# Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives

Francesco Casella

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy,
francesco.casella@polimi.it

## Abstract

State-of-the-art Modelica tools are very effective at converting declarative models based on differential-algebraic equations into ordinary differential equations. However, when confronted with large-scale models of distributed systems with a high number of states (1000 or more) or with large algebraic systems of equations (1000 or more unknowns), they face a number of serious efficiency issues, that hamper their practical use for system design. The paper analyses these issues in detail, points out strategies for improvement, and also introduces a library of scalable test models that can be used to assess existing tools, as well as to help developing advanced solution methods for large-scale systems.

*Keywords: Modelica Compilers, Large-Scale Models, Efficient Simulation*

## 1 Introduction

After almost 20 years from the first release of the Modelica language definition 1.0 (The Modelica Association, 1997), the Modelica language is well-established for system-level modelling tasks in many domains of engineering, such as automotive, robotics, mechatronics, energy, aerospace, in particular when multi-domain modelling is required.

To the best of the author's knowledge, based on published literature and personal experience, the standard work flow of state-of-the art Modelica tools can be summarised by the following steps, which are described in detail by Cellier and Kofman (2006).

1. *(Flattening)* The Modelica code is parsed; classes are expanded and instantiated, and eventually brought into the so-called flat form, i.e., a set of scalar hybrid differential-algebraic equations together with a set of scalar variables and parameters.
2. *(Causalisation)* Structural analysis of the differential-algebraic equations (DAEs) is performed, in order to solve them efficiently for the state derivatives and algebraic variables. This process includes equation ordering (BLT transformation), may require symbolic index reduction, and usually involves extensive symbolic processing, as well as the use of advanced techniques such as tearing or reshuffling for solving sub-systems of equations efficiently. In most cases, the use of numerical solvers for linear and non-linear systems of algebraic equations is required.
3. *(Time integration)* The code which results from the previous step is linked to some well-tested, general-purpose dense Ordinary Differential Equation (ODE) solver, including root-finding algorithms to handle state events in the case of hybrid models.

In principle, step 2 is not strictly necessary, as DAEs resulting from step 1 could be solved directly using numerical DAE solvers. In practice, this is not standard practice for two reasons: one is that object-oriented Modelica models very often end up having index greater than 1, that are challenging to solve numerically, the other is that the above-sketched process is usually more numerically robust and easier to initialize than the direct solution of the nonlinear DAEs.

As to step 3, most Modelica models end up being stiff, because the modular way of building the models very often generates some very fast dynamic phenomena that, albeit maybe not of interest for the modeller, cannot be easily removed from the model, because they stem from the interaction of equations placed in different components.

As a consequence, stiff solvers are usually needed, the choice usually falling onto DASSL (for multi-step algorithms) and on Radau IIa (for single-step algorithms), which implement sophisticated step-size and order adaptation with error control, as well as root-finding algorithms for state-event detection.

When explicit solvers are required (e.g., for real-time simulation applications) it is sometimes possible to carefully build a modular model so that stiffness is avoided, but this is not the standard way people build object-oriented models in most cases, and people usually take for granted that stiffness will be handled by the solver.

DOI
10.3384/ecp15118459

In the early days of Modelica tool development, the largest and most challenging object-oriented models were multi-body systems, for which the above-sketched process can be extremely effective.

Consider, for example, the well-known 6 d.o.f. robot model of the Modelica.MultiBody.Examples library: a system which is originally described by 1766 non-trivial DAEs is reduced to an ODE system having only 36 states, whose derivatives can be computed by forward assignments, except for one 6x6 linear system. Furthermore, the resulting ODE system is dense and very strongly coupled, as a change in torque at one joint influences the acceleration of all the robot's links, due to kinematic constraints. The choice of a general-purpose dense ODE solver is perfectly adequate in this case. Even more dramatic is the case of the EngineV6 model, which starts from 2083 non-trivial DAEs and ends up with a 4-th order ODE system.

Probably due to the success in these very demanding applications, this standard work flow has not changed much over the years, and is still the state of the art as of today. However, there are significant problems when following this approach with several categories of models, some of which are rapidly gaining significance.

For example, it is well-known among control practitioners that the simulation of the transient of a Modelica system model including a digital controller can be orders of magnitude slower than the simulation of the model with the corresponding continuous-time one. As a consequence, people often delay the simulation of the actual closed-loop behaviour with the digital controller until late in the project, even though some potentially critical control functionality (e.g., anti-wind-up logic) cannot be easily and accurately reproduced in a continuous-time framework. Also, when finally switching to digital control, they might resort to fixed-time-step simulation, which do not give any guarantee of precision, in order to keep the simulation time within acceptable limits.

Another field of growing importance is the simulation of large networked systems with decentralized control. One notable example is that of smart grids, where multiple producers and consumers of electrical and also possibly thermal energy cooperate to the goals of stable network behaviour, satisfaction of all the load requests, and system optimality. Another interesting example is the one of self-driving cars on highways. These systems can easily encompass hundreds or thousands of individual agents, some of which might be inactive or dormant for long periods of time, as well as multi-domain physical phenomena that span widely different time-scales.

