Achieving O(n) Complexity for Models from Modelica.Mechanics.MultiBody

Christian Schubert[a] Jens Frenkel[a] Günter Kunze[a] Michael Beitelschmidt[b]
[a] Professur für Baumaschinen- und Fördertechnik
[b] Professur für Dynamik und Mechanismentechnik
Technische Universität Dresden
01069 Dresden, Germany
{christian.schubert, jens.frenkel, guenter.kunze
michael.beitelschmidt}@tu-dresden.de

Abstract

This paper presents a graph theoretical interpretation of the well-known $O(n)$ algorithm for Multibody systems. It enables Modelica compilers to solve for the unknown accelerations of a Multibody model without the need of inverting a dense mass matrix which would require $O(n^3)$ operations.

Keywords: MultiBody, Relaxation, Gaussian Elimination, OpenModelica

1 Introduction

Simulation has become an indispensable tool in early development stages. Increasing computational power leads to a demand for more detailed models. Especially in the design of Mobile Machinery, Multibody systems are of major importance.

Currently, most Modelica compilers apply Tearing [1] to models from Modelica.Mechanics.MultiBody yielding a dense linear system of size proportional to n - the number of bodies. In order to solve for the unknown joint accelerations the system has to be inverted which requires $O(n^3)$ operations. Hence this approach is only recommendable for small to medium sized problems.

Efficient algorithms with $O(n)$ complexity are well known from literature [2], [4]. Unfortunately their application for Modelica.Mechanics.MultiBody proves to be difficult since these algorithms rely on special knowledge about the multibody systems which is not available in a general equation based framework like Modelica.

It has already been pointed out in the literature [5] that a technique called *Relaxation* is able to yield such an $O(n)$ formalism for multibody systems. However, adaptions to the model libraries as well as a specific model structure were required.

This paper presents a novel algorithm for general purpose Modelica compilers. It is based on a graph theoretical generalization of the well known $O(n)$ algorithm for multibody systems adapted to models from Modelica.Mechanics.MultiBody.

2 Multibody systems

2.1 Kinematic Graph

Every multibody system can be represented by a kinematic graph whose nodes represent both bodies and inertial frames and whose edges correspond to joints. If the kinematic graph contains closed loops, appropriate joints, so called *cut-joints*, are temporarily removed so that the resulting graph only consists of *trees*. In a tree, every node (body) has a unique parent, which is the next node on the path to the root (inertial frame). All bodies are numbered, such that every child body has a higher number than its parent. Each joint is numbered according to the child body it is connected to, thus forming pairs of bodies and joints. All remaining cut-joints are numbered consecutively. An example is given in Figure 1.

2.2 Equations of Motion

The equation of motion of a single body i can be written as

$$\mathbf{M}_i\mathbf{a}_i = \mathbf{p}_i + \mathbf{f}_i - \sum_{k\in\mu(i)} \mathbf{R}_{k,i}^T\mathbf{f}_k \quad (1)$$

$$\mathbf{p_i} = \mathbf{f}_{i,ext} - \mathbf{h}_i \quad (2)$$

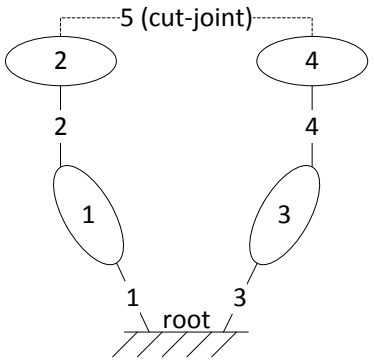

Figure 1: Kinematic Graph of Example System

where $\mathbf{M}_i \in \mathbb{R}^{6\times6}$ represents the mass matrix, $\mathbf{a}_i \in \mathbb{R}^6$ both translational and rotational acceleration of a fixed point on body i and $\mathbf{h}_i \in \mathbb{R}^6$ all gyroscopic terms. $\mathbf{p}_i$ is used as an abbrevation for $\mathbf{h}_i$ and all external forces and torques $\mathbf{f}_{i,ext}$. $\mathbf{f}_i, \mathbf{f}_k \in \mathbb{R}^6$ represent the joint reaction forces of the joint belonging to the body i as well as all the set of all its children $\mu(i)$. $\mathbf{R}_{k,i}$ transforms the force and torque from i to k whereas its transpose performs the opposite transformation.

