Towards a Benchmark Suite for Modelica Compilers: Large Models

Jens Frenkel[+], Christian Schubert[+], Günter Kunze[+], Peter Fritzson[*], Martin Sjölund[*], Adrian Pop*
[+]Dresden University of Technology, Institute of Mobile Machinery and Processing Machines
[*]PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
{jens.frenkel, christian.schubert, guenter.kunze}@tu-dresden.de,
{peter.fritzson,martin.sjolund,adrian.pop}@liu.se

Abstract

The paper presents a contribution to a Modelica benchmark suite. Basic ideas for a tool independent benchmark suite based on Python scripting along with models for testing the performance of Modelica compilers regarding large systems of equation are given. The automation of running the benchmark suite is demonstrated followed by a selection of benchmark results to determine the current limits of Modelica tools and how they scale for an increasing number of equations.

Keywords: benchmark, performance comparison, code generation, compiler.

1 Introduction

Benchmarks are a well-known method to compare the capabilities of different software products. Based on the results users are able to choose the best software for their application. Several commercial and non-commercial Modelica compilers are available on the market, like SimulationX, OpenModelica, JModelica, MathModelica, and Dymola.

Due to the growing number of compilers, a tool independent and standardized test is needed from which the strengths of each compiler can be determined. Such a benchmark might also be used by compiler developers to test their compilers for compliance with the Modelica standard. Furthermore it can be used to identify ways of improving simulation performance. This paper tries to develop such a benchmark suite called ModeliMark.

A standard for benchmarking Modelica compilers should cover the following topics:

1. languages features
2. symbolic manipulation power
3. numeric solver robustness
4. compiling performance
5. simulation/target code performance

In the first part all *language features* of Modelica are tested showing the coverage of each compiler.

Symbolic manipulation power refers to testing which simplifications and manipulations are undertaken by the compiler in order to improve simulation speed.

In *Numeric solver robustness* difficult models are simulated comparing their results. Difficult models might feature high indices, inconsistent initial values or singularities which might require dynamic state selection for example [1].

Compiling and *simulation performance* tests a set of predefined models and measures their time for translation or simulation respectively.

A main concern of this paper is to investigate how current modelica compilers cope with large models, i.e. many equations.

The next chapter gives an overview of previous work on comparisons for Modelica compilers. It is followed by an overview on model design for benchmarking the scalability with respect to model size. Chapter four focuses on how the execution of such a benchmark could be automated using Python. A first glance at some benchmark results is given in the fifth chapter.

2 Previous Work

Every development team of a Modelica compiler already has a wide range of tests to ensure that the compiler is working correctly. Also the Modelica language specification and the Modelica Library include numerous examples which can be included in tests. Some of them can be used to test for *language features* whereas others could be used for performance measurements.

At the Modelica Conference in 2008 [8] a benchmark library focussing on *numerical robustness* was presented. The authors tried to compare their own Modelica compiler MOSILAB with commercial tools. Several models ranging from simple tests for language

features to demanding electrical circuits with discontinuities.

Further possible benchmark models emerged from the efforts of implementing parallel Modelica compilers; see [4] and [5].

2 Large models for Performance Benchmarks

The scope of this paper lies not in trying to establish a general Modelica benchmark but in providing a set of large benchmark models.

With increasing popularity of Modelica the demand for more detailed models increases as well. This leads to larger models with a very high number of equations and variables. However, large systems is a challenging task for Modelica compilers due to the symbolic approach to Modelica compilation.

The following models are supposed to evaluate the performance and boundaries of available Modelica compilers regarding large models. The benchmark comprises a set of synthetic models testing different aspects.

The first model is called *flat model*, containing many variables and equations which are tree structured.

The *hierarchical model* yields similar complexity by recursive use of small submodels.

A further set of models is designed to test the symbolical effort to extract systems of equations. It consists of models with:

1. a large number of alias variables, for example "a=b" or "a=-b", and only a few other equations
2. a *linear system* of equations
3. a *nonlinear system* of equations
4. a linear system of equations with time discrete and continues variables (*mixed linear system*)
5. a nonlinear system of equations with time discrete and continues variables (*mixed nonlinear system*)

2.1 Flat Model

The flat model consists of *n* variables and *n* equations and has the following form.

```
model flatclass_n
  input Real inp;
  Real v_1;
  Real v_2;
  Real v_3;
  ...
  Real v_n;
equation
  v_1 = 1 + v_2;
  v_2 = 2 + v_3;
...…
  v_(n-1) = (n-1) + v_n;
  der(v_n) = v_1 + inp;
end flatclass_n;
```

The same model may be expressed using a for-loop.

```
model flatClass_N
  constant Integer N=100;
  input Real inp;
  Real v[N];
equation
  for i in 1:N-1 loop
    v[i] = i + v[i+1];
  end for;
  der(v[N]) = v[1] + inp;
end flatClass_N;
```

While both models give the same result their syntax is different leading to different workloads in the compiler. For this comparison only the first model has been considered.

