Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

SYSTEMATIC PERFORMANCE OPTIMIZATION FOR THE POWERDEVS SIMULATOR AND MODELS OF LARGE-SCALE REAL-WORLD APPLICATIONS

Ezequiel Pecker-Marcosig[1], Gerónimo Romczyk[2], Matías Bonaventura[1,3] and Rodrigo Castro[1,2]

[1]Instituto UBA-CONICET de Ciencias de la Computación, Buenos Aires, ARGENTINA
[2]Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Univ. de Buenos Aires, ARGENTINA
[3]CERN, Geneva, SWITZERLAND

ABSTRACT

As simulation models grow in size and complexity, their performance comes under increasing pressure. In order to deliver results within an acceptable timeframe, new mechanisms are required that reduce the simulation performance burden while avoiding simplifications of model behavior. We present a systematic methodology for optimizing the PowerDEVS toolkit, improving both the core simulation engine and general-purpose behavioural models in a way that they stand as independent yet compatible tasks, thanks to the strict separation between simulator and models promoted by the DEVS formalism. We tested our optimizations on a variety of models that emphasize different effects. The main target system of this work is the large-scale network model of the Data Acquisition System (DAQ) at CERN, which is used to derive hardware requirements for future accelerator upgrades. Speedups for the DAQ model more than doubles the number of experiments that can be run within the same amount of time.

1 INTRODUCTION

The steady emergence of new applications in science and engineering demands efficient design techniques to shorten system development cycles and analysis. Experimentation by model simulation is often the only viable alternative for studying increasingly large and complex systems, where typical analytical methods or heuristic approaches are not an option.

Yet, as models grow in size and complexity, simulation performance can become a critical stumbling block, rendering an entire simulation-based effort infeasible, thus undermining the very motivations for which it was chosen in the first place.

Therefore, the discipline of simulation performance optimization stands as a key element of numerical experimentation. Numerous approaches have been developed to improve the performance of simulation models based on varied techniques, such as improvements in code parallelization, data structure handling, compilation techniques, use of special hardware structures, and even refactoring of the model logic.

However, the reusability and potential impact of these approaches beyond specific systems are often rather low. The efforts that focus on specificities of a particular model are naturally limited in scope. On the other hand, when performance improvements are applied to generic-purpose simulation algorithms, it cannot be guaranteed that the changes introduced will be beneficial for all types of models compatible with such algorithms.

This is where those simulators adhering to formalisms that strictly separate modelling from simulation can offer unique advantages. Improvements introduced to a particular model do not pollute the performance of the simulation engine when applied to other models. In turn, improvements made to the simulation engine should impact all models compatible with such engine (with varying model-dependent intensities).

In this paper, we will focus on a simulator of this class. PowerDEVS (Bergero and Kofman 2011) is a tool with more than 12 years of evolution featuring an extensive library of models (and their related publications) covering disciplines from engineering to biology, from communication networks to social

behavior, to name a few. In turn, it adheres to the Discrete-EVent System specification (DEVS) formalism (Zeigler et al. 2018), known for its generality by allowing the representation of any kind of discrete system and combining them with precision-controlled approximations of continuous systems, permitting both deterministic and stochastic dynamics. In turn, DEVS has proven to be a common denominator of a variety of other mathematical formalisms that have an equivalent representation under DEVS (Vangheluwe 2000).

Due to the variety and large number of pre-existing models developed using component libraries in PowerDEVS, any performance improvement to be made in its engine should be accompanied by a robust and repeatable methodology, that allows measuring the impacts in a potentially large set of models, explaining the reasons behind the improvements within the architecture of the modified software.

1.1 Target Case Study: Large-Scale Data Acquisition Cluster at CERN

A salient application where performance is a critical issue is the Data Acquisition (DAQ) network simulation model used in the ATLAS experiment at CERN (Bonaventura et al. 2016). This is a real-world large-scale case study implemented in PowerDEVS that represents a distributed system of compute nodes interconnected by a complex data-network (involving more than 8700 DEVS submodels). Simulation execution performance plays a crucial role to provide experts and decision-makers with coherent results in a timely manner, ensuring that conclusions about DAQ performance obtained from simulation results can be drawn within reasonable time frames.

Based on this model, a recent M&S study estimated network hardware requirements providing relevant insights for the commissioning of a future system upgrade (Pozo Astigarraga et al. 2023). Yet, the simulated model represents only $\sim 10\,\%$ of the full system, due to prohibitively long execution times. Although acceptable for that particular study, such long execution times hinder the usability of the model for other system-wide studies, as experts must wait for days to draw conclusions. In addition, as the size of the system is expected to grow in the coming years, simulation execution times are likely to increase even further.

Different approaches are typically followed to improve performance of DEVS simulations. The fact that DEVS provides a strict separation between models and the simulation engine allows for making independent improvements at both the DEVS simulator (strictly speaking the *abstract simulator*) and the models.

There exist multiple strategies for a DEVS simulator to execute models. In all cases, the simulation of a discrete-event model involves the execution of a time-sorted sequence of events. Zeigler (2017) proposed the parallel processing of simultaneous events for DEVS models, leading to reductions of simulation times as shown later in (Lanuza et al. 2020; Trabes et al. 2023). Despite being a valid approach, the parallelization does not necessarily improve performance in all models, since synchronization contention might introduce relevant penalties when no good model partitioning is possible. This is indeed the case in our motivating data-network case study.

Since PowerDEVS features Classic DEVS, events are processed sequentially and therefore an efficient handling of the events list becomes a key issue performance-wise. In this work we study the efficiency of different techniques for dealing with event ordering and for handling data structures in the simulation engine.