It is assumed that every joint i has a set of joint-coordinates $\mathbf{q}_i$ as well as joint velocities $\mathbf{s}_i$ which fully determine its kinematic state. Thus, the acceleration of body i is given by

$$\mathbf{a}_i = \mathbf{R}_{i,h}\mathbf{a}_h + \mathbf{J}_i\dot{\mathbf{s}}_i + \mathbf{c}_i \tag{3}$$

where h is the index of the parent body of i. $\mathbf{J}_i$ describes the degrees of freedom of joint i and $\mathbf{c}_i$ collects all remaining terms which are neither linear in $\mathbf{a}_h$ nor $\dot{\mathbf{s}}_i$.

From d'Alamberts Principle it can be found that

$$\mathbf{J}_i^T\mathbf{f}_i = \tau_i \tag{4}$$

whith τ_i being the motor force driving the joint.

Since equations (1)-(4) are linear with respect to the accelerations and forces, one can merge the equations for every element of the multibody system into one single linear system of equations.

One of the most efficient $O(n)$ algorithms (see [3]) to solve this linear system of equations is defined through repeated application of

$$\dot{\mathbf{s}}_i = \rho_i^{-1}\left(\tau_i - \mathbf{J}_i^T\mathbf{M}_i^A\left(\mathbf{a}_{\lambda(i)} + \mathbf{c}_i\right) - \mathbf{J}_i^T\mathbf{p}_i^A\right) \tag{5}$$

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{J}_i\dot{\mathbf{s}}_i + \mathbf{c}_i \tag{6}$$

requiring the calculation of the following variables for each body starting at the highest index

$$\mathbf{M}_i^A = \mathbf{M}_i + \sum_{k\in\mu(i)} \mathbf{M}_k^a \tag{7}$$

$$\rho_i = \mathbf{J}_i^T\mathbf{M}_i^A\mathbf{J}_i \tag{8}$$

$$\mathbf{M}_i^a = \mathbf{M}_i^A - \mathbf{M}_i^A\mathbf{J}_i\rho_i^{-1}\mathbf{J}_i^T\mathbf{M}_i^A \tag{9}$$

$$\mathbf{p}_i^A = \mathbf{p}_i + \sum_{k\in\mu(i)} \mathbf{p}_k^a \tag{10}$$

$$\mathbf{p}_i^a = \mathbf{p}_i^A + \mathbf{M}_i^A\mathbf{c}_i + \mathbf{M}_i^A\mathbf{J}_i\rho_i^{-1}\left(\tau_i - \mathbf{J}_i^T\mathbf{p}_i^A\right) \tag{11}$$

It can be shown that this exact algorithm can be derived from a (sparse) Gaussian Elimination of the linear system of equations provided all equations and variables are ordered correctly.

In a general equation based framework, information such as the ordering of bodies is not readily available. Thus the algorithm cannot be applied directly. However, [5] has shown that the application of a technique called *Relaxation*, which is a type of Gaussian Elimination, may also lead to an $O(n)$ algorithm. The suitable ordering of the equations and variables was achieved by inserting a special *relax* operator into the model equations.

This paper follows another path in which the $O(n)$ algorithm is derived using graph theoretical techniques. To do so a graph representing the equations of motion is built from the multibody system. The key to an efficient $O(n)$ algorithm lies in the ordering of the graph. This is a problem for a Modelica compiler since the information about the structure is lost in the compilation process but is needed to achieve the efficiency of algorithms such as [3]. By trying to generalize the idea behind the algorithm from [3], a good ordering for general modelica models can be found which consequently leads to an efficient $O(n)$ algorithm. This approach is described in the following section.

