Selective Evaluation of RHS during Multi-Rate Simulation

Philip Hannebohm [1] Bernhard Bachmann [1]

[1]Institute for Data Science Solutions, Bielefeld University of Applied Sciences and Arts, Germany,
philip.hannebohm@hsbi.de, bernhard.bachmann@hsbi.de

Abstract

Modeling across physical domains can lead to coupled systems that operate on different time scales. Moreover, spatially discretized models can have dynamic behavior on only a small portion of the whole system, while that portion might move over time. With a single rate solver like DASSL, the step size is restricted to fast dynamics of a few states while the others could do much larger time steps, degrading the overall simulation speed. Recent development of the Generic Bi-rate ODE (GBODE) solver within the OpenModelica simulation environment addresses these models. It allows for adaptive partitioning of slow and fast states and progresses these partitions with different time steps. However, substantial performance improvements were not yet observed, in part because of the rigid evaluation structure of the generated code. During each small step only the derivatives of fast states are required, but all derivatives are computed. This paper demonstrates the use of selective equation evaluation to reduce the computational cost of multi-rate integration as shown on a scalable example of a distributed heating system. Further uses for selective evaluation during the simulation process are discussed and a complexity analysis is given.

Keywords: Equation-based modelling, Multi-rate simulation, Structural analysis

1 Introduction

In computer science strategies like partial or lazy evaluation are used to improve computation speeds. This paper introduces a similar approach for use in the simulation of equation-based models. These models are commonly represented as one coupled differential algebraic equation system (DAE), so for the purposes of this paper we consider the equation

$$0 = F(x, \dot{x}, u) \tag{1}$$

which is solved for $\dot{x}$ at each integration step assuming knowledge of all the states x. Further inputs u may influence the system dynamics as well.

Multi-rate integration methods are suited for systems with dynamics on multiple time scales (Thiele, Otter, and Matsson 2014). Furthermore, for models where a subsystem A undergoes rapid changes while another subsystem B stays relatively stationary during some interval of the simulation time, at a later time B may have dynamic changes while A does not. In these scenarios adaptive multi-rate simulation is clearly superior to statically partitioned multi-rate or even single-rate simulations. One such adaptive multi-rate solver is GBODE which stands for Generic Bi-rate ordinary differential equation (ODE) solver and is a generic implementation for any Runge-Kutta (RK) scheme (Bachmann 2023; Bonaventura et al. 2025). As a case study, our method is applied to the GBODE solver of OpenModelica.

During a multi-rate simulation step a subset of fast states is identified. For these states several small steps are taken while the remaining states take only one large step. At each fast step only a subset of $\dot{x}$ needs to be computed. Thus, computing the whole right-hand side (RHS) means superfluous calculations.

For efficiently evaluating parts of a model without evaluating the whole RHS, it is necessary to have a fast and flexible way to evaluate only a selected subset of equations. We demonstrate how this can be easily done within the OpenModelica Compiler (Fritzson, Pop, Abdelhak, et al. 2020). For the relatively newly developed GBODE solver this feature is crucial in providing improved simulation performance compared to other integration schemes.

2 Evaluation Structure

This section builds up the notation used in this paper. A general system of equations can be given in residual form as

