View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Archivio istituzionale della ricerca - Politecnico di Milano

Solving Large-scale Modelica Models: New Approaches and Experimental Results using OpenModelica

Willi Braun[1] Francesco Casella[2] Bernhard Bachmann[1]

[1]FH Bielefeld, Bielefeld, Germany, {willi.braun,bernhard.bachmann}@fh-bielefeld.org
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy, francesco.casella@polimi.it

Abstract

Modelica-based modeling and simulation is becoming increasingly important for the development of high quality engineering products. Therefore, the system size of interest in a Modelica-based simulation is continuously increasing and the traditional way of generating simulation code, e.g. involving symbolic transformations like matching, sorting, and tearing, must be adapted to this situation. This paper describes recently implemented sparse solver techniques in OpenModelica in order to efficiently compile and simulate large-scale Modelica models. A proof of concept is given by evaluating the performance of selected benchmark problems.

Keywords: Modelica, large-scale, sparse solver techniques

1 Introduction

The design and safe operation of modern large-scale cyber-physical systems requires the ability to model and simulate them efficiently. The Modelica language is optimally suited for the modelling task, thanks to the high-level declarative modelling approach and to the powerful object-oriented features such as inheritance and replaceable objects. On the other hand, as noted in (Casella, 2015), until recently the development of Modelica tools has been focused on the modelling of moderate-sized models, optimizing the simulation code as much as possible by means of structural analysis and symbolic processing of the system of equations.

Large system models are usually characterized by a high degree of sparsity, since each component interacts only with a few neighbours, so that each differential-algebraic equation in the model only depends on a handful of variables. The availability of reliable open-source sparse solvers (Hindmarsh et al., 2005; Davis and Natarajan, 2010) and of cheap computing power and memory even on low-end workstations opens up the possibility of tackling much large system models, featuring hundreds of thousands or possibly millions of equations, exploiting the sparsity of such models for their solution.

In particular, the interest in the use of Modelica for the modelling and simulation of national- and continental-sized power generation and transmission systems recently motivated a first exploratory effort in this direction, using OpenModelica as a development platform, see (Casella et al., 2016). The methods implemented for the power system studies also allowed to efficiently simulate the cooling blanket of the future DEMO nuclear fusion reactor, which requires the modelling of thousands of individual heat-exchanging pipes, see (Froio et al., 2016).

The goal of this paper is threefold: to discuss different strategies for the simulation of large-scale Modelica models using sparse solvers; to describe an implementation of such strategies in the OpenModelica Compiler (OMC), using open-source solvers; finally, to present and discuss the performance obtained in a number of benchmark cases. The numerical methods are discussed in Section 2. The simulation performance is analyzed on three sets of benchmarks: the ScalableTestSuite library (Casella, 2015; Casella and Sezginer, 2016), some large power system models (Casella et al., 2016), and large high-fidelity models of the cooling system of the future DEMO nuclear fusion plant (Froio et al., 2017); results are reported in Section 3. Finally, Section 4 concludes the paper and gives an outlook to future work.

2 Solving Modelica Models

2.1 ODE mode

2.1.1 Symbolic Transformation Steps

In common Modelica tools the compile process can be summarized with the following steps, which are also explained in (Cellier and Kofman, 2006):

Flattening The Modelica model is transformed by the front-end into a flat representation, consisting essentially of lists of variables, functions, equations and algorithms.

Pre-Optimization In this phase a basic structural analysis of the differential-algebraic equations (DAE) is performed, e.g. detecting the potential states and discrete variables, eliminating alias variables.

Causalization This is a basic step in a Modelica Compiler, the so-called BLT-Transformation. Matching, sorting, and index reduction algorithms are applied in order to causalize the DAE and transform it to a system of ordinary differential equations (ODE).

Post-Optimization In this phase further optimization processes are applied on the equation system, e.g. the optimization of algebraic loops, like tearing, or the generation of corresponding symbolic Jacobians.

Code-Generation The final step after the symbolic manipulation is the target code generation for the optimized system in order to perform the simulation.