Note that a Modelica compiler which is able to work with for-loops directly instead of expanding them may achieve significantly better results using the second formulation.

2.2 Hierarchical Models

The following hierarchical model features a mechanical system consisting of a long series of masses interconnected by springs. The base class uses the Modelica.Mechanics.Translational library and is shown in Figure 1:

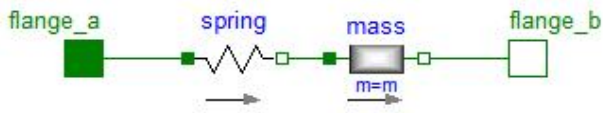

Figure 1. SpringMass Model 1**.**

In the next level two of these submodels are combined as shown in Figure 2. This step can be repeated for each individual level.

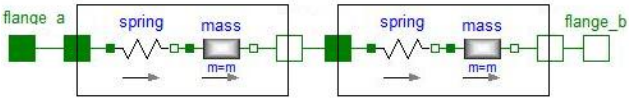

Figure 2. SpringMass Model 2**.**

The topmost model connects a submodel to a fixed flange. Therefore the number of equations increases with the level as given in Table 1.

Due to the structural information translation may be faster than the flat model with equal number of equations.

Level	Equations
1	21
2	42
3	84
4	168

Table 1: Number of equations for the SpringMass models.

2.3 Alias Model

Connect equations in Modelica models often lead to equations like “a=b” or “a=-b”. Such equations are considered to be alias equations since they merely introduce new names for known variables. Hence, additional alias equations should only lead to minimal overhead.

In order to test this behaviour the model `FlatAliasClass` has been designed. It is very similar to the `FlatModel` only with additional alias equations. The number *m* of additional alias equations can be altered to see their influence.

```
model Flataliasclass_n
  input Real inp;
  Real v_1;
  Real v_2;
  ...
  Real v_n;

  Real va_1;
  Real va_2;
  ...
  Real va_m;
equation
  va_1 = v_1;
  va_1 = 1 + v_2;
  va_2 = v_2;
  va_2 = 2 + v_3;
..…
  va_(n-1) = v_(n-1);
  va_(n-1) = (n-1) + v_n;
  der(v_n) = v_1 + inp;
end Flataliasclass_n;
```

In addition another model called `AliasClass_N` has been implemented in which alias relations emerge only if previous found aliases are replaced. In fact, if all alias relations are found only the first equation remains.

```
model AliasClass_N
  input Real inp;
  constant Integer N=4;
  Real a[2*N+1];
equation
  der(a[1]) = inp;
  a[2] = -a[1];
  a[3] = 2*a[2]+a[1];
  for i in 4:2:2*N loop
    a[i] = a[i-3] + a[i-2] – a[i-1];
    a[i+1] = i*a[i]+(i-1)*a[i-1];
  end for;
end AliasClass_N;
```

2.4 Model with linear or nonlinear Systems of Equations

Dealing with linear or nonlinear systems of equations is a basic requirement of every Modelica compiler. The main concern of the next four models is to answer the question up to which size an equation system can be handled by a compiler and what effort it takes.

2.4.1 Linear Model

First, there is the `Linersysclass_n` which possess a strong connected linear system of *n* equations which has the unique solution:

v = 1+inp/(n-2)*[1,1,…,1] for *n* > *2*.

```
model Linearsysclass_n
  input Real inp;
  Real v_0;
  Real v_1;
  Real v_2;
  ...
  Real v_n;
equation
  - v_0 + v_1 + v_2 … + v_n = n-2 + inp;
  + v_0 - v_1 + v_2 … + v_n = n-2 + inp;
  + v_0 + v_1 - v_2 … + v_n = n-2 + inp;
  …
  + v_0 + v_1 + v_2 … - v_n = n-2 + inp;
end Linearsysclass_n;
```

