Modeling of Large-Scale Power Generation and Transmission Networks in OpenModelica

Francesco Casella

(francesco.casella@polimi.it)

(francesco.casella@dynamica-it.com)

POLITECNICO
MILANO 1863

Dynamica

Introduction and Motivation

- Accelerating pace of innovation and change in European PG&T networks
    - Intermittent renewables
    - Distributed generation
    - Innovative components (DC links, power electronics)
    - Tighter integration between different countries

Introduction and Motivation

- Accelerating pace of innovation and change in European PG&T networks
  - Intermittent renewables
  - Distributed generation
  - Innovative components (DC links, power electronics)
  - Tighter integration between different countries

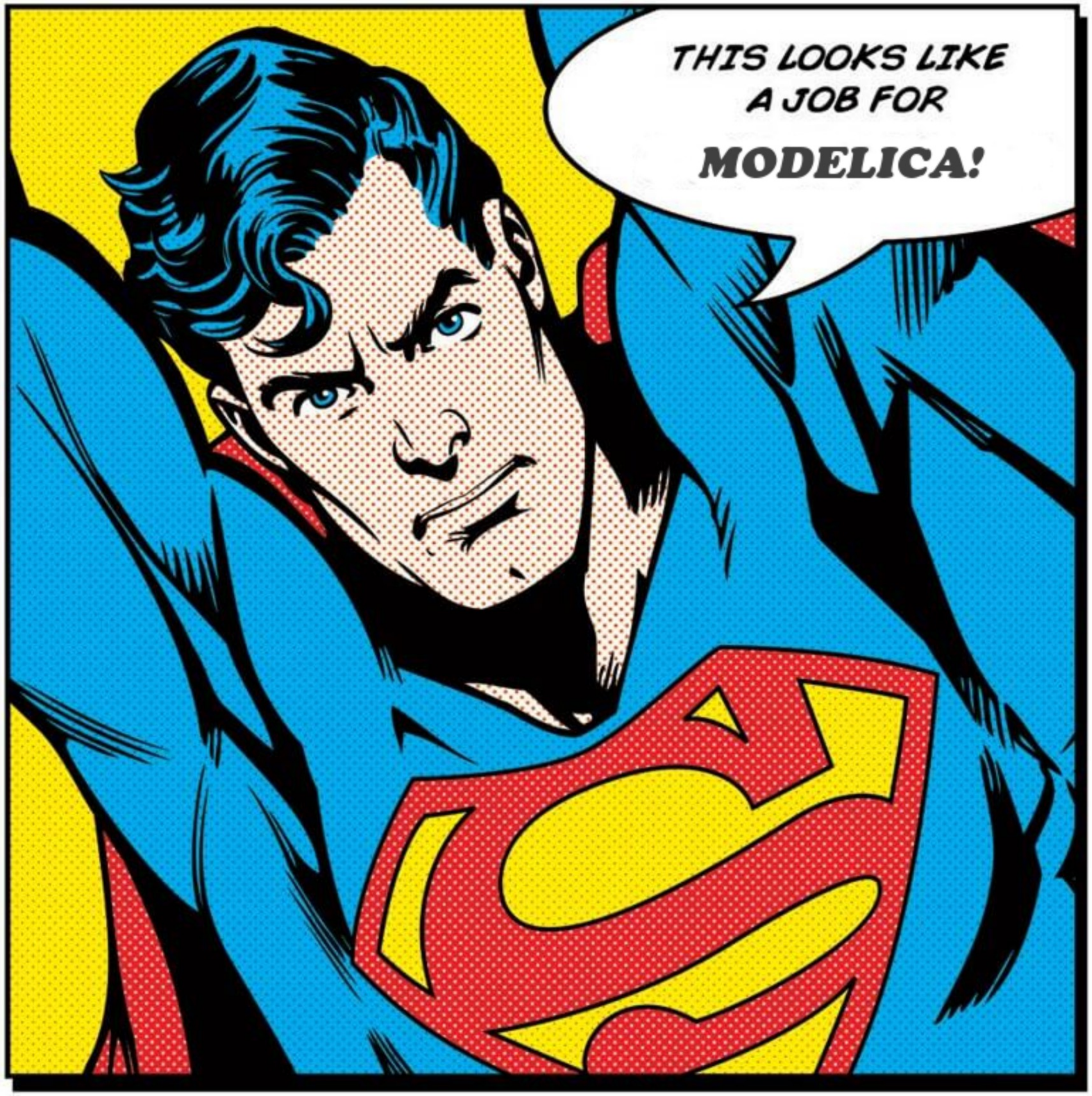
- Mandate to Transmission System Operators (TSO) to exchange data about the national systems
  - CIM standard (*parameters*) → good for static analysis
  - Transient analysis requires exchanging *models* (→ equations)

