Fast Simulation of Fluid Models with Colored Jacobians

Willi Braun[a] Stephanie Gallardo Yances[b] Kilian Link[b] Bernhard Bachmann[a]
[a]University of Applied Sciences Bielefeld, Bielefeld, Germany
[b]Siemens AG, Energy Sector, Erlangen

Abstract

The industrial usage of the open-source Modelica tool OpenModelica was limited so far for power plant applications, due to the performance of large fluid systems. This paper presents some efforts to improve the simulation time on benchmark fluid models proposed by Siemens Energy. The main aspects presented here to achieve a faster simulation are an efficient evaluation of the jacobian matrix by a coloring technique, that exploits the sparsity pattern of a modelica model. Therefore the techniques are scratched and applied to benchmark models provided by Siemens Energy.

Keywords: OpenModelica, Fluid Simulation, Benchmark, Simulation, Jacobian, Coloring, Sparsity-Pattern, DASSL

1 Introduction

In power plant applications, detailed analysis of the dynamic behaviour of heat recovery steam generators result in very large fluid systems.

Modelica is the preferred modeling language for dynamic simulations within Siemens Energy [5] due to its applicability for multi-domain modeling of physical systems, the high degree of maintainability of Modelica models and the possibility of rapid development of new components in Modelica.

The commercial tool Dymola is mainly used for modeling and simulation. The open-source Modelica enviroment OpenModelica for industrial and academic usage is getting more and more an alternative and has the large benefit that it is freely available. Fluid modeling with Openmodelica was limited by missing implementation of some special features like Modelica.Media. The OpenModelica compiler flattens now the complete Modelica.Media library. Nevertheless the missing functions are still replaced in all benchmark models by external libraries. In order to make OpenModelica an established Modelica tool, the accuracy and performance have to be comparable with Dymola.

The aim of the current paper is to present the improvement of the simulation time for special benchmark fluid models using an efficient technique to evaluate jacobians. The benchmark fluid models are developed by Siemens AG, Energy Sector, using the commercial Modelica environment Dymola. Siemens Energy has presented fluid models before, which are suitable for the benchmark of the accuracy and the performance of a Modelica Tool. The complexity of these models have been further refined to build up realistic plant models like used in daily business and to reach model sizes which are suitable for performance tests. On the other hand University of Applied Sciences Bielefeld has developed techniques to generate symbolic jacobians in OpenModelica before ([4],[3]). The derivatives are useful for simulating a model as well as for the sensitivity analysis or the optimization of models. Further, jacobians are necessary to support the next FMI[1] version 2.0 [1]. In the work before it was not possible to show improvements for the simulation. This can be explained mainly by the model size we had tested our implementation on, this was caused by the fact that the generation of symbolic jacobians was not applicable to large scale models. This is solved by generating generic partial derivatives and utilise them to compute the full jacobains. Here we catch up and apply the generation of symbolic jacobians on large scale models provided by Siemens Energy [6].

The paper is structured as follows: In section 2 the usage of the jacobian for the simulation purpose is specified. Further, the coloring and the determination of the sparsity pattern are stated and the application of the coloring to the solving process is described. In section 3 there are given some information about the used benchmark fluid models. Whose perfomance is measured in section 4. Section 5 summarizes the results of this paper and gives proposals for future work.

[1] http://fmi-standard.org/

2 Jacobian for Simulation

A Modelica model is typically translated to a basic mathematical representation of differential and algebraic equations (DAEs), before being able to simulate the model. Further, these DAEs are transformed to ODEs (ordinary differential equations) with an algebraic part, which is the starting point.

$$\begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} \underline{h}(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ \underline{k}(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix} \tag{1}$$

The jacobian of interest for simulation purpose consists of partial derivatives of the ODE-Block h with respect to the states.

$$J_A = \frac{\partial \underline{h}}{\partial \underline{x}} = \begin{pmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_n} \end{pmatrix} \tag{2}$$

