ScalableTestGrids - An Open-Source and Flexible Benchmark Suite to Assess Modelica Tool Performance on Large-Scale Power System Test Cases

Francesco Casella[1] Adrien Guironnet[2]

[1]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy, francesco.casella@polimi.it
[2]Réseau de Transport d'Electricité, France, adrien.guironnet@rte-france.com

Abstract

This paper introduces *ScalableTestGrids*, an open-source and flexible benchmark suite for assessing the performance of Modelica tools on large-scale power system test cases. The benchmark suite is built by using components from the PowerGrids library and generic utility scripts creating the final runnable Modelica models. It does not depend on confidential data; its structure makes any future needed modification or evolution easy and straightforward. Results obtained with the OpenModelica tool are also reported. The benchmark suite can be used by tool developers to assess the capability of their tools to handle large-scale power generation and transmission models.

Keywords: Benchmark, Power System Simulation, Large-Scale Simulation, Performance, Open-Source

1 Introduction

Power systems are evolving at a very fast pace due to a global demand for cleaner energy. This drives major changes in the system structure, with a growing penetration of Renewable Energy Sources (RES) and an important boom of High-Voltage Direct Current (HVDC) lines (ENTSO-E 2020; IEA 2021). To be able to ensure the system stability and to handle the new challenges arising in a satisfying manner, System Operators (SO) have to adapt the way they control, operate and design the grid. This notably implies that power system simulation tools should offer a great flexibility to cope with the pace of evolution and a high-level of transparency to facilitate collaboration and coordination between all the actors.

While traditional power system software use closed-source models, solvers and data, and are difficult to adapt to emerging technologies, Modelica offers an appealing alternative. Its open-source, declarative and high-level nature makes it a good candidate for modern, flexible and transparent power system modelling.

These advantages have boosted the development of several power system Modelica libraries in the recent years (Winkler 2017), from first efforts to port existing tool models (Bogodorova et al. 2013) to libraries carefully designed to take full advantage of the declarative modelling approach of the language while easing the transition for power system experts who are Modelica beginners (A. Bartolini, F. Casella, and Guironnet 2019). It has also led to academics usage of Modelica in other domains than classical electromechanical simulations, such as electromagnetic transient simulations (Masoom et al. 2020), dynamic phasor modeling (Mirz et al. 2019) or power-electronics dominated grid simulations (Cossart et al. 2020) as well as proof of concept for industrial use (Francesco Casella, Andrea Bartolini, et al. 2016; Guironnet et al. 2018).

At the same time, efforts have been made to ease a wide adoption by both development of automatic conversion methods for creation of standard test cases (Razik, Dinkelbach, et al. 2018; Gómez et al. 2019) and computation time improvements (Francesco Casella, Leva, and Andrea Bartolini 2017; Braun, Francesco Casella, and Bachmann 2017; Henningsson, Olsson, and Vanfretti 2019), for example through the use of DAE-mode integration. However, fundamental performance barriers remain on these two fronts, hampering the full operational use of general-purpose Modelica tools to handle simulations involving national- or continental-scale grids; this motivated for instance the development of a mixed C++/Modelica approach by RTE (Guironnet et al. 2018).

Indeed, it is important to remind the reader the operational constraints currently existing on automation and performance for time-domain simulations in power systems. One fundamental process to ensure power system stability is the so-called *Dynamic Security Assessment*, that consists in simulating a large number of contingencies to make sure that the grid is operating in a secure and stable way (Loud et al. 2010; Panciatici, Bareux, and Wehenkel 2012). For example, the French national grid control center launches 65 different simulation scenarios every 15 minutes for voltage stability, and 1270 different scenarios every 30 minutes for transient stability. In addition to this *real-time* use, similar calculations are done in day-ahead and week-ahead situations with comparable performance constraints.

Currently available general-purpose Modelica simulation tools are still not able to provide adequate support for

such applications, in particular because the standard approach followed for code generation requires flattening the models to scalar equations, which leads to unacceptable code generation time and generated code size, see, e.g. (Francesco Casella, Leva, and Andrea Bartolini 2017). A quantum leap is required in code generation technology, which should avoid as much as possible to generate repeated code when hundreds or thousands of similar components are instantiated in a system model, as is commonplace in national grid models. Significant improvements may also be required on the simulation runtimes, e.g. to handle events efficiently.