For this description and without lack of generality, and for clarity of the presentation, only the continuous part of the DAE is considered in the following. The result of the **Flattening** is the equation system:

$$\begin{aligned} F(t,\dot{x}(t),x(t),u(t),y(t),p) = 0, \quad t \in [t_0,t_f] \\ x(t_0) = x_0 \end{aligned} \tag{1}$$

where $\dot{x}(t) \in \mathbb{R}^{n_x}$ are the potential state derivatives, $x(t) \in \mathbb{R}^{n_x}$ are the potential states, $u(t) \in \mathbb{R}^{n_u}$ are the inputs and $y(t) \in \mathbb{R}^{n_y}$ are the algebraic variables. For simplicity, the initial conditions of the DAE states are given by x_0. Introducing $z = (\dot{x}\ y)$, denoting the unknowns of the DAE, and $v = (x\ u)$, denoting the known variables, the DAE can be re-written as

$$F(z,v) = 0 \tag{2}$$

that is basically the result of the **Pre-Optimization**.

The conceptual idea of the DAE **Causalization** commonly used in Modelica tools is to get an ordering of the unknowns $z(t)$, which enables to solve them sequentially

$$z = G(v) \in \mathbb{R}^{n_x+n_y} \tag{3}$$

If index reduction is necessary, some of the potential states and state derivatives become algebraic and the number of equations might change. The general form of the causalized system consists of a sequence of assignment statements including implicit systems of equations (algebraic loops)

$$0 = g_i(z_i, z_1, \ldots, z_{i-1}, x, u), \quad i = 1, \ldots, k \tag{4}$$

where

$$z = (z_1, \ldots, z_k), \quad z_i \in \mathbb{R}^{n_i}, \quad \sum_{i=1}^{k} n_i = n_x + n_y.$$

Finally, the ODE may be re-written

$$\dot{x} = f(x,u,p,t) \tag{5}$$

$$\hat{y} = h(x,u,p,t) \tag{6}$$

where $\hat{y}$ are the outputs of the system. Note that the other algebraic variables of y are considered to be internal to the ODE in this representation.

In the **Post-Optimization** mainly algebraic loops are torn down (Täuber et al., 2014) and the symbolical Jacobians are determined where applicable. Also the sparsity pattern of equation (5) is detected, which can be employed for the numerical jacobian calculation of the integration method (see also (Braun et al., 2012)).

2.1.2 Numerical Solving Process

For the simulation of the generated ODE equation (5) a numerical integration method for solving the differential equations as well as linear and non-linear system solvers for the implicit equations (4) are needed . In the next section the utilized methods and the exploitation scope for sparsity are described.

The numerical integration can be performed with explicit or implicit methods, whereby the implicit approaches are used in a Modelica environment more often, since most problems arising in practice are stiff. For explicit methods the next step can be calculated by

$$x(t+\delta t) = \Phi(x(t),u,p,t,\delta t,f), \tag{7}$$

whereby Φ is calculated by explicitly evaluating the function f in formula (5). Therefore, sparse methods can only be applied for calculating the solution of algebraic loops with respect to equation (4). The handling of sparse algebraic loops is described below.

For implicit methods the next step has to be calculated by

$$x(t+\delta t) = \Psi(x(t+\delta t),u,p,t,\delta t,f), \tag{8}$$

whereby the evaluation of Ψ involves the solution of a non-linear system using equation (5). The most widely used method for solving such non-linear systems is Newton's method and the core of it is to solve consecutive a linear system of the form

$$J \cdot (x(t+\delta t) - x(t)) = -F \tag{9}$$

where F denotes the residual form of equation (8) and J is the corresponding Jacobian matrix. The solution of this linear system offers some potential to gain performance for large-scale systems. Firstly, the matrix J can be calculated by exploiting the sparsity of the system, both numerically and symbolically. Naturally, this includes the storage of the matrix in a suitable sparse format to reduce the memory consumption. Secondly, in order to solve equation (9) sparse linear solvers (e.g. sparse LU factorization) can be utilized. For that purpose several methods have been developed and made publicly available (Davis and Natarajan, 2010; Davis, 2004).

For calculating the solution of algebraic loops with respect to equation (4) the same sparse solution methods can be utilized to gain some performance.