A DEVS model is typically a hierarchical composition of submodels that exchange data. The simulation process involves internal messaging needed by the *abstract simulator*. In large-scale models organized in several hierarchical levels, the amount of internal messages exchanged can grow considerably, contributing to enlarge the execution times. A typical way to deal with this issue is to simplify model structure, placing all behavioral submodels at the same level in the structure, thus reducing multi-level message handling overheads. This procedure is known as *global model flattening*. In this work we explore this approach and its effects on performance.

In some cases where there are multiple instances of identical submodels (that may differ only in their parameters), an extension of the DEVS formalism called VECDEVS (Bergero and Kofman 2014) can be

used. It helps to reduce the model's structure by replacing these multiple instances with a single *vectorial model* with vectorial inputs and outputs, obtaining a type of *local flattening*. However, this is not always applicable given the variety of pre-existing model libraries in PowerDEVS involving multiple different submodels.

For running a performance analysis of the changes introduced in PowerDEVS we had to develop a testing workbench. DEVStone (Glinsky and Wainer 2005) is a *de-facto* repeatable benchmarking method for DEVS simulators. This benchmark has been used to evaluate the performance improvements in different DEVS simulators (Glinsky and Wainer 2005; Lanuza et al. 2020; Cárdenas et al. 2022). DEVStone was recently revisited and extended in (Risco-Martín et al. 2017), where it was also used to run a performance comparison between five different popular DEVS simulators.

In this paper we present a number of performance improvements introduced in the core engine of PowerDEVS and in selected general-purpose atomic models, together with a systematic and repeatable benchmark method.

The resulting *relative speedups* (the difference between the new and previous execution times, relative to the latter) range from 2 % to more than 60 %, and in particular for the real-world DAQ network model the performance improved nearly 55 % (more than doubling the number of experiments that can be performed in the same amount of time).

The article is structured as follows: Section 2 provides the background, Section 3 details the changes made to PowerDEVS while in Section 4 these changes are validated against a battery of simulation models involving different selected application domains and also the synthetic DEVStone benchmark. The evaluation and discussion of the speedups for the data-network model used at CERN is presented in Section 5. Finally, Section 6 concludes the paper.

2 PRELIMINARY CONCEPTS

2.1 The DEVS Formalism and its Abstract Simulator

The Discrete-EVent Systems specification (DEVS) formalism is the most general mathematical formalism for M&S of discrete-event systems (Zeigler et al. 2018). Any discrete-event system, and in particular any discrete-time system, can be represented under this formalism (as long as it undergoes a finite number of events in a finite interval of time). Continuous-time systems can be approximated in DEVS with any degree of desired accuracy.

In DEVS there is a strict separation between a DEVS model and the simulation engine used to run the model. Since the engine is common for any model, every optimisation made on the first one will have an impact on the runtimes of all new and previously developed models without any additional effort.

DEVS defines a system's model as a hierarchical and modular composition of submodels, which can be behavioral (Atomic) models or structural (Coupled) models. This feature fosters model reusability and enables that every improvement in simpler models improves the performance of the models built on them.

An **Atomic DEVS model** is formally defined as $M_A = \langle X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta \rangle$, where X and Y represent the sets of input and output events, and S is the set of internal state values. The model's behavior is defined by four dynamic functions: the *external transition function* $\delta_{\text{ext}}()$ (for reactive behavior upon receiving events), the *internal transition function* $\delta_{\text{int}}()$ (for autonomous timed behavior), the *output function* $\lambda()$ (for emitting events) and the *time advance function* $ta()$ (for defining the lifetime of internal states in the absence of external events). The internal state of a model remains hidden to other models (*information hiding*), and the only way to know or modify a model state is by exchanging messages (input/output events).

A **Coupled DEVS model** defines a network of interacting DEVS models (Atomic or Coupled) arranged in a hierarchical/modular structure that serves as a message passing networked structure. A Classic DEVS *Coupled* model is formally described as: $M_C = \{X, Y, D, EIC, EOC, IC, Select\}$, where: X and Y are sets of input and output events, respectively, D is the set of components names, IC is the set of internal couplings among members of D, EIC is the external inputs coupling relation (links between external input ports and

internal components) and EOC is the external output coupling relation (links between internal components and external output ports). $Select : 2^D \to D$ is a tie–breaking function for simultaneous events.

DEVS features a generalized **Abstract Simulator** algorithm, with two main classes of objects: *simulators* and *coordinators*. The execution of each atomic model is controlled by its *simulator*, while *coordinators* manage the coordination of coupled models and are in charge of synchronizing their children simulators and coordinators. Hence, simulators and coordinators mimic the hierarchical structure of the atomic and coupled components present in the model. At the top of this hierarchy, an overseeing *root-coordinator* is in charge of triggering new simulation cycles and handling the advancement of the global simulation time. The synchronization between simulators and coordinators in the abstract simulator is performed by an internal message passing protocol.

Each coordinator in the abstract simulator maintains a sorted list of future scheduled events for its child models. The events in the list are popped out and executed sequentially, producing state transitions in the atomic models and possibly new events, which in turn are inserted into the list according to their timestamps.

2.2 The PowerDEVS Toolkit

The PowerDEVS toolkit (Bergero and Kofman 2011) is the flagship DEVS simulator for the Quantized State-Systems (QSS) family of numerical solvers (Castro et al. 2024), making it suitable for the simulation of hybrid dynamical systems. A distinctive advantage of PowerDEVS is its performance when compared to other DEVS simulators (which makes it advisable for applications where simulation performance is a key aspect) and has been recommended for modelers who are not familiar with DEVS and for users who want to use DEVS for the simulation of hybrid systems (Van Tendeloo and Vangheluwe 2017).

