D6.2.5: Prototype implementation of parallel jacobian evaluation with multi-core compilation

WP 6.2: Efficient operation and simulation on multi-core platforms

Work Package 6: Modeling and simulation services

MODRIO (11004)

Version 2.0
Date 11/30/2015

Authors Willi Braun FH Bielefeld
Bernhard Bachmann FH Bielefeld

Executive Summary

The main aspect of this delivery is the parallel evaluation of the symbolical jacobian matrix to achieve a faster simulation. The current implementation is a prototype for parallel evaluation of symbolically generated jacobian matrices. The parallel evaluation is based on partial derivatives combined with a compression technique and the pThreads library. This prototype works with symbolical jacobian and the effect is demonstrated on an example where the evaluation time is dominated by jacobian evaluations.

Summary

The main aspect of this delivery is the parallel evaluation of the symbolical jacobian matrix to achieve a faster simulation. It does includes the efficient evaluation of the jacobian matrix by a coloring technique, that exploits the sparsity pattern of a Modelica model. This is done by generating generic partial derivatives and utilize them to compute the full jacobains. The current implementation is a prototype for parallel evaluation of symbolically generated jacobian matrices. that considers for now only the parallel evaluation but not the compilation. The parallel evaluation is based on partial derivatives combined with a compression technique and the pThreads library.

1 Introduction

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 $\underline{x}(t)$ [5]. Thus the parallelization of this matrix can lead to a speed-up of the solving process.

In general the derivative matrices calculation in OpenModelica is based on generic partial derivatives, since the generation of the full symbolic jacobian requires n-times differentiation of every equation and that is an unacceptable cost. The generic directional derivative is generated by differentiating every equation once with respect to a seed vector $z(\underline{e}_k)$.

$$J_A = \frac{\partial \underline{h}}{\partial z}(\underline{e}_k) \quad \underline{e}_k \in \mathbb{R}^n := k - \text{th coordinate vector} \tag{3}$$

Then the full symbolic jacobian can be calculated by evaluating $\frac{\partial \underline{h}}{\partial z}(\underline{e}_k)$ with every $\underline{e}_k \in \mathbb{R}^n$. At this point the amount of calls could be reduced by exploiting the sparsity pattern and partition the columns by colors. This technique is elaborated in section 2. Section 3 contains a short description of the current implementation status.

2 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 3.

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

$$J = \begin{pmatrix} j_{11} & 0 & 0 & 0 & j_{15} \\ 0 & j_{22} & j_{23} & 0 & 0 \\ j_{31} & j_{32} & 0 & 0 & 0 \\ 0 & 0 & j_{43} & 0 & j_{45} \\ 0 & 0 & 0 & j_{54} & j_{55} \end{pmatrix} \tag{4}$$

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 $G = ((V_1, V_2), E)$ 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.

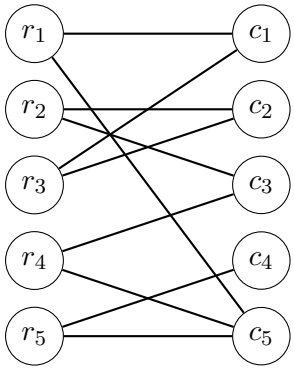

Figure 1: Bipartite graph G

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 [1], 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 [1]. 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 [1] 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.

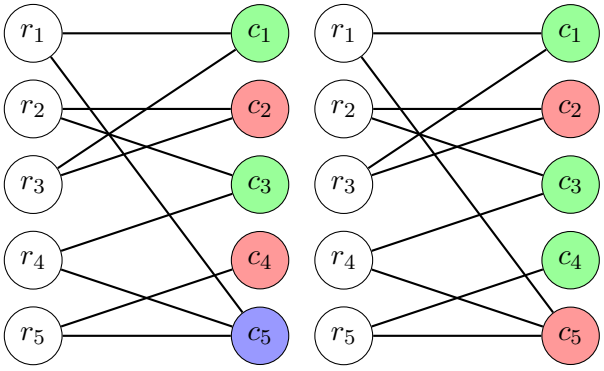

Figure 2: Bipartite graph G

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 $= \frac{|V_2|}{C}$.

