Efficient Symbolical and Numerical Algorithms for nonlinear model predictive control with OpenModelica

Bernhard Bachmann
Department of Engineering and Mathematics
University of Applied Sciences Bielefeld
A Modelica-based Tool Chain
(Johan Åkesson)

- Flattening of Modelica source code
  Compiler front-end
- Unstructured Flat DAE
- Symbolic manipulation
  Index reduction
  Analytic solution of simple equations
- Code generation
  Residual equations
  Analytic Jacobians
- Transformed flat ODE
  (index 1 system)
- C code
- Solution profiles
- Result
  Post processing
  Visualization
- Numerical solvers
  NLP algorithms
  Integrators
- C code

[Graphical representation of the tool chain process]
Outline
1. Excerpt of OpenModelica’s symbolic machinery

2. Symbolically derived Jacobians
   i. Directional derivatives
   ii. Sparsity pattern
   iii. Coloring of the Jacobian

3. Nonlinear Optimal Control Problem
   i. General Discretization Scheme
   ii. Multiple Shooting/Collocation
   iii. Total Collocation
   iv. Applications

4. Lessons learned & Outlook
Symbolic Machinery of OpenModelica

General representation of DAEs (continuous signals):

\[ 0 = f(t, \overset{\cdot}{x}(t), \overset{\cdot}{x}(t), x(t), \overset{\cdot}{y}(t), u(t), p) \]

- \( t \) time
- \( \overset{\cdot}{x}(t) \) vector of differentiated state variables
- \( x(t) \) vector of state variables
- \( \overset{\cdot}{y}(t) \) vector of algebraic variables
- \( u(t) \) vector of input variables
- \( p \) vector of parameters and/or constants
Basic Transformation Steps

Transformation to explicit state-space representation:

\[ 0 = f(t, \dot{x}(t), x(t), y(t), u(t), p) \]

\[ \dot{z}(t) = \begin{pmatrix} \dot{x}(t) \\ y(t) \end{pmatrix} = g(t, x(t), u(t), p) \]

\[ 0 = f(t, z(t), x(t), u(t), p) \]

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

\[ y(t) = k(t, x(t), u(t), p) \]

Implicit function theorem:

Necessary condition for the existence of the transformation is that the following matrix is regular at the point of interest:

\[ \det \left( \frac{\partial}{\partial z} f(t, z(t), x(t), u(t), p) \right) \neq 0 \]
Symbolic Transformation Algorithmic Steps

- DAEs and bipartite graph representation
  - Structural representation of the equation system

- The matching problem
  - Assign to each variable exact one equation
  - Same number of equations and unknowns

- Construct a directed graph
  - Find sinks, sources and strong components
  - Sorting the equation system

- Adjacence Matrix and structural regularity
  - Block-lower triangular form (BLT-Transformation)
DAEs and Bipartite Graph Representation

Example of a regular DAE:

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

\[
\begin{align*}
&f_1(z_3, z_4) = 0 \\
&f_2(z_2) = 0 \\
&f_3(z_2, z_3, z_5) = 0 \\
&f_4(z_1, z_2) = 0 \\
&f_5(z_1, z_3, z_5) = 0
\end{align*}
\]

Adjacence matrix

\[
\begin{pmatrix}
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1
\end{pmatrix}
\]
Solve the Matching Problem

Example of a regular DAE:

\[
\begin{align*}
    f_1(z_3, z_4) &= 0 \\
    f_2(z_2) &= 0 \\
    f_3(z_2, z_3, z_5) &= 0 \\
    f_4(z_1, z_2) &= 0 \\
    f_5(z_1, z_3, z_5) &= 0
\end{align*}
\]
Construct a Directed Graph

\[ f_1 \rightarrow z_1 \]
\[ f_2 \rightarrow z_1 \]
\[ f_3 \rightarrow z_1 \]
\[ f_4 \rightarrow z_1 \]
\[ f_5 \rightarrow z_1 \]

\[ f_1 \rightarrow z_2 \]
\[ f_2 \rightarrow z_2 \]
\[ f_3 \rightarrow z_2 \]
\[ f_4 \rightarrow z_2 \]
\[ f_5 \rightarrow z_2 \]

\[ f_1 \rightarrow z_3 \]
\[ f_2 \rightarrow z_3 \]
\[ f_3 \rightarrow z_3 \]
\[ f_4 \rightarrow z_3 \]
\[ f_5 \rightarrow z_3 \]