2.4.2 Mixed Linear Model

Modelica models often include if-equations which lead to time discrete components which themselves may be part of an equation system. Such systems of equations have to be solved using iterative methods which are tested by the following model. `Mixedlinersysclass` leads to a strongly connected linear system including if-equations.

```
model Mixedlinearsysclass_n
  input Real inp;
  Real v_0;
  Boolean b_0;
  Real v_1;
  Boolean b_1;
  Real v_2;
  Boolean b_2;
  ...
  Real v_n;
  Boolean b_n;
```

```
 equation
   b_0 = v_0 > 0;
   (if b_0 then -v_0 else -2*v_0) + v_1 +
 v_2 … + v_n  = n-2+inp;

   b_1 = v_1 > 0;
   + v_0 + (if b_1 then -v_1 else -2*v_1) +
 v_2 … + v_n  = n-2+inp;

   b_2 = v_2 > 0;
   + v_0 + v_1 +(if b_2 then -v_2 else -
 2*v_2) … + v_n  = n-2+inp;
   …

   b_n = v_n > 0;
   + v_0 + v_1 + v_2 … +(if b_n then -v_n
 else -2*v_n)  = n-2+inp;
end Mixedlinearsysclass_n;
```

2.4.3 Nonlinear Model

Similar to the linear case, there are models which test the Modelica compiler's ability to solve nonlinear equations by a slight alteration of the aforementioned models.

```
 model Nnlinearsysclass_n
   input Real inp;
   Real v_0;
   Real v_1;
   Real v_2;
   ...
   Real v_n;
 equation
   - sin(v_0) + v_1 + v_2 … + v_n=n-2+inp;
   + v_0 - sin(v_1) + v_2 … + v_n=n-2+inp;
   + v_0 + v_1 - sin(v_2) … + v_n=n-2+inp;
   …
   + v_0 + v_1 + v_2 … - sin(v_n)=n-2+inp;
end Nonlinearsysclass_n;

model Mixednonlinearsysclass_n
   input Real inp;
   Real v_0;
   Boolean b_0;
   Real v_1;
   Boolean b_1;
   Real v_2;
   Boolean b_2;
   ...
   Real v_n;
   Boolean b_n;
 equation
   b_0 = v_0 > 0;
   (if b_0 then sin(v_0) else cos(v_0)) +
 v_1 + v_2 … + v_n = n-2+inp;
   b_1 = v_1 > 0;
   + v_0 + (if b_1 then sin(v_1) else
 cos(v_1)) + v_2 … + v_n = n-2+inp;
   b_2 = v_2 > 0;
   + v_0 + v_1 +(if b_2 then sin(v_2) else
 cos(v_2)) … + v_n = n-2+inp;
   …
   b_n = v_n > 0;
   + v_0 + v_1 + v_2 … +(if b_n then
 sin(v_n) else cos(v_n)) = n-2+inp;
end Mixednonlinearsysclass_n;
```

3 Automating the Benchmark Suite

The main concern of this paper was to get an answer on how current Modelica compilers cope with large models, i.e. many equations. To get this answer a lot of models with an increasing number of equations had to be translated and simulated. Hence, the generation of the models as well as the control of the Modelica compilers should be fully automated.

The programming language Python proves to be well suited as it allows importing C-Code, starting external processes or even accessing COM-Components under Microsoft Windows. In addition Python is an object oriented script language which is easy to read and for which comprehensive libraries are available.

The first part of the solution is a model generator as shown in Figure 3. It chooses appropriate models, values for n (number of equations) and writes Modelica code which shall then be tested. Each test model is stored in a separate Python class which returns Modelica code for a given n.

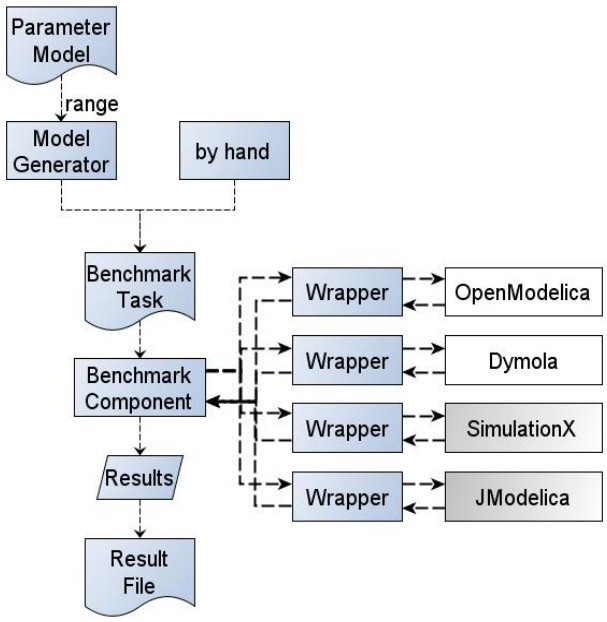

Figure 3. Benchmark Framework**.**

These models are handed over to the Benchmark Component. It controls the benchmark process and communicates via a wrapper with the corresponding Modelica compiler. Such a wrapper has to be created for each individual compiler. So far wrappers for OpenModelica, JModelica and Dymola have been implemented.