3 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 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).

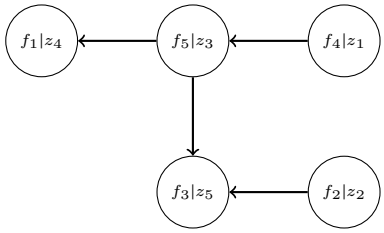

Figure 3: Directed graph for sorting the example system

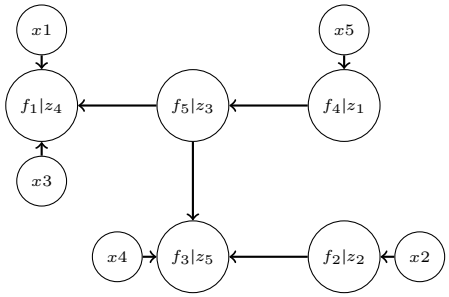

Figure 4: Expanded directed graph for sorting the example system

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 (5) 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.

$$J = \begin{pmatrix} * & 0 & * & 0 & * \\ 0 & * & 0 & 0 & 0 \\ 0 & * & 0 & * & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix} \tag{5}$$

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 (6) for the system above with the expansion about the states and the equation where they occur.

$$\begin{array}{c} \\ f4 \\ f5 \\ f1 \\ f2 \\ f3 \end{array} \begin{array}{c} \begin{array}{ccccc|ccccc} z1 & z3 & z4 & z2 & z5 & x1 & x2 & x3 & x4 & x5 \end{array} \\ \left(\begin{array}{ccccc||ccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right) \end{array} \tag{6}$$

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.

4 Parallel Evaluation Implementation and Results

The coloring technique as discussed in section 2 is also useful to discover computational subtasks for parallel evaluation of equation 3. All columns with the same color are compressed to one and evaluated at once. Thus every compressed column with a different color can be evaluated in parallel, because it is completely independent from the others.

The first part of this delivery was the preparation of a generic derivative module in OpenModelica, that is used for all differentiation tasks in OpenModelica. Further the generated c-code of OpenModelica was improved in sense of the thread-safeness.

The parallel evaluation is implemented with the means of the pThreads library. It allows a program to control multiple different flows of work that overlap in time. One an important point is to divide the work in appropriate bits to distribute them on the available threads, since to separate every