On the other hand, testing the performance of such advanced or experimental Modelica tools on large-scale, realistic national power grid models is not a straightforward task; in general, it requires a lot of domain-specific knowledge to set up realistic test models and meaningful simulation scenarios, and possibly relies on confidential or restricted-access data. The latter issue also makes it difficult to compare the performance of different tools, since different tool developers may not have access to the same models.

In order to fill this gap, the development of a Modelica library based on the PowerGrids library (A. Bartolini, F. Casella, and Guironnet 2019) was launched, to offer open-source, easy-to-use, customizable and scalable benchmarks to all Modelica tools providers. The library captures the essential features of large-scale electromechanical power grid models, while keeping the complexity at the minimum possible level.

This library, called *ScalableTestGrids*, can be used by people doing research and development on advanced methods to handle large-scale Modelica models, to test the ability of their methods and tools to handle the kind of models that are required by SOs to run their daily operation. It can also be used to compare tools against each other, to assess the improvements of a given tool over time, and ultimately to make educated guesses possible about when general-purpose Modelica tools could eventually be used for industrial-grade power system simulations.

The rest of the paper will be organized in the following way. Section 2 presents the requirements used for specifying the benchmark suite while Section 3 introduces the selected design. Section 4 provides the current benchmark suite and gives the results obtained with OpenModelica while sketching evolutions that could further improve the performance. Finally, Section 5 serves as the conclusion.

2 Requirements

This section will introduce the main requirements used to define the benchmark suite and their motivations.

The first requirement is that the benchmark suite is representative of real power system test cases. This point is very important and ensures that advanced tools take advantage of structural conditions really existing in large-scale networks, rather than exploit artificial structural conditions only appearing in the benchmark suite. In particular, large-scale power system test cases have two main characteristics.

The first one is the very sparse structure of power system. Indeed, in power systems, each electrical node is only connected to a few other ones and the other physical components are either connections between two nodes or are interfaced to a single node. Controls are essentially local even if a few wide-area ones exist but only affect a very restricted subset of variables (frequency regulation for example). Table 1 shows representative sparsity levels for large-scale networks and Figure 1 shows the French power system and enables to directly see the sparse nature of the grid.

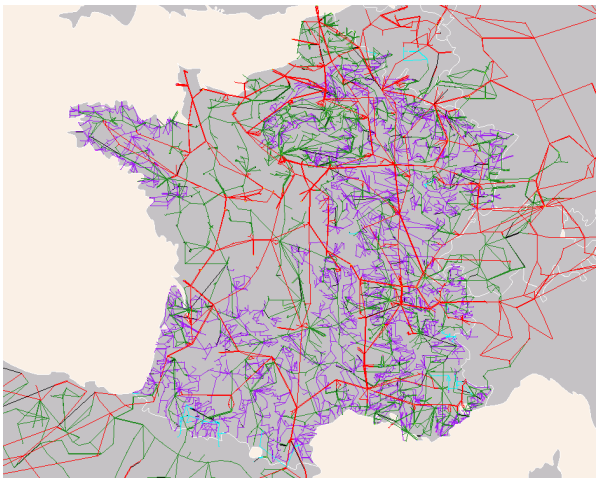

Figure 1. French Transmission System

The second characteristic is that large-scale networks are built using relatively few component models, which are instantiated a large number of times. This feature should be exploited by the tool, which should avoid wasting time and memory to generate repeated code structures. The components are instantiated and connected on a one-by-one basis, coming from a netlist description of the grid structure, which is irregular, as shown in Figure 1. Table 2 illustrates this property with the number of components for different large-scale test cases; the figures are taken from the Horizon 2020 Pegase project reports.

The third requirement for the benchmark suite is that it should be easily modified. In its current version the benchmark suite focuses on the impact of the system size, particularly the number of continuous variables, on the tool performance. This already offers a lot of tough challenges to address, but it could further evolve, for instance taking into account issues arising with large numbers of discrete variables and events. It is definitely of prime importance that such changes can be included in a straightforward way, considering the current pace of evolution in power systems, ensuring that the benchmark suite consistently captures the challenges posed to Modelica tools by such

DOI 10.3384/ecp21181351

Table 1. Sparsity levels for representative test cases with K nodes, N equations, NNZ non-zero elements in the Jacobian, density factor $d = \frac{NNZ}{N.N}$ (Razik, Schumacher, et al. 2019)