The design of the control strategy for such large-scale systems is usually based on hierarchical approaches, using cascaded control strategies and abstracting low-level behaviour within higher levels. However, at some point in the design cycle it becomes important to verify the system performance by taking into account the detailed physical behaviour, in particular to test how the system reacts to borderline or anomalous conditions. For example, what if a power generation unit cannot keep up with the required load ramp rates, due to limitations in the energy conversion process? What if a self-driving car brakes too hard on slippery ground and loses traction in a rush-hour traffic scenario? Today's Modelica tools are clearly inadequate to handle the simulation of such large systems, because their standard work flow does not scale up well with the system size.

The goals of this paper are thus to point out the fundamental limitations of the current approach that hinder the use of Modelica tools in these areas, to highlight some recent relevant research developments going in the right directions, to make concrete proposals for further research, and finally to urge the Modelica community to undertake a more systematic and aggressive strategy to make the object-oriented simulation of such systems easy and efficient for tomorrow's system designers.

This is the outline of the paper: in Section 2, the issues of current state-of-the-art Modelica solvers with large-scale models are reviewed; Section 3 introduces a library that is intended to collect a wide array of benchmark of scalable (very) large Modelica models to support the development and testing of innovative methods and algorithms; Section 4 reviews some promising research trends to address the challenges of efficient simulation of large-scale models. Finally, Section 5 closes the paper with some concluding remarks.

## 2 Issues with State-of-the-Art Solvers and Large Models

In this section, the limitations of the standard work flow presented in the Introduction when dealing with large-scale models are discussed.

### 2.1 Localized Interaction is Not Exploited

Object-oriented system models are built in a modular way by the hierarchical composition of components and sub-systems via causal and a-causal connectors. The a-causal connection paradigm makes it possible to propagate instantaneous constraints (i.e., algebraic equations) through large portions of the system, with the consequence that the ODE $dx/dt = f(x,t)$ obtained after causalization is tightly coupled and has a dense Jacobian $\partial f/\partial x$ with comparably few non-zero terms.

In practice, however, this almost only happens in the case of multi-body systems, where the Jacobian corresponding to the states of kinematic chains turns out to be dense. In most other cases, the interaction between subsystems is based on flows that depend only on nearby states. Although the a-causal modelling paradigm allows for algebraic constraints resulting in a tight coupling between the state derivatives across components, these constraints are usually confined to small portions of the system. In other words, the derivative of any given state variable only depends on the values of a few neighbouring states, which means that each row of the Jacobian $\partial f/\partial x$ only has a few non-zero elements.

As a consequence, the Jacobian $\partial f/\partial x$ has a high degree of sparsity. Even more importantly, the number of non-zero element on each row is an invariant property of the system structure. Therefore, as the number of states $N$ grows, e.g., because more and more individual agents or sub-systems are added to it, the number of non-zero terms only grows as $O(N)$, not as $O(N^2)$, as it is the case with tightly coupled systems.

Apparently, this basic fact is still not exploited by Modelica tool developers, which still employ dense solvers as the mainstream option. This has two severe consequences. The first is that the workload of Hessian inversion in implicit solvers, which grows as $O(N^3)$, eventually becomes the bottleneck for large enough systems, as there are no other tasks in the simulation process whose complexity grows at this rate. Probably, this is currently masked by the fact that other tasks become infeasible earlier as the size grows, before this break-even point is ever reached, but as other tasks are streamlined, this will become all-important.

The other consequence is that the memory allocated to store the values of the Jacobian $\frac{\partial f}{\partial x}$ and of the factored Hessian $(h\frac{\partial f}{\partial x} - I)^{-1}$ matrices for the solver can become unnecessarily quite large, triggering lots of cache memory misses that can severely degrade the simulation speed. For example, a system with 1000 state variables requires 16 Mbytes to store the these two matrices in double-precision floating-point arithmetics; this is already well beyond above the size of the on-CPU cache in modern processors. A system with 10000 states requires 1.6 Gbytes of storage just for that purpose, which is totally unreasonable, as most of that space is occupied by zeros.

## 2.2 Localized Activity is Not Exploited

In many large-scale systems, local phenomena may occur within one sub-system or agent, that require short time steps for an accurate description, but on the other hand have negligible influence over the other sub-systems or agents on the time span of such short steps.

For example, consider the model of an urban district heating system. When a local temperature controller is switched on or off, or when a window is opened in one heated unit, relatively fast transients are triggered that involve local state variables, but have a negligible influence on the temperature of the water in the distribution system over that short time span, due to its large heat capacity. As a consequence, they also have a negligible effect on the other heated units. In order to describe the fast local transients within the specified accuracy, standard ODE solvers will reduce the time step length and compute several time steps within a short time interval.

This is very inefficient when the system is very large, (say, 100 or 1000 heated units), since the short time steps span the *entire* system, requiring the computation of a very large derivative vector and of an extremely large Jacobian matrix, as well as the inversion of an extremely large Hessian matrix. This tremendous effort is in fact useless, as all other state variable will hardly change at all during these short steps.

## 2.3 Systems with Activity on Widely Different Time Scales Are Penalized

In many cases, multi-domain systems are characterized by physical phenomena taking place over widely different time scales. For example, power plant models built by connecting a boiler-turbine model, a synchronous electrical generator, and a transmission line to a network strong point, are characterized by slower thermal dynamic phenomena, taking place over time scales from a few seconds to a few hundred seconds, and faster electrical phenomena taking place over time scales from 10 to 100 milliseconds.