2.2 Simulation in DAE mode

An other way to go is to pass-through the whole system of equation (2) directly to an DAE solver, instead of using the ODE solver for integration and solve the implicit parts of equation (5) explicitly by algebraic solvers. Due to the fact that the index reduction is an important step for better convergence to the solution (Brenan et al., 1996), it is preferable to pass the system with index 1 (eq. (3)). Also, if the simulation is performed with equation (3), some time consuming steps in the **Post-Optimization** compiling process, which deal with algebraic loops, namely tearing and the generation of symbolic Jacobians, can be skipped. A DAE solver is always an implicit solver and has to solve an non-linear system, which is usually solved using some variants of the Newton's method (Brenan et al., 1996). Thus, all the local implicit algebraic loops are solved all together by the global routine. One effect of using equation (3) instead of solving equation (5) is that the Jacobian matrix gets bigger, since the integration method needs to solve for the variables x, $\dot{x}$ and y instead only for x. But this also preserves the sparse structure of the equation system with respect to Modelica models. In the ODE mode the corresponding Jacobian matrix is more dense due to the fact that the algebraic variables y are considered as internal variables.

Note, that in the current status of the DAE mode implementation it is still mandatory to generate the causalized code for proper handling of synchronous events and discrete variables. Therefore, OpenModelica generates currently an additional system in DAE mode. However, it is possible to skip unnecessary compiling steps by some specific compiler flags, which are documented in the OpenModelica User's Guide (Open Source Modelica Consortium).

2.3 Implementation in OpenModelica

The default simulation in OpenModelica is performed by solving system (5) using DASSL as a pure ODE solver. Hereby, the implicit parts (algebraic loops) are solved explicitly with algebraic equation solvers, the linear parts with lapack and the non-linear parts with a newton-based solver implemented in OpenModelica (Bachmann et al., 2015).

For the simulation of large scale Modelica models the most important part is a suitable sparse linear solver as depicted in section 2.1.2. Currently, one of the best under public domain available direct sparse linear solver for unsymmetric problems is the KLU solver (Davis and Natarajan, 2010) from the sparse matrix suite SuiteSparse. This solver is designed for solving sequences of unsymmetric sparse linear systems that arise from differential-algebraic equations, occurring when simulating electronic circuits. In fact, the linear systems arisen when simulating Modelica models are in general unsymmetric and often sparse, both in ODE and DAE mode. The open-source software family called SUNDIALS offers as a DAE solver the IDA solver (Hindmarsh et al., 2005). The IDA solver stands for Implicit Differential-Algebraic solver and is based on DASSL, but is written in ANSI-standard C. Further, for the solution of the underlying non-linear system at each time step, the IDA solver offers an interface to the sparse linear solver KLU. Furthermore, the SUNDIALS suite includes also a newton-based non-linear solver KINSOL, which is also able to use the KLU solver for the underling linear system. Through the connection of SUNDIALS and SuiteSparse suite to the OpenModelica environment it is now possible to rely on sparse methods at every step of the numerical simulation process.

3 Performance Results

3.1 Benchmarks from the ScalableTestSuite

3.1.1 Test set-up

The ScalableTestSuite (Casella, 2015; Casella and Sezginer, 2016) contains a number of different benchmark models, whose size can easily be chosen by setting one or more Integer parameters. The benchmarks are designed to stress some aspect of the code generation and execution, e.g. by possessing large implicit systems of algebraic equations, large number of states, large number of event-generating functions, etc. Please refer to the library documentation for further details.

This section reports the performance of a selection of nine benchmark models, each one coming in three different sizes.The results obtained with four different numerical solution strategies are presented and compared. Note that the current set of benchmarks does not include systems with large implicit systems of nonlinear equations – these will be added in the final version of the paper.

The first solution strategy, labelled *OD* in the result table, is the default approach to solving Modelica models implemented in the OpenModelica tool (see section 2). The DAEs are turned into ODEs by solving them for the derivatives, using the BLT transformation to do so efficiently, applying symbolic index reduction if the system has index greater than one. The implicit equations in the BLT corresponding to strong components in the dependency graph are solved with dense linear and nonlinear equation solvers, using tearing to reduce the size of the implicit part of the problem and thus somehow exploiting sparsity. The ODEs are then solved by the DASSL BDF integrator, using a dense linear solver for its internal operations.