Introduction and Motivation

- Accelerating pace of innovation and change in European PG&T networks
  - Intermittent renewables
  - Distributed generation
  - Innovative components (DC links, power electronics)
  - Tighter integration between different countries

- Mandate to Transmission System Operators (TSO) to exchange data about the national systems
  - CIM standard (*parameters*) → good for static analysis
  - Transient analysis requires exchanging *models* (→ equations)

- Currently used network simulation tools
  - In-house legacy codes (SICRE, EUROSTAG)
  - Commercial power system simulation tools (PSS/E, DigSILENT)
  - Old-fashioned, procedural, inflexible, closed-source, etc.
  - Adding new models is difficult
  - Doing things other than simulation is difficult (calibration, estimation, optimization, etc.)


THIS LOOKS LIKE
A JOB FOR
MODELICA!

To Probe Further

Please attend Luigi Vanfretti's tutorial and presentation tomorrow @ MODPROD workshop on the use of Modelica in the field of Power System Simulation

Research Question:

Research Question:

Can OpenModelica handle regional / national / continental sized PG&T system models?

Electro-Mechanical Models of PG&T Systems

- Power Generation units
  - Low-order models of prime mover and governor
  - Low-order model of synchronous generator in Park coordinates
  - Rotating mechanical generator with inertia
  - Linear dynamics plus saturations
  - Causal models, heritage of developements in the '90s

Electro-Mechanical Models of PG&T Systems

- Power Generation units
  - Low-order models of prime mover and governor
  - Low-order model of synchronous generator in Park coordinates
  - Rotating mechanical generator with inertia
  - Linear dynamics plus saturations
  - Causal models, heritage of developements in the '90s

- Transmission lines, transformers
  - Sinusoidal steady-state models (electromagnetic transients neglected)
  - Phasor-based models (complex numbers)
  - (Very large) linear implicit systems of algebraic equations

Electro-Mechanical Models of PG&T Systems

- Power Generation units
  - Low-order models of prime mover and governor
  - Low-order model of synchronous generator in Park coordinates
  - Rotating mechanical generator with inertia
  - Linear dynamics plus saturations
  - Causal models, heritage of developements in the '90s

- Transmission lines, transformers
  - Sinusoidal steady-state models (electromagnetic transients neglected)
  - Phasor-based models (complex numbers)
  - (Very large) linear implicit systems of algebraic equations

- Load models
  - Linear models (constant impedance)
  - PQ models (nonlinear)
  - Pseudo-PQ models (controlled impedance)

Electro-Mechanical Models of PG&T Systems

- Mathematical structure of the DAEs
  - Very large, very sparse DAEs
    (few nonzero elements for each equation)
  - Index-1 DAEs

Electro-Mechanical Models of PG&T Systems

- Mathematical structure of the DAEs
  - Very large, very sparse DAEs
    (few nonzero elements for each equation)
  - Index-1 DAEs
- Mathematical structure of the causalized ODEs
  - One big implicit linear system (transmission lines, loads, transformers)
  - Explicit assigments or small linear systems for the rest of the equations
  - Dense Jacobian $\partial f / \partial x \rightarrow$ unsuitable for implicit solvers!

Electro-Mechanical Models of PG&T Systems

- Mathematical structure of the DAEs
  - Very large, very sparse DAEs
    (few nonzero elements for each equation)
  - Index-1 DAEs
- Mathematical structure of the causalized ODEs
  - One big implicit linear system (transmission lines, loads, transformers)
  - Explicit assigments for the rest of the equations
  - Dense Jacobian $\partial f / \partial x \rightarrow$ unsuitable for implicit solvers!
- Explicit ODE solvers (Runge-Kutta)
  - Fixed time step around 20 ms
  - Sparse solver for the large implicit linear system – no tearing!
  - Avoid nonlinear systems (stemming from rigorous PQ loads)