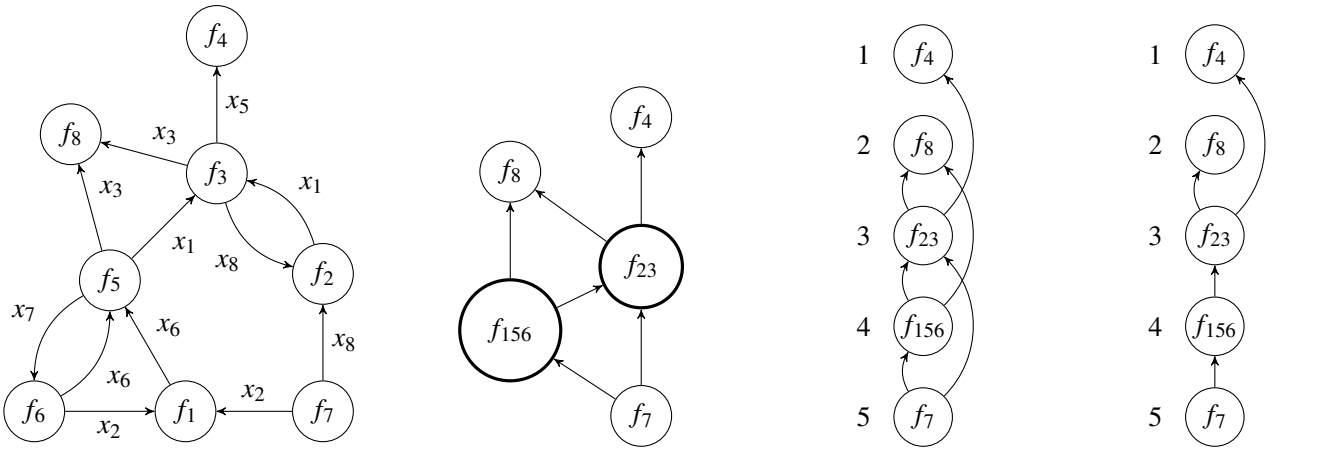
$$\begin{aligned} 0 &= f_1(x_{I_1(1)}, \ldots, x_{I_1(d_1)}) \\ &\vdots \\ 0 &= f_n(x_{I_n(1)}, \ldots, x_{I_n(d_n)}) \end{aligned} \tag{2}$$

where d_k is the number of variables occurring in equation k and $I_k(1), \ldots, I_k(d_k)$ are their corresponding indices.

If this system is given symbolically, or the incidence is given explicitly by the $I_1, \ldots, I_n$, Equation (2) can be transformed into semi-explicit form. The OpenModelica Compiler does this via the following steps.

A *perfect matching* between variables and equations is found (Pantelides 1988). The result can be interpreted as a directed graph with matched pairs of variables and equations as nodes and edges from equations to variables used in the equation.

In the context of this paper it is advantageous to define the edges in the opposite direction such that $f_i \to f_j$ when f_i depends on f_j, instead of the usual definition where we would have $f_j \to f_i$ as f_j influences f_i.



(a) Directed graph based on the given matching.

(b) DAG constructed by contracting the SCCs $f_{23} = \{f_2, f_3\}$ and $f_{156} = \{f_1, f_5, f_6\}$.

(c) Topologically sorted DAG.

(d) Transitive reduction of the DAG in (c).

Figure 1. Structure of the system in Equation (3).

As an example take the following system of equations and assume the equations are matched to the underlined variables:

$$\begin{aligned} 0 &= f_1(\underline{x_2}, x_6) & 0 &= f_5(x_1, x_3, \underline{x_6}, x_7) \\ 0 &= f_2(x_1, \underline{x_8}) & 0 &= f_6(x_2, x_6, \underline{x_7}) \\ 0 &= f_3(\underline{x_1}, x_3, x_5, x_8) & 0 &= f_7(x_2, \underline{x_4}, x_8) \\ 0 &= f_4(\underline{x_5}) & 0 &= f_8(\underline{x_3}) \end{aligned} \tag{3}$$

Figure 1a shows the graph resulting from Equation (3).

All *strongly connected components* (SCCs) of the graph are contracted into single nodes resulting in a *directed acyclic graph* (DAG). This DAG is the basis for evaluating the RHS of any ODE. The nodes of a DAG can be *topologically sorted*, i.e. they can be labeled with integer indices in such a way that $i > j$ for every edge $i \to j$. One such choice of indices is shown in Figure 1c. A valid order of execution is obtained by progressing from nodes with lower index to nodes with higher index. These two steps, contracting SCCs and topological sorting, are done simultaneously (Tarjan 1972).