When transients take place in the faster sub-system(s), standard ODE solvers reduce the system-wide time step length to very small values, causing the wasteful re-computation of the slower thermal states, which hardly change at all across those time steps. On top of that, if accurate equations of state are used to describe the fluid properties, these unnecessary computations involve those equations, leading to an enormous and useless overhead. Once again, the slow-down factor gets bigger as the size of the system (i.e., the number of its states) increases.

## 2.4 Localized Influence of Events and Discontinuities is Not Exploited

When hybrid systems and events are involved, the situation illustrated in the previous sub-section gets even worse. The standard approach described in Section C of the Modelica Language Specification (The Modelica Association, 2014) requires that every time an event is triggered, the integration of the continuous-time ODE is halted at the event instant, the event is processed, global event iteration (involving the entire system!) is performed until convergence, and finally the simulation is restarted, usually with a very short time step because the discontinuities triggered by the event usually cause fast changes in some state variables, requiring small enough steps to stay within the allowed error tolerance.

Although this approach gives theoretical guarantees of accuracy in the numerical solution, it quickly becomes prohibitively expensive for all but the simplest systems.

For example, consider the model of an innovative energy conversion system for distributed generation, where a waste heat recovery unit, powered by an Organic Rankine Cycle (ORC) engine, drives an electrical generator which is connected to the grid by means of a switched AC/AC converter. The ORC system has time constants from a few seconds to a few tens of seconds, and is characterized by a very high CPU load to compute its state derivatives, as complex equations of state are needed to describe the fluid properties. It might also feature a digital controller model with a periodic sampling time around 500 ms or so. On the other hand, the switched converter model triggers state events whenever currents or voltages on each of the three phases of both sides cross certain thresholds. Since a small ORC turbine can easily exceed 60000 rpm rotational speed and there are multiple events for each turn on each phase, the average frequency of events may easily exceed 10000 per second.

It is clear that recomputing the entire state derivative record, which requires computing all the fluid properties around the circuit, at this kind of rate can make the simulation four or more orders of magnitude slower than necessary, which basically means this kind of simulations is currently infeasible with state-of-the-art tools. On the other hand, this state of affairs is by no means necessary: it is easily understood that the effect of any single switching on the turbine rotational speed is negligible, due to its comparatively large inertia. Since the turbine is the only interface between the electrical part and the thermal part, there is obviously no need at all of recomputing the thermal states 10000 times per second or more, in order to achieve an accurate simulation.

### 2.5 Systems with large-scale algebraic constraints are not considered

At the other end of the spectrum, there are interesting system models characterized by very large systems of algebraic equations. In the Modelica world, a model with one or more such systems is often considered evil, or at least the result of inappropriate modelling practices, and the common wisdom calls for adding a few more states to the model in order to break them down to smaller systems. However, this is not always appropriate.

Consider the study of power generation and transmission systems, which has recently gained interest in the Modelica community, see Vanfretti et al. (2014). The models used to assess the network stability consider the dynamic phenomena taking place in the power generators, while neglecting the much faster electrical phenomena in the transmission network, which is described by algebraic equations (dynamic phasors). Nation-wide or continent-wide models can thus easily contain thousands of state variables, as well as one, very big algebraic system of a hundred thousands or more linear equations. They key factor here is that this system will be extremely sparse, thanks to the transmission network topology.

State-of-the art tools try to cope with this system using tearing, which is prohibitively expensive at this scale.

### 2.6 Repetitive Structures are Not Exploited

Large-scale models usually involve repetitive structures, such as arrays of variables or models, and for-loops in equation sections. Even if for-loops are not used, it might be the case that the same model is instantiated a very high number of times. For instance, a model of a digital circuits might contain a very large number of NAND gates; a 64 bit adder might be built hierarchically by assembling 4-bit and 16-bit adders.

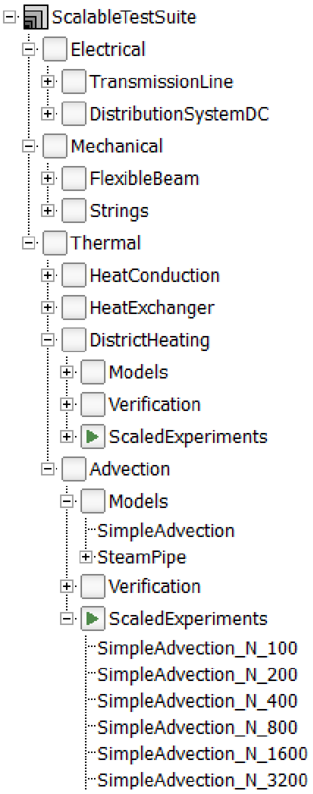
As mentioned in the Introduction, the mainstream approach of state-of-the-art tools is to flatten the entire model all the way down to scalar variables and equations, then analyse the structural properties of the system of equations and generate the code to compute the state derivatives. If the model is very large and has a lot of repetitive structures, the analysis phase (which is part of the compilation process) might require a very large amount of time, which could be spared if the analysis is carried out at the array level.

Also, following the standard approach, separate code is generated to solve each equation in the DAE, so that, in the case of repetitive models, the code has a large number of duplications, i.e., chunks of code that carry out exactly the same operation, albeit on different chunks of data. This can lead to unnecessarily high memory allocation, which brings in the previously discussed overhead due to off-cache memory access and cache misses. If the output of the Modelica tool is C code that needs to be compiled into executable code, very large source code files might also cause the C compiler to fail, or at least to become very slow, particularly if optimized code is generated.