Electro-Mechanical Models of PG&T Systems

- Mathematical structure of the DAEs
  - Very large, very sparse DAEs
    (few nonzero elements for each equation)
  - Index-1 DAEs
- Mathematical structure of the causalized ODEs
  - One big implicit linear system (transmission lines, loads, transformers)
  - Explicit assigments for the rest of the equations
  - Dense Jacobian $\partial f / \partial x \rightarrow$ unsuitable for implicit solvers!
- Explicit ODE solvers (Runge-Kutta)
  - Fixed time step around 20 ms
  - Sparse solver for the large implicit linear system – no tearing!
  - Avoid nonlinear systems (stemming from rigorous PQ loads)
- Implicit DAE Sparse Solvers (IDA/Kinsol)
  - Exploiting sparsity is essential (up to one million equations!)
  - Can handle much larger time steps if nothing happens on the system
  - Event handling can be problematic (large event iterations)
  - Nonlinear (PQ) loads are not a problem, only one nonlinear iteration for each time step

Feasibility Study in OMC

- Study carried out by Dynamica for CESI in partnership with Politecnico
- Basic standard models for all components, no fancy stuff
- Use appropriate formalisms
  - Complex numbers for equation-based phasor models
  - Block diagrams for controllers (IEEE standards)
- Focus on performance (compilation & simulation), not on results

Feasibility Study in OMC

- Study carried out by Dynamica for CESI in partnership with Politecnico
- Basic standard models for all components, no fancy stuff
- Use appropriate formalisms
  - Complex numbers for equation-based phasor models
  - Block diagrams for controllers (IEEE standards)
- Focus on performance (compilation & simulation), not on results
- Integration strategy tested so far
  - Causalized ODEs
  - Explict Runge-Kutta 4th order @ 20 ms time step
  - KLU sparse solver for the large linear network model
- Three test cases
  - Rete C (Ireland, 600 nodes, public domain)
  - Rete E (Italy, 1800 nodes, proprietary)
  - Rete D (France, detailed, 5000 nodes, public domain)

Feasibility Study in OMC

- Study carried out by Dynamica for CESI in partnership with Politecnico
- Basic standard models for all components, no fancy stuff
- Use appropriate formalisms
  - Complex numbers for equation-based phasor models
  - Block diagrams for controllers (IEEE standards)
- Focus on performance (compilation & simulation), not on results
- Integration strategy tested so far
  - Causalized ODEs
  - Explict Runge-Kutta 4th order @ 20 ms time step
  - KLU sparse solver for the large linear network model
- Three test cases
  - Rete C (Ireland, 600 nodes, public domain)
  - Rete E (Italy, 1800 nodes, proprietary)
  - Rete D (France, detailed, 5000 nodes, public domain)
- Modelica code generated automatically from PSS/E netlists by a Python script
- Small library developed for this project (rapid prototyping approach)

Code Samples from the Library – Equation-Based

```
package Types
  operator record ComplexVoltage = Complex(redeclare SI.Voltage re, redeclare SI.Voltage im);
  operator record ComplexCurrent = Complex(redeclare SI.Current re, redeclare SI.Current im);
  operator record Admittance = Complex(redeclare SI.Conductance re, redeclare SI.Susceptance im);
  operator record Impedance = Complex(redeclare SI.Resistance re, redeclare SI.Reactance im);
  operator record ComplexPower = Complex(redeclare SI.Power re, redeclare SI.ReactivePower im);
end Types;
```