Test case	K	N	NNZ	d [%]
French Extra High Voltage with Simplified Loads	2000	26432	92718	**0.013**
French Extra High Voltage with Voltage-Dependent Loads	2000	60236	188666	**0.0051**
French + one neighbor Extra High Voltage with Simplified Loads	3000	47900	205663	**0.0089**
French + one neighbor Extra High Voltage with Voltage Dependent Load	3000	75300	266958	**0.0047**
French + neighb. countries Extra High Voltage with Simplified Loads	7500	70434	267116	**0.0054**
French Extra High Voltage + regional High Voltage with Simplified Loads	4000	90940	316280	**0.0038**
French Extra High Voltage + regional High Voltage with Voltage Dependant Loads	4000	197288	586745	**0.0015**
French + neighb. countries Extra High Voltage with Voltage Dependent Loads	7500	220828	693442	**0.0014**

Table 2. Number of components instances for representative test cases

Test case	Generators	Loads	Lines	Transformers
French EHV	607	2905	2668	1040
French EHV + HV	725	7875	8592	2577
Continental European EHV	3483	7211	~ 16000	~ 5000

models.

The fourth requirement concerns the benchmark suite scalability. The goal being to assess tool performance on large-scale simulations, creating examples of different size in a simple way is a *must have*. It means that from a small test case, utility functions should exist to obtain larger test cases with similar properties. These functions should be as generic as possible to facilitate their use for any kind of test case and should contain parameters enabling to define the final test case (its size for example).

Finally, the fifth requirement considered is related to the usability of the benchmark suite. In order to maximize its potential use, it is necessary that the initialization and simulation work fine in different Modelica environments. On the one hand, this implies that initialization should be straightforward and robust. On the other hand, simulation scenarios which are relevant but straightforward to simulate should be defined, ensuring that there are no potential simulation issues even with the large-sized systems; at the same time, the test cases should remain well representative of real-life scenarios at all sizes. The goal of these benchmarks is to show that a tool can handle systems of realistic size with good performance, not to test corner cases or numerically challenging situations.

3 Design

3.1 Package Structure

Considering the requirements laid out in Section 2, the library has been structured in three main packages:

- A *Components* package, which defines the components (based on the PowerGrids library) used to assemble the test cases.
- A *GridModelGenerators* package that contains the utilities functions to create the actual test cases.
- A *Models* package, containing some instances of automatically generated test cases for convenience, e.g. to get them easily run by continuous integration frameworks.

In the *GridModelGenerators* package, Modelica functions are employed to create large-scale network model using for-loops. This ensures that the library is fully self-contained and does not rely on other languages (e.g. Python or C) to generate the code of the test cases.

3.2 Model structure

A key feature of these models is that components and connect statements are instantiated individually, as in a real-life cases such as the one shown in Figure 1. Relying on arrays of components and for-loop connection equations could generate system structures that could be further optimized by smart tools, but that would be irrelevant to assess the performance on real-life models, where such regular structures are absent.

The structure of the scalable grid model which is currently implemented as a benchmark in the library is shown in Figure 2. At the top level, a meshed grid is represented, includings some nodes with large power generation systems, and some nodes with connection to local radial grids, which have a linear or tree-like structure. Twin transmission lines are sometimes used to ensure higher transmission capacity in the meshed part.

These structural features were represented by an idealized square, $2N \times 2N$-node meshed grid, with alternating generation (round) and load (square) nodes. The nodes are connected in the east-west direction by means of single transmission lines, while twin parallel transmission lines are used in the north-south direction.

Generator nodes contain a full-fledged synchronous machine model, equipped with automatic voltage regulation (IEEE AC4A type), power system stabilizer (IEEE PSS2A type), and turbine governor with primary frequency control (IEEE TGOV type), built with components from the PowerGrids library. The generator is connected to the meshed grid by a step-up transformer.

Load nodes contain a sub-system, built by the linear connection of M transmission lines, each connected to a PQ load at its end, and ultimately connected to the meshed network by a step-up transformer.