PowerDEVS features a Graphical User Interface (GUI), which hides away the internal aspects of DEVS models facilitating its use by modelers unfamiliar with DEVS. A model is then built by interconnecting graphical representations of DEVS atomic and coupled models (blocks with input/output ports that are dragged, dropped and wired) including the setting of their model parameters. Alternatively, py2PowerDEVS (Pecker–Marcosig et al. 2022) boosts the capabilities when constructing topologically large models by using Python scripting as an alternative (and a complement) to the PowerDEVS GUI. py2PowerDEVS allows PowerDEVS classes defined in C++ to be accessed directly from Python code, while execution occurs strictly in C++, hence with no performance penalties.

All DEVS atomic models in PowerDEVS are derived classes from the `Simulator` class. According to their behavior, these classes implement their corresponding state definition, the internal and external transition functions, the output function, and the time advance function. The coordinators in PowerDEVS belong to the `Coupling` class. Due to the closure under coupling property this is a derived class from `Simulator`. The class `RootCoupling` derives from `Coupling` and differs in the parent object. According to the definition, a DEVS coupled model contains a list of internal/external connections with atomic and coupled child models, and maintains a sorted list of events for them. In PowerDEVS the list of sorted events in a `Coupling` object is a binary heap called `ChildHeap`, where the time of the event is stored in a C++ vector and the model in a array of pointers to `Simulator`. The internal connections (IC), external input connections (EIC) and external output connections (EOC) in a `Coupling` object are managed as arrays of pointers to `Connection` objects, hence whenever an event arrives or is propagated the corresponding lists must be run through. The `Connection` class is formed by four integer numbers representing the source and sink models and their corresponding ports. The `RootSimulator` class in PowerDEVS is in charge of running the simulation by making the time advance, and then plays the role of the root-coordinator in the DEVS abstract simulator.

3 CLASSIFICATION OF PERFORMANCE OPTIMIZATIONS IN POWERDEVS

This section covers all optimisations performed in PowerDEVS, grouped into those performed on the core simulator and on a selection of general purpose atomic models. Performance bottlenecks were detected using mainly standard profiling tools. A test methodology was defined to ensure that the performance comparison is consistent. All enhancements are finally tested in section 4 with a set of well-selected models.

3.1 Identification of Performance Losses in PowerDEVS

We begin by developing a framework for running PowerDEVS models using a set of industry-standard debugging and profiling tools (`ValGrind` and `CallGrind`). The resulting data, together with the visualisations generated by `KCacheGrind`, are used to produce valuable graphs showing the distribution of resources usage (CPU and Memory) during the simulation. These tools are also valid for analysing DEVS models built with py2PowerDEVS, since Python is only used to build the model structure, while the executable code is entirely C++.

We applied this framework to identify performance bottlenecks and areas for potential improvement. For example, it helped with identifying and fixing several memory leaks. In this article, performance is measured in terms of the end-to-end time required to run a given simulation.

We undertook a systematic analysis of the causes of performance losses and the changes introduced in PowerDEVS to address them. In the next two sections we report on these changes, grouped as changes to the core simulator and to general purpose atomic models.

3.2 Optimizations Performed in the PowerDEVS Core Simulator

Below is a list of changes made to the core simulator to address the performance bottlenecks identified. The speedups achieved for the motivating case study at CERN as a result of these changes are reported in more detail in Section 5.

Event Queues We improved the management of the sorted list of events in the coordinators (`ChildHeap` in PowerDEVS) by replacing a *binary heap* with a *pairing heap*. This type of heap is known to provide better performance for event management, helping to reduce the time needed to find the next event.

Connections Handling We replaced the arrays of pointers used to store the EIC, EOC and IC connections for the coupled models in the `Coupling` objects with ordered dictionaries sorted by model and port. Apart from the performance impact, this helped to simplify the code and make it more readable.

Core Simulator Debugging A further contribution to reducing simulation time has been made by replacing the core simulator debug logging functions with a macro. The mechanism for collecting and displaying such debug information has also been improved.

Model Flattening Finally, we have included the optional automatic flattening of a model. Model flattening helps to simplify the structure of a model by reducing the internal messaging of the simulator. In the flattened model, since there is a single DEVS coupled model and a single coordinator who also plays the role of root coordinator, there is a single event queue in the model. Therefore, the policy used to manage the event queue in a flattened model is expected to have a significant impact. As the model flattening procedure must be available for PowerDEVS models created with both the GUI and py2PowerDEVS, model flattening is performed at runtime during model initialization, contributing to the simulation time (see Section 4.3). Model flattening is enabled by a command line parameter. It should be noted that for VECDEVS models this approach does not lead to any improvement, as they have already been optimized when the structure created by Coupled DEVS models was replaced by VECDEVS Atomic models.

3.3 Optimizations Performed on Selected General-Purpose PowerDEVS Atomic Models

Although the improvements to the atomic models are not as general as those in the core simulator, the compositional construction of DEVS models takes advantage of the changes made to general-purpose atomic models, such as numerical integrators, data loggers and network entities. Their impact on the speedup for the motivating case study at CERN is reported in Section 5.

QSS3 Integrator We replaced the function used by the QSS3 integrator atomic model to find the roots of a third order polynomial. To evaluate this change, we built a model for the motion of a charged particle in a constant magnetic field, described by the Lorentz equations, consisting of six QSS3 integrators. This model was chosen because it has a known analytical solution. We reduced the simulation time by 19 % and slightly reduced the error compared to the analytical solution.

Data-Loggers We introduced changes to the data structures used by the data loggers (classes `DiscreteSamplerLogger` and `SamplerLogger`) to determine specific values, such as maximum, minimum, moving average and so on, from the data generated by the invoking atomic models.