## 3 The ScalableTestSuite Library

The assessment of the performance of existing tools when dealing with large scale systems, as well as the testing of innovative algorithms and solution strategies, calls for a library of benchmark cases. In the author's opinion, the requirements of this library are

- The size of the model should be easily selected by setting one (or more) integer parameters, while meaningful values of all other physical parameters should be automatically set by the model.
- The models should stress all the aspects mentioned in Section 2, either one at a time, or possibly also in a combined fashion.
- The models should be physically meaningful and representative of real-life modelling problems.
- The library should be self-contained and only depend on the Modelica Standard Library (MSL) to ensure maximum portability.
- At least some models should be defined as plain equations in a single Modelica class, so that they



```
ScalableTestSuite
  Electrical
    TransmissionLine
    DistributionSystemDC
  Mechanical
    FlexibleBeam
    Strings
  Thermal
    HeatConduction
    HeatExchanger
    DistrictHeating
      Models
      Verification
      ScaledExperiments
    Advection
      Models
        SimpleAdvection
        SteamPipe
      Verification
      ScaledExperiments
        SimpleAdvection_N_100
        SimpleAdvection_N_200
        SimpleAdvection_N_400
        SimpleAdvection_N_800
        SimpleAdvection_N_1600
        SimpleAdvection_N_3200
```

**Figure 1.** Structure of the ScalableTestSuite library.

might also be tried out easily with other simulation tools that do not support Modelica.

- At least some models should be defined both by plain equations and by modular descriptions, to assess how efficiently the tool can handle the overhead of a modular, object-oriented description of the model.
- The library should be widely advertised among tool developers and researchers, be open to contributions, and eventually become the accepted reference benchmark for the community.

A related work, the Modelimark test suite, was presented at the 2011 Modelica conference by Frenkel et al. (2011). The main goal in that case was the same of the library presented in this paper, i.e., provide scalable test models for tool benchmarking.

The result of that work is a Python software which can automatically generate Modelica code of models with scalable complexity. This approach is actually quite powerful and gives a lot of flexibility in terms of what can actually be tested; on the other hand, it makes the development, deployment and maintenance of the test suite more complex than a plain Modelica package containing the direct definitions of the benchmarks, in particular if many contributors are expected.

Also, the focus of that work is mainly (though not exclusively) aimed at evaluating the compiler performance, i.e. how much time is needed to obtain the executable simulation code from the Modelica source, while the present work is more focused on the solver performance, though obviously the compiler performance can be evaluated as well. Finally, compared to the Modelimark test suite, the present work puts more emphasis on testing the simulation performance on physically meaningful models, which are representative of some class of real-life modelling problem.

At the 2014 EOOLT workshop in Berlin, after the discussion following the presentation of the paper (Ranade and Casella, 2014), the author decided to start working on his library with a master thesis project. Version 1.0 of the library, released on May 11th, 2015, is the end result of Kaan Sezginer's master's thesis (Sezginer, 2015). The library, which is open source and hosted on GitHub (Sezginer and Casella, 2015) is under continuous development and has already grown since then.

As of the time of this writing, the library contains 16 different test models, belonging to the electrical, mechanical, and thermal domains. The size can be set by suitable integer parameters - the library already contains the definition of test cases of size growing as the powers of 2, complete with experiment annotation, so that the benchmarks can be run by just checking out the library from the repository and compiling any of those classes.

The structure of the library is shown in Figure 1. More specifically, the following models are currently included:

- Electrical transmission line, directly modelled by equations or by connection of MSL components
- DC distribution networks, modelled by MSL components
- Flexible cantilevered beam, modelled by connection of MSL multi-body components or by the finite element method, taken from Schiavo et al. (2006)
- String suspended in a gravitational field, modelled by connection of MSL multi-body components
- One-dimensional heat conduction, with two different types of boundary conditions, directly modelled by equations or by connection of MSL components
- One-dimensional heat exchanger, in co-current and counter-current configuration, assuming constant fluid density and constant fluid heat capacity
- Models of 1D thermal advection, one assuming constant density and heat capacity, the other using the detailed IF97 model of steam and including compressibility effects
- Models of a district heating (Ranade and Casella, 2014) and of distributed cooling system (Floros et al., 2014). The former is a continuous time model, using nonlinear (and very stiff) systems with

bifurcations to model the local on-off temperature controllers; the latter employs events for the same purpose.

The models have been verified against known analytical solutions, whenever available. Each model stresses one or more of the aspects discussed in Section 2.

All the models in the library except the ones built with the MultiBody library have a high degree of sparsity, see Table 1 for some example values. As already anticipated in Section 2.1, the density of the Jacobian for the multibody models does not depend on the size and is about 50%, making a dense solver perfectly adequate to handle them. Conversely, for all other models the number of non-zero elements grows as $O(N)$, so that the density of the Jacobian is inversely proportional to the system size and already much below 1% even for moderately large models with about 1000 state variables.

The district heating model is characterized by a strongly localized action: the on-off transitions of the local temperature controllers require many time steps to be computed accurately, but take place in much less than one second, during which the temperature of all other units do not change significantly; due to sligtly different heat capacity parameters of the different units, all the transitions take place asynchronously, so that each transition involves one unit at a time.