3 Graphtheoretical Interpretation

3.1 Graph of a system of equations

Given a set of equations, an undirected bipartite graph can be defined which contains two sets of nodes representing equations and variables respectively. There is an edge between a variable and an equation if and only if that equation depends on that variable. The graph of the equation system belonging to the example system has been sketched in Figure 2. Every square node represents an equation whereas every circle represents a variable. Nodes representing eq. (1) have been named I_i, eq. (3) is called A_i and (4) is labelled D_i. In addition to the definition above, the edges carry the partial derivative of the equation with respect to the variable.

It can be seen that the graph exhibits two *legs*. The first leg contains all kinematic quantities (A_i, a_i)

DOI 10.3384/ecp12076705

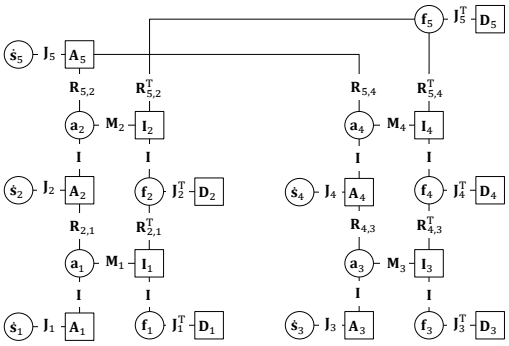

Figure 2: Equation Graph of Example System

whereas the second leg comprises all kinetic quantities (I_i, f_i). The two legs are interconnected through *steps* given by the inertial equations I_i (eq. 1). Equations D_i (eq. 4) and variables $\dot{\mathbf{s}}_i$ appear as *handles* to the legs, thus forming a ladder like structure. All nodes with the same index represent a body along with its joint and shall be denoted as *body structure*. A body structure is called *terminal* if the body it represents does not have any children.

3.2 Gaussian Elimination

Gaussian Elimination can be applied to a linear system of equations $\mathbf{Ax} = \mathbf{b}$. Therefore one has to reproduce $\mathbf{A}$ from the equation graph of the multibody system. This requires the numbering of all equation and variable nodes, i.e. allocating them to rows and columns of $\mathbf{A}$. The algorithm then iterates over all elements on the main diagonal of $\mathbf{A}$, which are called *pivots*. Multiples of the current row are added to all rows below such that all elements below the pivot are eliminated. Thus $\mathbf{A}$ is reduced to $\hat{\mathbf{A}}$ which has a (block) upper triangular form.

Given the numbered graph, Gaussian Elimination can be applied directly:

1. Begin at $i = 1$
2. Check that there is an (invertible) edge between equation node i and variable node i (equivalent to pivot element)
3. Create Edges between all pairs of equations and variables connected to equation and variable nodes i
4. Remove equation and variable nodes i along with all adjacent edges from the graph
5. Continue at 2 with $i := i + 1$ until all nodes are removed

Figure 3 shows this process for a body structure (see section 3.1) as found in Figure 2.

