test scenarios: an extension of the simplified case study described in Section 2.3 using a 3D cube mesh geometry (Figure 6a), and the realistic HEP setup consisting in the CMS detector application (Figure 6b).

All simulations were run on a dedicated Intel Core i7-6700K CPU (clocked at 4.00 GHz) machine with 16 GB of RAM and Ubuntu 16.04.1 LTS x86_64 (`4.13.0-36-generic` kernel) OS. Both Geant4 versions (i.e., with and without GQLink enabled) were compiled with `gcc` 5.4.0 (Ubuntu 5.4.0-6ubuntu1~16.04.9) in release mode (i.e., with optimization flags turned on). We used Geant4 version 10.03.p03 and the QSS Solver engine from version 3.0 for GQLink. Each simulation was single-threaded.

4.1 Case Studies

The model in use throughout the whole experimentation is given by the usual equations of motion of charged particles in a magnetic field (Lorentz equations). Figure 5 shows the respective ODE system along with a snippet of the μ-Modelica model used in GQLink. There, q and m are the charge and mass of the particle, respectively; c is the speed of light; γ is the Lorentz factor and B is the magnetic field.

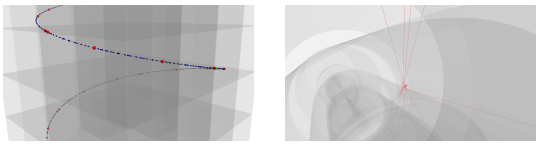
$$\begin{cases} \dot{x} = v_x & \dot{v_x} = \frac{qc^2}{m\gamma} \cdot (v_y B_z - v_z B_y) \\ \dot{y} = v_y & \dot{v_y} = \frac{qc^2}{m\gamma} \cdot (v_z B_x - v_x B_z) \\ \dot{z} = v_z & \dot{v_z} = \frac{qc^2}{m\gamma} \cdot (v_x B_y - v_y B_x) \end{cases}$$

```
model UsualEq
...
parameter Real invMGamma = 1 / (m * gamma);
parameter Real coeff     = q*c*c * invMGamma;
...
equation
(Bx,By,Bz) = GQLink_GetB(x,y,z);
der(x) = vx;   der(vx) = coeff * (Bz*vy - By*vz);
der(y) = vy;   der(vy) = coeff * (Bx*vz - Bz*vx);
der(z) = vz;   der(vz) = coeff * (By*vx - Bx*vy);
...
end UsualEq;
```

Figure 5: Lorentz equations of motion (left) and their μ-Modelica implementation (right).

4.1.1 Perfect 3D Helical Motion In A Cubic Lattice: A Simple Synthetic Baseline

Scenario description This scenario consists in a single electron under a uniform, static magnetic field along the $\hat{z}$ plane, i.e., $\vec{B} = (0,0,B) = B\hat{z}$, where $B = 1$ Tesla, and initial velocity $\vec{v} = (0.9797958971132712\,v\hat{x}, 0, 0.2\,v\hat{z})$, where $v = 0.999\,c$ and $c = 299.792458$ mm/ns. The particle then follows a helical trajectory where its position in $\hat{z}$ is increasing and linear with respect to the time. We stress that this is a tracking-only scenario where physics interactions were intentionally turned off. As for the geometry used, we designed a 3D cube mesh with a user-controlled size for each cube in order to simplify the experimentation for an increasing number of boundary crossings. Clearly, as the cube size decreases, the number of boundary crossings along the trajectory of the particle increases. This extends the scenario studied in our previous works as we now might have boundary crossings in each of the three dimensions and in different angles. Figure 6a shows a 3D mesh of cubes with edge size of 20 mm and a GQLink simulation for a particle trajectory. Green dots indicate the start of a new step, blue dots represent new QSS substeps and red dots show the intersection points where the particle enters into a new cube.



(a) 3D helical motion (b) CMS application

Figure 6: Examples of GQLink simulations for each test scenario.