For solving equation 1 with an integration method like DASSL, the derivatives are needed with respect to the states $\underline{x}(t)$ [7]. After all, DASSL uses the iteration matrix

$$M = \frac{\partial \underline{h}}{\partial \underline{x}} + cj * \frac{\partial \underline{h}}{\partial \dot{\underline{x}}} \tag{3}$$

for solving a nonlinear system in each step by a modified newton method. This matrix M is almost the same as the partial derivatives with respect to the states beside the $cj * \frac{\partial \underline{h}}{\partial \dot{\underline{x}}}$ part. But that part is the identity matrix multiplied with a scalar value calculated by DASSL. By default DASSL calculates the iteration matrix M by means of numerical finite differentiation. Therefore it is necessary to evaluate the ODE function h $n+1$ times. However, it is also possible to equip DASSL with an user-specific routine that provides manually calculated iteration matrix M. Considering issues of performance, the calculation of M is the most critical part. In table 1 are summarized the results for one simulation of two different benchmark models (see 3), where are denoted t_s as simulation time, J_{evals} as number of jacobain evaluations and J_{time} as time of evaluation of the jacobian J_{evals} times. One can see that the calculation of the jacobian matrix takes the major time of the simulation time.

N	x	eqns	N	x	eqns
19	231	942	10	140	826
t_s	J_{evals}	J_{time}	t_s	J_{evals}	J_{time}
10.8	111	9.7	2.4	69	1.4

Table 1: Simulation times vs. Jacobian evaluations

So at that point it's possible to reduce the DASSL solving time. It is quite evident that this could be tackled by exploiting the sparse structure of a Modelica Model. One approach which uses the sparsity pattern to reduce the amount of ODE-function calls is the partitioning of columns in colors and calculating them at once [2]. Additionally the matrix M can be determined in a symbolic way and combined with the coloring approach.

Therefore we test 4 different methods to calculate the jacobians:

- finite difference approximations.
- finite difference approximations with coloring.
- symbolical jacobian generated by OpenModelica.
- symbolical jacobian generated by OpenModelica with coloring.

For the numerical approximation of the jacobian the forward finite differentiation is used, where h is determined by DASSL and it depends on x, $\dot{y}$, current step size.

$$\dot{y} = \frac{f(x+h) - f(x)}{h} \tag{4}$$

The symbolical jacobians are generated within the OpenModelica compiler (for more details see [3],[1]).

2.1 Coloring Jacobians

The coloring of a matrix means first of all to color columns that have no non-zero-elements in the same row. Thus, the starting point for coloring is the sparsity pattern of a matrix. The determination of the sparsity pattern of a Modelica model is described in the next section 2.2.

Assuming the matrix J with it's sparsity pattern is given as:

$$J = \begin{pmatrix} j_{11} & 0 & 0 & 0 & j_{15} \\ 0 & j_{22} & j_{23} & 0 & 0 \\ j_{31} & j_{32} & 0 & 0 & 0 \\ 0 & 0 & j_{43} & 0 & j_{45} \\ 0 & 0 & 0 & j_{54} & j_{55} \end{pmatrix} \tag{5}$$

In this matrix J for example the columns 1 and 3 and also the columns 2 and 4 have no shared non-zero elements in the rows. Thus, this columns could be calculated at once, since they are structural orthogonal. Finding those structural orthogonal rows could be done by re-formulating the problem as graph coloring of a bipartite graph. The bipartite graph $G =$

DOI 10.3384/ecp12076247

$((V_1, V_2), E)$ consists of vertexes V_1, V_2, where V_1 are all rows and V_2 are all columns. And for every non-zero element an edge e_i is defined between the involved row and the corresponding column, vice versa. For the matrix above the corresponding bipartite graph is drawn in figure 1.

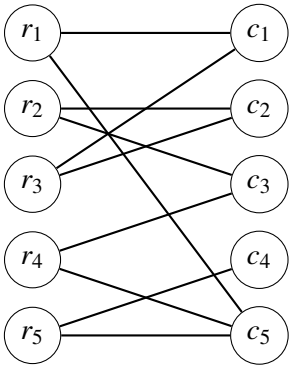

Figure 1: Bipartite graph G

Next step is coloring the column vertexes with the minimum number of colors, so that no row vertex has a connection to columns with the same color. This problem is well-known as NP-hard [2], but for the current purpose it's not very critical to find the optimum, so a fast approximation is well-suited. Therefore a modified partial distance-2 coloring algorithm for bipartite graphs is used as suggested also in [2]. In our tests it reveals a good performance meaning that the solution was really close to the chromatic number $\chi(G, V_2)$, which describes the optimal solution. This observation could be done since there exists a lower bound for $\chi(G, V_2)$. It is also shown in [2] that $\chi(G, V_2) \geq \Delta V_1$ is true. This sounds intuitional for the reason that the minimal partition size depends on the maximum number of non-zero elements in the rows. The time complexity for the algorithm is $O(|E| * \Delta V_1)$, where ΔV_1 is the maximum degree of the vertex $v_i \in V_1$. For example in the jacobian above, it's easy to see that there are several possible solutions as shown in figure 2.