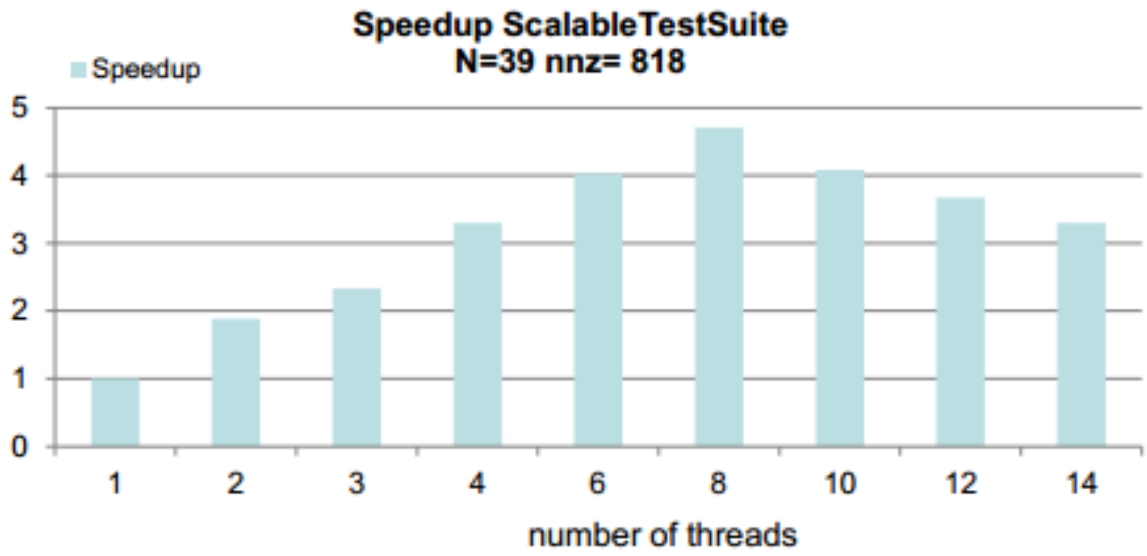

Figure 5: SpeedUp by activating parallel evaluation for a model

column on a single thread would produce to much overhead. This is solved by creating equal groups for every thread.

In figure 5 the speed-up for a simulation run with an in parallel evaluated symbolic jacobian is depict. In this model Listing 1 the evaluation of the symbolic jacobian is a dominated part of the whole simulation run. Thus the parallel evaluation has such an big effect of the overall time. Also there is a break-even point where the speed-up doesn't increase anymore, this is when the overhead costs for the parallel evaluation overrun the costs of one column evaluation.

References

[1] 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.

[2] Braun W, Gallardo Yances S, Link K, Bachmann B. Fast Simulation of Fluid Models with Colored Jacobians . In: Proceedings of the 9th Modelica Conference, Munich, Germany, Modelica Association, 2012.

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

Listing 1: Scaleable testcase of DistributionSystemModelica used with N=40 and M=40

```
model DistributionSystemModelica
  parameter Integer N = 4
    "Number of segments of the primary distribution line";
  parameter Integer M = N
    "Number of segments of each secondary distribution line";
  parameter Real alpha = 2 "Distribution line oversizing factor";
  parameter Modelica.SIunits.Resistance R_l = 1
    "Resistance of a single load";
  parameter Modelica.SIunits.Resistance R_d2 = R_l/(M^2*alpha)
    "Resistance of a secondary distribution segment";
  parameter Modelica.SIunits.Resistance R_d1 = R_l/(M^2*N^2*alpha)
    "Resistance of a primary distribution segment";
  parameter Modelica.SIunits.Voltage V_ref = 600 "Reference source voltage";

  Modelica.Electrical.Analog.Basic.Resistor primary[N](each R = R_d1)
    "Primary distribution line segments";
  Modelica.Electrical.Analog.Basic.Resistor secondary[N,M](each R = R_d2)
    "Secondary distribution line segments";
  Modelica.Electrical.Analog.Basic.Resistor load[N,M](each R = R_l)
    "Individual load resistors";
  Modelica.Electrical.Analog.Basic.Ground ground[N,M] "Load ground";
  Modelica.Electrical.Analog.Basic.Ground sourceGround "Source ground";

  Modelica.Electrical.Analog.Sources.RampVoltage V_source(V = V_ref, duration = 1)
    "Voltage source";
equation
  connect(primary[1].p, V_source.p);
  connect(sourceGround.p, V_source.n);
  for i in 1:N-1 loop
    connect(primary[i].n, primary[i+1].p);
  end for;
  for i in 1:N loop
    connect(primary[i].n, secondary[i,1].p);
    for j in 1:M-1 loop
      connect(secondary[i,j].n, secondary[i,j+1].p);
    end for;
    for j in 1:M loop
      connect(secondary[i,j].n, load[i,j].p);
      connect(load[i,j].n, ground[i,j].p);
    end for;
  end for;

  annotation (Documentation(info="<html>
<p>This model represnts a DC current distribution system, whose complexity depends on two
    parameters
N and M. A voltage source is connected to primary resistive distribution line which is split
    into
N segments, each with a resistance R_d1. At the end of each segment, a secondary distribution
line is attached with M elements each of resistance R_d2. At the end of each secondary segment,
a load resistor of resistance R_l is connected, which is grounded on the other side.</p>
</html>"));
end DistributionSystemModelica;
```