Verification Due to the simplicity of this model, we have a closed form analytic solution which facilitates the error analysis and in turn provides a direct test of the algorithmic correctness of the communication between Geant4 and QSS. We ensured that the errors in the particle position (i.e., the difference between the simulated and the analytic position of the particle in each of the three axes) produced by both simulators are comparable. For each cube size, the errors in x and y were between 0.3 and 0.31 mm for Geant4 and between 0.38 and 0.39 mm for GQLink. As for the error in z, it was between 0.0123 and 0.0127 mm for Geant4 and between 0.0049 and 0.007 mm for GQLink. Relative accuracy (ε parameter for Geant4; ΔQRel for GQLink) was set to 10^{-6} (with $\Delta\text{QMin} = 10^{-5}$ mm in the case of GQLink), and we used default values for the other Geant4 accuracy parameters. The substep block size was set to $n = 3$ in GQLink. We defined a track length of 100 m for every simulation.

Performance comparison Figure 7a shows the *speedup* of GQLink with respect to Geant4 as a function of the cube size (or, alternatively, as shown in the top x axis, as a function of the total number of boundaries crossed). Each point in the graph shows the mean value of 20 equivalent simulations (with error bars for standard deviation). We distinguish the case where both simulators achieve the same theoretical simulation time and label it *ratio 1*. Ratio 1 is achieved when the mesh is made up of cubes with a 0.5 mm edge, corresponding to a total of $\sim$289000 boundary crossings. From this point onwards, GQLink offers exponentially increasing speedups, reaching a value of nearly 200% when the cube edge length is 0.05

mm. In other words, GQLink simulates about three times faster than Geant4. This experiment provides further evidences that reinforce a conclusion reached in previous contributions, but with simpler geometries: frequent discontinuities favor the performance of GQLink (more precisely, of QSS integrators).

We continue with the analysis in Figure 7b, where we can see the average CPU time consumed by `EstimateIntersectionPoint` per boundary crossing as a function of the cube size (i.e. number of boundary crossings). We note that this CPU time is not only essentially constant for every cube size in the case of GQLink but also, and more remarkably, between 6x and 8x lower than Geant4's. This follows from the substep packing strategy and its associated efficiency improvement when replacing the Geant4 adaptive-stepping algorithm implemented in `AccurateAdvance` with `GQLink_advance_constrained`. Being the substep block size constant throughout the experiments, it is reasonable to expect stable computing times for this piece of code. Surprisingly, on the other hand, the Geant4 curve exhibits a decreasing tendency. This is possibly due to the fact that, the smaller the cube size, the smaller the distances d that `AccurateAdvance` has to cover each time it is invoked. Thus, the number of iterations it must perform to fulfill those lengths tend to decrease, resulting in smaller computing times. Although the average CPU time per boundary crossing is at least 6x smaller in GQLink, the overall simulation time is not necessarily smaller, as it is clearly shown in Figure 7a below the ratio 1 threshold. This can be explained by Figure 7c, which shows the average number of QSS subteps per step (dashed blue line, right y axis) along with the average stepping time (i.e., the CPU time spent in stepping routines, not including boundary crossing procedures) for Geant4 and GQLink (red and blue solid lines, respectively; left y axis).

QSS substeps decrease systematically as the cube size decreases. This is due to the fact that steps tend to be shorter in scenarios with smaller cubes, since boundary crossings will occur more often, early interrupting the step computation. We can also see that GQLink's stepping time decreases following a tendency very similar to that of the substeps. This is an expected behavior, as few substeps demand less CPU instructions. On the other hand, Geant4's stepping time is essentially constant, which can be explained by the fact that, in general, only one iteration of the `ComputeStep` main loop is enough to cover the steps in these scenarios. The ratio 1 threshold is crossed around a cube edge of 0.5 mm. There, the combination of number of boundary crossings ($\sim$289000) with average number of QSS substeps ($\sim$22) start to favor GQLink, outperforming Geant4. Another interesting remark is that, when the number of substeps is small enough (< 15), GQLink's stepping time is smaller than Geant4's, allowing for eventual speedups in other scenarios with lighter boundary crossing activity on which particle trajectories can be approximated by QSS using a small number of substeps.