After a coloring C of the columns is found, it's possible to apply it to the calculation of the jacobians. Now all columns with the same color are structural orthogonal and can be calculated at once. Therefore the expected speed up for the calculation is speedup $= \frac{|V_2|}{C}$.

2.2 Sparsity Pattern

The sparsity pattern for J_A (see equation (2)) of a Modelica Model could also be determined by means of graph theory, because roughly spoken the sparsity

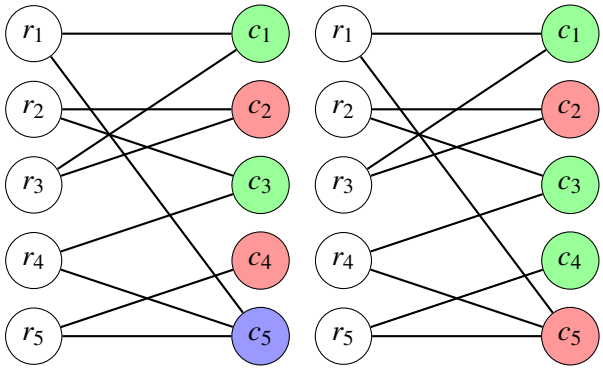

Figure 2: Bipartite graph G

pattern expresses which output variable has a connection to which state. So this could be formulated as a st-connectivity problem in a directed graph. The st-connectivity is a decision problem that asks if the vertex t is reachable from the vertex s. A directed graph is also naturally used in a Modelica tool for the sorting of the equations with the tarjan algorithm. For example if one has a system with 5 equations, and 5 states a directed graph for sorting could look like the one in figure (3).

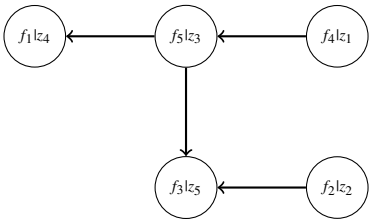

Figure 3: Directed graph for sorting the example system

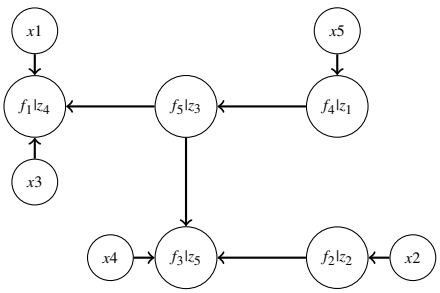

Figure 4: Expanded directed graph for sorting the example system

For the ordinary sorting task by tarjan only the unknowns are considered, since the states are assumed to be known. So for the determination of sparsity pattern one would need to expand the graph by the states. This is done in the way that every equation vertex gets an

additional incoming connection by the states that are present in it. Finally the directed graph could look like the one in figure (4).

The sparsity pattern in equation (6) could than be obtained by finding all reachable vertexes for every state. For every connection that could be found the corresponding element is unequal zero. Finding the reachable vertexes for one state results in one column of the sparsity pattern.

$$J = \begin{pmatrix} * & 0 & * & 0 & * \\ 0 & * & 0 & 0 & 0 \\ 0 & * & 0 & * & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix} \tag{6}$$

However, the determination of the sparsity pattern via st-connectivity would require to traverse the whole graph for every state, what is of course not applicable for a large system. Thus one could benefit from the already sorted system and also use additional information from the adjacency matrix. For example consider the following possible sorted adjacency matrix (7) for the system above with the expansion about the states and the equation where they occur.

$$\begin{array}{c} \\ f4 \\ f5 \\ f1 \\ f2 \\ f3 \end{array} \left(\begin{array}{ccccc||ccccc} z1 & z3 & z4 & z2 & z5 & x1 & x2 & x3 & x4 & x5 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{array} \right) \tag{7}$$

In this BLT-sorted adjacency matrix we consider row after row and propagate the dependent states downwards to every equation. The accumulation of the non-zero elements is arranged in an array of lists for every equation. For the first equation we just add the dependent states x_5 to the corresponding list. For the second equation there are no direct dependencies, but we need to propagate the dependencies for the involved variables. In this case for the variable z_1 which occurs in the first column the lists of f_5 and f_4 are joined. For the next row it is necessary to add the direct dependent variables x_1, x_3 and union them with the indirect dependencies from variable z_3 and so on. This approach results in algorithm with a complexity that depends on the amount of non-zero elements. Our tests indicate even a logarithmic dependence for non-zero elements. Thus the sparsity pattern can be determined efficiently.