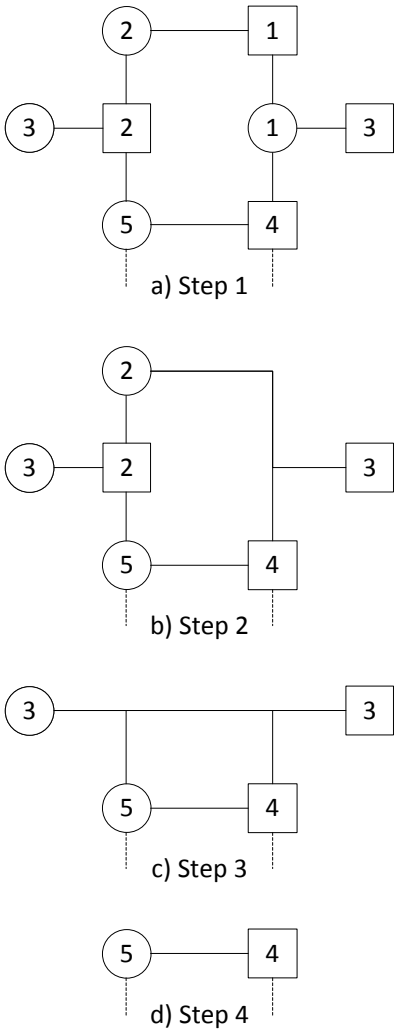

Figure 3: Steps of Gaussian Elimination

3.3 $O(n)$ algorithm

The numbering shown in Figure 3 leads to the efficient $O(n)$ algorithm from [3].

Removing the closed loop (A_5, D_5, f_5, $\dot{s}_5$) from the graph given in Figure 2 yields two terminal body structures. These can be eliminated as shown in Figure 3 revealing new terminal body structures. This process can be repeated until all body structures are eliminated.

Looking at the numbering employed in Figure 3 one may note that

1. Resulting pivots are chosen to be identity matrices if possible
2. Each body structure is treated seperately, beginning at the terminal ones
3. All nodes between the handles of the body structure are being numbered consecutively beginning at the equation handle
4. Equation and variable handle are given the same number although there is initially no connection between them (zero pivot)
5. Entries in the lower triangular part of **A**only occur due to the D_i nodes as well as the I_i nodes of non terminal body structures.

Please note that the handles nodes correspond to a suitable choice of tearing variables and residual equations, as described in [1].

One may expect that the application of these rules to the equation graph found in models from Modelica.Mechanics.MultiBody may yield a numbering which leads to an efficient $O(n)$ algorithm for multibody systems for a general purpose Modelica compiler.

4 Application to Modelica.Mechanics.MultiBody

4.1 Equation Graph

Due to the object oriented nature of Modelica.Mechanics.MultiBody the equations are the same as in 2.2 but are not written in such a compact form. Equation (1) is found in the *Body* model. Equations (3) and (4) are found in the different joint models. The transformation matrices $\mathbf{R}_{i,k}$ (see Eq. (1)) are defined through the *FixedTranslation* and *FixedRotation* models. The linear system of equations under consideration is found as a strong connected component through Tarjan's algorithm [8] after index reduction has been applied [7]. Moreover, most Modelica compilers apply symbolic simplifications to the equations of motion. Figure 4 shows the graph of the sample system with which a Modelica compiler has to deal with.

Application of the $O(n)$ algorithm requires three steps:

1. Recover the graph structure
2. Find a suitable ordering

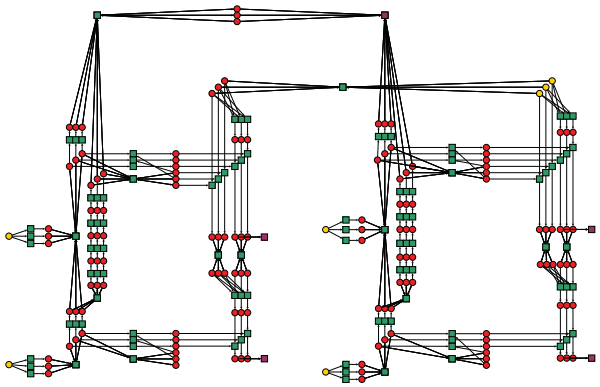

Figure 4: Equation Graph of Example System modelled with Modelica.Mechanics.MultiBody

3. Apply Gaussian Elimination

Every step will be discussed in the following.

4.2 Tree Structured Systems

4.2.1 Recovering the graph structure

The first rule (see section 3.3) says, that if possible the pivots shall be chosen to be identity matrices. Therefore pairs of equations and variables have to be found whose partial derivative is an identity matrix. This process shall be called *Natural Matching*. In a first step, every vectorial equation is tested if it can be solved for its unmatched vectorial variables with only using addition and subtraction. If that is the case, this equation and variable are *matched*. Afterwards, all remaining equations and variables are expanded to their scalar representation. All remaining scalar equations are tested if they can be solved for their unmatched vectorial variables with only using addition and subtraction. If that is the case, this equation and variable are also *matched*. Then a classic matching algorithm [6] is applied, leaving a set of variables and equations unmatched. These are the candidates for the tearing variables and residual equations. This procedure has already been suggested in [9].