\[ f_1 \rightarrow z_4 \]
\[ f_2 \rightarrow z_4 \]
\[ f_3 \rightarrow z_4 \]
\[ f_4 \rightarrow z_4 \]
\[ f_5 \rightarrow z_4 \]

\[ f_1 \rightarrow z_5 \]
\[ f_2 \rightarrow z_5 \]
\[ f_3 \rightarrow z_5 \]
\[ f_4 \rightarrow z_5 \]
\[ f_5 \rightarrow z_5 \]
Construct a Directed Graph
Construct a Directed Graph

\[ f_1 | z_4 \]

\[ f_2 \]

\[ f_3 \]

\[ f_4 \]

\[ f_5 \]

\[ z_1 \]

\[ z_2 \]

\[ z_3 \]

\[ z_5 \]
Construct a Directed Graph
Construct a Directed Graph
Construct a Directed Graph

\[ f_1 | z_4 \]

\[ f_5 | z_3 \]

\[ f_4 | z_1 \]

\[ f_3 | z_5 \]

\[ f_2 \]

\[ z_2 \]
Construct a Directed Graph

sink

strong component

source

\[ f_1(z_4) = 0 \]
\[ f_2(z_2) = 0 \]
\[ f_3(z_2, z_3, z_5) = 0 \]
\[ f_4(z_1, z_3, z_5) = 0 \]
\[ f_5(z_3, z_4) = 0 \]

\[ Z_2 \begin{bmatrix} f_2 & f_4 & f_3 & f_5 & f_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \]

Tarjan’s algorithm
Further Efficiency Issues - Dummy-Derivative Method

- Matching algorithm fails
  - System is structurally singular
  - Find minimal subset of equations
    - more equations than unknown variables
  - Singularity is due to equations, constraining states

- Differentiate subset of equations
  - Static state selection during compile time
    - choose one state and corresponding derivative as purely algebraic variable
      - so-called dummy state and dummy derivative
    - by differentiation introduced variables are algebraic
    - continue matching algorithm
    - check initial conditions
  - Dynamic state selection during simulation time
    - store information on constrained states
    - make selection dynamically based on stability criteria
    - new state selection triggers an event (re-initialize states)
Further Efficiency Issues – Algebraic Loops

• Solution of linear equation systems
  – Advanced solver packages (e.g. LAPACK) are used
  – Calculate LU-Decomposition for constant matrices
    • Small systems are inverted symbolically

• Solution of nonlinear systems
  – Advanced solver packages are used
    • Performance is depending on good starting values
  – Analytical Jacobian is provided symbolically

• Tearing systems of equations
  – Reducing the iteration variables dramatically

• Analytical Jacobians of the overall system
  – Minimize simulation/integration time needed
Outline

1. Excerpt of OpenModelica’s symbolic machinery

2. Symbolically derived Jacobians
   i. Directional derivatives
   ii. Sparsity pattern
   iii. Coloring of the Jacobian

3. Nonlinear Optimal Control Problem
   i. General Discretization Scheme
   ii. Multiple Shooting/Collocation
   iii. Total Collocation
   iv. Applications

4. Lessons learned & Outlook
Fast Simulation of Fluid Models with Colored Jacobians

Willi Braun, Bernhard Bachmann
Department of Engineering and Mathematics
University of Applied Sciences Bielefeld

Stephanie Gallardo Yances, Kilian Link
Siemens AG, Energy Section
Erlangen

(see 9th International Modelica Conference)
Symbolically Generation of Jacobians

How is simulation time effected by Jacobians?

Fluid Test Model

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>231</td>
</tr>
<tr>
<td>Equations</td>
<td>942</td>
</tr>
<tr>
<td>Simulation time</td>
<td>10.8</td>
</tr>
<tr>
<td>J evaluations</td>
<td>111</td>
</tr>
<tr>
<td>J evaluation time</td>
<td>9.7</td>
</tr>
</tbody>
</table>

The evaluation of Jacobians effects the simulation time a lot!
Symbolically Generation of Jacobians

State-Space Equations

\[
\begin{pmatrix}
\dot{x}(t) \\
\dot{y}(t)
\end{pmatrix}
= \begin{pmatrix}
h(x(t), u(t), p, t) \\
k(x(t), u(t), p, t)
\end{pmatrix}
\]

Simulation

- Many integration algorithms need "the Jacobian": 
  \[ A(t) = \frac{\partial h}{\partial x} \]
- Integrator DASSL