3 Benchmark Fluid Models

The first benchmark model (see figure 5) consists of three heated pipes in a row. The first pipe in flow direction is connected to a water source which supplies the liquid flow. The one-dimensional energy, mass and momentum balances are discretized in flow direction. The number of nodes which represent the connection between the discrete elements is N. The heated metal wall of the pipe represents a cylindrical metal wall with L numbers of layers.

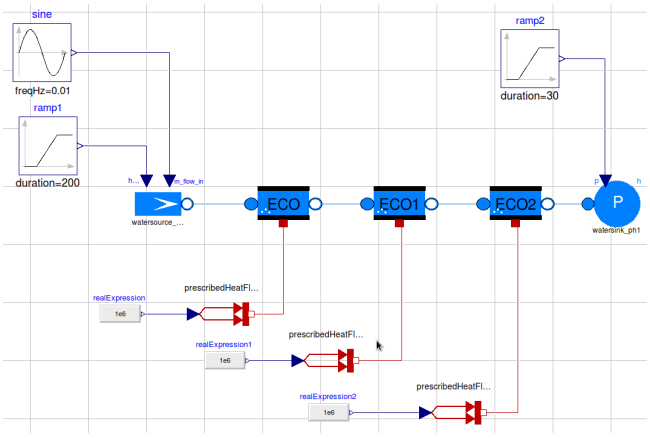

Figure 5: Pipes benchmark model

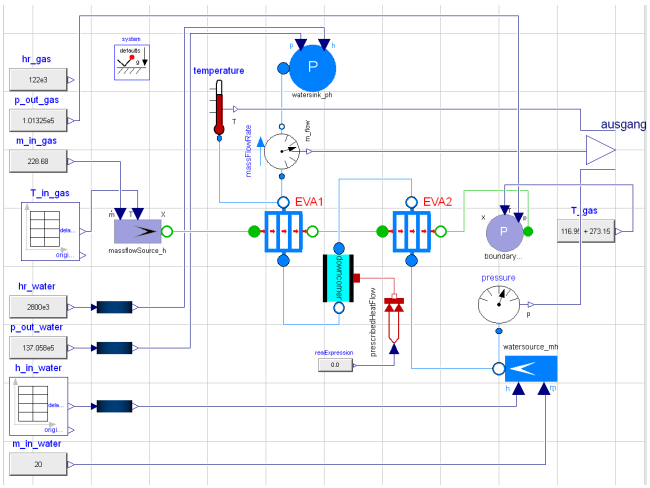

Figure 6: Heat exchanger benchmark model

The central part of our second more complex benchmark is an evaporator model (see figure 6) with parallel tube rows. A parallel flow evaporator consists of several heated tubes connected by an internal splitter at the inlet and an internal mixer at the outlet. For each of the N_l (number of parallel layers) exists a subaggregate which also models the gas-side, using a simple quasi stationary pressure drop. The water and steam flow and the inner heat transfer is modeled using the

DOI 10.3384/ecp12076247

	N	x	eqns	colors	N	x	eqns	colors	N	x	eqns	colors
	19	231	942	79	50	603	2430	203	100	1203	4830	403
method	steps	F-Eval	J-Eval	time	steps	F-Eval	J-Eval	time	steps	F-Eval	J-Eval	time
num	922	27184	111	10.8	1014	72854	118	85.3	1058	144874	119	372.3
numC	922	8929	94	4.5	1023	26914	124	38.4	1064	46835	112	144.2
sym	937	1539	103	8.5	976	1643	119	65.2	1052	1732	126	287.2
symC	937	1539	103	4.3	976	1643	119	30.3	1052	1732	126	139.3
Dymola	783	8772	90	1.6	915	23453	106	11.3	1035	43707	103	53.4

