TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing – TU Dresden

PARADOM

Parallel Algorithmic Differentiation in OpenModelica for Energy-Related Simulations and Optimizations

Willi Braun, Kshitij Kulshreshtha and Martin Flehmig

OpenModelica Workshop 2017, 06.02.2017

E-Mail: martin.flehmig@tu-dresden.de

Center for Information Services & High Performance Computing

Partners

PARADOM

SIEMENS

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Rexroth
Bosch Group

ABB

FH Bielefeld
University of Applied Sciences

LTX Simulation GmbH

TECHNISCHE UNIVERSITÄT DRESDEN

SPONSORED BY THE

Federal Ministry of Education and Research

Motivation I

Change in Generation of Electricity

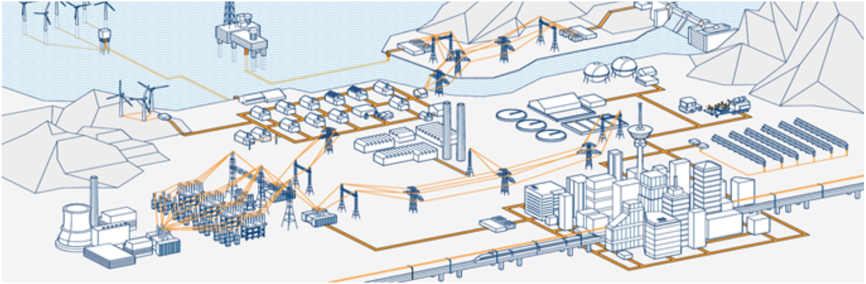
In Former Times

- Classically large power plants with predetermined plans of action

Brown coal power station in Jänschwalde (Germany), guentherhh, CC BY 2.0

Today and in Future

- Many distributed small power producers (technologies: solar power, wind energy, biogas, etc.)
- Conventional power stations ensure basic demand and peaks



©ABB

Challenges

- Combine small producers and conventional power plants into virtual power plants
- Real time optimization of all producers, i.e., flexible adaption to demands

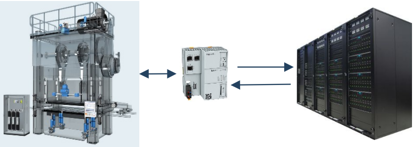
Motivation II

Optimal Control and Model Predictive Control

Motivation II

Optimal Control

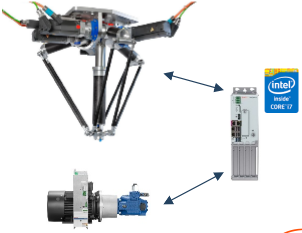
- Measured data of the real process are automatically transfered to the HPC-system
- Controller triggers a dynamic optimization at defined time
- Optimization is executed on HPC-system
- Adaption to changed production conditions



©Bosch Rexroth

Model Predictive Control

- Predictive control basing on a model
- Dynamic optimization problem is solved in every controller cycle
- Real-time requirements
- High-performance multi-core control hardware



©Bosch Rexroth

Round-up of Motivation

Tasks

- Modelling of the energy-related facilities and their components
- Simulation and Optimization
  - Components and processes
  - Performance of products in applications
- Online optimization to allow flexible adaption to demands and conditions

Challenges

- Rapidly increasing systems
- More and more comprehensive and complex models
- Limits of available optimization technologies will be reached in near future

Computation of Derivatives

The considered simulations and optimizations fundamentally base on the efficient computation of first and higher order derivatives.

How to obtain derivatives?

- Hand Coded
  - Implement analytical expression for the derivatives
- Finite Differences
  - Approximation of the derivatives by difference quotients, e.g., $f'(x) \approx \frac{f(x+h)-f(x)}{h}$
- Symbolic Differentiation
  - Make use of computer algebra systems

Computation of Derivatives

The considered simulations and optimizations fundamentally base on the efficient computation of first and higher order derivatives.

How to obtain derivatives?

- Hand Coded
  - Implement analytical expression for the derivatives
- Finite Differences
  - Approximation of the derivatives by difference quotients, e.g., $f'(x) \approx \frac{f(x+h)-f(x)}{h}$
- Symbolic Differentiation
  - Make use of computer algebra systems

Well know downsides.

But, can we do better?

Computation of Derivatives

The considered simulations and optimizations fundamentally base on the efficient computation of first and higher order derivatives.

How to obtain derivatives?

- Hand Coded
  - Implement analytical expression for the derivatives
- Finite Differences
  - Approximation of the derivatives by difference quotients, e.g., $f'(x) \approx \frac{f(x+h)-f(x)}{h}$
- Symbolic Differentiation
  - Make use of computer algebra systems

Well know downsides.

But, can we do better?

Yes, we can!