The second strategy, labelled *OS*, still resorts to causalization; however, the implicit equations corresponding to the strong components in the BLT are solved by the Kinsol/KLU sparse solvers, while the ODEs are solved by the IDA BDF integrator, relying on the KLU sparse linear solver internally.

In this case, tearing is not applied to solve the implicit equations corresponding to strong components in the BLT. The rationale behind this decision is that on the one hand, the sparse solver already vastly reduces the computational complexity, if the system is highly sparse. On the other hand, tearing very large systems might take a disproportionately large amount of time by the compiler back-end, so that the time savings at run time are likely to be more than offset by the much longer code generation time. In fact, this trade-off would itself deserve to be studied, but that goes beyond the scope of the present paper.

The third strategy, labelled *DA*, is to only apply symbolic index reduction (if needed) to the DAEs, and then use the sparse IDA solver directly to solve them over time.

The fourth strategy, labelled *DD*, is a variant of the former one, in which the subset of the DAEs that is strictly required to be solved in order to compute the state variables at the next time step is identified and passed to the sparse IDA solver. Once the new time step has been computed and accepted as valid by the error estimation routine, the remaining equations are solved for the remaining variables by OpenModelica-generated code, exploiting the usual BLT decomposition to solve them efficiently.

This strategy can be advantageous because it avoids computing unnecessary variables during the internal solver iterations, particularly when tentatively computing a step that may then be rejected by the error estimation routine. Also, it is often the case that the dependencies in the systems are such that, once the variables required to advance the states have been computed, the remaining ones can be computed by explicit assignments. Furthermore, these are only computed once instead of getting unnecessarily involved many times in the iterative solution of the implicit sparse nonlinear DAEs.

As to the initialization problem, with the first strategy the standard dense linear and nonlinear solvers with tearing are used; with the other three, the sparse solvers Kinsol/KLU without tearing are used instead.

All tests were carried out on the Open Source Modelica Consortium continuous testing infrastructure, using the development version 1.12.0 of OpenModelica. The computer used to run the tests is a 16-core Intel i7-6900K CPU @ 3.20 GHz, with 132 GB RAM.

3.1.2 Results and discussion

Table 1 reports some selected results, showing the number of equations *NE*, number of states *NS* and the running times of the simulations in seconds, including the time spent for initialization, for the four above-mentioned strategies. The full online report for each strategy can be retrieved by clicking on the corresponding label in the table headings of the PDF file.

Note that all the employed solvers are stiff and equipped with automatic order and step-size adaptation, with relative tolerance set to 10^{-6}, so that accuracy of the simulation results is comparable and the comparison among simulation times is fair and meaningful.

First of all, it is apparent how the adoption of sparse solvers turns out to be beneficial for 7 out of 9 benchmark models, reducing the simulation times by factors ranging from about 2 (SteamPipe and OneDHeatTransferTT_Modelica) to about 60 (TransmissionLineEquations_N_1280). It is also not significantly harmful in the remaining two.

Although the models in the ScalableTestSuite might be somewhat artificial and thus possibly bring higher benefits than real-life models, in the author's opinion this result is a clear indication that sparse solvers are the recommended option to simulate large-scale Modelica models.

The improvement in performance can be ascribed both to the more efficient solution of the large implicit systems of equations involved in the solution process, and probably also to the lower number of time-consuming memory cache misses, due to the much smaller memory footprint of the simulation executable.

For some models, a large fraction of the simulation time is spent computing the right-hand-sides of the equations, rather then solving them, as in the case of the SteamPipe, where most of the time is spent computing the steam properties. In these cases, the adoption of a sparse solver cannot change the situation dramatically. On the contrary, sparse methods can bring huge benefits to models like TransmissionLineEquations, which have a large number of state variables, and an easy-to-compute right-hand side of the ODEs, with a very sparse Jacobian.

The advantage of using a sparse DAE solver over a sparse ODE solver is instead much less clear, and depends a lot on the specific case.

The multi-body models StringModelica, a suspended string modelled as a chain of rigid bodies and free rotational joints, and FlexibleBeamModelica, a cantilevered beam modelled as a chain of rigid bodies with elastic rotational joints, perform much better with the DAE solver, for reasons currently under investigation; also the SimpleAdvection models show a factor 2 improvement when using the DAE solver.