Table 2: Simulation time for Tube3Test

	N	x	eqns	colors	N	x	eqns	colors	N	x	eqns	colors
	40	500	2986	95	80	980	5866	175	160	1940	11626	335
method	steps	F-Eval	J-Eval	time	steps	F-Eval	J-Eval	time	steps	F-Eval	J-Eval	time
num	492	38192	75	23.3	537	79131	80	94.7	542	140390	72	436.5
numC	516	10841	106	9.9	505	13810	75	27.3	596	28595	83	152.4
sym	544	774	74	44.9	536	726	77	176.7	556	752	83	792.1
symC	544	774	74	11.9	536	726	77	42.8	556	752	83	206.8
Dymola	359	7306	69	7.36	387	12531	67	22.4	408	23964	69	142

Table 3: Simulation times for HeatExchanger

pipe model. The outer heat transfer is assumed to be constant. This model is suitable for building up huge systems with many states since the number of tube layers of the evaporator can be adapted easily. Compared to a complete and detailed heat recovery steam generator model the model in figure 6 is still small. This justifies the requirement to improve the performance to use in future OpenModelica for power plant simulations.

4 Performance Measurements

The performance measurements are done on a workstation machine(Intel CPU Q9550 @ 2.83GHz). For the time measurements we run the simulation five times take the mean, in addition the initialization process is deducted. Here are depicted the results for the benchmark models 3 with the four modes described above in OpenModelica. Additionally, the results are compared to Dymola. For all simulation was chosen a tolerance of $1e^{-6}$, which is propagate in OpenModelica as absolute and relative tolerance. This may be one reason for the difference in the steps performed by the Integrator.

In the top of the tables 2 and 3 are stated the model details, where the variable N is used for resizing the model resulting in numbers of states, equations for the ODE-function and the colors. The method called "num" calculates the jacobians numerically and the method "sym" performs it symbolically. The additional "C" marks that the coloring is applied to these methods.

First, it can be stated that the simulation time is effected a lot by the coloring as expected. The factor is a bit lower than expected due to the different number of steps and thus a different number of jacobian evaluation in each simulation. This can be considered as numerical artefacts which are propagated and then induce small differences in the step-size chosen by the integrator. This effect can't be observed for the symbolic solution. Further, it can be stated for the numerical solution the amount of ODE-function evaluation is reduced dramatically and it tends to be close to Dymola. This suggests that Dymola uses a similar techniques.

5 Conclusions

The aim of this paper was to show that one key element for a Modelica Tool to perform a fast simulation is the exploiting of the sparsity pattern for the determination of jacobians. Therefore it is necessary to determine the sparsity pattern and partition the jacobians calculation in order to reduce the evaluation time. This is realized by graph theoretical means in OpenModelica. Further it was shown on the presented benchmark models that

the effect is significant, moreover this feature pushes OpenModelica further to an efficient simulation environment for relevant industrial problems.

Acknowledgments

The German Ministry BMBF has partially funded this work (BMBF Förderkennzeichen: 01IS09029C) within the ITEA2 project OPENPROD (http://www.openprod.org).

References

[1] Åkesson J, Braun W, Lindholm P, Bachmann B. Generation of Sparse Jacobians in the Function Mock-Up Interface 2.0. In: Proceedings of the 9th Modelica Conference, Munich, Germany, Modelica Association, 2012.

[2] Assefaw H. Gebremedhin, Fredrik Manne, and Alex Pothen. What color is your jacobian? graph coloring for computing derivatives. SIAM Rev., 47(4):629–705, 2005.

[3] Braun W, Ochel L, Bachmann B. Symbolically Derived Jacobians Using Automatic Differentiation - Enhancement of the OpenModelica Compiler. In: Proceedings of the 8th Modelica Conference, Dresden, Germany, Modelica Association, 2010.

[4] Fritzson P. et. al.: OpenModelica System Documentation, PELAB, Department of Computer and Information, Linköpings universitet, 2010.

[5] Siemens Energy: https://www.energy.siemens.com

[6] Link K, Vogel S, Mynttinen I. Fluid Simulation and Optimization using Open Source Tools. In: In: Proceedings of the 8th Modelica Conference 2010, Dresden, Germany, Modelica Association, 20th to 22nd 2011 2010.

[7] Petzold L. R.: A Description of DASSL: A Differential/Algebraic System Solver, Sandia National Laboratories Livermore, 1982.