Efficient Minimal Tearing of Hybrid Algebraic Loops for Large-scale System Simulation

OpenModelica Annual Workshop 2020

Andreas Heuermann     Bernhard Bachmann

FH Bielefeld
University of Applied Science
Faculty of Engineering and Mathematics
Modelica Compiler Overview
Symbolic Transformation
Matching & Sorting

Tearing
Overview
Minimal Tearing
Efficient Minimal Tearing of Hybrid Algebraic Loops for Large-scale System Simulation

Modelica Compiler Overview
Modelica Compiler Overview
How to get a mathematical representation from a model

Equations

\[ u_0 = A \sin(2\pi ft) \]
\[ u_0 = u_1 + u_3 \]
\[ u_1 = R_1 \cdot i_1 \]
\[ u_L = u_1 + u_2 \]
\[ u_2 = R_2 \cdot i_2 \]
\[ u_3 = u_2 \]
\[ u_3 = R_3 \cdot i_3 \]
\[ i_0 = i_1 + i_L \]
\[ u_L = L \cdot i_L \]
\[ i_1 = i_2 + i_3 \]
Modelica Compiler Overview
Symbolic Transformation

Steps to perform

- Causalization
  - Assign each variable to exactly one equation
- Matching and sorting
  - Find strong components and sorting
- Adjacency matrix and structural regularity
  - BLT transformation

Group variables

\[ 0 = f(x(t), \dot{x}(t), y(t), u(t), p, t) \]
\[ \dot{x}(t) = (i_L) \quad x(t) = (i_L) \]
\[ y(t) = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_L \\ i_0 \\ i_1 \\ i_2 \\ i_3 \end{pmatrix} \]
\[ p = \begin{pmatrix} A \\ f \\ R_1 \\ R_2 \\ R_3 \\ L \end{pmatrix} \]
\[ u(t) = ( ) \]
Modelica Compiler Overview
Symbolic Transformation

- **Matching:** Assign variables to equations
- **Sorting:** Construct directed graph
- **Get ordered state form**

Symbolic transformation

\[
0 = f(x(t), z(t), u(t), t), \quad z(t) = \begin{pmatrix} \dot{x}(t) \\ y(t) \end{pmatrix}
\]

\[
z(t) = g(x(t), u(t), p, t)
\]

\[
\dot{x}(t) = h(x(t), u(t), p, t)
\]

\[
y(t) = k(x(t), u(t), p, t)
\]
Modelica Compiler Overview
Matching & Sorting

Adjacency matrix

<table>
<thead>
<tr>
<th></th>
<th>(z_1)</th>
<th>(z_2)</th>
<th>(z_3)</th>
<th>(z_4)</th>
<th>(z_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_1)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(f_2)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(f_3)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(f_4)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(f_5)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Bipartite Graph

Minimal Tearing  •  February 3, 2020
Heuermann, Bachmann
Adjacency matrix

\[
\begin{pmatrix}
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1
\end{pmatrix}
\]
Modelica Compiler Overview

Matching & Sorting

Sink

\[ f_1 \mid z_4 \]

Loop

\[ f_3 \mid z_5 \]

Source

\[ f_2 \mid z_2 \]

Bipartite Graph

\[ f_1 \]

\[ f_2 \]

\[ f_3 \]

\[ f_4 \]

\[ f_5 \]

\[ z_1 \]

\[ z_2 \]

\[ z_3 \]

\[ z_4 \]

\[ z_5 \]
For our start example we get:

\[
\begin{pmatrix}
  u_0 & u_1 & i_1 & u_2 & i_2 & u_3 & i_3 & u_L & i_L & i_o \\
  f_1 & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
  f_2 & \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
  f_3 & \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
  f_4 & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \\
  f_6 & \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \\
  f_8 & \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
  f_{10} & \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \\
  f_7 & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \\
  f_5 & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \\
  f_9 & \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}
\end{pmatrix}
\]
Efficient Minimal Tearing of Hybrid Algebraic Loops for Large-scale System Simulation

Tearing

Overview
Tearing

Target:
- Efficient computation with numerical solver
- Reduces size of algebraic loops

General Idea
- Assume variables are already known (=Tearing variables)
- Causalize equations with this assumption (=inner equations)
- Remaining Equations are residual equations
Choose good tearing variables

Example

\[
\begin{align*}
    u_1 - R_1 \cdot i_1 &= 0 \\
    u_2 - R_2 \cdot i_2 &= 0 \\
    u_3 - R_3 \cdot i_3 &= 0 \\
    u_1 + u_3 &= u_0 \\
    u_2 - u_3 &= 0 \\
    i_1 - i_2 - i_3 &= 0
\end{align*}
\]

- Find minimal number of iteration variables
  - Mind solvability
  - Problem is NP-hard
- Use heuristics to choose in polynomial time
  - --tearingMethod = noTearing
  - --tearingMethod = minimalTearing
  - --tearingMethod = omcTearing
  - --tearingMethod = cellier
Tearing

Choose good tearing variables

Assuem $i_3$ is known

\[
\begin{align*}
  u_3 &= R_3 \cdot i_3 \\
  u_1 &= u_0 - u_3 \\
  i_1 &= \frac{u_1}{R_1} \\
  u_2 &= u_3 \\
  i_2 &= \frac{u_2}{R_2}
\end{align*}
\]

Residual equation:

\[0 = i_1 - i_2 - i_3\]

- Try to minimize number of iteration variables
  - Mind solvability
  - Problem is NP-hard
- Use heuristics to choose in polynomial time
  - --tearingMethod = noTearing
  - --tearingMethod = minimalTearing
  - --tearingMethod = omcTearing
  - --tearingMethod = cellier
Tearing
Hybrid Algebraic Loops

What if I want to disable tearing?

- `omcTearing` or `cellier` to time consuming
- Use sparse non-linear solvers
  - That’s what `--daeMode` is doing
- Debugging and library development

Problem

Hybrid algebraic loops need tearing!
Tearing

Hybrid Algebraic Loops


- IdealDiode1.off, ..., IdealDiode6.off inside algebraic loops
- "Switching state" type: Boolean
Efficient Minimal Tearing of Hybrid Algebraic Loops for Large-scale System Simulation

Tearing

Minimal Tearing
Minimal Tearing

Perform the bare minimum of optimization
Want to causalization of:

- Discrete variables (Boolean, Integer, ...)
- Variables solved inside an algorithm with discrete variables as outputs

General Idea

1. Search all discrete variables, tearingSelect=never-variables and variables from algorithms
2. Match found variables to equations of algebraic loop
   ⇒ Set as inner variables and equations
3. Remaining variables are iteration variables
--tearingMethod=minimalTearing

- implemented in OpenModelica v1.14
  - Cases for when-equations, if-equations and algorithms will be implemented in v1.16
- Performance: Matching on discrete variables of strong component
  - \( O(|V| \cdot |E|) \) with \( V \) discrete variables of loop, \( E \) connected equations
- Supersede --tearingMethod=noTearing
Efficient Minimal Tearing of Hybrid Algebraic Loops for Large-scale System Simulation

Thank you for your attention

Questions