Based on the usual Modelica translation process, which is divided into flattening and symbolic manipulation, the wrappers expose three functions:

- flatten
- translate

- simulate

Flatten instructs the compiler to parse the Modelica code and return the flat model. Translate first flattens the model and turns it into a state which can be simulated (executable for example). Simulate flattens, translates and simulates the model using the standard solver and a predefined output interval and stop time.

Since not every Modelica compiler provides functions to measure the execution time of the flattening, translation and simulation process the functionalities from the Python library *time* is used. All results are written to a text file. The source code as well as the models is freely available at http://code.google.com/p/modelimark, and linked from www.openmodelica.org.

4 Benchmark Results

The following benchmarks were accomplished using a Windows 7, 64 Bit System with Intel Core i7 860, 2.80 GHz and 4.0 GB RAM.

4.1 Modelica Compilers

For this benchmark three different Modelica compilers were used:

- OpenModelica compiler Revision 7745 from 21/01/2011
- JModelica 1.4
- Dymola 7.4

4.2 Flat Model

As can be seen in Figure 4 Dymola needs the least time for translation followed by OpenModelica and JModelica.

However time increases roughly with the third power of n which makes Modelica uneconomical for very large models. It was found that the upper limit for the number of equations is not defined by time but by the compiler itself.

While JModelica and OpenModelica failed at around 2000 and 60 000 equations respectively, Dymola managed to translate a model with 160 000 equations but Visual Studio 2008 failed to compile the executable.

4.3 Hierarchical Models

Figure 5 shows the results for the hierarchical Model in comparison to the flat one. It can be seen that JModelica and OpenModelica do not benefit. In Dymola however, the time now only increases with the second power of n. It is assumed that the internal look up process in Dymola exploits the model structure. Nevertheless, twice the equations still leads to a fourfold time for translation.

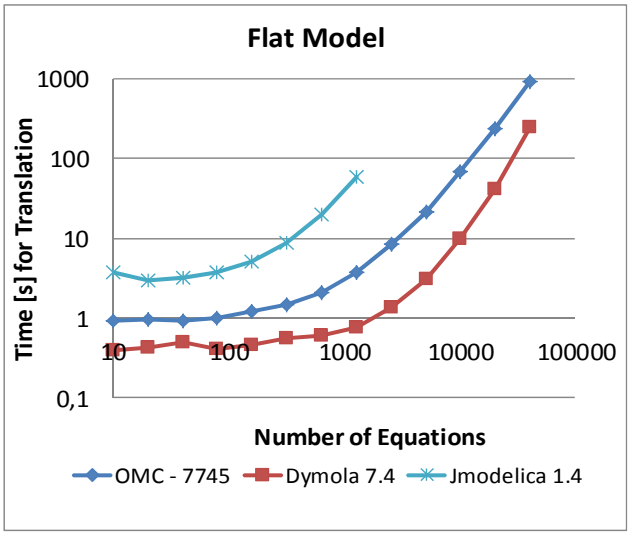

Figure 4: Benchmark Results Flat Model

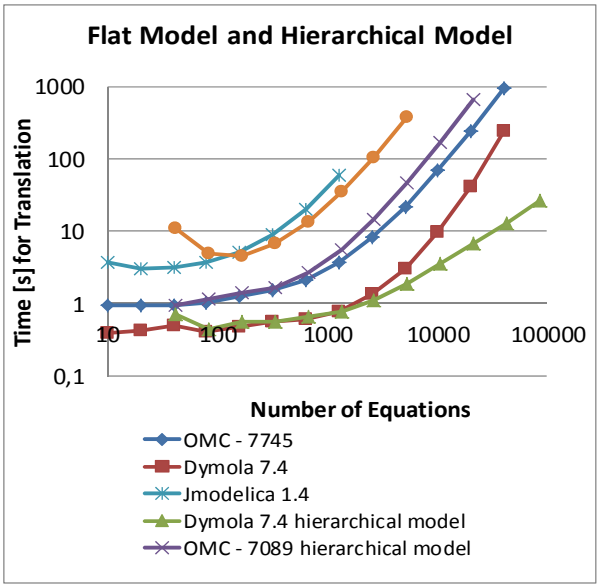

Figure 5: Flat Model and hierarchical Model

4.4 Alias Model

Each graph in the Figures 6, 7, 8 show for each compiler how the time for translation changes with increasing percentage of alias equations for a given number of equations.