Jacobian matrices

- \[ A(t) = \frac{\partial h}{\partial x} \]
- \[ B(t) = \frac{\partial h}{\partial u} \]
- \[ C(t) = \frac{\partial k}{\partial x} \]
- \[ D(t) = \frac{\partial k}{\partial u} \]
Symbolically Generation of Jacobians

Jacobian

\[ J_A = \frac{\partial h}{\partial x} = \begin{pmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_n} \end{pmatrix} \]

Full Symbolic Jacobian

Generation of the full symbolic jacobian requires \( n \)-times differentiation of every equation.
Symbolically Generation of Jacobians

**Jacobian**

\[
J_A = \frac{\partial h}{\partial x} = \begin{pmatrix}
\frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_n}
\end{pmatrix}
\]

**Generic Directional Derivative**

\[
J_A = \frac{\partial h}{\partial z}(e_k)
\]

\[e_k \in \mathbb{R}^n := k - \text{th coordinate vector}\]
Symbolically Generation of Jacobians

Example

model twoflattankmodel
  Real h1, h2;
  Real F1, F2;
  input Real F;
  parameter Real A1=2, A2=0.5;
  parameter Real R1=2, R2=1;
  equation
    der(h1) = (F/A1) - (F1/A1);
    der(h2) = (F1/A2) - (F2/A2);
    F1 = R1 * sqrt(h1 - h2);
    F2 = R2 * sqrt(h2);
end twoflattankmodel;

Jacobian

\[
J_A = \frac{\partial h}{\partial z} \left( \frac{\partial x}{\partial z} \right) = \left( \begin{array}{c} \frac{\partial \text{der}(h1)}{\partial z} \left( \frac{\partial h1}{\partial z}, \frac{\partial h2}{\partial z} \right) \\ \frac{\partial \text{der}(h2)}{\partial z} \left( \frac{\partial h1}{\partial z}, \frac{\partial h2}{\partial z} \right) \end{array} \right)
\]

\[
\frac{\partial \text{der}(h2)}{\partial z} = R1 \left( \frac{\partial h1}{\partial z} - \frac{\partial h2}{\partial z} \right) \left( \begin{array}{c} A1 \frac{\partial F1}{\partial z} + A2 \cdot P1 \cdot \frac{\partial A2}{\partial z} \\ A2 \frac{\partial F2}{\partial z} + A2 \cdot P2 \cdot \frac{\partial A2}{\partial z} \end{array} \right)
\]

\[
\frac{\partial \text{der}(h2)}{\partial z} = R2 \frac{\partial h2}{\partial z} \left( \begin{array}{c} A1 \frac{\partial F1}{\partial z} + A2 \cdot P1 \cdot \frac{\partial A2}{\partial z} \\ A2 \frac{\partial F2}{\partial z} + A2 \cdot P2 \cdot \frac{\partial A2}{\partial z} \end{array} \right)
\]
Symbolically Generation of Jacobians

**Numerical**

$$\frac{\partial h}{\partial x} = \frac{(h(x + \delta e_k) - h(x))}{\delta}$$

$e_k \in R^n := k - \text{th coordinate vector}$

Calculate the Jacobian numerical needs $n + 1$ call of the ODE-Block $h$.

**Symbolical**

$$J_A = \frac{\partial h}{\partial z}(e_k)$$

$e_k \in R^n := k - \text{th coordinate vector}$

Evaluate the Jacobian symbolical needs $n$ calls of Directional Derivative.

The amount of calls could be reduced by exploiting the sparsity pattern and partitioning the columns by colors.
Compute sparsity pattern of the Jacobians

Example system

\[ z(t) = f(x(t), t) \]
Compute sparsity pattern of the Jacobians

\[ J = \begin{pmatrix}
0 & 0 & 0 & 0 & * \\
0 & 0 & 0 & 0 & * \\
* & 0 & * & 0 & * \\
0 & * & 0 & 0 & 0 \\
0 & * & 0 & * & * 
\end{pmatrix} \]