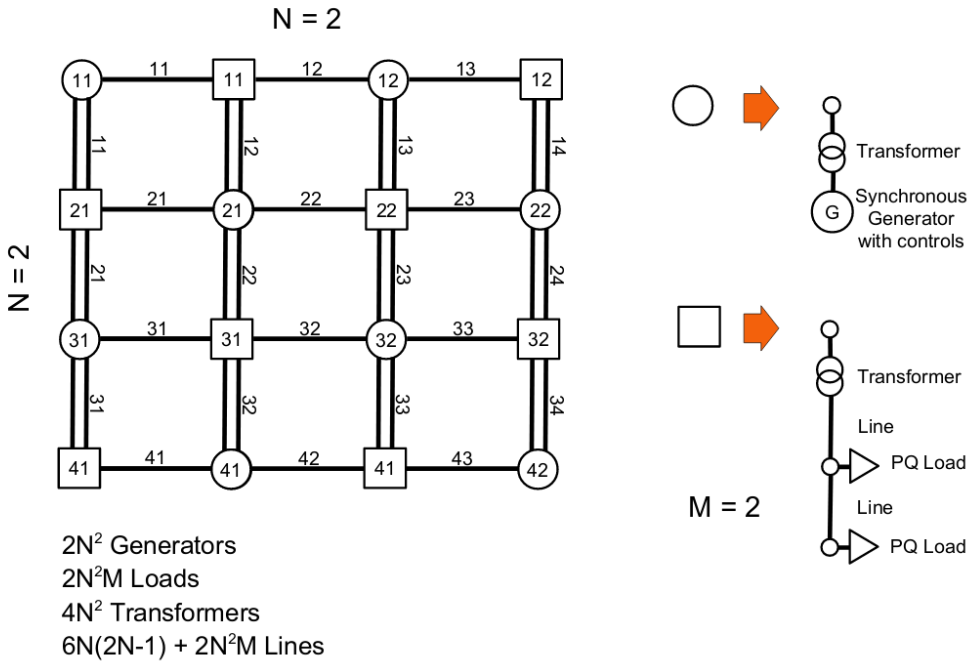

Figure 2. Test case structure with N = M = 2

As one can notice on the figure, the final test case comprises $2N^2$ generators, $2N^2M$ loads, $4N^2$ transformers and $6N(2N-1)+2N^2M$ lines. Thus, by setting N one can change the overall system size, while changing M allows to modify the sparsity pattern and include a more or less important part of the radial network in the system model.

The spatial structure of the grid is somewhat idealized, having a very regular pattern that can be easily scaled up in size. On the other hand, individual components are instantiated and connected one-by-one, and full-fledged realistic component models are used, so that the performance of a Modelica tool on a benchmark suite containing the number of nodes listed in Figure 2 can be considered to be fully representative of the performance on a model of a real-life system of the same size.

3.3 Initialization and simulation

Regarding initialization, dynamic power grid models, including the ones built with the *PowerGrids* library, are strongly nonlinear, and thus require the results of a static power flow computation to set up the start values for initialization; this is normally taken from the output of a separate tool. However, this would really be inconvenient in the case of this benchmark suite. The test model was thus conceived in order to have an initial power flow that is easily computed by the generated Modelica model, exploiting symmetry features.

Consider first an idealized case, where an infinite number of nodes is present, with full symmetry of voltages and power flows. Each generator node would then be surrounded by four load nodes of identical voltage, while each load node would be surrounded by four generator nodes of identical voltage. Also, there is zero net active power exchange between pairs of synchronous generators, which thus have all the same phase. Assuming that the load nodes have their nominal voltage magnitude (1 p.u.), and that the generator node voltages have a zero phase, it is possible to compute the complex power flows through the lines surrounding each load node, and hence the power flows and voltage phase and magnitude at both generator and load nodes. The active power output of each generator is then set to be equal to the total active power input of each radial sub-system, where each load takes $1/M$-th of the total active load, plus a small extra term for resistive losses across the transmission lines.

In this way, power flow values can be easily computed analytically and the results can be used to directly set the start values of voltage and complex power at the generator and load ports. The actual values of this ideal power flow are thus hard-wired in the code generator of the system model.

In fact, the actual grid has a finite extent, so it shows some border effects, compared to the ideal symmetric infinite grid; for example, the generator at node 11 needs to send its power through three lines only, instead of six, thus it requires a somewhat higher voltage. However, the difference with respect to the symmetric case is small enough that the symmetric power flow results can be used as start values for the finite grid steady-state initialization problem, without causing any convergence issue.