Network atomic models We changed the way selected atomic models in the Network library manage the names of variables (metrics) to log, because it was too time consuming.

3.4 New Core Logging Capabilities and Verification of Correctness of the Improved Models

In order to address the analysis of the performance in a sound way, we endowed PowerDEVS with a new data-logging mechanism for its core simulator. This new logging mechanism is independent of the model logging that runs at the model level. We have instrumented the `Simulators`, `Coupling`, `RootSimulator` and `RootCoupling` classes so that each time a message is sent or an event is propagated, one or two lines are logged in a comma-separated value log file (`pdevsDebug.log`) containing relevant data such as the full name of the object, the type of event, the virtual simulation time and the real execution time. This full name includes the names of all the parents of a given object, separated by dots. The verbosity of the logging can be set with a debug level.

We have developed algorithms to systematically compare debug logs for different runs of a given model. For each line in one of the log files, the corresponding line in the other log is identified by the full name of the object involved. This system is key to validating the model flattening mechanism, ensuring that the different entities in the core simulator exchange equivalent internal messages to the original hierarchical model. By equivalent we mean that for models that do not change their structure, their internal messages should be exactly the same.

Once the optimisations described in the previous sections have been applied and the simulation execution has been successfully accelerated, we need to ensure that the results are correct. This means that both the simulation results (produced by the model) and the internal messages exchanged in the core simulator before and after the introduction of the changes must be equivalent. In the following sections we will use these new logging capabilities and tools to perform the comparisons and verify the correctness of the improvements. Logging was enabled only when comparing the correctness of event traces (before and after improvements), while keeping it disabled when comparing simulation performance (to eliminate the logging overhead).

4 ANALYSIS AND VALIDATION OF PERFORMANCE OPTIMIZATIONS

This section presents a battery of tests designed to measure the performance of PowerDEVS in terms of the time required to run a simulation. The aim is to check if the changes made in Section 3 lead to performance optimisations, and if there is an improvement, to identify which change caused it.

We define a *relative speedup* as:

$$RS = (et_{bc} - et_{ac}) / et_{bc} \cdot 100$$

where *RS* is the relative speedup in percent, *et* is the execution time (measured in real wall clock time), and *bc* and *ac* stand for *before* and *after* introducing the *changes*, respectively. A positive value for *RS* indicates that the simulation performance is improved after the introduction of the changes (detailed in Section 3). A relative speedup of 0 % represents no speedup (equivalent to a speedup factor of 1 time).

We start with a series of experiments on various examples distributed with PowerDEVS, covering a variety of domains and formalisms. The idea behind using these relatively simple examples is to ensure backward compatibility.

We continue with a large-scale example involving the coupling of a large number of SIR (Succeptible-Infected-Recovered) submodels with a high degree of connectivity, introduced in (Pecker–Marcosig et al. 2022). This model helped us to perform a stress test on the benefits of optimisation on both continuous and hybrid models, since each SIR consists of three QSS integrators. We then conclude the validation with a popular benchmark for DEVS simulators called DEVStone (Glinsky and Wainer 2005). Each test is repeated three times and the average execution times are determined.

All these examples cover the two ways to create a model in PowerDEVS, either using the GUI or using a Python script with Py2PowerDEVS. These examples have been chosen to show different aspects of the optimisations introduced in PowerDEVS.

4.1 Performance Optimization for a Series of Models in Different Domains and Formalisms

A performance study has been carried out on more than 20 models covering different areas. The aim of these tests is to assess the impact of the changes on previously developed models released with the PowerDEVS distribution (SEDLab 2024). We also want to analyse whether the impacts of the improvements depend on the model type.

The examples are grouped according to the nature of the models. As for the continuous-time models expressed by sets of ordinary differential equations (ODEs), we considered: the classic prey-predator Lotka-Volterra model (`lotka volterra`), a nonlinear Stiff model (`stiff`) and a lumped model of a lossless transmission line formed by a network of 50 LC (inductor and capacitor) tank circuit models (`transmission line`) built with VECDEVS. Second, we considered three representative examples of models described by sets of Delay Differential Equations (DDEs), including a spiking neural network (`cellular network spikes`) and two other representative examples from the literature (`oberle and pesch` and `hairer et al`). For discrete-time models, we considered a second-order discrete-time oscillator model built with the serial connection of two delays (`q-operator`) and the popular Nicholson-Bailey model for the evolution of two populations (`nicholson bailey`), which is closely related to the Lotka-Volterra model but in discrete time.

For the stochastic models we used a basic network (`network basic`) consisting of: a packet sender, a receiver, a router and a queueing model (`queueing`) with a job generator, a job queue and a task processor. Finally, we considered several examples of hybrid systems: a bouncing ball falling down stairs (`Bouncing Ball`), a DC-DC switching Buck converter (`Buck Controlled Coupled`), a DC motor drive (`DC Drive` and `DC Drive Buck`), a spiking neuron (`Spiking Neuron`), and an inverted pendulum in closed-loop control (`Inverted Pendulum`).

Figure 1 summarizes the relative speedups (y-axis) obtained for all simulation examples (x-axis). These relative speedups are sorted from higher or most optimized (left) to lower or less optimized (right). Each simulation is repeated three times, taking into account a relevant virtual simulation time for each model, and the relative speedups are averaged. They cover a wide range of values, from over 60 % to nearly 0 % (speedup equal to 1), and reaching a negative value for the case of the DEVStone LI model (details are left to the next section). The negative relative speedup for the LI model around -2 % means that the performance is slightly degraded after applying the changes.