Computation of Derivatives

Algorithmic Differentiation (AD)

- Computing analytic derivatives of functions present in source code
- Exact derivatives within machine precision
- Low overhead

Basic idea

- Function present in source code is can be seen as a sequence of elementary arithmetic operations and functions
- Analytic differentiation of elementary functions + propagation by chain rule

Two basic modes: Forward and reverse

AD forward by Example

$$y = f(x_1, x_2, x_3) = \sin(x_1 x_2) x_3$$

- Decompose original function f into intrinsic functions

$$\begin{array}{lcl} v_1 & = & x_1 x_2 \\ v_2 & = & \sin(v_1) \\ v_3 & = & v_2 x_3 \\ \hline y & = & v_3 \end{array}$$

AD forward by Example

$$y = f(x_1, x_2, x_3) = \sin(x_1 x_2) x_3$$

- Decompose original function f into intrinsic functions
- Associate each intermediate variable v with a derivative $\dot{v} = \frac{\partial v_i}{\partial x}$
- Apply chain rule

$$\begin{array}{rcl} v_1 & = & x_1 x_2 \\ v_2 & = & \sin(v_1) \\ v_3 & = & v_2 x_3 \\ \hline y & = & v_3 \end{array}$$

$$\begin{array}{rcl} \dot{v}_1 & = & \dot{x}_1 x_2 + x_1 \dot{x}_2 \\ \dot{v}_2 & = & \cos(v_1) \dot{v}_1 \\ \dot{v}_3 & = & \dot{v}_2 x_3 + v_2 \dot{x}_3 \\ \hline \dot{y} & = & \dot{v}_3 \end{array}$$

AD forward by Example

$y = f(x_1, x_2, x_3) = \sin(x_1 x_2) x_3,$

What is $\frac{\partial y}{\partial x_1}$ at $(1, 3, 7)$?

- Chose x_1 as only independent variable, thus $\dot{x}_1 = 1$, $\dot{x}_2 = 0$ and $\dot{x}_3 = 0$

$$
\begin{aligned}
v_1 &= x_1 x_2 &&= 3 \\
\dot{v}_1 &= \dot{x}_1 x_2 + x_1 \dot{x}_2 &&= 3 \\
v_2 &= \sin(v_1) &&= 0.14112 \\
\dot{v}_2 &= \cos(v_1) \dot{v}_1 &&= -2.96997 \\
v_3 &= v_2 x_3 &&= 0.98784 \\
\dot{v}_3 &= \dot{v}_2 x_3 + v_2 \dot{x}_3 &&= -20.78984 \\
\hline
y &= v_3 &&= 0.98784 \\
\dot{y} &= \dot{v}_3 &&= -20.78984
\end{aligned}
$$

AD forward by Example

$y = f(x_1, x_2, x_3) = \sin(x_1 x_2) x_3,$

What is $\frac{\partial y}{\partial x_1}$ at $(1, 3, 7)$?

- Chose x_1 as only independent variable, thus $\dot{x}_1 = 1$, $\dot{x}_2 = 0$ and $\dot{x}_3 = 0$

$$
\begin{array}{lclcl}
v_1 & = & x_1 x_2 & = & 3 \\
\dot{v}_1 & = & \dot{x}_1 x_2 + x_1 \dot{x}_2 & = & 3 \\
v_2 & = & \sin(v_1) & = & 0.14112 \\
\dot{v}_2 & = & \cos(v_1)\dot{v}_1 & = & -2.96997 \\
v_3 & = & v_2 x_3 & = & 0.98784 \\
\dot{v}_3 & = & \dot{v}_2 x_3 + v_2 \dot{x}_3 & = & -20.78984 \\
\hline
y & = & v_3 & = & 0.98784 \\
\dot{y} & = & \dot{v}_3 & = & -20.78984
\end{array}
$$

TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

AD forward by Example

$y = f(x_1, x_2, x_3) = \sin(x_1 x_2) x_3,$

What is $\frac{\partial y}{\partial x_1}$ at $(1, 3, 7)$?

- Chose x_1 as only independent variable, thus $\dot{x}_1 = 1$, $\dot{x}_2 = 0$ and $\dot{x}_3 = 0$

$$
\begin{array}{lcllcl}
v_1 & = & x_1 x_2 & = & 3 \\
\dot{v}_1 & = & \dot{x}_1 x_2 + x_1 \dot{x}_2 & = & 3 \\
v_2 & = & \sin(v_1) & = & 0.14112 \\
\dot{v}_2 & = & \cos(v_1) \dot{v}_1 & = & -2.96997 \\
v_3 & = & v_2 x_3 & = & 0.98784 \\
\dot{v}_3 & = & \dot{v}_2 x_3 + v_2 \dot{x}_3 & = & -20.78984 \\
\hline
y & = & v_3 & = & 0.98784 \\
\dot{y} & = & \dot{v}_3 & = & -20.78984
\end{array}
$$

- Derivatives within working accuracy
- All gradients cost $\mathcal{O}(n)$ function evaluations

AD in OpenModelica

So, should we implement AD functionality in OpenModelica?

No, use a well established tool, like ADOL-C!