Since all equations in the Modelica.Mechanics.MultiBody library have been written down in a manner which is most suitable for computation, it happened in all our tests that the set of candidates may be used without further modification as tearing variables and residual equations. The tests also showed, that mostly joint accelerations were used as tearing variables and, depending on symbolical simplifications, the D_i equations or close neighbours were used as residual equations.

DOI 10.3384/ecp12076705

The result of the matching algorithm is visualized in the graph by assigning directions to all edges. An edge from an equation to a variable means that this equation is used The result of the matching algorithm is visualized in the graph by assigning directions to all edges. An edge from an equation to a variable means that this equation is used to calculate that variable. An edge from a variable to an equation means that this variable is needed in the calculation of that variable. All tearing varibales are assumed to be known whereas all residual equations do not have any variables that they are solved for. The result for the example system including the kinematic loop is shown in Figure 4.

Next, the order between the tearing variables has to be found. Therefore the algorithm starts at a tearing variable and follows the edges in opposite direction, thus running down the kinematic leg. When another tearing variable is found, it must be the predecessor and the traversal is stopped. Thus, the predecessor to every tearing variable can be found defining an order between them which corresponds to the kinematic graph of the mechanical system. Please note, that this only works for tree structured systems. Otherwise a body, and therefore a tearing variable, may have more than one predecessor.

In a next step, the residual equation to each tearing variable has to be found. Again, a breadth-first graph traversal is started from every tearing variable following each edge. The first residual equation, that is found is assigned to the tearing variable.

4.2.2 Finding a suitable ordering

From the kinematic graph, obtained in the previous step, the terminal pairs of tearing variables and residual equations are known. Starting at a terminal residual equation all paths to its tearing variable can be found by following the in opposite direction. Valid paths may also include eliminated nodes. Once all paths have been found, decreasing numbers are assigned to the nodes using a breadth-first-search starting at the tearing variable. Afterwards the nodes of the residual equation and the tearing variable are numbered. Then all numbered nodes are eliminated from the graph as well as the tearing variable from the kinematic graph. This process is repeated for the next terminal tearing variable until all tearing variables are eliminated. In a last step all remaining nodes are numbered. Thus, a number has been assigned to every node allowing to apply Gaussian Elimination.

4.2.3 Applying Gaussian Elimination

Given the numbering of all nodes, the matrix $\mathbf{A}$ can be constructed. Next Symbolic Gaussian Elimination is applied to the matrix $\mathbf{G} = \left[\begin{array}{cc} \mathbf{A} & \mathbf{b} \end{array} \right]$ yielding an upper-triangular $\mathbf{G}'$, see section 3.2. The equations of the strong connected component are then replaced by $\mathbf{x} = \mathbf{G}'^{-1}\mathbf{b}$.

When performing Gaussian Elimination temporary variables should be introduced after every elimination step. Otherwise the symbolic expressions in the entries of $\mathbf{G}'$ may grow very fast.

4.3 Closed Loop Systems

Adapting the algorithm to cope with closed kinematic loops is part of the ongoing work. This section shall outline the problem and possible solutions.

Natural Matching still works reasonably well, choosing joint accelerations and the constraint forces of the loop as tearing variables. The search for the predecessors of the tearing variables, however, breaks down. Firstly because the tearing variables of the kinematic loop may have more than one predecessor and secondly because they are located in the kinetic leg.

The ordering between the tearing variables then has to be modeified such that, the tearing variables of the loop closure joint are treated after all other tearing variables belonging to the same kinematic loop.