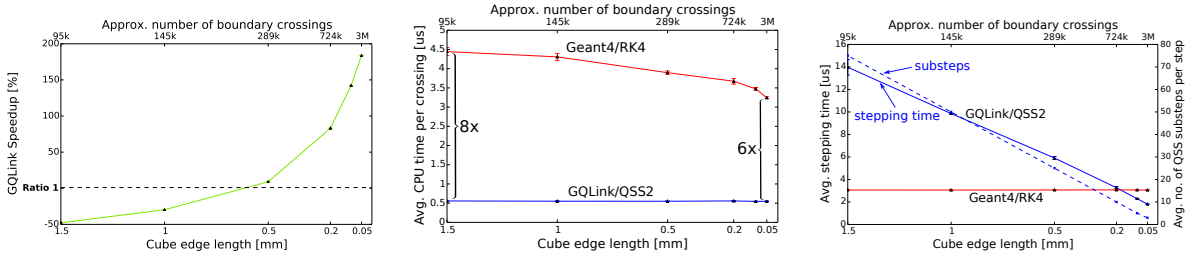
4.1.2 The Compact Muon Solenoid: A Real-World Complex System (A Very Tough Nut To Crack)

Scenario description This test scenario models the Compact Muon Solenoid (CMS) experiment, one of the main particle detectors on the Large Hadron Collider at CERN (https://cms.cern/detector). A standalone Geant4 application models the full CMS (Run1) geometry, volumebase magnetic field excerpted from CMS Offline Software (https://github.com/cms-sw/cmssw) and a particle gun shooting one π^- particle per event with a kinetic energy of 10 GeV. The particle gun injects a particle into the detector with user-configurable parameters (such as kinetic energy or momentum direction). In our experiments the direction of the primary particle is chosen randomly within the η-ϕ space with pseudorapidity $\eta \in [-1/2, 1/2]$ and azimuthal angle $\phi \in [-\pi, \pi]$. We used the reference physics list `FTFP_BERT` recommended in (Dotti et al. 2011). Although a single particle is initially shot per event, more than 62000 secondary particles per event are successively generated as they interact with matter across the detector (see Figure 6b showing a zoomed-in GQLink simulation for CMS, secondary tracks in red).

Verification Given that a closed-form analytic solution is no longer available, we resort instead to verify the accuracy of GQLink simulations in terms of their statistical consistency against Geant4 equivalents. This is achieved through the two-sample Kolmogorov-Smirnov test using a significance level $\alpha = 0.01$. We successfully tested the number of steps and tracks produced for all particles tracked in the application (negative and positive pions, electrons, positrons, gamma photons and a single type for every other particle).

We conducted a parameter sweeping for QSS accuracy in order to find a ΔQRel value good enough to keep the statistical consistency of the simulations while keeping the computing times as low as possible. Figure 8b shows means and standard deviations for simulation times in 30 independent simulations per experiment. We found that $\Delta\text{QRel} = 7.1125 \times 10^{-4}$ provides such a balance (with $\Delta\text{QMin} = 10^{-3} \cdot \Delta\text{QRel}$ mm). For Geant4 we used $\varepsilon = 1 \times 10^{-5}$ and default values for all remaining accuracy parameters.

Performance comparison Figure 8a shows the relative speedups of GQLink and Geant4 for 100 independent runs of CMS, each consisting of 2000 particle gun events. Each run was configured using the same set of accuracy parameters and seeds for the random number generators for both Geant4 and GQLink. A total of 70 runs resulted favorable for GQLink, whereas the remaining 30 were favorable for Geant4. The dashed light brown line shows an overall average speedup of 0.61% in favor of GQLink, with a maximum of 3.5% (leftmost blue bar) and an average of 1.23% (dashed blue line), against a maximum of 3.2% (leftmost red bar) and an average of 0.83% (dashed red line) for Geant4. The CMS application presents a small proportion of boundary crossings, i.e., about 8% of the total number of steps. This is not enough to leverage the substep packing improvements discussed above. Yet, the favorable speedups can be explained by the fact that particle trajectories in these simulations tend to be computed with a small average number of substeps per step (about 1.4). Therefore, we fall in the situation discussed in Section 4.1.1 supported by Figure 7c. These results show considerable improvements over the proof-of-concept GQLink discussed in Section 2.3, derived from low-level optimizations (e.g. branch mispredictions, cache misses) boosting the overall performance of GQLink. We conclude that currently GQLink achieves a performance comparable to the default Geant4 integrator in a realistic benchmark such as CMS, with a slight bias favoring GQLink (supported by the $^{70}/_{30}$ ratio of positive speedups). We assess this as a promising result considering that this application is a tough one for GQLink (small proportion of boundary crossings).