Package ADOL-C (**A**utomatic **D**ifferentiation by **O**ver**L**oading in **C**++)

- Open-Source
- Used in many applications
- Hugh range of functions
- Bases on operator overloading in C/C++

```
class adouble {
  double val;
  double dot;
}
```

```
adouble operator* (adouble a, adouble b) {
  adouble c;
  c.val = a.val * b.val;
  c.dot = a.dot * b.val + a.val * b.dot;
  return c;
}
```

Connecting ADOL-C and OpenModelica

OpenModelica + ADOL-C

- First prototype 2014 with C++ code and `adouble`
- New prototype generates directly a *trace*
- No compilation needed, just read the *trace*

Connecting ADOL-C and OpenModelica

OpenModelica + ADOL-C

- First prototype 2014 with C++ code and `adouble`
- New prototype generates directly a *trace*
- No compilation needed, just read the *trace*

Example:

```
model A
  parameter Real a=-0.25;
  Real x,y;
equation
 der(y) = y/x + x*3.0 + a;
 der(x) = x + log(x)*(-3.0);
end A;
```

TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

Connecting ADOL-C and OpenModelica

OpenModelica + ADOL-C

- First prototype 2014 with C++ code and `adouble`
- New prototype generates directly a *trace*
- No compilation needed, just read the *trace*

Example:

```
model A
  parameter Real a=-0.25;
  Real x,y;
equation
 der(y) = y/x + x*3.0 + a;
 der(x) = x + log(x)*(-3.0);
end A;
```

Trace for model A:

```
// allocation of used variables
{ op:assign_d_zero loc:0 }
{ op:assign_d_zero loc:1 }
{ op:assign_d_zero loc:2 }
{ op:assign_d_zero loc:3 }
// define independent -> x, y
{ op:assign_ind loc:0 }
{ op:assign_ind loc:1 }
// operations
{ op:div_a_a loc:1 loc:0 loc:4 }
{ op:mult_d_a loc:0 loc:5 val:3.0 }
{ op:assign_p loc:1 loc:6 }
{ op:plus_a_a loc:5 loc:6 loc:7 }
{ op:plus_a_a loc:4 loc:7 loc:3 }
{ op:log_op loc:0 loc:4 }
{ op:mult_d_a loc:4 loc:5 val:-3.0 }
{ op:plus_a_a loc:0 loc:5 loc:2 }
// define depenpendent -> der(x), der(y)
{ op:assign_dep loc:2 }
{ op:assign_dep loc:3 }
// death_not
{ op:death_not loc:0 loc:9 }
// num real parameters
{ op:set_numparam loc:1 }
```

OpenModelica + ADOL-C - First Results

Example:
ScalableTestSuite.Elementary.SimpleODE.Models.CascadedFirstOrder

Sparse Jacobian Evaluation:

N	ADOL-C	OM Symbolical
100	0.000480442	0.000156783
200	0.000830835	0.000413299
400	0.00157551	0.000952923
800	0.00294508	0.00209405
1600	0.00676732	0.00536921
3200	0.0141433	0.012003
6400	0.0390204	0.0310391
12800	0.0771545	0.0756394

Generate and Read Performance:

	ADOL-C		OM Sym.
N	generate	read	generate
100	0.0008721	0.0289017	0.0255
200	0.001601	0.0569519	0.04937
400	0.004216	0.114088	0.1044
800	0.006973	0.227438	0.2311
1600	0.01347	0.521812	0.557
3200	0.0259	0.898992	1.732
6400	0.05123	1.8135	4.436
12800	0.1087	3.62892	43.57

OpenModelica + ADOL-C - Status

Status: *Early-pre-alpha* prototype

We can create traces for (simple) expressions using

- standard operators (e.g., $+, -, *, /$)
- and standard functions (e.g., $\sin, \cos, \log, \exp$).

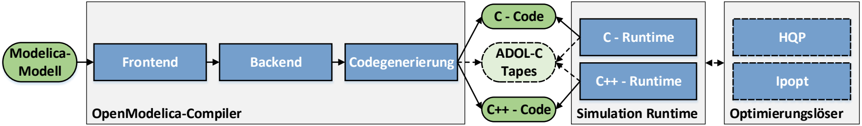
Outlook

- If expressions
- Arrays
- Records, functions, algorithms
- Algebraic loops

Goals within PARADOM

Development of parallel algorithms and application of efficient optimization methods within the OpenModelica environment with respect to HPC systems

- Integration of ADOL-C into the OpenModelica Compiler and the simulation runtimes C and C++
- Linking of the optimization solvers Ipopt and HQP and OpenModelica
- Provisioning of interfaces to suitable solvers for systems of equations (e.g., MUMPS, SuperLU, SuiteSparse)
- Parallelization of derivative computation in ADOL-C
- Development of parallel multiple shooting methods within HQP



TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

Wide Appeal and Sustainability

OpenModelica, ADOL-C and HQP are open-source software projects

- We will develop on the corresponding repositories
- Developments can be used immediately by the communities
- Feedback from the users
- Early and continuous build-up of know-how on user side

ADOL-C: parallel computation of derivatives

- Independent from OpenModelica
- Guarantees future

OpenModelica

- Enable and speed-up large simulations
- New users due to new possibilities/capabilities

Thank you for your attention.