In other cases, such qas TransmissionLineEquations and PowerSystemStepLoad, the advantage is more limited. The TransmissionLineModelica model turns out to be five time faster with the sparse ODE solver than with the full DAE solver (*DA* strategy). The penalty is reduced to about a factor 2 when using the more advanced *DD* strategy, which is understandable, as the model is built with basic Resistor and Capacitor models from the Modelica Standard Library and thus has a lot of redundant equations.

Finally, it seems that the *DA* strategy never turns out to provide any substantial advantage over the second best choice.

3.2 Large-scale power generation and transmission system models

The interest in Modelica modelling of national- and continental-size power generation and transmission systems is growing. A first feasibility study in this field was reported in (Casella et al., 2016). The relevant features of the benchmark models are reported here for convenience; the interested reader is referred to the above-mentioned reference for background information and more details.

Three benchmark test cases from that study are considered in this paper, whose main features are reported in Table 3. Note that the size of these models is much larger than the typical size of the ScalableTestSuite examples reported in the previous section.

RETE_C is a model of the Irish power generation and high-voltage power transmission system, while RETE_E and RETE_G are a medium- and a high-fidelity model

Table 1. Simulation times of ScalableTestSuite benchmarks in seconds.

Benchmark	*NE*	*NS*	*OD*	*OS*	*DA*	*DD*
SimpleAdvection_N_3200	6402	3199	20.81	4.851	3.087	2.561
SimpleAdvection_N_6400	12802	6399	104.9	13.27	6.107	6.781
SimpleAdvection_N_12800	25602	12799	642.2	41.15	19.17	18.38
SteamPipe_N_640	8966	1280	169.2	148.4	158.7	139.3
SteamPipe_N_1280	17926	2560	395.8	316.8	357.8	302.9
SteamPipe_N_2560	35846	5120	1165	651.0	801.9	679.9
TransmissionLineEquations_N_320	642	640	4.344	0.5742	0.2626	0.3563
TransmissionLineEquations_N_640	1282	1280	23.52	1.133	0.8848	0.7923
TransmissionLineEquations_N_1280	2562	2560	241.1	6.099	4.973	4.621
TransmissionLineModelica_N_320	6755	642	3.677	1.100	3.337	1.937
TransmissionLineModelica_N_640	13475	1282	29.15	2.090	11.63	7.59
TransmissionLineModelica_N_1280	26915	2562	235.0	9.012	47.80	20.96
FlexibleBeamModelica_N_16	5949	32	26.74	21.65	14.4	9.611
FlexibleBeamModelica_N_32	10877	64	111.9	64.87	38.12	28.30
FlexibleBeamModelica_N_64	20733	128	1819	393.8	n.a.	65.47
StringModelica_N_16	5887	34	1.801	1.410	0.4385	1.012
StringModelica_N_32	10783	66	9.710	10.02	1.541	1.897
StringModelica_N_64	20575	130	86.48	25.91	3.756	n.a.
PowerSystemStepLoad_N_16_M_4	1059	193	0.2272	0.1477	0.1329	0.4610
PowerSystemStepLoad_N_32_M_4	3139	385	0.7197	0.632	0.4116	0.5558
PowerSystemStepLoad_N_64_M_4	10371	769	2.713	2.961	2.277	2.867
OneDHeatTransferTT_Modelica_N_320	3190	318	0.322	0.2358	0.1794	0.3176
OneDHeatTransferTT_Modelica_N_640	6390	638	0.9237	0.3579	0.4711	0.4736
OneDHeatTransferTT_Modelica_N_1280	12790	1278	1.822	1.038	0.9342	1.058
HeatingSystem_N_20	103	41	16.76	20.26	n.a.	n.a.
HeatingSystem_N_40	203	81	113.7	155.6	n.a.	n.a.
HeatingSystem_N_80	403	161	827.2	831.5	n.a.	n.a.

Table 2. Number of synchronous generators, transmission lines, transformers and equations of the benchmark models

Network	Gen's	Lines	Trafo's	Equations
RETE_C	74	369	583	56386
RETE_E	267	1458	1202	157022
RETE_G	407	6833	2824	593886

of the Italian high-voltage power generation and transmission system, with an equivalent simplified representation of the interconnection to the pan-European grid.