The only requirements on the Modelica tool used to simulate the test case are that it should use a *sparse* non-

DOI 10.3384/ecp21181351

linear solver to handle the initialization problem, since its size prevents using a dense solver in all cases except the smaller ones, and that it should preferably use homotopy for the initialization process, to avoid issues related to the controller saturations.

Once the system has been initialized at steady state, the simulation scenario assumes that all loads in the upper half of the network reduce their active power consumption by 10% at $t = 1$, regardless of the system size. This starts a transient in which all synchronous generators initially accelerate, due to the overall power imbalance; then, the primary frequency controllers reduce the turbine power outputs and stabilize the grid at a slightly higher frequency. Some local and inter-area damped oscillations of voltage and frequency ensue.

The symmetric nature of this perturbation is such that the transients of individual currents and power flows in the grid components in certain areas of the grid are similar regardless of the system size. This makes the comparison of simulation times across different grid sizes fair, since more or less the same things happen in each component, regardless of the grid size. This would not be the case if, e.g., the perturbation was applied to one load only, since the relative effect of such a perturbation would become smaller with increasing system size, potentially requiring less time steps from the variable step-size solver.

3.4 Generation of system models

Figure 3 shows a code fragment of the Modelica function that generates the system models, showing how individual components are instantiated and how individual connect-equations are added to the system.

The *Components* package contains the component models that are instantiated by the system model. In the current library version, as already mentioned, the generator model utilized is the connection of a synchronous generator with a voltage regulator control (IEEE AC4A), a power system stabilizer (IEEE PSS2A) and a speed regulator (IEEE TGOV). All these models are connected together to create the *ControlledGenerator* model that is instantiated in the system model.

This model structure makes it easy to modify or extend the test case, to include a new model or new levels of complexity. For example:

- in order to assess the impact of discrete variables and events, it is possible to create a composite model using the transformer physical model and a tap-changer logic (available in the original PowerGrids library) in the *Components* package, using it in place of the simple transformer physical model when instantiating the radial part of the network;

- if one wants to see the effect of distributed RES on performances, the load nodes can be enriched by connecting a RES model in parallel with the PQ loads;

- to include a new structure in the test cases, such as a few HVDC lines between two sub-networks, an extra for-loop could be included in the code generation function.

All these variants could be handled by Boolean parameters of the code generation function, which would activate the generation of the corresponding code.

4 Benchmark suite and results

The library has been used to create a first set of test cases using the basic model described in Section 3.2: synchronous generators with frequency and voltage regulations, voltage-dependent loads, classical linear transmission lines and transformers models (no regulations nor saturations).

Test cases were run with different values of *N* and *M* using OpenModelica version 1.18.0-dev-263-g3806526c07, setting the the most favourable options: use of *DAE-mode*, fixed-step homotopy solver for the initialization part, sparse Kinsol/KLU solver for the time-domain part, no tearing, which is too cumbersome for large systems, and -O0 optimization of the C code compilation, to avoid spending too much time on executable code optimizations.

The same experiments were performed on two different machines: a 20-core Xeon E5-2650 workstation with 72 GB of RAM under Ubuntu 20.04, and on an 8-core i7-8550U laptop with 16 GB of RAM under Windows 10 Pro, both 64-bits. During code generation, OpenModelica can exploit multiple cores by running several threads in parallel, e.g. for garbage collection; the generated C code is also split in several files, that can be compiled in parallel. Hence, simulations were run one at a time, to exploit parallelism a much as possible.

The results obtained are collected in Table 3. Simulations were ran successfully up to N = 11, M = 4 on the workstation and up to N = 6, M = 4 on the laptop; larger test cases could not be run due to memory limitations. However, the expected performance figures for those cases were extrapolated based on smaller test case results, and shown in italics in the table.

Concerning the simulation part, performance remain acceptable up to about 4000 components (i.e., generators, transformers, lines, and loads) in the system. It is comparable to the performance of existing domain-specific tools and demonstrates that the computation time is compatible with industrial uses of Modelica-based solutions for power system stability. Above this size, further improvements are needed, both on the software side - symbolic Jacobians in *DAE-mode* for simulation, more efficient and streamlined run-time code - and on the hardware side - using last-generation high-performance hardware, for example.