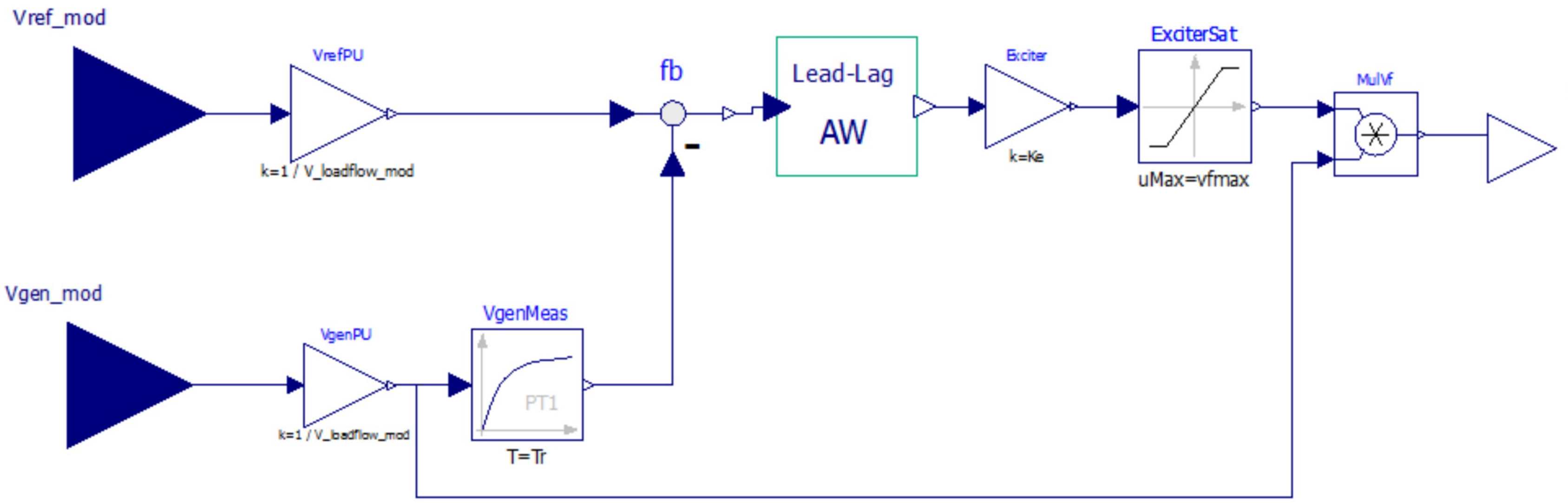
Code Samples from the Library – Equation-based

```
package Types
  operator record ComplexVoltage = Complex(redeclare SI.Voltage re, redeclare SI.Voltage im);
  operator record ComplexCurrent = Complex(redeclare SI.Current re, redeclare SI.Current im);
  operator record Admittance = Complex(redeclare SI.Conductance re, redeclare SI.Susceptance im);
  operator record Impedance = Complex(redeclare SI.Resistance re, redeclare SI.Reactance im);
  operator record ComplexPower = Complex(redeclare SI.Power re, redeclare SI.ReactivePower im);
end Types;

connector Pin
  Types.ComplexVoltage V "Line-to neutral voltage";
  flow Types.ComplexCurrent I "Line current";
end Pin;
```

Code Samples from the Library – Equation-Based

```
package Types
  operator record ComplexVoltage = Complex(redeclare SI.Voltage re, redeclare SI.Voltage im);
  operator record ComplexCurrent = Complex(redeclare SI.Current re, redeclare SI.Current im);
  operator record Admittance = Complex(redeclare SI.Conductance re, redeclare SI.Susceptance im);
  operator record Impedance = Complex(redeclare SI.Resistance re, redeclare SI.Reactance im);
  operator record ComplexPower = Complex(redeclare SI.Power re, redeclare SI.ReactivePower im);
end Types;

connector Pin
  Types.ComplexVoltage V "Line-to neutral voltage";
  flow Types.ComplexCurrent I "Line current";
end Pin;

model TransmissionLineAdm
  extends BaseClasses.TwoPort;
  parameter SI.Resistance R "Line resistance ";
  parameter SI.Reactance X "Line reactance";
  parameter SI.Susceptance B = 0 "Shunt susceptance";
  final parameter Types.Admittance Yl = Complex(1,0)/Complex(R,X) "Line admittance";
  final parameter Types.Admittance Ys = Types.Admittance(0, B/2) "Shunt admittance";

  Real LineBreakerClosed = 1 "1=closed, 0=open";

  Types.Admittance Yl_act "Actual line admittance, including breaker";
  Types.Admittance Ys_act "Actual shunt admittance, including breaker";
  Types.ComplexVoltage Vl "Voltage across the line";
  Types.ComplexCurrent Il, Isa, Isb;
  SI.Angle Vl_arg = CM.arg(Vl) if auxCalcModArg "Vz angle relative to ref. rotating at omega_ref";
  SI.Voltage Vl_mod = CM.'abs'(Vl) if auxCalcModArg "modulus of Vz";
equation
  Yl_act = Yl * Complex(LineBreakerClosed, 0);
  Ys_act = Ys * Complex(LineBreakerClosed, 0);
  Ia = Il + Isa;
  Il + Ib = Isb;
  Isa = Ys * Va;
  Isb = Ys * Vb;
  Il = Yl * Vl;
  Va = Vl + Vb;
end TransmissionLineAdm;
```

