Dymola Sparse Solvers for Large-Scale Simulations

Large-scale models with many states can be found in several applications, especially when some form of spatial distribution is involved. Consider for example models of electric power distribution or high-fidelity descriptions of the thermal convection of steam.

Using traditional solvers for these models is often not feasible since the run times and memory requirements scale poorly with the size of the simulated system.

However, such models are typically sparse since they mainly have local interaction between states. Intelligent solvers can utilize this sparsity for efficient simulation of very large models.

In Dymola 2018 FD01 the numerical integrators have been enhanced with sparse solver capabilities.

Introduction

To enable efficient simulation of Modelica models Dymola applies advanced symbolic and numeric techniques to break down and compact models. This process, which takes place during translation, often results in smaller systems with comparatively few but tightly coupled states. However, for many models a large number of states still remain.

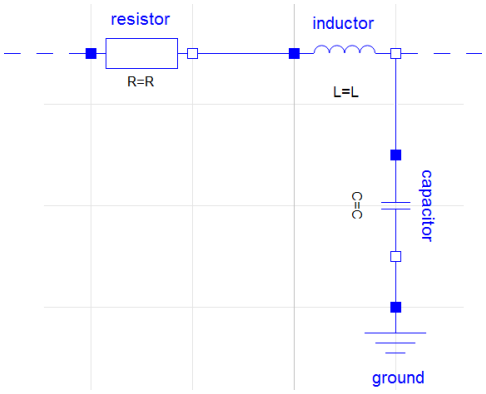
To simulate the translated model sophisticated solvers like DASSL, Radau, or CVode are typically used. At the core of these integrators several linear systems of equations are solved. These systems generally involve all the model's states and can thus become big when large-scale models are simulated.

The traditional approach to solving a linear equation system takes work effort that grows as n^3, where n is the number of states. Additionally, one or more matrices with the size of at least n x n must be stored. At model sizes of about one thousand states the solution of these linear equation systems often becomes a major bottleneck in the overall simulation procedure.

In many cases sparse linear equation solvers can be employed to remedy this problem. These solvers are very efficient when each state of the system only has direct interaction with a few other states. This typically means that the required computation effort and the amount of data to be stored scale linearly with the number of states. Furthermore, sparse linear equation solving can be easily parallelized for even faster simulations.

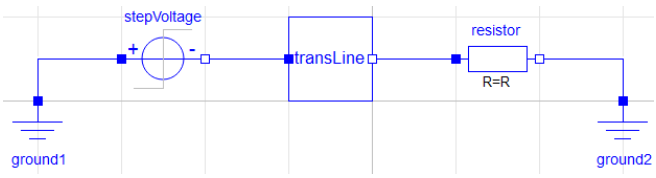
An example application

As a prototypical example consider a simple model of an electrical transmission line. It consists of N segments, each built up from a resistor, an inductor, and a capacitor, connected as in the following figure.



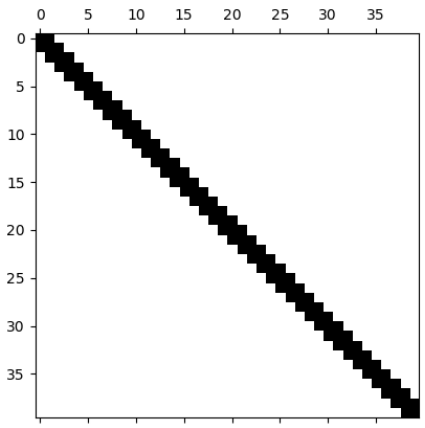
The negative (right) pin of the inductor in one segment is connected to the positive (left) pin of the resistor in the next segment. The accuracy of the model depends on the number of segments.

Finally, to test the transmission line it is connected to a step voltage source at the start of the line and a resistor at the end.



Since each component is only connected to one or two other components the direct interaction between states is localized. This is mirrored by the sparsity pattern of the linear systems that are solved at the core of for example DASSL.

With $N = 20$ segments we get 40 states after translation and the following incidence matrix.



The dark squares represent matrix elements that may be different from zero, the rest are zero throughout the simulation.

Sparse linear equation solvers use this pattern and only do work where it is required. In contrast, traditional solvers work with all 400 matrix elements.

As N increases we still only have three non-zero elements per row making the benefits of sparse solvers even greater. This is illustrated by the concluding efficiency experiments.

Activating sparse solvers in Dymola

It is often difficult to decide beforehand which will be faster; dense or sparse algebra. To help with the choice Dymola makes an automatic estimate during translation. If Dymola deems the translated model to be suitable for sparse solvers it issues the following message in the translation log.

> Sparse solver handling possible: false.
> Due to flag Advanced.SparseActivate=false.
> Model sparse and large enough: true.
> Sparse solvers are available for dassl, lsodar, cvode, radau, esdirk*, sdirk* (using OpenMP, set number of cores with Advanced.NumberOfCores).

Sparse solvers can then be activated by setting:

```
Advanced.SparseActivate = true
```

It is by default set to false. The simulation can then be started as normal using one of the listed methods.

Methods not listed are explicit and thus do not need to solve large equation systems at their core. All of Dymola's implicit solvers support sparse linear algebra in Dymola 2018 FD01.

By default one processor core is used. Depending on the model and the computer running the simulation faster run times may be achieved through parallelization. The number of cores to be used for matrix factorization can be specified by setting:

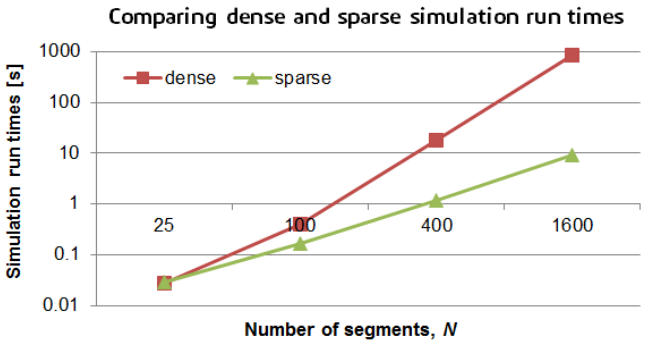
```
Advanced.NumberOfCores
```

Due to small variations in round-off errors between dense and sparse linear algebra some solvers may use slightly different number of steps, function evaluations, and Jacobian evaluations when sparse is activated.

Finally note that sparse linear algebra is only applied to the systems of equations that arise inside the integrators, for example inside DASSL. This is not to be confused with the systems of equations inside the model, that is, those whose sizes are listed under *Statistics* in the translation log. Those systems are typically small and are treated in the traditional fashion.

Efficiency experiments

To illustrate the efficiency of sparse linear algebra the previously discussed transmission line model is simulated using Radau. Test cases with an increasing number of segments are considered and simulation run times are measured when dense and sparse algebra is employed, respectively. The results are presented in logarithmic scale in the below diagram.



Conclusion

In Dymola 2018 FD01 the numerical integrators have been enhanced with powerful sparse solver capabilities which allows for efficient simulation of large-scale models using one or several cores.