We observe that the highest improvement was achieved for the queueing example (built with both flavors: GUI and Py2PowerDEVS), closely followed by DEVStone models and the TDAQ example (examined in Sections 4.3 and 5 respectively).

Finally, it should be noted that for the `network basic` model the performance gain depends on whether the model is built with the GUI or with py2PowerDEVS, and this is because they are not exactly the same.

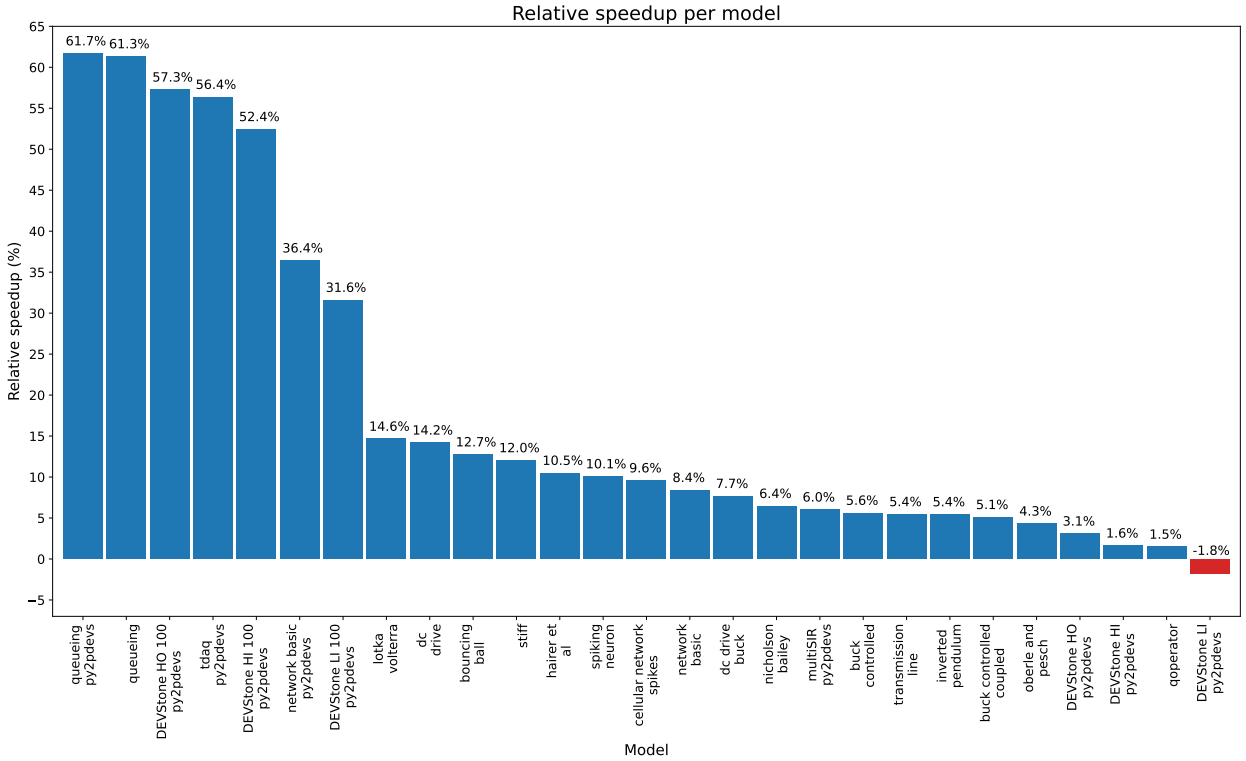


Figure 1: Average relative speedups for a series of DEVS models in PowerDEVS convering a wide range of applications and built with the GUI and py2PowerDEVS.

4.2 Performance Optimization for a Large System of Hybrid Models

In this case we evaluate the performance of a large-scale epidemiological model consisting of a network of 1431 classic SIR (Susceptible-Infected-Recovered) submodels interconnected by 9632 input/output links organized in 2 levels. Each node in the resulting graph corresponds to a different geographical region. The behaviour of the node is modelled with a set of three non-linear ODEs representing the evolution of independent population sets S (Succeptible), I (Infected) and R (Recovered) compartments. The connections between SIR submodels represent infections between cities or towns in terms of the influence of neighboring SIR models (see (Pecker–Marcosig et al. 2022) for details). We choose this model because it is appropriate to evaluate the speed-ups that can be obtained applying the changes presented in Section 3 to a large hybrid model with strong connectivity. Due to its complex structure, this model was built using py2PowerDEVS.

Figure 1 shows that the relative speedup in this case (`multiSIR`) is in the order of 6 %, mainly due to the improved management of event queues and input/output connections, and the changes to the QSS3 integrators.

4.3 Performance Assessment with the Synthetic DEVStone Benchmark

DEVStone (Glinsky and Wainer 2005) defines several *synthetic* DEVS models with different structures, sizes and connectivities. A DEVStone model is a coupled DEVS model containing recursive repetitions of a regular structure, with different number of nested levels. Each level contains a number of coupled

and atomic models with different interconnection patterns. The most inner model in the recursion is the simplest, consisting in a coupled model with a single atomic model. Five parameters customize a DEVStone model: (a) *Type* specifies the model structure, (b) *Width (w)* specifies the number of atomic components per level, (c) *Depth (d)* specifies the number of nested coupled models, (d,e) *Internal/External transition delay* specify the time spent at each internal/external transition function (the Drystone benchmark code is used for spending the required amount of time in the internal/external transition functions). The original version of DEVStone (Glinsky and Wainer 2005) defines three types of models: (a) *LI models* present the simplest structure with a low number of interconnections for each coupled model, (b) *HI models* extend the LI models with a higher level of input couplings, and (c) *HO models* extend the HI models by increasing the number of ports, input and output couplings.