There is a map $\mathscr{M}$ from variable to index corresponding to the node that computes the variable. This can be seen as a modification of the matching where variables solved in the same SCC are given the same index. For the system of Equation (3) this would be

$$\begin{aligned} &\mathscr{M}(x_1) = \mathscr{M}(x_8) = 3 \\ &\mathscr{M}(x_2) = \mathscr{M}(x_6) = \mathscr{M}(x_7) = 4 \end{aligned} \qquad \begin{aligned} &\mathscr{M}(x_3) = 2 \\ &\mathscr{M}(x_4) = 5 \\ &\mathscr{M}(x_5) = 1 \end{aligned} \tag{4}$$

Similarly, a modified incidence map $\mathscr{I}$ from index to variable sets can be constructed based on the I_k as defined in Equation (2) and the contraction of SCCs. The variables solved in the respective SCC are removed so only its dependencies remain. For Equation (3) this would be

$$\begin{aligned} &\mathscr{I}(1) = \mathscr{I}(2) = \{\} && \mathscr{I}(4) = \{x_1, x_3\} \\ &\mathscr{I}(3) = \{x_3, x_5\} && \mathscr{I}(5) = \{x_2, x_8\} \end{aligned} \tag{5}$$

Applying $\mathscr{M}$ to each element of $\mathscr{I}(k)$, we get the direct dependencies of the SCC with index k. This corresponds to the direct children of node k. In general we get the direct dependency map $\mathscr{D}$ as

$$\mathscr{D}(k) = \{\mathscr{M}(x) \mid x \in \mathscr{I}(k)\}. \tag{6}$$

Again, for Equation (3) the direct dependencies are

$$\begin{aligned} &\mathscr{D}(1) = \mathscr{D}(2) = \{\} && \mathscr{D}(4) = \{3, 2\} \\ &\mathscr{D}(3) = \{2, 1\} && \mathscr{D}(5) = \{4, 3\} \end{aligned} \tag{7}$$

Note that the indices in $\mathscr{D}(k)$ are all smaller than k. This is by construction of the topological ordering.

3 Selective Evaluation

The goal is to only evaluate the minimal part of the system that is needed to compute values for a set X of selected variables.

3.1 Computing Reachability

Finding all dependencies of X is equivalent to computing the reachability of the corresponding nodes in the DAG. Since the DAG is topologically sorted, the reachability can be found without the need for depth-first search. This is done in Algorithm 1 by going through the nodes from last to first according to the topological order and on each selected node k marking its direct dependencies $\mathscr{D}(k)$ as selected. All nodes in $\mathscr{D}(k)$ are smaller than k and their direct dependencies will be selected once the outer loop in line 3 gets to them. In this way the deeper paths are marked automatically by later loop iterations. The

DOI 10.3384/ecp218943

Algorithm 1 mark reachability by backwards loop

1: **for** $x \in X$ **do**
2: select $\mathscr{M}(x)$
3: **for** $k = N, \ldots, 1$ **do** ▷ backwards
4: **if** k is selected **then**
5: select all $j \in \mathscr{D}(k)$ ▷ $j < k$

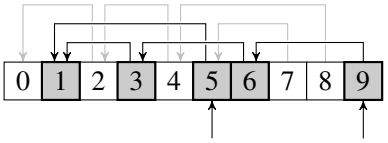


Figure 2. Example of reachability computation exploiting the topological ordering of the DAG.

complexity of Algorithm 1 is $O(|X| + |V| + |E|)$ where $|E| = \sum_k \mathscr{D}(k)$ is the number of edges in the DAG and $|V| = N$ is the number of vertices.

This is demonstrated in Figure 2 with a small example. First, nodes 5 and 9 corresponding to fast states are selected, then direct dependencies of selected nodes are selected going from right to left, so the order of selection is: $9 \to 6$, $6 \to 3$, $5 \to 1$, $3 \to 1$.

3.2 Transitive Reduction

The *transitive reduction* R of a DAG G is a subgraph of G which has the same reachability relation but minimal number of edges (Aho, Garey, and Ullman 1972). If there is a longer path from i to j then a direct edge $i \to j$ can be removed, keeping the overall reachability the same. Figure 3 shows the simplest scenario where the edge from 3 to 1 can be removed because dependency is still given through 2. The cost of computing R is $O(|V||E|)$ for sparse graphs (Goralčíková and Koubek 1979).

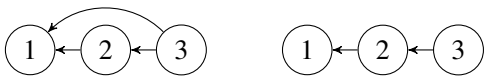


Figure 3. One step of transitive reduction.