<table>
<thead>
<tr>
<th>(z_1)</th>
<th>(z_3)</th>
<th>(z_4)</th>
<th>(z_2)</th>
<th>(z_5)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(x_4)</th>
<th>(x_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Accumulation Lists**
- \(f_4\): \(<5>\)
- \(f_5\): \(<5>\)
- \(f_1\): \(<1,3,5>\)
- \(f_2\): \(<2>\)
- \(f_3\): \(<5,2>\)
Utilize sparsity pattern of the Jacobians

\[
J = \begin{pmatrix}
  \hat{j}_{11} & \hat{j}_{12} & 0 & 0 & \hat{j}_{15} \\
  0 & 0 & \hat{j}_{23} & 0 & 0 \\
  0 & \hat{j}_{32} & \hat{j}_{33} & \hat{j}_{34} & 0 \\
  \hat{j}_{41} & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & \hat{j}_{54} & \hat{j}_{55}
\end{pmatrix}
\]
Utilize sparsity pattern of the Jacobians

**Jacobian**

\[ J = \begin{pmatrix}
  j_{11} & j_{12} & 0 & 0 & j_{15} \\
  0 & 0 & j_{23} & 0 & 0 \\
  0 & j_{32} & j_{33} & j_{34} & 0 \\
  j_{41} & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & j_{54} & j_{55}
\end{pmatrix} \]

\[ J_R = \begin{pmatrix}
  j_{11} & j_{12} & j_{15} \\
  0 & 0 & j_{23} \\
  j_{34} & j_{32} & j_{33} \\
  j_{41} & 0 & 0 \\
  j_{54} & 0 & j_{55}
\end{pmatrix} \]
Performance gain of implementation

Model details
- States: 231
- Equations: 1,006
- JacElements: 53,361
- NonZero: 3,032
- Colors: 79

Simulation statistics

<table>
<thead>
<tr>
<th>method</th>
<th>steps</th>
<th>F-Eval</th>
<th>J-Eval</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>922</td>
<td>27184</td>
<td>111</td>
<td>10.8</td>
</tr>
<tr>
<td>numC</td>
<td>922</td>
<td>8929</td>
<td>94</td>
<td>4.5</td>
</tr>
<tr>
<td>sym</td>
<td>937</td>
<td>1539</td>
<td>103</td>
<td>8.5</td>
</tr>
<tr>
<td>symC</td>
<td>937</td>
<td>1539</td>
<td>103</td>
<td>4.3</td>
</tr>
<tr>
<td>Dymola</td>
<td>783</td>
<td>8772</td>
<td>90</td>
<td>1.6</td>
</tr>
</tbody>
</table>
Outline

1. Excerpt of OpenModelica’s symbolic machinery

2. Symbolically derived Jacobians
   i. Directional derivatives
   ii. Sparsity pattern
   iii. Coloring of the Jacobian

3. Nonlinear Optimal Control Problem
   i. General Discretization Scheme
   ii. Multiple Shooting/Collocation
   iii. Total Collocation
   iv. Applications

4. Lessons learned & Outlook
Parallel Multiple-Shooting and Collocation Optimization with OpenModelica

Bernhard Bachmann, Lennart Ochel, Vitalij Ruge
Mathematics and Engineering
University of Applied Sciences Bielefeld

Mahder Gebremedhin, Peter Fritzson,
PELAB – Programming Environment Lab

Vaheed Nezhadali, Lars Eriksson, Martin Sivertsson
Vehicular Systems
Linköping University

(see 9th International Modelica Conference)
Nonlinear Optimal Control Problem (NOCP)

Mathematical problem formulation

- objective function

$$\min_{u(t)} J(x(t), u(t), t) = E(x(t_f)) + \int_{t_0}^{t_f} L(x(t), u(t), t) \, dt$$

- subject to

$$x(t_0) = h_0$$  \hspace{1cm} initial conditions

$$\dot{x}(t) = f(x(t), u(t), t)$$  \hspace{1cm} \text{DAEs, Modelica}

$$g(x(t), y(t), u(t), t) \geq 0$$  \hspace{1cm} path constraints

$$r(x(t_f), y(t_f)) = 0$$  \hspace{1cm} terminal constraints
Theoretical Background
General discretization scheme

\[ x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t), u(t), t) \, dt \]

\[ x_i(t_i) = h_i \]
Theoretical Background

Multiple Shooting/Collocation

- Solve sub-problem in each sub-interval

\[ x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t), u(t), t) \, dt \approx F(t_i, t_{i+1}, h_i, u_i), \quad x_i(t_i) = h_i \]
Theoretical Background

Multiple Shooting / Collocation Optimization

- Discretized Nonlinear Optimal Control Problem
  - objective function (integral approximation by trapezoidal rule)

  $$\min_{u(t)} J(x(t), u(t), t) = E(h_n) + \frac{\Delta t}{2} \sum_{i=0}^{n-1} L(h_i, u_i, t_i) + L(h_{i+1}, u_i, t_{i+1})$$