We implemented the LI, HI and HO models from the DEVStone benchmark in PowerDEVS (making use of Py2PowerDEVS for scripted model generation). We built a customizable model involving $w = 30$ models per level arranged in $d = 10$ levels. As a verification for these newly created synthetic models, we checked the analytical expressions for the number of events, the number of executions of the internal and external transition functions in (Risco-Martín et al. 2017).

The simulation results before and after the performace optimizations (including the model flattening) are shown in Figure 1. Comparing the `devstone LI`, `devstone HI` and `devstone HO` for a single input event, we observe that the best optimization was achieved with the HO model, with a relative speedup below 5 %, closely followed by the HI model with a relative speedup of about 2 %.

We also observe that for the LI model the relative speedup is negative. It is worth recalling that the flattening process for a model takes place in runtime during the model initialization phase, so it adds to the total execution time. This effect has a more significant impact on the LI model than on the HI and HO models since the former has the lower degree of connectivity. Figure 2 distinguishes between execution times for the *full run* (i.e. with the initialization phase included) and for a version *without initialization* considered, thus reporting two different relative speedups. We can see that the LI model actually achieved a positive relative speedup (blue bar) when excluding from the analysis the time spent in the model flattening activity during the initialization phase. This same effect is seen for the HI and HO models, going from an average relative speedup of 2 % and 3 % (orange bars) to 7.2 % and almost 10 % (blue bars), respectively.

To reduce the impact of the time spent in the initialization on the total simulation time, we repeated the study but considering 100 input events instead of only one. It can be noted that for longer simulation times, the difference between the relative speedups for the full run (orange) and removing the time spent in the initialization (blue) is almost negligible (see Figure 2).

5 RESULTS FOR THE DATA ACQUISITION SYSTEM MODEL AT THE ATLAS EXPERIMENT IN CERN

To validate our optimization approaches on a large-scale real-world case study, this section shows the performance improvements achieved in the DAQ network simulation model used in the ATLAS experiment at CERN.

In the coming years, the ATLAS experiment's DAQ system will be extensively upgraded to take full advantage of the High-Luminosity LHC (HL-LHC) upgrade (ATLAS Collaboration 2017). This upgrade will allow data to be recorded at unprecedented rates. The detector will be read out at 1 MHz, generating over 5 TB/s of data. This new design poses significant challenges for the Ethernet-based network, which will have to carry 20 times more data than the actual system.

As the desired HL-LHC system hardware is not currently available and the size of the test laboratory is significantly smaller, a packet-level network simulation model was built in PowerDEVS (Pozo Astigarraga et al. 2023) to extrapolate the network hardware requirements to the expected operating point. Figure 3(a) shows the network and hardware configuration of the experimental laboratory.

The tests in the lab comprises 48 Readout (RO) nodes, and 1 rack with 32 Event Filter (EF) servers. The devices under test (DUT), are the top-of-rack (ToR) switches that aggregate the EF nodes, which have

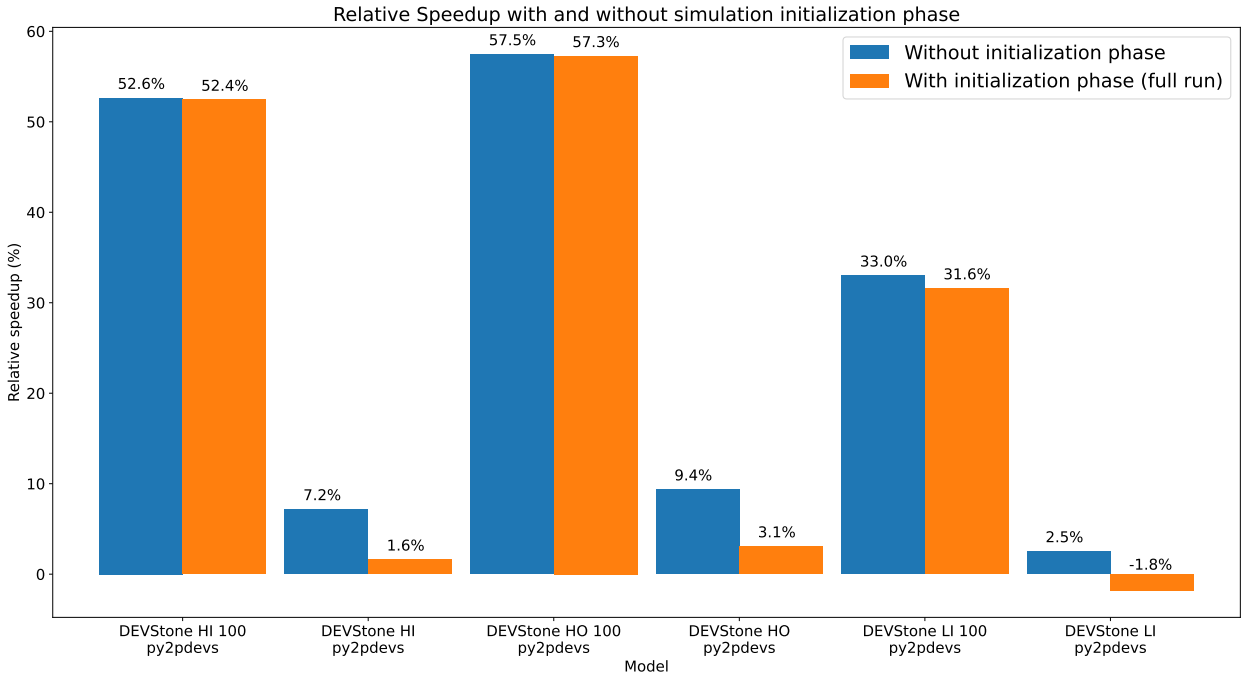


Figure 2: Relative speed-ups for DEVStone models for the full run (orange) and without the initialization stage (blue). The orange bars coincide with Figure 1.

an aggregated uplink capacity of 800 Gbps to the core routers, and an access link capacity of 25 Gbps to each of the 32 EF nodes. The model includes representations for the relevant distributed applications that generate data, network devices and servers.