The distributed cooling system model combines the feature of the previous model with events having only a local influence on the corresponding unit temperature.

The transmission line models also show localized action, as the simulated transient correspond to one sharp voltage and current wave crossing the transmission line once; consequently, each individual voltage has a sharp transition only when the wave passes through it, and is practically constant during the rest of the time.

The DC distribution system models allow to experiment with test cases featuring large sparse systems of linear algebraic equations.

As to repetitive structures, most examples are based on arrays of parametric size of variables and/or models, and make use of for loops in the equation section, so they can also be used for testing the ability of the compiler to cope with these structures efficiently. In the case of the DC distribution system, Modelica code is also provided that automatically generates the code of large system models with explicit declarations of individual components and connections, in order to test the ability of compilers to factor out common code also in this case. Some examples of automatically generated code are already included in the library.

More models with events, as well as multi-physics models with widely different time scales, will be added in the near future, possibly before the writing of the final version of this paper.

# 4 Research Trends for the Efficient Simulation of Large-Scale Models

This section points out some recent research trends that might improve the performance of Modelica tool dramatically when dealing with large-scale models. The aim is to encourage the community at investing more in this direction and bring these advanced methodologies in the mainstream as standard options for tomorrow's Modelica tools, so that they can keep up with the challenges posed by large-scale system models.

## 4.1 Sparse Solvers

A literature search on the topic of using sparse ODE solvers for the simulation of Modelica models found surprisingly little relevant results. To the author's knowledge, the only really relevant reference is a Modelica Conference paper (Link et al., 2009), where the authors advocated the use of sparse DAE solvers to improve the performance of power plant simulation in Modelica. Apparently, they have not been listened to so far.

To further motivate the analysis carried out in Section 2.1, Table 2 reports the simulation results of the SimpleAdvection model from the ScalableTestSuite library. The model is linear and the causalization can be performed exclusively by forward assignments, so that the Hessian inversion quickly becomes the performance bottleneck. Dymola 2015 FD01 has been used to run the test on a laptop with an Intel i5-4200U CPU and 8 GB of RAM, using DASSL as the ODE solver. Similar results have been obtained using OpenModelica on the same machine, with the same ODE solver.

The simulation time scales up as $O(N^{2.6})$ and it is really hard to believe that much better results couldn't be obtained with the sparse matrix version of DASSL. Also note the disproportionate amount of memory allocated by the simulation process (over 1 GByte for the largest model), mostly to accommodate the Jacobian and Hessian matrix values. Finally, note that in most scientific computing circles, a dynamic model with 12800 state variables is considered a small one, not a very large one.

The other application for sparse solvers is in models with large linear algebraic systems of equations. For very large systems, tearing takes too much time during code generation and leads to a set of residual equations which is still very sparse. Sparse numerical solvers should instead be automatically selected and used in these cases.

## 4.2 Multi-Rate Algorithms

In all those cases showing localized activity and/or widely different time scales, multi-rate algorithms can improve the simulation performance dramatically and increasingly with the system size.

Multi-rate algorithms have been studied since the early 1960s, but never made it into the mainstream sim-

DOI 10.3384/ecp15118459

**Table 1.** Sparsity of some large-size models.

| *Model* | *# of states* | *# of non-zero Jac. entries* | *Jac. density* |
|---|---:|---:|---:|
| Transmission line 320 elements | 642 | 1603 | 0.38% |
| Transmission line 640 elements | 1282 | 3203 | 0.19% |
| Steam pipe advection 320 volumes | 640 | 3194 | 0.78% |
| Steam pipe advection 640 volumes | 1280 | 6394 | 0.39% |
| Heat exchanger 320 volumes | 957 | 3505 | 0.38% |
| Heat exchanger 640 volumes | 1917 | 7025 | 0.19% |
| String models 32 segments | 66 | 2147 | 49.29% |
| String models 64 segments | 130 | 8387 | 49.63% |

**Table 2.** Simulation performance of the SimpleAdvection model.

| *Model* | *# of states* | *Simulation time [s]* | *Memory allocation [MB]* |
|---|---:|---:|---:|
| Simple advection, 1600 nodes | 1599 | 6.7 | 22 |
| Simple advection, 3200 nodes | 3199 | 30.8 | 84 |
| Simple advection, 6400 nodes | 6399 | 183 | 326 |
| Simple advection, 12800 nodes | 12799 | 979 | 1293 |

ulation tools. The basic idea behind them is that only a few *global* steps involving the entire system should be performed. If the error estimates after one such step exceed the set tolerance only for a sub-set of states (i.e., the fast ones, or the ones with localized activity), then the time step grid is refined for those states only (the *active* states), while interpolating the found result for the *latent* states, whose precision has already been achieved with the global step. This approach can also be applied recursively.

The end result of using multi-rate algorithms is that whenever localized activity and/or transients in a fast subsystem take place, refinement steps are taken which only involve the (small!) relevant portion of the system, avoiding useless computation of other state derivatives, and also only requiring the inversion of small Hessians, in case of implicit solvers. Of course the advantage of using these algorithms grows with the size of the system.