Code generation and compilation have reasonable performance up to about 300 components in the system. This means that general-purpose Modelica-based tools such as OpenModelica can be used for research studies on small

```
algorithm
  when initial() then
    Modelica.Utilities.Files.remove(f);
    print("within ScalableTestGrids.Models;", f);
    print("model Type1_N_" + String(N) + "_M_" + String(M), f);
    print("  extends Modelica.Icons.Example;", f);
    print("  inner PowerGrids.Electrical.System systemPowerGrids(", f);
    print("    initOpt = PowerGrids.Types.Choices.InitializationOption.globalSteadyStateFixedPowerFlow);", f);
    for i in 1:2 * N loop
      for j in 1:N loop
        if i == N and j == div(N + 1, 2) then
          print("  PowerGrids.Electrical.Buses.ReferenceBus BUS_GEN_EHV_" + String(i) + "_" + String(j) + "(SNom = 1e9, UNom = 400e3,
              UStart = 400e3 * 0.966, portVariablesPhases = true);", f);
        else
          print("  PowerGrids.Electrical.Buses.Bus BUS_GEN_EHV_" + String(i) + "_" + String(j) + "(SNom = 1e9, UNom = 400e3,
            portVariablesPhases = true);", f);
        end if;
      end for;
    end for;
    for i in 1:2 * N loop
      for j in 1:N loop
        print("  PowerGrids.Electrical.Buses.Bus BUS_LOAD_EHV_" + String(i) + "_" + String(j) + "(SNom = 1e9, UNom = 400e3,
            portVariablesPhases = true);", f);
      end for;
    end for;
    for i in 1:2 * N loop
      for j in 1:N loop
        print("  Components.ControlledGenerator GEN_" + String(i) + "_" + String(j) + "(GEN(SNom = 1e9, PStart = -806e6, QStart = -300
            e6));", f);
      end for;
    end for;
    for i in 1:N loop
      for j in 1:N loop
        for k in 1:M loop
          print("  PowerGrids.Electrical.Loads.LoadPQVoltageDependence LOAD_" + String(i) + "_" + String(j) + "_" + String(k) + "(PRef
              = Pvar, QRef = Qvar, UNom = 63e3, SNom = " + String(1e9 / M) + ", PStart = " + String(800e6 / M) + ", QStart = " +
            String(100e6 / M) + ");", f);
        end for;
      end for;
    end for;

    ...

    print("equation", f);
    for i in 1:2 * N loop
      for j in 1:N loop
        print("  connect(BUS_GEN_EHV_" + String(i) + "_" + String(j) + ".terminal, TRANSFORMER_GEN_" + String(i) + "_" + String(j) + "
            .terminalB);", f);
      end for;
    end for;
    for i in 1:2 * N loop
      for j in 1:N loop
        print("  connect(GEN_" + String(i) + "_" + String(j) + ".terminal, TRANSFORMER_GEN_" + String(i) + "_" + String(j) + ".
            terminalA);", f);
      end for;
    end for;

    ...
```

Figure 3. Part of the algorithm implemented to create the Modelica code of the system models

Table 3. Performance results for different system sizes

										Xeon E5-2650, Ubuntu 20.04				i7 85506, Windows 10			
N	M	# equations	# non-trivial eqs.	# nodes	# generators	# transformers	# lines	# loads	# solver steps	code gen. time / s	C compile time / s	exec size / MB	sim. time / s	code gen. time / s	C compile time / s	executable size / MB	simulation time / s
2	4	12174	5091	80	8	16	68	32	297	17.4	3.6	13.5	0.8	26.7	12.7	21.0	1.0
3	4	28284	11801	180	18	36	162	72	319	40.7	8.9	31.2	1.9	66.8	25.1	36.7	2.2
4	4	51078	21307	320	32	64	296	128	315	75.3	15.7	56.2	3.5	125.3	41.2	59.0	4.2
6	4	116718	48683	720	72	144	684	288	294	178.5	36.8	128.2	8.2	305.4	66.0	123.1	10.7
8	4	209094	87211	1280	128	256	1232	512	300	329.1	69.6	229.6	15.8	*600.0*	*140.0*	*240.0*	*20.0*
11	4	397788	165913	2420	242	484	2354	968	322	737.5	163.8	438.0	34.6	*1200.0*	*300.0*	*500.0*	*40.0*
16	4	*800000*	*350000*	5120	512	1024	5024	2048	*300*	*1500.0*	*350.0*	*900.0*	*70.0*	*2400.0*	*600.0*	*1000.0*	*80.0*
23	4	*1600000*	*660000*	10580	1058	2116	10442	4232	*300*	*3000.0*	*800.0*	*1800.0*	*150.0*	*5000.0*	*1200.0*	*2000.0*	*160.0*
32	4	*3200000*	*1400000*	20480	2048	4096	20288	8192	*300*	*6000.0*	*1800.0*	*3600.0*	*320.0*	*10000.0*	*2500.0*	*4000.0*	*320.0*
45	4	*6400000*	*2600000*	40500	4050	8100	40230	16200	*300*	*12000.0*	*4000.0*	*7500.0*	*700.0*	*20000.0*	*5000.0*	*8000.0*	*640.0*
64	4	*12800000*	*5600000*	81920	8192	16384	81536	32768	*300*	*24000.0*	*9000.0*	*18000.0*	*1500.0*	*40000.0*	*10000.0*	*16000.0*	*1280.0*