  - subject to

  $$x(t_0) = h_0$$
  $$F(t_i, t_{i+1}, h_i, u_i) = h_{i+1}$$
  $$g(h_i, u_i, t_i) \geq 0$$
  $$g(h_{i+1}, u_i, t_{i+1}) \geq 0$$
Theoretical Background

Multiple Shooting / Collocation Optimization

Local Discretization Problem

Optimizer

Shooting Optimization

new value

evaluate

solve

Optimum

Efficient Symbolical and Numerical Algorithms for nonlinear model predictive control with OpenModelica

Bernhard Bachmann, et. al.
Theoretical Background

Total Collocation Optimization

- Total conditions
- Optimizer
- optimum

evaluate

new value
Theoretical Background

Total Collocation Optimization

- Discretized Nonlinear Optimal Control Problem
  - objective function (integral approximation by Gauß quadrature)

\[
\min_{u(t)} J(x(t), u(t), t) = E(h_n) + \Delta t \sum_{j=0}^{m} w_j \cdot \sum_{i=0}^{n-1} L(h_i^{(j)}, u_i, t_i + s_j)
\]

- subject to

\[
\begin{align*}
x(t_0) &= h_0 \\
g(h_i, u_i, t_i) &\geq 0 \\
g(h_{i+1}, u_i, t_{i+1}) &\geq 0
\end{align*}
\]

additional collocation conditions
Theoretical Background

Collocation Condition – Approximation of States

• Assumption:
  States are locally polynomial

\[ x_i(t_i + \hat{s} \cdot \Delta t) = p_0(\hat{s}) \cdot h_{i-1}^{(m)} + \sum_{j=1}^{m} p_j(\hat{s}) \cdot h_i^{(j)} \]

where \( x_i(t_i + \hat{s}_k \cdot \Delta t) = \delta_{k,0} \cdot h_{i-1}^{(m)} + \sum_{j=1}^{m} \delta_{k,j} \cdot h_i^{(j)} = h_{i}^{(k)} \)

\( \hat{s}_k \) are the Radau points
\( p_j(\hat{s}) \) are the Lagrange Basis polynomial to the nodes \( \hat{s}_k \)

• Collocation conditions

\[ \Delta t \cdot f(h_i^{(j)}, u_i, t_i + \hat{s}_k \cdot \Delta t) = p_0'(\hat{s}_k) \cdot h_{i-1}^{(m)} + \sum_{j=1}^{m} p_j'(\hat{s}_k) \cdot h_i^{(j)} \]
Theoretical Background

Collocation Condition – Approximation of State Derivatives

- Assumption:
  State derivatives are locally polynomial

\[ \Delta t \cdot f(x_i(t_i + \hat{s} \cdot \Delta t), u_i, t_i + \hat{s} \cdot \Delta t) = \sum_{j=0}^{m} p_j(\hat{s}) \cdot f_i^{(j)} \]

where \( \Delta t \cdot f(h_i^{(k)}, u_i, t_i + \hat{s}_k \cdot \Delta t) = \sum_{j=0}^{m} \delta_{k,j} \cdot f_i^{(j)} = f_i^{(k)} \)

\( \hat{s}_j \) are the Lobatto points

\( p_j(\hat{s}) \) are the Lagrange Basis polynomial to the nodes \( \hat{s}_j \)

- Collocation conditions

\[ h_i^{(k)} = \sum_{j=0}^{m} \int p_j(\hat{s}_k) \cdot f_i^{(j)} + h_i^{(m)} \]
Applications – Diesel Electric Powertrain

• Find fuel optimal control and state trajectories from idling condition to a certain power level

• Nonlinear mean value engine model

• Only diesel operating condition

• Mathematical problem formulation:
  – 2 inputs \((u_f, u_{wg})\)
  – 4 states \((\omega_{ice}, p_{im}, p_{em}, \omega_{tc})\)
  – 32 algebraic equations
Applications – Diesel Electric Powertrain