On-going work at Politecnico di Milano is aimed at developing a multi-rate version of the TR-BDF2 algorithm, which is particularly attractive as the coefficients to interpolate the latent state are a by-product of the solution process, so they don't need any additional overhead. Very promising results have already been obtained using a multi-rate version of Rosenbrock's algorithm, applied to the district heating model of the ScalableTestSuite library. In particular, while the simulation time using DASSL grows as $O(N^{2.6})$, the simulation time of the prototype versions of multi-rate algorithms written in Matlab grows in a range from $O(N^{1.3})$ to $O(N^{1.8})$. Although the performance should be evaluated on an efficient version of the code written in C, the shown trend is nevertheless pointing in the right direction.

Also on-going at the moment of writing this paper is work at Bielefeld University to interface such solver to the OpenModelica compiler.

Please refer to (Ranade and Casella, 2014) for a more in-depth discussion of multi-rate algorithms, references to relevant literature, and preliminary results. Refer to (Casella, 2015) for ideas on how to generate efficient code to support such algorithms starting from declarative DAE-based Modelica models.

## 4.3 Smart Multi-Rate Event Handling

As discussed in Section 2.4, a straightforward implementation of the algorithm sketched in Appendix C of the Modelica Specification guarantees the correctness of the simulation results, but can be extremely inefficient in the case of large systems with frequently spaced events.

Some ideas have already been explored in the literature. Sanz et al. (2014) discuss how to avoid useless event iterations when it can be established a priori that a further event iteration is not necessary; they also discuss how to limit the event iterations to the smallest necessary sub-set of equations. In (Höger, 2013), a somewhat similar analysis is carried out to avoid unnecessary function evaluation during root finding, when the exact time instant of a state event is being computed. In both cases, the advantage grows with the size of the system.

All adaptive step-size solvers with error control of order $n$ assume that the right-hand side of the ODEs is $n$ times continuously differentiable. Whenever an event is triggered in a Modelica model, some derivatives can change discontinuously, which violates the assumption of continuous differentiability. The standard approach is to integrate the equations up to the event time, process the event determining the new initial conditions, then restart the integration from these initial conditions completely disregarding the previous results. As said above, this approach is safe but usually very inefficient.

A less naive solution strategy could be worth exploring. Assume that the variable $v$ with discontinuous behaviour due to events only influences the derivative of one state variable $x_1$, that in turn $x_1$ only influences the derivative of $x_1$ and $x_2$, and those in turn only influence the derivative of $x_1$, $x_2$, and $x_3$. It follows that $x_1$ is continuous (though not differentiable), $x_2$ is continuously differentiable, and $x_3$ is twice continuously differentiable. By assumption, the rest of the system is not influenced by $v$, $x_1$, and $x_2$, so its inputs are twice continuously differentiable despite the event. Consequently, a second-order error estimation algorithm would give reliable indications about the error on all the states other than $x_1$, $x_2$, and $x_3$ across time step including the event.

Assume now that during a global step the zero-crossing function of the event changes sign, but the error estimator remains below the tolerance for all these other states. In the context of a multi-rate algorithm, it could then be possible accept the global step without any event handling, and only process the event in a refinement step, involving only the small set of active variables $\{x_1, x_2, x_3\}$.

In other terms, assuming that there is enough low-pass action between the discontinuous variables and the bulk of the system states, it might not be necessary to stop the integration of the latter ones, but only to refine the solution of those states which are more directly affected by the discontinuity.

There are of course a lot of details to be worked out to implement this approach, but it is the author's opinion that the performance improvements could be dramatic for most large models of systems subject to digital control; even more so if there are different sub-systems with different time scales and different sampling rates of the corresponding controllers.

## 4.4 QSS Algorithms

Quantized State Systems methods (Cellier and Kofman, 2006) adopt an alternative strategy for the solution of ODE systems, i.e., instead of discretizing over time they discretize over the set of state values, thus turning ODE systems into Discrete EVent Systems (DEVS). Second- and third-order accurate methods QSS2 and QSS3 have been developed, as well as a linearly implicit method LIQSS (Migoni et al., 2013), which can deal with stiff system, a key feature for generic Modelica models.

Two features of QSS algorithms are relevant in the context of this paper. The first is that these algorithms naturally exploit localized interaction, as discussed in Section 2.1, handling therm very efficiently, as demonstrated by Floros et al. (2014). The second is that, as soon as the continuous-time ODEs are turned into event-based systems, the handling of events blends in the framework seamlessly without any significant overhead. Thus, also the issue raised in Section 2.4 can be solved efficiently by these algorithms.

Experiments have been already made at handling Modelica models with QSS algorithms.

Floros et al. (2011) report a first attempt at implementing an interface to a PowerDEVS-based QSS solver in the OpenModelica back-end. Unfortunately, this is no longer supported by OpenModelica due to later back-end refactoring.

Later on, the same research group developed a standalone QSS solver with a interface based on a small subset of Modelica ($\mu$-Modelica), and a back-end module for OpenModelica that generates $\mu$-Modelica code from regular Modelica code, see (Bergero et al., 2012).

Although these early experiments have lead to interesting published results, to the author's knowledge, as of the time of writing of this paper there is still no mainstream, well-tested and maintained tool that can process generic Modelica models, possibly involving advanced language features, and successfully generate simulation code based on QSS. Further work is required in this direction to consolidate the above-mentioned results.