DOI 10.3384/ecp21181351

test cases without any problem; as soon as the number of components increases more, code generation becomes too time-consuming for practical industrial use. As already mentioned in the Introduction, as long as the structural analysis and symbolic processing of the system equations is done on a scalar basis, the code generation process and the generated code size do not scale up well enough.

The use of non-expanded arrays in all the stages of the compilation of the Modelica code into simulation code, which would exploit the repeated instantiation of the same model a large number of times, seems to be the most promising path to push back these limits. To the author's knowledge, such methods are currently not yet implemented in any production-grade, general-purpose standard Modelica tool. They are in fact already available in the IDA simulation tool (Sahlin et al. 2019), which however supports a variant of the language, IDA Modelica, a subset of the full Modelica language, with some extensions for separate compilation. Unfortunately the authors did not have access to that tool at the time of this writing, so they were unable to assess its performance with the proposed suite of benchmark models.

The results obtained clearly show that the most important barrier for large-scale simulations is currently found in the code generation and compilation time. This is an even more difficult challenge, knowing that state-of-the-art, domain-specific power system simulation tools don't need such phases during their operation and that one key process is stability assessment demanding a large number of simulations. On the other hand, the particular structure of large-scale power system simulations, built by a small number of different components which are instantiated a large number of times in the system, is not yet really exploited in the back-end process, meaning that a large room for improvement exist.

5 Conclusions

This paper introduces the *ScalableTestGrids* benchmark suite, created on top of the *PowerGrids* library. This open-source library, which is hosted on GitHub (*ScalableTestGrids library* 2021), offers a simple way to build benchmark models of electro-mechanical power generation and transmission systems, that are scalable up to very large size, representative of real-life system models, easy to customize, and easy to simulate, except for their sheer size.

Experiments with the latest version of OpenModelica were carried out on the benchmark suite, proving that the simulation times that can be achieved with general-purpose Modelica tools are already satisfactory. On the other hand, results showed that fundamental progress is still needed in the code generation and compilation phases, to envision an industrial use of general-purpose Modelica tools for large-scale power system stability simulations. In particular, a quantum leap in code generation methods is required, to exploit the feature of these system models, that contain very large numbers of instances of relatively few component models.

In the future, the authors plan to use the benchmark suite to measure the improvements achieved in Modelica tools for large-scale simulations, especially through the use of vectorized code generation. Efforts are ongoing in this direction, for example the LargeDyn project at Linkoping University, and HiPerMod project at Politecnico di Milano. They will also continue enriching it – e.g., by introducing more discrete variables or by adding new power system components such as RES or HVDC lines – to be able to assess the tools performance with diverse power system structures.

As a final remark, the comparison of the performance of different Modelica tools on the presented benchmark suite goes beyond the scope of the present paper, since the challenging nature of the benchmark suite may require to use experimental or undocumented features of the tools (which are not know to the authors of this paper), to obtain the best performance, as was the case with OpenModelica. Other Modelica tool developers are thus encouraged to test their tools on this benchmark suite, to report their best results, and to use it as a reference case to improve the support of large power system modelling in their tools.

References

Bartolini, A., F. Casella, and A. Guironnet (2019-02). "Towards Pan-European Power Grid Modelling in Modelica: Design Principles and a Prototype for a Reference Power System Library". In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. Linköing University Electronic Press.

Bogodorova, T. et al. (2013). "A Modelica power-system library for phasor time-domain simulation". In: *Proc. 4th IEEE PES ISGT Europe*.