The model is exercised with different input parameters and executed repeatedly to derive statistically sound metrics. Despite running these models on dedicated high-end servers, the reported simulation times range typically from a few minutes to several hours (for each single simulation), while a typical full parameter sweep takes approximately 48 hs.

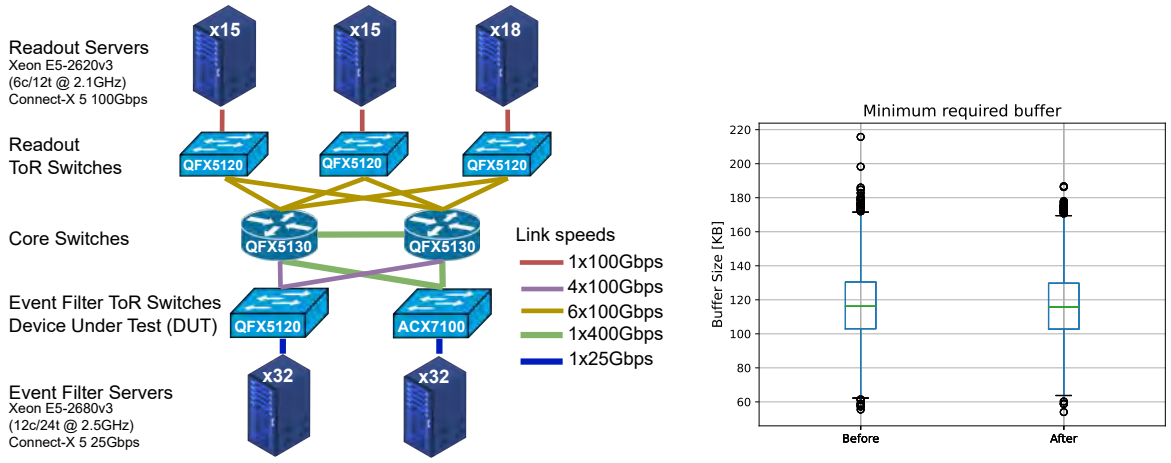
Hence, the reduction of DAQ model simulation times is of utmost relevance to provide timely information to experts, and to enable the evolution towards simulation of larger models within reasonable timeframes.

The DAQ simulation model represents network traffic at the packet level, including protocols, network transfer delays, and OSI layers 3 and above. In this section, due to the long execution times, we considered an event-size of 0.5 MB and an event-rate of 4.882 kHz with the simulation model comprising 79 servers. The model structure consists of 8728 atomic models arranged in 3 levels that generate millions of events per simulation run.

The fourth bar in Figure 1 (from left to right) shows that with the optimizations introduced both in the simulator engine and in certain general-purpose atomic models, an overall relative speedup of nearly 56 % can be observed for the TDAQ example.

The execution time to simulate 5 s of virtual simulation time was on average 524 s before the changes and 229 s with all optimizations included. A detailed analysis using the new logging capabilities ensured that the events before and after the introduction of the changes remained equivalent.

Due to the stochastic nature of this model, the metric used to compare results is the Minimum Buffer Size Required for the switches under test, whose difference remains between acceptable limits (see Figure 3(b)).



(a) (b)

Figure 3: (a) Topology of the ATLAS Data Acquisition model prototype for testing the future HL-LHC collider at CERN. The PowerDEVS model includes 8728 atomic models. (b) Simulation-determined Minimum Buffer Size Required for the EF ToR switches under test (before and after the changes).

6 CONCLUSIONS AND NEXT STEPS

In this work we presented a series of enhancements aimed at optimizing the performance of the PowerDEVS simulation toolkit, measured in terms of reducing simulation execution time. We presented a methodology for detecting potential performance losses and assessing optimization. Thanks to the strict separation between simulation engine and simulation models enforced by DEVS we were able to conduct changes simultaneously and independently to the PowerDEVS simulation engine and selected general-purpose atomic models.

Given the wide range of applications where PowerDEVS has been applied to and the vast list of models developed with it, it is crucial to ensure that their performance is not worsened by any change.

We successfully verified that the new optimizations hold for varied models that belong to the standard distribution of PowerDEVS, and that are built with both the GUI and with the Py2PowerDEVS extension for automated model building.

Our experiments with the standard DEVStone synthetic reference helped to increase the levels of confidence in the changes introduced in a robust and systematic way.

After resorting to a variety of example models from dissimilar domains and structures we conclude that the PowerDEVS performance was significantly improved and there is still room for additional improvements.

Finally, we used the optimized version of PowerDEVS for the real-world case study of the DAQ data network simulation model currently used at the ATLAS experiment at CERN, achieving an average relative speedup of well over 50 %, a substantial result for this project.

In terms of future work, we will continue to use the tools developed in this paper and look for additional performance optimizations. We are also interested in using the implementation of the DEVStone benchmark built in this work for PowerDEVS to compare it with other popular DEVS simulators. Finally, we plan to continue investigating alternative algorithms for handling event queues in the PowerDEVS core simulator, provided we identify potential impacts on simulation performance.