In the case of Dymola the influence of alias equations is similar to normal equations.

In OpenModelica and JModelica alias equations are treated more efficiently since their influence is almost linear and independent of n.

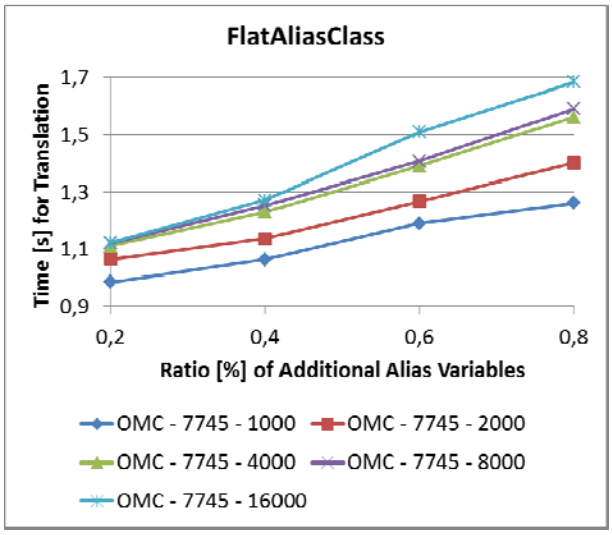

Figure 6: FlatAlilasClass - OMC 7745

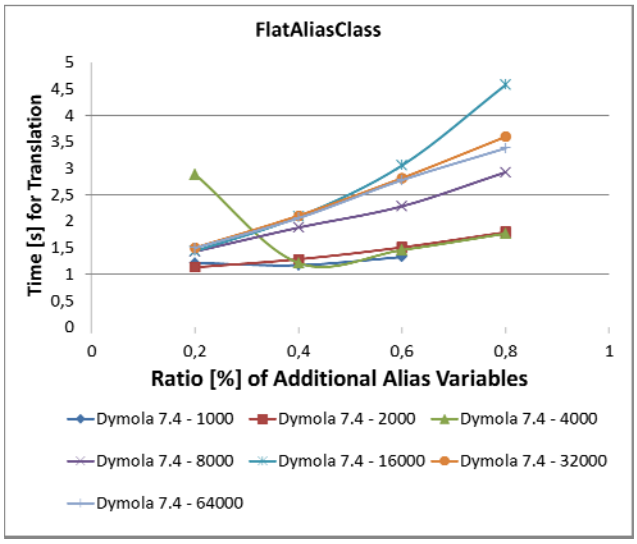

Figure 7: FlatAliasClass Dymola 7.4

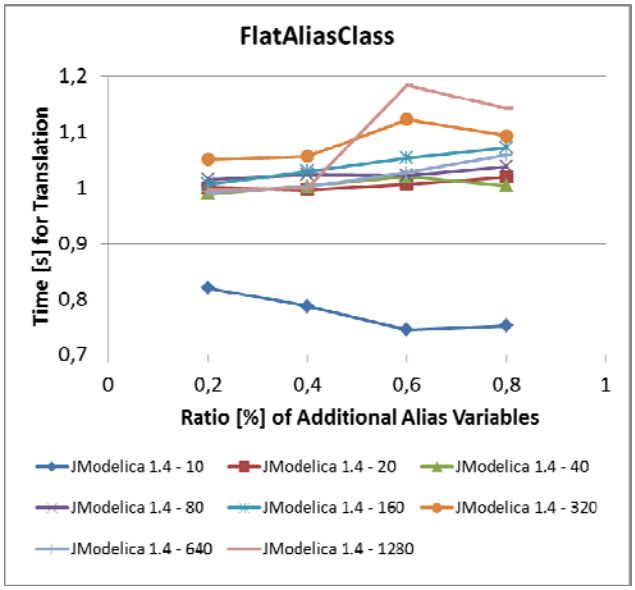

Figure 8: FlatAliasClass JModelica 1.4

For the model `AliasClass` where a recursive substitution is needed, to find all alias relations Dymola seems to be more efficient (Figure 9). Further investigations have shown that the Dymola compiler replaces only the first 11 alias variables. All the other alias variables are not detected and calculated for each simulation step.