These models have a peculiar feature, i.e., their DAE representation is highly sparse, but their ODE representation is dense, because all the synchronous generators interact instantaneously with each other, due to the phasor-based algebraic description of the transmission network. As a consequence, the use of implicit ODE solvers is not recommended, because the corresponding Jacobian is very large and dense.

At the time of the writing of (Casella et al., 2016), the sparse DAE solver only worked on the smallest test case, so for the larger ones a variant of the the *OD* strategy was employed, using an explicit Runge-Kutta solver to avoid computing the dense Jacobian. Linearized load models were required in order to use the linear sparse solver KLU to compute the causalized equations. However, this approach was clearly sub-optimal, because a) realistic load models are non-linear and b) the system models are significantly stiff. Using fully implicit sparse DAE solvers with variable step size is clearly preferrable from a performance point of view.

In this paper, we can now report the simulation performance obtained with the *DA* strategy, using an Intel Xeon CPU E5-2650 server running at 2.30GHz with 72 GB of RAM installed. All simulations start with the system in steady-state, then at time t = 1 s a big load is disconnected from the grid, causing an imbalance between generated and consumed power. The system undergoes a transient with some voltage and frequency oscillations, until the voltage and frequency controllers re-establish a new equilibrium in about 10-15 seconds. The simulation time span is 20 seconds, in order to check that the system actually returns to steady-state.

Performance results are reported in Table 2. It is worth noting that these results were obtained with a first implementation of the *DA* strategy; the authors are confident that the optimization of the IDA solver parameters and a more thorough scaling of the problem, which is badly scaled due to the use of SI units, could further improve the performance significantly.

3.3 Large-scale models of nuclear fusion reactor components

The development of a conceptual design of the European Demonstration Fusion Power Reactor (EU DEMO) is one of the goals defined in the EU fusion roadmap Horizon 2020. The future DEMO reactor aims at demonstrating industrial-scale electrical power production from nuclear fusion processes.

Table 3. Simulation performance with *DA* strategy

Network	Rel. tol.	No. of steps	Sim. time [s]
RETE_C	10^{-4}	39	0.96
RETE_C	10^{-6}	146	3.18
RETE_E	10^{-4}	140	8.80
RETE_E	10^{-6}	364	15.22
RETE_G	10^{-4}	221	59.95
RETE_G	10^{-6}	615	123.19

Politecnico di Torino, in cooperation with Politecnico di Milano, is developing a global thermal-hydraulic model of the entire system, using Modelica. One important part of that is the breeding blanket cooling system, in which pressurized water flows through a very complex and large system of tubes, collecting the heat generated from the nuclear fusion process and delivering it to a standard steam generator and turbine system, similar to those used for traditional PWR nuclear power plants. The breeding blanket cooling system is highly modular and has a repetitive structure, but its sub-components have different geometric features, so that it necessary to simulate each and every tube individually. As a result, models of this system can have a very large size. The interested reader can refer to (Froio et al., 2017) for more details.

The model reported in the above-mentioned reference has 289126 equations and 20772 states. The simulation of a transient of interest for the study of such system requires 64 s with the *DD* strategy and 146 s with the *DA* strategy.

The model has been benchmarked and validated against more detailed 3D CFD models. Given the simulation times shown above, which are obviously much faster than those of the CFD simulation, the model is suitable for use in parametric optimization studies, aimed at the optimal design of the coolant flow distribution.

4 Conclusion

This paper introduces methods and strategies to solve large-scale Modelica models and reports the performance of their implementation in OpenModelica on selected benchmark problems.

The main result of this study is that the use of sparse solvers is almost always beneficial, sometimes very substantially, over the traditional use of dense solvers supported by thorough symbolic manipulation. The comparison between sparse DAE solvers and sparse ODE solvers has many different outcomes, depending on the specific problems at hand.

Another interesting result is that we have demonstrated the feasibility of using such sparse solvers to successfully simulate Modelica models of industrially relevant systems with size up to over half a million DAEs.