REFERENCES

ATLAS Collaboration 2017. “Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System”. Technical Report No. CERN-LHCC-2017-020 / ATLAS-TDR-029, CERN, Geneva.

Bergero, F. and E. Kofman. 2011. "PowerDEVS: A Tool for Hybrid System Modeling and Real-Time Simulation". *Simulation* 87(1-2):113–132.

Bergero, F. and E. Kofman. 2014. "A Vectorial DEVS Extension for Large Scale System Modeling and Parallel Simulation". *Simulation* 90(5):522–546.

Bonaventura, M., D. Foguelman, and R. Castro. 2016. "Discrete Event Modeling and Simulation-Driven Engineering for the ATLAS Data Acquisition Network". *Computing in Science & Engineering* 18(3):70–83.

Cárdenas, R., K. Henares, P. Arroba, J. L. Risco-Martín and G. A. Wainer. 2022. "The DEVStone Metric: Performance Analysis of DEVS Simulation Engines". *ACM Transactions on Modeling and Computer Simulation* 32(3):1–20.

Castro, R., M. Bergonzi, E. Pecker-Marcosig, J. Fernández and E. Kofman. 2024. "Discrete-Event Simulation of Continuous-Time Systems: Evolution and State of the Art of Quantized State System Methods". *Simulation* 100(6):613–638.

Glinsky, E. and G. Wainer. 2005. "DEVStone: a Benchmarking Technique for Studying Performance of DEVS Modeling and Simulation Environments". In *Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT)*. Oct. 10th-12th, NW Washington, DC, USA, 265-272.

Lanuza, J., G. G. Trabes, and G. A. Wainer. 2020. "Parallel Execution of DEVS in Shared-Memory Multicore Architectures". In *Proceedings of 2020 Spring Simulation Conference (SpringSim)*. May 19th-21th, Fairfax, VA, USA, 1-11.

Pecker–Marcosig, E., M. Bonaventura, E. Lanzarotti, L. Santi and R. Castro. 2022. "py2PowerDEVS: Construction and Manipulation of Large Complex Structures for PowerDEVS Models via Python Scripting". In *2022 Winter Simulation Conference (WSC)*, 2594–2605 https://doi.org/10.1109/WSC57314.2022.10015479.

Pozo Astigarraga, E., M. Bonaventura, J. Maple, E. Pecker Marcosig, G. Levrini and R. D. Castro. 2023. "Benchmarking Data Acquisition event building network performance for the ATLAS HL-LHC upgrade". In *Proc. of the 26th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2023)*. May 8th-12th, Norfolk, VA, USA, 1-11.

Risco-Martín, J. L., S. Mittal, J. C. F. Jiménez, M. Zapater and R. H. Correa. 2017. "Reconsidering the Performance of DEVS Modeling and Simulation Environments using the DEVStone Benchmark". *Simulation* 93(6):459–476.

SEDLab 2024. "PowerDEVS". Available at https://git-modsimu.exp.dc.uba.ar/matiasb/powerdevs-CERN/, accessed 28th March.

Trabes, G. G., V. Gil-Costa, and G. A. Wainer. 2023. "Energy and Power Evaluation of Parallel DEVS Simulations on Multicore Architectures". In *Proceedings of the 2023 Annual Modeling and Simulation Conference (ANNSIM)*. May 23th-26th, Hamilton, ON, Canada, 696-707.

Van Tendeloo, Y. and H. Vangheluwe. 2017. "An Evaluation of DEVS Simulation Tools". *Simulation* 93:103–121.

Vangheluwe, H. 2000. "DEVS As a Common Denominator for Multi-Formalism Hybrid Systems Modelling". In *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design (CACSD'00)*. Sept. 25th-27th, Anchorage, AK, USA, 129-134.

Zeigler, B. P. 2017. "Using the Parallel DEVS Protocol for General Robust Simulation with Near Optimal Performance". *Computing in Science & Engineering* 19(3):68–77.

Zeigler, B. P., A. Muzy, and E. Kofman. 2018. *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations*. 3rd ed. San Diego, CA, USA: Academic Press.

AUTHOR BIOGRAPHIES

EZEQUIEL PECKER-MARCOSIG received the Electronic Engineer degree and his PhD in 2011 and 2021, respectively, from the University of Buenos Aires (UBA, Argentina). He has been appointed Assistant Researcher in CONICET (National Research Council of Argentina) and holds an Adjunct Professor position at UBA. His research interests include M&S-driven controller design for cyber-physical systems. His email address is emarcosig@dc.uba.ar.

GERÓNIMO ROMCZYK is an MASc student in Computer Science in the Computing Department, Faculty of Exact and Natural Sciences, University of Buenos Aires, and scholar at the Discrete Event Simulation Lab. His thesis explores the optimization of simulation algorithms focused on large-scale data-networks. His e-mail address is gromczyk@dc.uba.ar.

MATÍAS BONAVENTURA received his Ph.D. in Computer Science in 2019 from the University of Buenos Aires, Argentina. He is now the main Software Engineer for the Dataflow system in the ATLAS TDAQ group at CERN. His research interests include simulation, high-performance networked and distributed systems. His e-mail address is matias.alejandro.bonaventura@cern.ch.

RODRIGO CASTRO received his Electronic Engineer degree and his Ph.D. in 2004 and 2010, respectively, both from Universidad Nacional de Rosario (UNR, Argentina). He is Professor with the University of Buenos Aires (UBA, Argentina) and Head of the Laboratory of Discrete Event Simulation at the Research Institute of Computer Science (ICC) of CONICET (National Research Council of Argentina). His research interests include the simulation and control of complex hybrid systems. His e-mail address is rcastro@dc.uba.ar.