## 4.5 Exploiting repetitive structures

Interesting ideas have been published about methods to avoid flattening the models to the level of scalar equations and variable, exploiting for loops or hierarchical model structure, see e.g. (Zimmer, 2009; Höger, 2011; Arzt et al., 2014). These methods may lead to considerable savings in the memory footprint of the simulation executable, as well as to considerable speed-up in the compilation and structural analysis phases.

However, at the time of the writing of this paper, these methods have only been demonstrated by prototype implementations, but are still unavailable in mainstream Modelica tools. More development, implementation and testing work is required to make them standard features of state-of-the-art Modelica tools.

## 4.6 Exploiting parallel CPUs

Research work on the parallelization of simulation code generated from Modelica models started as early as Peter Aronsson's PhD work (Aronsson, 2006).

The field is now finally approaching maturity. Parallel simulation code generation recently became available as an advanced feature in Dymola 2015 FD01, using algorithms described by Elmqvist et al. (2014), so it has already passed the stage of prototype implementation and entered the stage of advanced feature in at least one Modelica tool.

Two prototype parallel code generation back-ends in OpenModelica are documented in the literature:

(Sjölund et al., 2013), mainly focusing on TLM-based partitioning, and (Walther et al., 2014), based on ideas from Casella (2013). A new infrastructure to implement parallelized code is currently being built in the same tool, as documented by Gebremedhin and Fritzson (2014). To the author's knowledge, no other published information is available concerning other Modelica tools.

The author hopes that this kind of algorithms eventually becomes a standard feature that does not require special settings by end user - the tool should figure out autonomously what is the best configuration for the problem at hand, given the available hardware resources (i.e., number of cores), and the exploitation of parallelism should become transparent to the end user, as most other low-level details of symbolic and numerical processing.

All the above-mentioned works exploit the parallelism inherent in the causalization process. It was clearly pointed out in the most recent papers that the problem of clustering the atomic tasks into bigger ones is essential to avoid excessive task set-up and switching overhead, which could more than offset the gain obtained by parallelism. Some heuristics are needed to estimate the actual workload, which is one key ingredient of clustering algorithm. One may expect that these heuristics will become more mature and reliable in the next few years.

Another useful feature when dealing with implicit solvers is the parallelization of the computation of the Jacobian $\partial f/\partial x$. This is almost trivial to achieve in the cases when the Jacobian is computed numerically, and also trivially obtained when the Jacobian is computed symbolically, once the problem of computing $f(x)$ efficiently has been solved. Significant speed-ups could be obtained here, in particular for large models.

As a side remark, it is surprising that the performance test whose results are reported in the above-mentioned references are carried out with low-end computers with few parallel cores, such as laptops. It would be interesting to see what kind of speed-up factors can be obtained if high-end workstations with latest-generation with 20+ logical cores are employed. This will give some understanding on what will be possible with run-of-the-mill hardware within 3-6 years.

As a final remark, it is the author's opinion that parallel computation strategies should be combined with appropriate techniques exploiting sparsity and locality of large-scale Modelica models, in order to obtain truly outstanding performance improvement. Brute-force speed-up by parallelization is not able by itself to offset the inefficiencies pointed out in Section 2, even when larger numbers of cores will eventually become available at low cost on standard workstations.

## 5 Conclusions

In the future, large-scale system-level models of smart grids and distributed cyber-physical systems will become a strategic asset in the development of such systems. Although the Modelica language has a lot of potential in this field, current state-of-the-art Modelica tools employ methods and algorithms that suffer from fundamental limitations as the size of the system model increases, quickly leading to unsatisfactory performance even for moderately large models. In order to meet the challenges posed by large, hybrid, distributed models, fundamental advances are required in the integration methods and also in the structural analysis and compilation phases.

This paper tries to draw the attention of the Modelica community on this topic, highlighting some fundamental issues, pointing out promising research trends that have potential to solve them effectively, and urging tool developers to pursue the goal of efficient simulation of large-scale models with determination.

Finally, the paper introduces the ScalableTestSuite Modelica package, a library of scalable models for benchmarking existing tools and helping the development of innovative methods and algorithms to cope with large-scale Modelica models. Contributions are welcome to improve and expand the scope of this library, in order to make it the reference collection of benchmarks.

## 6 Acknowledgement

The author thanks his former master's student Kaan Sezginer for his contribution in developing version 1.0 of the ScalableTestSuite model library.

## References

P. Aronsson. *Automatic Parallelization of Equation-Based Simulation Programs*. PhD thesis, Linköping University, Department of Computer and Information Science, 2006.

Matthias Arzt, Volker Waurich, and Jörg Wensch. Towards utilizing repeating structures for constant time compilation of large Modelica models. In David Broman and Peter Pepper, editors, *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 35–38, Berlin, Germany, Oct 10 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666207.

F. Bergero, X. Floros, J. Fernandez, E. Kofman, and F. Cellier. Simulating modelica models with a stand-alone quantized state systems solver. In *Proceedings 9th International Modelica Conference*, pages 237–246–442, Munich, Germany, Sep. 2012. Modelica Association. doi:10.3384/ecp12076237.

Francesco Casella. A strategy for parallel simulation of declarative object-oriented models of generalized physical networks. In Henrik Nilsson, editor, *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)*, pages 45–51, Nottingham, UK, Apr 19 2013. ISBN 978-91-7519-621-3. URL `http://www.ep.liu.se/ecp/084/006/ecp13084006.pdf`.