Further developments of this work are already planned. First of all, it would be interesting to using the DAE solver to simultaneously handle the differential equations and the nonlinear algebraic implicit equations corresponding to strong components of the BLT, while still exploiting the BLT to compute the residuals of the DAEs by sequences of explicit assignments. This would combine the advantages of the sparse ODE and sparse DAE approaches discussed in this paper, avoiding the nested iterations of the nonlinear strong components solver and of the implicit ODE solver, possibly further improving the results reported in this paper on some classes of models.

It will also be necessary to further optimize the code generation for use with sparse solvers, as the current implementation is such that the code generation time is typically much larger than the simulation time, particularly for very large models. Radically new approaches to the code generation process are needed to break the one million equation barrier with reasonable executable code sizes and code generation times.

Last, but not least, although the handling of hybrid models with DAE sparse solvers is already implemented in OpenModelica, it has not been specifically optimized for efficient handling of large-size models. Such optimization would be another interesting research direction.

5 Acknowledgments

The presented work is partly financed by the PARADOM project, that is funded by the Federal Ministry of Education and Research (BMBF) under the support code 01IH15002B.

CESI S.p.A. is gratefully acknowledged for making the power system models available for this study.

References

B. Bachmann, W. Braun, L. Ochel, and V. Ruge. Symbolical and numerical approaches for solving nonlinear systems. Annual OpenModelica Workshop 2015, 2015. URL `https://www.openmodelica.org/images/docs/openmodelica2015/OpenModelica2015-talk04-Bernhard-Bachmann-NLSinOpenModelica.pdf`.

W. Braun, S. Gallardo Yances, K. Link, and B. Bachmann. Fast simulation of fluid models with colored jacobians. In *Proceedings of the 9th International Modelica Conference*, pages 247–252, Munich, Germany, Sep. 3–5 2012. Modelica Association. doi:10.3384/ecp12076247.

K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, 1996. doi:10.1137/1.9781611971224.fm.

F. Casella and K. Sezginer. The ScalableTestSuite Modelica Library, 2016. URL `https://github.com/casella/ScalableTestSuite`.

Francesco Casella. Simulation of large-scale models in Modelica: State of the art and future perspectives. In Peter Fritzson and Hilding Elmqvist, editors, *Proceedings 11th International Modelica Conference*, pages 459–468, Versailles, France, Sep 21–23 2015. The Modelica Association. ISBN 978-91-7685-955-1. doi:10.3384/ecp15118459.

Francesco Casella, Andrea Bartolini, Simone Pasquini, and Luca Bonuglia. 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*, pages 0–6, Firenze, Italy, Oct. 24-27 2016. IEEE, IEEE. ISBN 978-1-5090-3474-1.

F. E. Cellier and E. Kofman. *Continuous System Simulation*. Springer-Verlag, 2006.

T. A. Davis. Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method. *ACM Transactions On Mathematical Software*, 30(2):196–199, June 2004. ISSN 0098-3500. doi:10.1145/992200.992206. URL `http://dx.doi.org/10.1145/992200.992206`.

T. A. Davis and E. Palamadai Natarajan. Algorithm 907: Klu, a direct sparse solver for circuit simulation problems. *ACM Trans. Math. Softw.*, 37(3):36:1–36:17, September 2010. ISSN 0098-3500. doi:10.1145/1824801.1824814. URL `http://doi.acm.org/10.1145/1824801.1824814`.

Antonio Froio, Francesco Casella, Fabio Cismondi, Alessandro Del Nevo, Laura Savoldi, and Roberto Zanino. Dynamic thermal-hydraulic modelling of the eu demo wcll breeding blanket cooling loops. *Fusion Engineering and Design*, in press, available online:1–5, 2017. doi:10.1016/j.fusengdes.2017.01.062.

C. Froio, F. Casella, F. Cismondi, A. Del Nevo, L. Savoldi, and R. Zanino. Dynamic thermal-hydraulic modelling of the eu demo wcll breeding blanket cooling loops. In *Proc. 29th Symposium on Fusion Technology (abstract)*, Prague, Czech Republic, 2016.

A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

Open Source Modelica Consortium. OpenModelica User's Guide. Online. URL `https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/`.

P. Täuber, L. Ochel, W. Braun, and B. Bachmann. Practical realization and adaptation of cellier's tearing method. In *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, EOOLT '14, pages 11–19, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666204. URL `http://doi.acm.org/10.1145/2666202.2666204`.