So far, the search for the predecessors has been extended so that it finds every tearing variable which is a parent in the kinematic graph. It leads to a dramatic increase in effort for both the search as well as the assembly of the kinematic graph. Tests have shown that the whole algorithm suceeds for some models, including the sample model, but it fails for others. Failing is mainly caused because the search for predecessors sometimes returns unexpected results.

5 Numerical Tests

The described algorithm has been implemented into the OpenModelica Compiler. It has been tested on multibody systems with tree structure only.

The following models have been used for testing:

1. Planar Pendulum - A sequence of n submodels consisting of a revolute joint, a body and a fixed-Translation

2. Split Pendulum - Same as Planar Pendulum but with a short extra branch of constant length

3. Alternating Pendulum - Same as Planar Pendulum, but with alternating axes of rotation

4. Multi Pendulum - Each body is followed by two more pendulum bodies with a limited recursion depth (see Figure 5

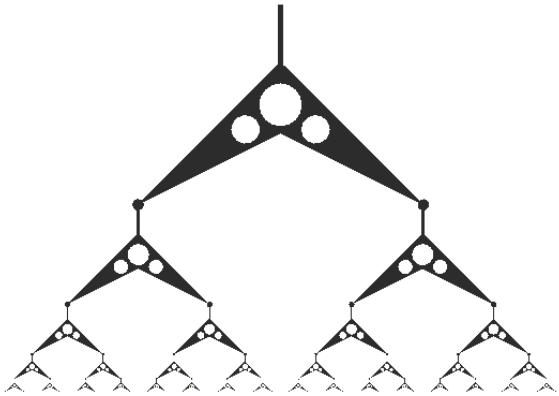

Figure 5: Multi Pendulum

Figures 6, 7, 8 and 9 show the required operation counts needed to calculate the whole model (including the accelerations) for the four test cases. As can be seen each curve exhibits a linear dependence on the number of bodies and therefore the number of degrees of freedom.

For comparison, the results when using tearing [1] which is ($O(n^3)$) have also been included. One may see that for planar systems the $O(n)$ algorithm produces much lower operation counts as the number of bodies grow. In the 3D case, however, the tearing algorithm outperforms the proposed $O(n)$ algorithm. Investigations have shown that this is partly due to the limited symbolic simplification capabilities of the OpenModelica Compiler.