Braun, Willi, Francesco Casella, and Bernhard Bachmann (2017-05). "Solving large-scale Modelica models: new approaches and experimental results using OpenModelica". In: *Proc. 12th International Modelica Conference*. Prague, Czech Republic, pp. 557–563. DOI: 10.3384/ecp17132557.

Casella, Francesco, Andrea Bartolini, et al. (2016-10). "Object-Oriented Modelling and Simulation of Large-Scale Electrical Power Systems using Modelica: a First Feasibility Study". In: *Proceedings of the 42nd Annual Conference of the IEEE Industrial Electronics Society IECON 2016*. IEEE. Firenze, Italy: IEEE, pp. 0–6. ISBN: 978-1-5090-3474-1.

Casella, Francesco, Alberto Leva, and Andrea Bartolini (2017). "Simulation of large grids in OpenModelica: reflections and perspectives". In: *Proc. 12th International Modelica Conference*. Prague, Czech Republic, pp. 227–233. DOI: 10.3384/ecp17132227.

Cossart, Q. et al. (2020-10). "An Open-Source Implementation of Grid-Forming Converters Using Modelica". In: *2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*. IEEE.

ENTSO-E (2020). *ENTSO-E Research, Development and Innovation Roadmap 2020-2030*. Tech. rep. ENTSO-E. URL: https://eepublicdownloads.azureedge.net/clean-documents/Publications/RDC%20publications/entso-e-rdi_roadmap-2020-2030.pdf.

Gómez, Francisco J. et al. (2019-01). "CIM-2-mod: A CIM to modelica mapping and model-2-model transformation engine". In: *SoftwareX* 9, pp. 161–167. DOI: 10.1016/j.softx.2019.01.013.

Guironnet, A. et al. (2018-10). "Towards an open-source solution using Modelica for time-domain simulation of power systems". In: *Proc. 8th IEEE PES ISGT Europe*. Sarajevo, Bosnia and Herzegovina.

Henningsson, E., H. Olsson, and L. Vanfretti (2019-02). "DAE Solvers for Large-Scale Hybrid Models". In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. Linköing University Electronic Press.

IEA, RTE (2021). *Conditions and Requirements for the Technical Feasibility of a Power System with a High Share of Renewables in France Towards 2050*. Tech. rep. IEA, RTE. URL: https://assets.rte-france.com/prod/public/2021-01/RTE-AIE_rapport%20complet%20ENR%20horizon%202050_EN.pdf.

Loud, Lester et al. (2010-08). "Hydro-Québec's challenges and experiences in on-line DSA applications". In: pp. 1–8. DOI: 10.1109/PES.2010.5588120.

Masoom, A. et al. (2020-12). "Simulation of electromagnetic transients with Modelica, accuracy and performance assessment for transmission line models". In: *Electric Power Systems Research* 189, p. 106799.

Mirz, M. et al. (2019-07). "DPsim—A dynamic phasor real-time simulator for power systems". In: *SoftwareX* 10, p. 100253.

Panciatici, Patrick, Gabriel Bareux, and Louis Wehenkel (2012-09). "Operating in the Fog: Security Management Under Uncertainty". In: *Power and Energy Magazine, IEEE* 10, pp. 40–49. DOI: 10.1109/MPE.2012.2205318.

Razik, Lukas, Jan Dinkelbach, et al. (2018-10). "CIMverter–a template-based flexibly extensible open-source converter from CIM to Modelica". In: *Energy Informatics* 1.S1. DOI: 10.1186/s42162-018-0031-5.

Razik, Lukas, Lennart Schumacher, et al. (2019-06). "A Comparative Analysis of LU Decomposition Methods for Power System Simulations". In: *Proc. 2019 IEEE PowerTech*. IEEE. Milan, Italy, pp. 1–6. DOI: 10.1109/PTC.2019.8810616.

Sahlin, Per et al. (2019-09). "On the scalability of equation-based building and district simulation models". In: *Proceedings 16th IBPSA International Conference and Exhibition*. Rome, Italy, pp. 2584–2590. DOI: 10.26868/25222708.2019.210130.

ScalableTestGrids library (2021). URL: https://github.com/PowerGrids/ScalableTestGrids (visited on 2021-05-07).

Winkler, D. (2017-09). "Electrical Power System Modelling in Modelica - Comparing Open-source Library Options". In: *Proc. 58th SIMS*. Reykjavik, Iceland.