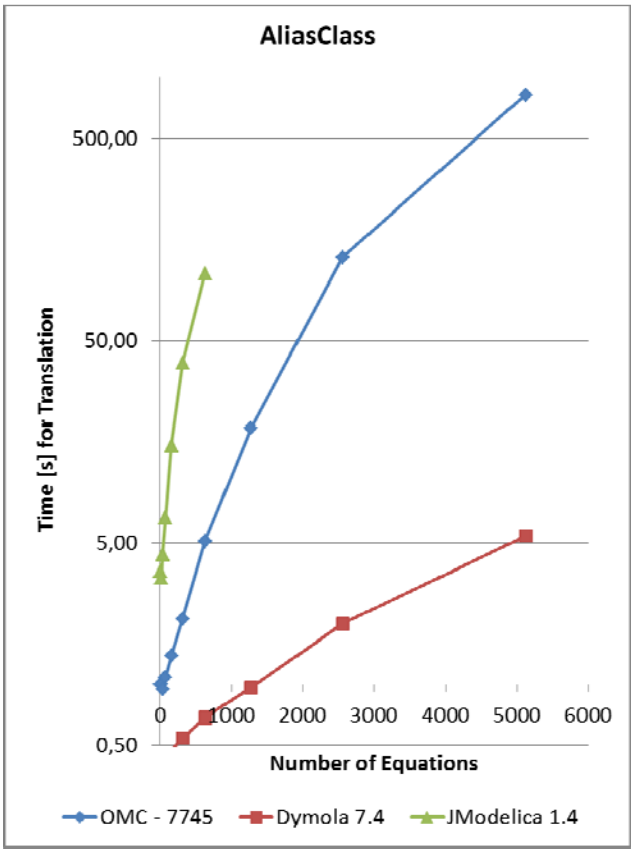

Figure 9: AliasClass Model

4.5 Linear or Nonlinear Systems of Equations

Looking at Figure 10, 11, 12 it can be seen that the results for the different types of equation systems hardly differ. Again, the implementation in Dymola seems to be more efficient compared to JModelica and OpenModelica. Note, that the maximum number of unknowns which could be solved for in Dymola was 320 compared to a 160 in OpenModelica and 80 in JModelica.

5 Conclusions

This paper tried to encourage the development of a standard benchmark suite for Modelica compilers. It would give compiler developers insights to find possibilities for improvements and give users the chance to compare different compilers.

The scope of this paper was limited to the behavior of current Modelica compilers regarding large models. It could be found that Dymola was generally faster than OpenModelica and JModelica. However, even Dymola does not seem suitable for very large models as it cannot cope with models that have more than 160000 equations.

Furthermore it was found that, depending on the model, the time needed for translation grows with second or third power of the number of equations.

In order to continue establishing Modelica as the major simulation language better ways of dealing with large models have to be found. Some first promising ideas are given in [6] and [7].

All the models as well as the Python code are freely available at http://code.google.com/p/modelimark/, and linked from www.openmodelica.org.