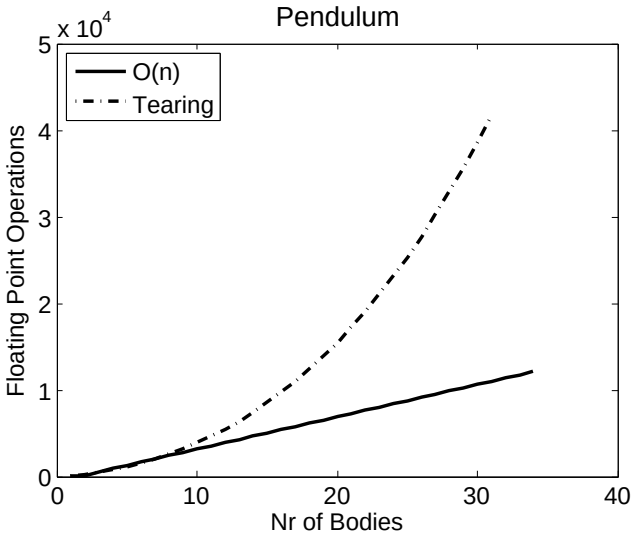

Figure 6: Results - Pendulum

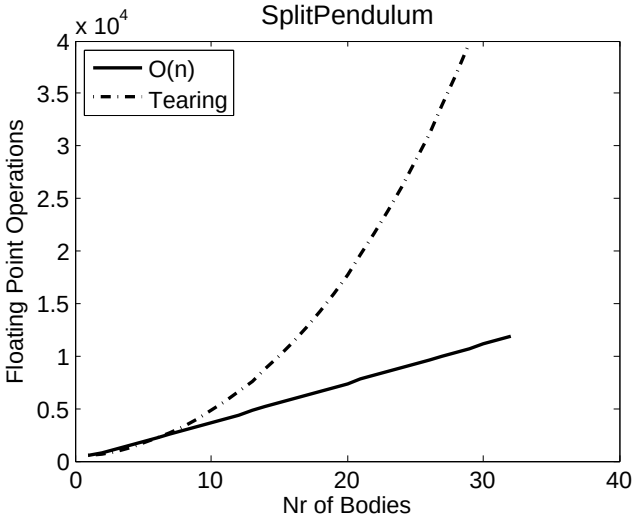

Figure 7: Results - Split Pendulum

6 Discussion

6.1 Applicability

The algorithm has been tested on several different multibody models. It relies on the structural properties of the linear system as discussed in the earlier sections. Due to the fact that Tarjan's Algorithm [6] decomposes the system into seperate strong connected components, the use of force elements does not influence the algorithm as long as their value does not depend on accelerations or forces in the system. Hence, Accounting for dry friction (tangential force depending on the normal force) for example, might cause the algorithm to fail.

Structural singularities are found during compile time, since during symbolic Gaussian Elimination each pivot is checked if it is non-zero. Problems like numerical cancellation or division by zero are not detected by the compiler and have to be reported as errors at runtime.

Due to the problems which may be encountered on some models, the algorithm should not be enabled by default. Instead it provides an interesting alternative for users who try to tune their models for faster execution times, as it would be the case in real time applications, for example.

DOI 10.3384/ecp12076705

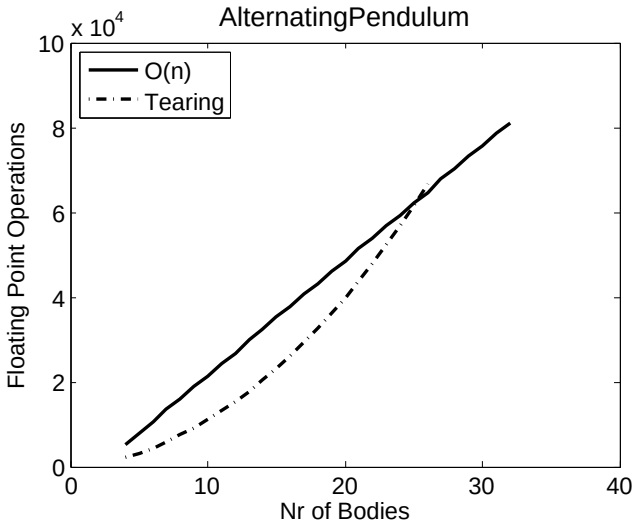

Figure 8: Results - Alternating Pendulum (3D)

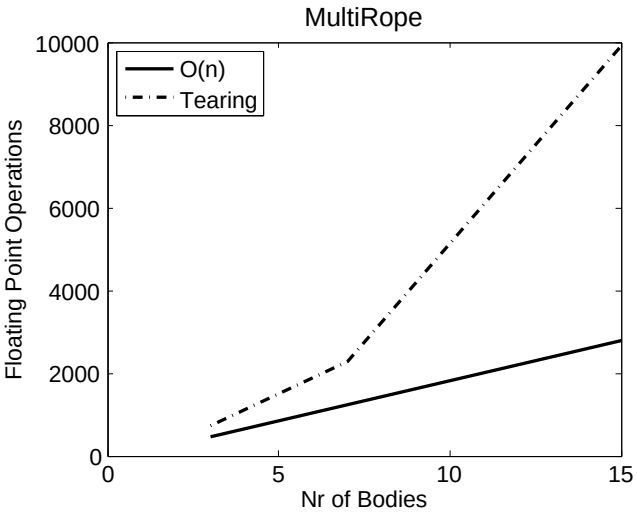

Figure 9: Results - Multi Pendulum

6.2 O(n) or Tearing?

If Gaussian Elimination fails during compile time, the current implementation switches back to Tearing. However, the question arises which strong connected components should the proposed $O(n)$ algorithm be applied to. The current (presumably non-efficient) implementation is controlled by a compiler flag. If it is set, the $O(n)$ algorithm is applied to every strong connected component. Should it fail, Tearing is applied istead. A possible improvement could be, to control that either by an annotation or by comparing the operation count.