(a) GQLink speedup vs. cube size/boundary crossings

(b) Average CPU time consumed by *EstimateIntersectionPoint* vs. cube size/boundary crossings

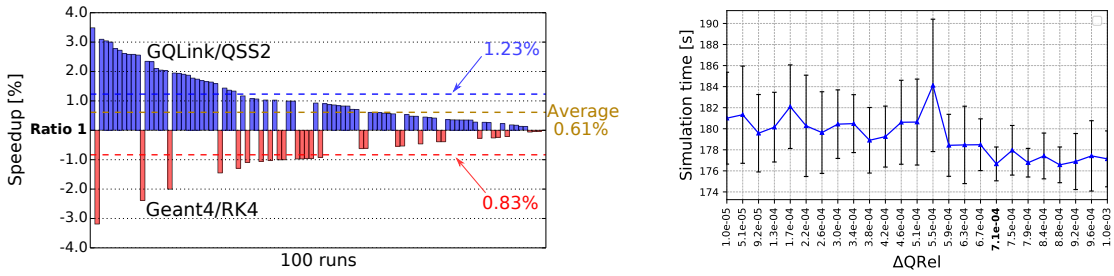
(c) Average stepping time (left *y* axis, solid lines) and average QSS substeps per step (right *y* axis, dashed line) vs. cube size/boundary crossings

Figure 7: Performance comparison for the 3D helical motion setup.

5 CONCLUDING REMARKS AND NEXT STEPS

We presented a co-simulation scheme for applying the Quantized State System (QSS) numerical methods to the problem of particle tracking in high-energy physics (HEP) experiments. The main hypothesis is that the efficiency of QSS for handling discontinuities in ODE systems can be leveraged to obtain speedups and better error control. We relied upon Geant4, the most widely used simulator for HEP. We then produced an optimized version of the QSS Solver simulator, and designed a co-simulation strategy that combines both toolkits while clearly preserving the independence of their core engines. Co-simulation mechanisms were provided and justified after separation-of-concerns principles in software engineering. A novel QSS substep packing strategy proved essential to leverage the QSS polynomial dense output, tackling the problem of finding intersection points within bounded accuracy constraints. Results showed significant speedups up to 3x in a synthetic representative setup (a single electron describing a 3D helical trajectory within a lattice of cubes). Substep packing reduced the average CPU time spent in intersection-finding routines in the order

of 6x to 8x. For a complex realistic HEP case study we simulated a benchmark physics experiment on the Compact Muon Solenoid (CMS) detector at CERN. CMS proved a tough scenario for GQLink, as it features a very small proportion of volume crossings. Low-level software optimizations were decisive in achieving a competitive performance with respect to the standard Geant4 Runge-Kutta-based integrators. A moderate positive speedup bias (∼0.6% on average, ∼3% best case) favors GQLink. We interpret this as a promising result: GQLink performs just slightly better (on average) in a realistic tough scenario of lightweight discontinuity activity, while it offers large potential gains in cases of very frequent boundary crossings. Yet, co-simulation adds a new software layer, imposing its own overhead which should be always minimized (a side effect to pay for the sake of scalability and modularity). Our next steps include testing other realistic HEP applications such as the ATLAS detector at CERN, comparing performance against new default solvers available in Geant4, and experimenting with QSS methods implemented natively within Geant4. Our work in progress includes devising methods to select automatically optimal accuracy and subteps block sizes, and implementing geometry control within the standalone QSS Solver to perform autonomous QSS tracking within 3D spaces with faceted polyhedrons.



(a) Performance comparison for 100 runs

(b) ΔQRel parameter sweeping for QSS

Figure 8: CMS application analysis.

ACKNOWLEDGMENTS

The authors thank S. Y. Jun, K. Genser and D. Elvira (Fermi National Accelerator Lab.) for their support with HEP applications, and Federico Lois (Corvalius) for his support with low-level algorithmic optimization.

REFERENCES

Allison, J., K. Amako, J. Apostolakis, P. Arce, M. Asai, T. Aso, E. Bagli, A. Bagulya, S. Banerjee, G. Barrand, and S. Guatelli. 2016. "Recent Developments In Geant4". *Nuclear Instruments and Methods In Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 835:186–225.

Bergero, F., J. Fernández, E. Kofman, and M. Portapila. 2016. "Time Discretization Versus State Quantization In The Simulation of A One-Dimensional Advection-Diffusion-Reaction Equation". *Simulation* 92 (1): 47–61.