- Mathematical problem formulation
  - Object function
    \[
    \min_{u(t)} \sum_{i=1}^{4} (x_i(t_f) - x_i^{ref})^2 + \int_0^T m_f \, dt
    \]
  - subject to
    \[
    \begin{align*}
    \dot{x}_1 &= f_1(x_2, x_3, u_1) \\
    \dot{x}_2 &= f_2(x_1, x_2, x_4) \\
    \dot{x}_3 &= f_3(x_1, x_2, x_3, u_1, u_2) \\
    \dot{x}_4 &= f_4(x_2, x_3, x_4, u_2)
    \end{align*}
    \]
    \[
    x_{lb_i} \leq x_i \leq x_{ub_i}, \ i = 1, \ldots, 4 \\
    0 \leq u_1, u_2 \leq 1
    \]

Engine is accelerated only near the end of the time interval to meet the end constraints while minimizing the fuel consumption
Implementation Details – Current Status

- Realization with OpenModelica Environment
- Optimica prototype implementation is available
- Using Ipopt for solution process
- Necessary derivatives are numerically calculated
  – Gradients, Jacobians, Hessians, ...
- **But:** Complete tool chain not yet implemented

Test Environment

- Processor:
  - 2xIntel Xeon CPU E5-2650
  - 16 cores @ 2.00GHz
- OpenMP
Implementation Details – Ipopt & Parallelization

- Schematic view of the required components of Ipopt
Implementation Details - Numerical Optimization

- Enormous speed-up when utilizing sparse Jacobian matrix
- Speed-up for the over-all optimization
- Sparse-structure model independent
### Results - Diesel Electric Powertrain

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPLE SHOOTING</td>
<td>921.6s</td>
</tr>
<tr>
<td>MULTIPLE COLLOCATION</td>
<td>29519.8s</td>
</tr>
<tr>
<td>TOTAL COLLOCATION</td>
<td>9.5s</td>
</tr>
<tr>
<td>TOTAL COLLOCATION 2</td>
<td>15.6s</td>
</tr>
</tbody>
</table>

![State trajectories](image-url)
Results - Diesel Electric Powertrain

- Ipopt runs in serial mode
- Most execution time is elapsed in Jacobian calculation and solution process of the local discretization problem
- Reasonable speed-up
- Factors are non-optimal due to memory handling
  - Further investigations will be performed
Results - Diesel Electric Powertrain

- Ipopt runs in serial mode
- Less execution time is elapsed in Jacobian calculation
- Reasonable speed-up for Jacobian calculation
- Factors are non-optimal due to memory handling
- Overall Speed-up increases with model complexity
  - Parallelizing of Ipopt necessary

**TOTAL COLLOCATION**

![Graph 1: TOTAL COLLOCATION](image1)

**TOTAL COLLOCATION 2**

![Graph 2: TOTAL COLLOCATION 2](image2)
Lessons learned

• Symbolic calculation of derivatives improve performance
  – Jacobian, Gradient, ...

• Utilizing sparsity pattern is crucial

• In serial mode total collocation methods are superior to multiple shooting/collocation methods

• Parallelizing the algorithms performs better on multiple shooting/collocation methods

• Symbolic transformation to ODE form is a key issue for the realization of an automatic tool chain
Example 1 – From the dark side (Francesco Casella)

optimization Example1A(
  objective = a1/b1*(x1 - x10)^2 +
  a2/b2*(x2 - x20)^2 +
  a3/b3*(x3 - x30)^2,

  ...
)
end Example1A;

optimization Example1B(
  objective = f1 + f2 + f3,

  ...
)
equation
  f1 = a1/b1*(x1 - x10)^2;
  f2 = a2/b2*(x2 - x20)^2;
  f3 = a3/b3*(x3 - x30)^2;

  ...
end Example1B;
Example 2 – From the dark side (Francesco Casella)

```plaintext
optimization Example2A(
    parameter Real PR = 10;
    ...
    equation
        p_in/p_out = PR "Turbine pressure ratio";
    ...
    end Example1A;
)

optimization Example2A(
    parameter Real PR = 10;
    ...
    equation
        p_in = PR*p_out "Turbine pressure ratio";
    ...
    end Example1A;
)```
Future work

• Implement complete tool chain in OpenModelica
• Automatic generation of simulation code based on Optimica
• Utilizing symbolically derived derivative information
  – Gradient, Jacobian, Hessian, ...
• Further improvements with appropriate scaling
• Exploiting parallel evaluation of the optimization method
• Advanced use of OMC symbolic machinery
  – Efficient handling of model dependent algebraic loops
• Generalization of NOCP problem formulation
  – e.g. time minimal optimization, parameter estimation

• Further testing on industrial-relevant problems
Thank you

Questions?