Francesco Casella. Efficient computation of state derivatives for multi-rate integration of object-oriented models. In I. Troch F. Breitenecker, A. Kugi, editor, *Proceedings 8th Vienna International Conference on Mathematical Modelling*, pages 262–267, Vienna, Austria, Feb. 18–20 2015.

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

Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. Parallel model execution on many cores. In *Proceedings of the 10 th International ModelicaConference*, Lund, Sweden, Mar. 10-12 2014. The Modelica Association. doi:10.3384/ECP14096363.

X. Floros, F. Bergero, F. Cellier, and E. Kofman. Automated simulation of modelica models with qss methods - the discontinuous case. In *Proceedings 8th International Modelica Conference*, pages 657–667, Dresden, Germany, Mar 20-22 2011. Modelica Association.

Xenofon Floros, Federico Bergero, Nicola Ceriani, Francesco Casella, Ernesto Kofman, and François Cellier. Simulation of smart-grid models using quantization-based integration methods. In Hubertus Tummescheit and Karl-Erik Årzén, editors, *Proceedings 10th International Modelica Conference*, pages 787–797, Lund, Sweden, Mar 10-12 2014. The Modelica Association. ISBN 978-91-7519-380-9. doi:10.3384/ECP14096787.

Jens Frenkel, Christian Schubert, Günter Kunze, Peter Fritzson, Martin Sjölund, and Adrian Pop. Towards a benchmark suite for Modelica compilers: Large models. In *Proceedings 8th International Modelica Conference*, pages 143–152, Dresden, Germany, Mar 20-22 2011. Modelica Association.

Mahder Gebremedhin and Peter Fritzson. Automatic task based analysis and parallelization in the context of equation based languages. In David Broman and Peter Pepper, editors, *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 49–52, Berlin, Germany, Oct 10 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666210.

C. Höger. Separate compilation of causalized equations. In *Proceedings 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools.*, pages 113–120, Sep. 2011.

Christoph Höger. Sparse causalisation of differential algebraic equations for efficient event detection. In *Proceedings of the 8$^{th}$ EUROSIM Congress on Modelling and Simulation*, pages 351–356, Washington, DC, USA, 2013.

K. Link, H. Steuer, and A. Butter. Deficiencies of modelica and its simulation environments for large fluid systems. In *Proceedings 7th International Modelica Conference*, pages 341–344, Como, Italy, Sep. 20–22 2009. The Modelica Association. ISBN 978-91-7393-513-5. doi:DOI: 10.3384/ecp09430034.

G. Migoni, M. Bartolotto, E. Kofman, and F. Cellier. Linearly implicit quantization-based integration methods for stiff ordinary differential equations. *Simulation Modelling Practice and Theory*, 35:118–136, 2013. doi:10.1016/j.simpat.2013.03.004.

Akshay Ranade and Francesco Casella. Multi-rate integration algorithms: a path towards efficient simulation of object-oriented models of very large systems. In David Broman and Peter Pepper, editors, *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 79–82, Berlin, Germany, Oct 10 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666214.

Victorino Sanz, Alfonso Urquia, and Francesco Casella. Improving efficiency of hybrid system simulation in modelica. In David Broman and Peter Pepper, editors, *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 21–28, Berlin, Germany, Oct 10 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666205.

Francesco Schiavo, Luca Viganó, and Giann Ferretti. Object-oriented modelling of flexible beams. *Multibody System Dynamics*, 15(3):263–286, 2006.

K. Sezginer and F. Casella. The ScalableTestSuite Modelica library, 2015. URL `https://github.com/casella/ScalableTestSuite`.

Kaan Sezginer. A test suite of large scalable models for Modelica tool evaluation. Master's thesis, Politecnico di Milano, April 2015. Supervisor: prof. F. Casella.

Martin Sjölund, Mahder Gebremedhin, and Peter Fritzson. Parallelizing equation-based models for simulation on multi-core platforms by utilizing model structure. In Alain Darte, editor, *Proceedings of the 17th Workshop on Compilers for Parallel Computing*, July 2013.

The Modelica Association. Modelica - A unified object-oriented language for physical systems modeling - Language specification version 1.0. Online, Sep 1997. URL `http://www.modelica.org/news_items/documents/ModelicaSpec30.pdf`.

The Modelica Association. Modelica - A unified object-oriented language for physical systems modeling - Language specification version 3.3 revision 1. Online, Jul. 11 2014. URL `https://www.modelica.org/documents/ModelicaSpec33Revision1.pdf`.

Luigi Vanfretti, Tetiana Bogodorova, and Maxime Baudette. A Modelica power system component library for model validation and parameter identification. In *Proceedings 10th International Modelica Conference*, pages 1195–1203, Lund, Sweden, Mar 10-12 2014. The Modelica Association. doi:10.3384/ECP140961195.

Marcus Walther, Volker Waurich, Christian Schubert, and Ines Gubsch. Equation based parallelization of modelica models. In *Proceedings of the 10 th International ModelicaConference*, Lund, Sweden, Mar. 10-12 2014. The Modelica Association. doi:10.3384/ECP140961213.

D. Zimmer. Module-preserving compilation of modelica models. In *Proceedings 7th International Modelica Conference*, pages 880–889, Como, Italy, Sep. 20–22 2009. The Modelica Association. doi:10.3384/ecp09430028.