Using R instead of G we can reduce the amount of work in line 5 of Algorithm 1 because fewer nodes are in direct dependencies. Since R needs to be computed only once in a preprocessing step, its cost is negligible when doing a reachability analysis many times during simulation. In highly connected systems this might improve the overall performance significantly. For the example system in Equation (3) this removes two out of six edges as depicted in Figure 1d.

3.3 Implementation

Prior to our changes the simulation code generated by the OpenModelica Compiler had the following flat structure:

```
void functionODE()
{
  eqFunction_1();
  // [...]
  eqFunction_N();
}
```

While very simple and efficient, the code does not allow for partial evaluation. One way to achieve this is by giving each `eqFunction_i` a flag to tell whether or not it should be evaluated. However, this would still lead to $O(N)$ operations. Replacing the function with the following evaluation scheme gives the desired flexibility with minimal overhead:

```
void functionODE()
{
  static f_ptr eqFunctions[N] = {
    eqFunction_1,
    // [...]
    eqFunction_N
  };
  for (int i = 0; i < evalN; i++)
    eqFunctions[evalI[i]]();
}
```

Here `evalI` is an array of SCC indices and `evalN` is the length of `evalI`, whereas `eqFunctions` is an array of function pointers `f_ptr`.

During simulation, `evalI` needs to be recomputed via Algorithm 1 each time X changes. If there is a rapid swap between two unchanging sets X_s and X_f, as is the case for GBODE bi-rate integration, the corresponding `evalI` can be buffered to avoid recomputation.

Some additional overhead can be expected from making equation evaluation flexible in this way because of the additional pointer indirection and as a consequence the less efficient cache handling. However, this overhead should be negligible for large systems when only a fraction of equations needs to be evaluated.

4 Preliminary Results

A modified version of GBODE was tested on the model `HeatingSystem` from the ScalableTestSuite Modelica Library (Casella 2015). It has the following equations:

```
Text = 278.15 + 8*sin(2*pi*time/86400)"f1";
Cd*der(Td) = Qd - sum(Qh)             "f2";
Qd = sat(Kp*(Td0 - Td),0, Qmax)       "f3";
for i in 1:N loop
  Qh[i] = Gh*(Td - Tu[i])*u[i]        "f4";
  Que[i] = Gu*(Tu[i] - Text)          "f5";
  Cu[i]*der(Tu[i]) = Qh[i] - Que[i]   "f6";
  der(x[i]) = a*hist(x[i], Tu0 - Tu[i],
                     Teps)            "f7";
  u[i] = sat(b*x[i], -0.5, 0.5) + 0.5 "f8";
end for;
```

Its equation structure, depicted in Figure 4, is well suited to selective evaluation when solving for fast states. There are no algebraic loops involved and thus every variable can be computed explicitly. Because of this, each node corresponds exactly to one variable.

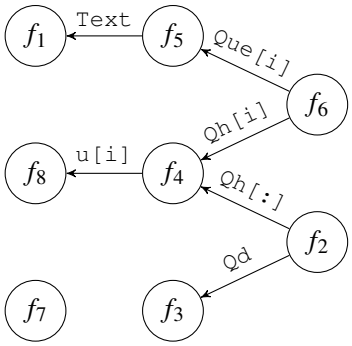


Figure 4. DAG of `HeatingSystem`. Arrays and for-loops are kept as single nodes for better readability.

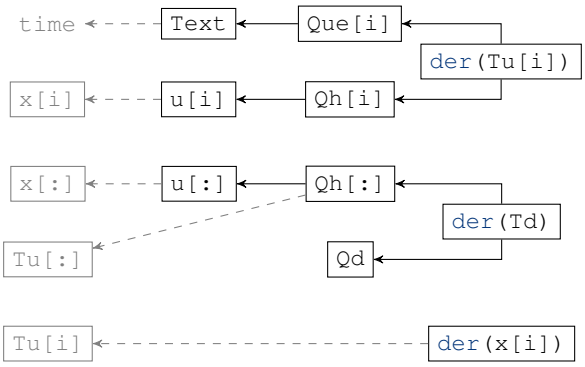


Figure 5. Sub-DAGs of `HeatingSystem` for each state derivative. In dashed are dependencies on states that may need to be interpolated. Self-dependencies are omitted for readability.