6.3 Efficiency

The investigations suggest that this algorithm indeed achieves $O(n)$ performance and the results show that it is often more efficient than Tearing. However, there is still much potential for optimization. The most promising optimization would be to exploit symmetry. This could be achieved by looking for common sub expressions.

The current version of the Modelica.Mechanics.MultiBody library however, is not suited for exploiting symmetry since all translational variables are written with respect to the world frame. Thus, for equation (1) and (3) the relationship $\mathbf{R}_{i,j} = \mathbf{R}_{j,i}^T$ does not hold. Preliminary tests have shown a 20% decrease in operation count, without the usage of a common sub expression search, when the symmetry is established by writing all translational variables with respect to the local frame_a.

7 Outlook

Next steps include adaptions to make the algorithm work reliably on models with kinematic loops. It is also worth extending the module which performs symbolic simplification by analyzing the assignments before code generation. This may also be combined with trying to exploit symmetry in order to lower the number of operations.

Lastly, it would be interesting to see if that algorithm may also be applied successfully to models from other domains, like electrical networks or chemical processes.

8 Conclusion

In this paper a special $O(n)$ algorithm for calculating the joint accelerations of a multibody system has been adapted. With the novel graph theoretic interpretation, general purpose Modelica compilers are able to solve models from Modelica.Mechanics.MultiBody with a computational effort proportional to number of bodies n compared to the usual $O(n^3)$ algorithms based on the mass matrix. A working implementation for the OpenModelica Compiler has shown a linear relationship between the operation count and degrees of freedom. When comparing the results to the tearing algorithm, it became apparent that it outperforms the proposed algorithm for non-planar models. This is partly due to the limited symbolic simplification carried out by the OpenModelica Compiler.

References

[1] H. Elmqvist, and M. Otter: Methods for Tearing Systems of Equations in Object Oriented Modeling, Proc. ESM'94, European Simulation Multiconference, Barcelona, Spain, June 13, 1994, pp. 326-332.

[2] R. Featherstone: The calculation of robot dynamics using articulated-body inertias. International Journal of Robotics Research 2, May 1983, pp. 13-30

[3] R. Featherstone: Rigid Body Dynamics Algorithms. Springer, New York, 2008

[4] H. Brandl, R. Johanni and M. Otter: A very efficient algorithm for the simulation of robots and similar multibody systems without inversion of the mass matrix, In Kopacek, P., Troth, I. and Desoyer, K. (Eds.), Theory of Robots, Oxford, Pergamon Press, 1988, pp. 95- 100

[5] M. Otter, H. Elmqvist, and F.E. Cellier: Relaxing: A symbolic sparse matrix method exploiting the model structure in generating efficient simulation code, Proc. Symp. Modelling, Analysis, and Simulation, CESA'96, IMACS MultiConference on Computational Engineering in Systems Applications, Lille, France, vol.1, 1995, pp. 1-12

[6] I.S. Duff, A.M. Ersiman, and J.K. Reid: Direct Methods for Sparse Matrices, Oxford University Press, 1986

[7] S.E. Mattsson and G. Söderlind: Index Reduction in Differential Algebraic Equations Using Dummy Derivatives, SIAM Journal on Scientific Computing, Vol. 14, No. 3, pp. 677-692, 1993

[8] R. Tarjan: Depth-first search and linear graph algorithms, 12th Annual Symposium on Switching and Automata Theory, 13-15 Oct., 1971, pp. 114 - 121

[9] E. Carpanzano: Order Reduction of General Nonlinear DAE Systems by Automatic Tearing, Mathematical and Computer Modelling of Dynamical Systems, Vol. 6, Iss. 2, 2000

DOI 10.3384/ecp12076705