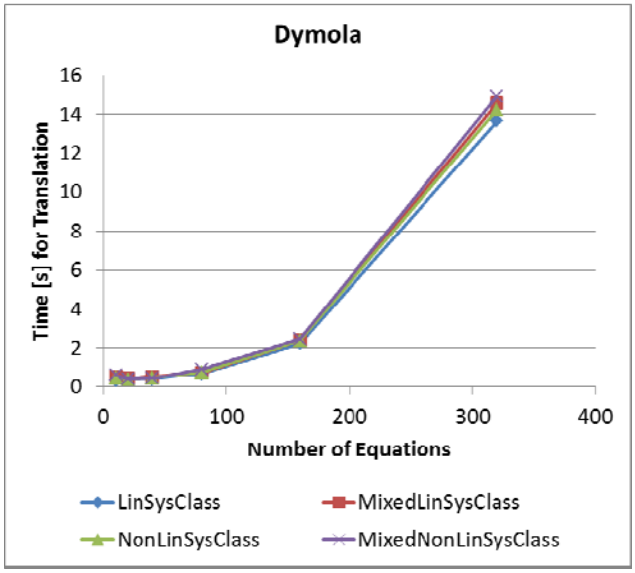

Figure 10: Linear and Nonlinear Systems of Equations Dymola 7.4

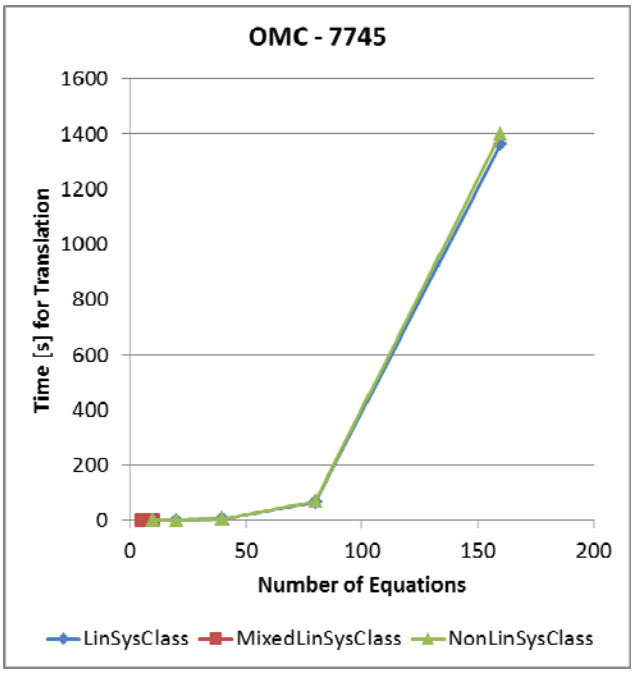

Figure 11: Linear and Nonlinear Systems of Equations OMC – 7745

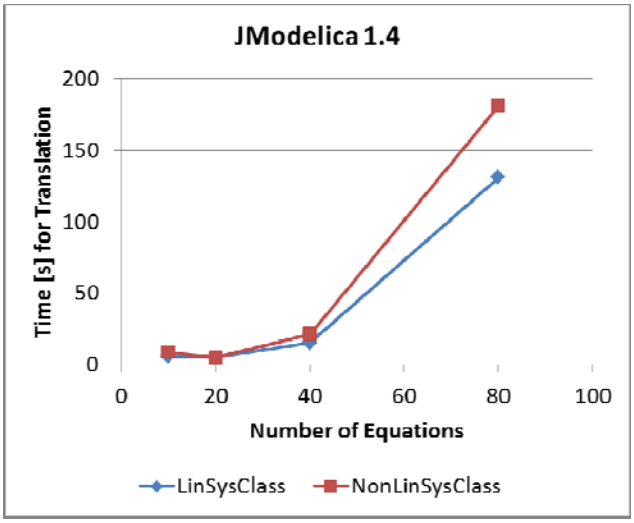

Figure 12: Linear and Nonlinear Systems of Equations JModelica 1.4

References

[1] S. Mattsson, G. Söderlind: Index reduction in differential-algebraic equations using dummy derivatives. SIAM J. Sci. Statist. Comput., 14:677– 692, 1993.

[2] Peter Fritzson: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, Page 57ff, Wiley IEEE Press, 2004.

[3] O. Enge-Rosenblatt, C. Clauß, P. Schwarz, F. Breitenecker, C. Nytsch-Geusen: Comparison of Different Modelica-Based Simulators Using Benchmark Tasks, in Procedings of Modelica Conference 2008

[4] M. Maggio, K. Stavåker, F. Donida, F. Casella, P.Fritzson: Parallel Simulation of Equation-based Object-Oriented Models with Quantized State Systems on a GPU, in Proceedings of the 7th Modelica Conference 2009

[5] H. Lundvall, K. Stavåker, P. Fritzson, C. Kessler: Automatic parallelization of simulation code for equation-based models with software pipelining and measurements on three platforms. in ACM SIGARCH Computer Architecture News, Vol. 36, No. 5, December 2008.

[6] D Zimmer: Module-Preserving Compila-tion of Modelica Models,Proceedings 7th Modelica Conference, Como, Italy, Sep. 20-22, 2009.