Bergero, F., X. Floros, J. Fernández, E. Kofman, and F. E. Cellier. 2012. "Simulating Modelica Models With A Stand–Alone Quantized State Systems Solver". In *Proceedings of The 9th International Modelica Conference*. September 3Rd-5Th, Munich, Germany, 237-246.

Blochwitz, T., M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauß, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J. Peetz, and S. Wolf. 2011. "The Functional Mockup Interface For Tool Independent Exchange of Simulation Models". In *Proceedings of The 8th International Modelica Conference*. March 20Th-22Nd, Dresden, Germany, 105-114.

Butcher, J. C. 1987. *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*. 1 ed. New York: Wiley-Interscience.

Cellier, F. E., and E. Kofman. 2006. *Continuous System Simulation*. 1 ed. Berlin: Springer-Verlag.

De Win, B., F. Piessens, W. Joosen, and T. Verhanneman. 2002. "On The Importance of The Separation-of-Concerns Principle In Secure Software Engineering". In *Workshop For Application of Engineering Principles To System Security Design (Waepssd) Proceedings*. November 6Th-8Th, Boston, Usa, 1-10.

Dotti, A., J. Apostolakis, G. Folger, V. Grichine, V. Ivanchenko, M. Kosov, A. Ribon, V. Uzhinsky, and D. Wright. 2011. "Recent Improvements On The Description of Hadronic Interactions In Geant4". *Journal of Physics: Conference Series* 293 (1): 012022.

Elvira, D. 2017. "Impact of Detector Simulation In Particle Physics Collider Experiments". *Physics Reports* 695:1–54.

Fernández, J., and E. Kofman. 2014. "A Stand-Alone Quantized State System Solver For Continuous System Simulation". *Simulation* 90 (7): 782–799.

Fonseca, P. 2013. "Co-Simulation Using Specification and Description Language". In *Proceedings of The 2013 Winter Simulation Conference*, 4022–4023, edited by R. Pasupathy et al., Piscataway, New Jersey. IEEE.

Fritzson, P. 2015. *Principles of Object-Oriented Modeling and Simulation With Modelica 3.3: A Cyber-Physical Approach*. 2 ed. Hoboken: Wiley.

Kofman, E. 2004. "Discrete Event Simulation of Hybrid Systems". *Siam Journal On Scientific Computing* 25 (5): 1771–1797.

Kofman, E., and S. Junco. 2001. "Quantized State Systems. A Devs Approach For Continuous System Simulation". *Transactions of Scs* 18 (3): 123–132.

Moallemi, M., G. Wainer, F. Bergero, and R. Castro. 2011. "Component-Oriented Interoperation of Real-Time Devs Engines". In *Proceedings of The 44th Annual Simulation Symposium*. April 4Th-9Th, Boston, Usa, 127-134.

Munawar, A., T. Yoshizawa, T. ishikawa, and S. Shimizu. 2013. "On-Time Data Exchange In Fully-Parallelized Co-Simulation With Conservative Synchronization". In *In Proceedings of The 2013 Winter Simulation Conference*, 2127–2138, edited by R. Pasupathy et al., Piscataway, New Jersey:IEEE.

Santi, L., F. Bergero, S. Y. Jun, K. Genser, D. Elvira, and R. Castro. 2018. "Gqlink: An Implementation of Quantized State Systems (Qss) Methods In Geant4". *Journal of Physics: Conference Series*. [In Press].

Santi, L., N. Ponieman, S. Y. Jun, K. Genser, D. Elvira, and R. Castro. 2017. "Application of State Quantization-Based Methods In Hep Particle Transport Simulation". *Journal of Physics: Conference Series* 898 (4): 042049.

Wirth, N., P. Bayrasy, B. Landvogt, K. Wolf, F. Cecutti, and T. Lewandowski. 2017. *Analysis and Optimization of Flow Around Flexible Wings and Blades Using The Standard Co-Simulation Interface Mpcci*. 1 ed., 283–321. Cham: Springer International Publishing.

AUTHOR BIOGRAPHIES

LUCIO SANTI is a PhD student in the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. His e-mail address is lsanti@dc.uba.ar.

RODRIGO CASTRO is a Professor in the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, head of the Simulation Lab, and a researcher at CONICET. His research interests include simulation and control of hybrid systems. His email address is rcastro@dc.uba.ar.