Code Samples from the Library – Block-Diagram Based



Improving The Compilation & Simulation Performance

- Using a sparse linear solver and avoiding tearing on the large algebraic system of equations is essential
  - UMFPACK tried out first
  - KLU implemented recently, turns out to be much faster (2X or more) on this kind of problems

Improving The Compilation & Simulation Performance

- Using a sparse linear solver and avoiding tearing on the large algebraic system of equations is essential
  - UMFPACK tried out first
  - KLU implemented recently, turns out to be much faster (2X or more) on this kind of problems
- Many stack overflow problems, solved by using tail recursion
- Some functions in the back-end scaled very badly with size, the worst ones have been fixed (more on this later)
- The two largest examples require 64-bit OMC because of memory requirements, all examples run under Linux
- The preOpt and postOpt settings need to be carefully selected to avoid excessive code generation times

Improving The Compilation & Simulation Performance

- Using a sparse linear solver and avoiding tearing on the large algebraic system of equations is essential
  - UMFPACK tried out first
  - KLU implemented recently, turns out to be much faster (2X or more) on this kind of problems
- Many stack overflow problems, solved by using tail recursion
- Some functions in the back-end scaled very badly with size, the worst ones have been fixed (more on this later)
- The two largest examples require 64-bit OMC because of memory requirements, all examples run under Linux
- The preOpt and postOpt settings need to be carefully selected to avoid excessive code generation times

- Very significant progress between Oct 2015 and Jan 2016, but still much remains to be done

Data of the Test Models

	# of Nodes of the power network	Eqs/ Vars	State vars	Linear systems	Linear system density [%]	Assignments
Rete C	751	60135	615	1x12246 + 42x2	1x <0,01 + 42x 100	14227
Rete E	1815	156729	1894	1x36400 + 6x2	1x <0,01 + 6x 100	35536
Rete D	8376	470000	16219	1x90208	1x <0,01	160408

Data of the Test Models

	Objects	Nodes	Connec tions	Loads	Genera tors	Transm lines	Transf ormers
Rete C	1661	751	423	255	73	359	551
Rete E	4178	1815	807	742	266	1338	1025
Rete D	12809	8376	2676	3383	2317	1944	2489

Results of the Feasibility Study

Tests run under Linux on a Intel Xeon E5-2670 @ 2.6GHz with 160 GB RAM

Simulation: 20 s @ 20 ms time step, Runge-Kutta 4th order, KLU linear solver

	Objects	*Nodes*	*Eqs/ Vars*	**Front-end [s]**	**Back-end [s]**	**Sim Code [s]**	**Templates [s]**	**C compile [s]**	**Total compile [min]**	**Sim time [s]**
Rete C	*1661*	*751*	*60135*	57	47	26	15	15	**2,7**	**22,7**
Rete E	*4178*	*1815*	*156729*	185	161	104	33	41,9	**8,8**	**74,6**
Rete D	*12809*	*8376*	*470000*	1099	1852	2060	182	186	**89**	**203,7**
Scaling w.r.t. Rete C										
Rete E	*2,52*	*2,42*	*2,61*	3,23	3,43	3,98	2,22	2,86	**3,28**	**3,28**
Rete D	*7,71*	*11,15*	*7,82*	19,14	39,4	78,8	12,2	12,7	**33,6**	**8,96**

Lessons learned

- The front-end processing time scales as O(N). The new one migh be faster.
- If arrays of objects were not unrolled by the front-end, performance might benefit a lot – currently no difference between using arrays or not.
- Connections will always be declared individually (no repetitive structure)

Lessons learned

- The front-end processing time scales as O(N). The new one migh be faster.
- If arrays of objects were not unrolled by the front-end, performance might benefit a lot – currently no difference between using arrays or not.
- Connections will always be declared individually (no repetitive structure)
- Many functions in the Back-End and SimCode phase scale as $O(N^3)$
    - Expected performance is usually O(N) or $O(N^2)$
    - From a certain size up they become the bottleneck
    - Naive implementations in terms of efficiency;
inefficiency only shows up for large N
    - Some of these functions have already been fixed, others need to do so