The various sub-DAGs for computing single derivatives are shown in Figure 5. Here variables are used as names for the nodes to make the structure easier to recognize. There is some overlap between the dependencies of each of the `der(Tu[i])` and `der(Td)` due to `Qh`.

We performed a simulation of `HeatingSystem` with the solver settings `-s=gbode -gbratio=0.5`. Each time a new set of fast states was identified, the fraction of equations needed to evaluate the derivatives was determined by Algorithm 1 and logged. The top row of Table 1 shows the number of equations to evaluate. The bottom row shows how often these many equations were needed to evaluate the new set of fast states. In most cases the number of equations needed is less than 7 out of a total of 64, which is about 11%. More than half of the time only two equations (3%) were needed, while the number never went beyond 17 (27%).

Table 1. Number of phases that require evaluation of `evalN` equations for fast states. Entries not in the table have a count of zero.

`evalN`	2	4	6	13	15	17
#	365	184	75	4	3	3

5 Conclusions and Future Work

We showed that selectively computing parts of the equation system of a DAE is practically possible and most of the work needed is already done by Modelica compilers. We also demonstrated that it would be desirable for multi-rate simulation of at least some models. Selective evaluation is also used to compute only needed parts of the Jacobian when solving fast states with implicit methods. In particular for the `HeatingSystem` there are significant savings in computational load. This is not a direct measure of performance gain, but it gives a rough idea that selective evaluation should lead to significantly faster simulations.

The technique of selective RHS-evaluation and the corresponding dependency analysis can be used even more broadly. During event search of hybrid DAE simulations only the zero crossings need to be computed in order to narrow down the exact event time. Then during event iteration only parts influenced by discrete changes need to be recomputed iteratively. For this the dependency analysis needs to be extended on discrete variables and a slightly different selection mechanism needs to be developed. This will be investigated in future works.

Acknowledgements

This work was conducted as part of the OpenSCALING project (Grant No. 01IS23062E) at the University of Applied Sciences and Arts Bielefeld, in collaboration with Linköping University. The authors would like to express their sincere appreciation to both the OpenSCALING project and the Open Source Modelica Consortium (OSMC) for their support, collaboration, and shared commitment to advancing open-source modeling and simulation technologies.

References

Aho, A. V., M. R. Garey, and J. D. Ullman (1972). "The Transitive Reduction of a Directed Graph". In: *SIAM Journal on Computing* 1.2, pp. 131–137. DOI: 10.1137/0201008.

Bachmann, Bernhard (2023). "GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica". In: *OpenModelica Annual Workshop*. Linköping, Sweden. URL: https : / / openmodelica . org / images / M _ images / OpenModelicaWorkshop _ 2023 / GBODE - OpenModelicaWorkshop_2023.pdf.

Bonaventura, Luca et al. (2025). "Self-Adjusting Multi-Rate Runge-Kutta Methods: Analysis and Efficient Implementation in An Open Source Framework". In: *Journal of Scientific Computing* 105. DOI: 10.1007/s10915-025-03049-y.

Casella, Francesco (2015). "Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives". In: *Proceedings of the 11th International Modelica Conference*. Versailles, France, pp. 459–468. DOI: 10.3384/ecp15118459.

Fritzson, Peter, Adrian Pop, Karim Abdelhak, et al. (2020-10). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control: A Norwegian Research Bulletin* 41, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Goralčíková, Alla and Václav Koubek (1979). “A reduct-and-closure algorithm for graphs”. In: *Mathematical Foundations of Computer Science 1979*. Ed. by Jiří Bečvář. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 301–307. DOI: 10.1007/3-540-09526-8_27.

Pantelides, Constantinos C. (1988). “The Consistent Initialization of Differential-Algebraic Systems”. In: *SIAM Journal on Scientific and Statistical Computing* 9.2, pp. 213–231. DOI: 10.1137/0909014.

Tarjan, Robert (1972). “Depth-First Search and Linear Graph Algorithms”. In: *SIAM Journal on Computing* 1.2, pp. 146–160. DOI: 10.1137/0201010.

Thiele, Bernhard, Martin Otter, and Sven Erik Matsson (2014). “Modular Multi-Rate and Multi-Method Real-Time Simulation”. In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, pp. 381–393. DOI: 10.3384/ecp14096381.