Lessons learned

- The front-end processing time scales as O(N). The new one migh be faster.
- If arrays of objects were not unrolled by the front-end, performance might benefit a lot – currently no difference between using arrays or not.
- Connections will always be declared individually (no repetitive structure)
- Many functions in the Back-End and SimCode phase scale as $O(N^3)$
  - Expected performance is usually O(N) or $O(N^2)$
  - From a certain size up they become the bottleneck
  - Naive implementations in terms of efficiency;
    inefficiency only shows up for large N
  - Some of these functions have already been fixed, others need to do so
- Some unnecessary (or utterly useless) optimizations can be skipped
  - detectJacobianSparsePattern: explicit algorithms don't need this
  - disableLinearTearing: mandatory for the largest algebraic loop
  - indexReductionMethod=uode: we know a priori the system is index-1

Lessons learned

- The front-end processing time scales as O(N). The new one migh be faster.
- If arrays of objects were not unrolled by the front-end, performance might benefit a lot – currently no difference between using arrays or not.
- Connections will always be declared individually (no repetitive structure)
- Many functions in the Back-End and SimCode phase scale as $O(N^3)$
  - Expected performance is usually O(N) or $O(N^2)$
  - From a certain size up they become the bottleneck
  - Naive implementations in terms of efficiency;
    inefficiency only shows up for large N
  - Some of these functions have already been fixed, others need to do so
- Some unnecessary (or utterly useless) optimizations can be skipped
  - detectJacobianSparsePattern: explicit algorithms don't need this
  - disableLinearTearing: mandatory for the largest algebraic loop
  - indexReductionMethod=uode: we know a priori the system is index-1

↓

Premature optimization is the root of all evil,
but it is now time to do something about it!

Lessons learned - II

- All stages of code generation are very expensive in terms of RAM usage:
  - about 100 kBytes (!) of memory are allocated for each scalar equation
  - The test case requires to instantiate only a handful of different classes
  - There is arguably *a lot* of uselessly repeated work in the process
- The generated C-code and simulation executables are unnecessary large
  - The size of the executable for the largest test case (ReteD) is 435 MB!
- The simulation executable allocates a lot of RAM to store the results
  - A test case generating 4.8 GB of result data allocates over 40 GB during the simulation!

Lessons learned - II

- All stages of code generation are very expensive in terms of RAM usage:
  - about 100 kBytes (!) of memory are allocated for each scalar equation
  - The test case requires to instantiate only a handful of different classes
  - There is arguably *a lot* of uselessly repeated work in the process
- The generated C-code and simulation executables are unnecessary large
  - The size of the executable for the largest test case (ReteD) is 435 MB!
- The simulation executable allocates a lot of RAM to store the results
  - A test case generating 4.8 GB of result data allocates over 40 GB during the simulation!

Despite the inefficiencies in the code generation process
the simulation code already outperforms
legacy domain-specific tools!

Research Question:

Can OpenModelica handle regional / national / continental sized PG&T system models?

Research Question:

Can OpenModelica handle regional / national / continental sized PG&T system models?

Yes, definitely!

(better with some more work...)

Future Developments and Optimizations

- Test native sparse DAE solvers (e.g., IDA/Kinsol)
- Investigate optimizations for DAE solvers (w/o causalization)
- Investigate efficient event handling for these large systems

Future Developments and Optimizations

- Test native sparse DAE solvers (e.g., IDA/Kinsol)
- Investigate optimizations for DAE solvers (w/o causalization)
- Investigate efficient event handling for these large systems
- Avoid loop unrolling in the front-end when declaring arrays of objects of the same type
- Apply basic simplifications (e.g. removing alias equations) to classes, before instantiating them N times and doing the same thing N times
- Avoid generating N instances of the same C-code for each object, call the same function with different parameters instead
  - Much less code generation time *and* much less memory footprint

Future Developments and Optimizations

- Test native sparse DAE solvers (e.g., IDA/Kinsol)
- Investigate optimizations for DAE solvers (w/o causalization)
- Investigate efficient event handling for these large systems
- Avoid loop unrolling in the front-end when declaring arrays of objects of the same type
- Apply basic simplifications (e.g. removing alias equations) to classes, before instantiating them N times and doing the same thing N times
- Avoid generating N instances of the same C-code for each object, call the same function with different parameters instead
    - Much less code generation time *and* much less memory footprint
- Streamline all Back-End and SimCode functions so that they scale properly with the system size
- Link specialized solvers for these models into OMC to get even faster simulation performance (could be done as proprietary development)

Acknowledgement

Special thanks to:

Willi Braun
Adrian Pop
Vitalij Ruge
Martin Sjölund
Volker Waurich

for their help in getting these preliminary results out of OMC in the last three months!

Thank you for you kind attention!

(I hope I'll be able to take questions via Skype...)