[7] C. Höger, F. Lorenzen, P. Pepper: Notes on the Separate Compilation of Modelica, The 3rd International Work-shop on Equation-Based Object-Oriented Modeling Languages and Tools, Oslo, Norway, October 3, 2010

Appendix

a. FlatModel

n	OMC - 7745		Dymola 7.4		Jmodelica 1.4	
	T	S	T	S	T	S
10	0,92	0,11	0,39	0,11	3,73	0,21
20	0,94	0,07	0,42	0,09	3,00	0,15
40	0,91	0,09	0,50	0,03	3,19	0,24
80	1,00	0,13	0,41	0,19	3,73	0,25
160	1,21	0,21	0,46	0,14	5,04	0,50
320	1,49	0,35	0,55	0,33	8,73	1,45
640	2,11	0,68	0,59	0,55	19,76	5,66
1280	3,73	1,27	0,75	1,11	59,57	22,30
2560	8,37	2,61	1,34	2,00		
5120	21,75	5,01	3,05	4,06		
10240	69,13	10,01	9,83	7,66		
20480	239,78	21,47	41,18	16,25		
40960	932,63	42,87	245,29	35,01		

b. Hierarchical Models

n	OMC – 7745		Dymola 7.4		JModelica 1.4	
	T	S	T	S	T	S
42	0,94	0,21	0,70	0,27	10,72	1,57
84	1,14	0,13	0,44	0,07	4,86	0,15
168	1,39	0,3	0,55	0,00	4,55	0,23
336	1,61	0,29	0,56	0,11	6,78	0,41
672	2,63	0,57	0,66	0,13	13,52	1,13
1344	5,58	1,07	0,76	0,35	34,38	3,59
2688	14,67	2,14	1,10	0,36	103,74	13,30
5376	46,60	4,47	1,86	1,70	377,37	50,66
10752	165,47	10,26	3,58	0,94		
21504	666,54	26,01	6,69	7,21		
43008			13,03	7,29		
86016			26,37	46,29		

c. FlatAliasClass

n	alias	OMC - 7745		Dymola 7.4	
		T	S	T	S
1000	0	3,27	1,57	0,59	0,25
1200	200	3,22	1,83	0,72	0,82
1400	400	3,47	2,01	0,69	1,06
1600	600	3,89	2,15	0,79	1,13
1800	800	4,13	1,91	0,79	1,27
2000	0	6,52	2,24	1,03	1,65
2400	400	6,94	2,68	1,16	1,77
2800	800	7,41	2,93	1,32	2,06
3200	1200	8,26	3,67	1,55	2,30
3600	1600	9,14	3,92	1,85	2,41
4000	0	15,80	4,13	2,27	3,10
4800	800	17,56	5,02	2,89	3,34
5600	1600	19,46	5,90	3,50	3,92

6400	2400	21,99	6,61	4,20	4,52
7200	3200	24,69	7,53	5,10	5,19
8000	0	47,53	8,19	6,48	5,97
9600	1600	53,25	9,96	9,25	6,91
11200	3200	59,60	11,45	12,18	7,71
12800	4800	66,90	12,48	14,79	9,23
14400	6400	75,67	14,36	18,97	9,43
16000	0	155,14	16,84	24,88	11,30
19300	3200	174,05	20,12	35,72	14,21
22400	6400	197,41	23,53	51,89	15,18
25600	9600	234,25	26,64	76,05	16,55
28800	12800	261,50	29,90	113,93	9,23
32000	0			114,03	23,75
38400	6400			216,26	28,89
44800	12800			303,47	35,67
51200	19200			405,83	32,61
57600	25600			518,32	32,39
64000	0			669,37	39,99
76800	12800			1001,91	107,16
89600	25600			1379,49	160,05
102400	38400			1860,19	44,43
115200	51200			2262,47	33,65

n	alias	JModelica 1.4	
		T	S
10	0	3,64	0,18
12	2	2,99	0,21
14	4	2,87	0,14
16	6	2,72	0,12
18	8	2,74	0,14
20	0	2,85	0,14
24	4	2,85	0,15
28	8	2,84	0,13
32	12	2,86	0,15
36	16	2,90	0,15
40	0	3,04	0,15
48	8	3,00	0,18
56	16	3,04	0,16
64	24	3,10	0,18
72	32	3,05	0,18
80	0	3,50	0,25
96	16	3,55	0,26
112	32	3,58	0,23
128	48	3,57	0,26
144	64	3,63	0,23
160	0	4,87	0,48
192	32	4,91	0,47
224	64	5,02	0,49
256	96	5,13	0,47
288	128	5,22	0,51
320	0	8,29	1,38
384	64	8,70	1,38
448	128	8,75	1,40
512	192	9,30	1,43
576	256	9,06	1,42
640	0	19,37	5,37
768	128	19,22	5,27
896	256	19,40	5,21
1024	384	19,89	5,28
1152	512	20,50	5,31
1280	0	60,08	21,35
1536	256	59,92	21,11
1792	512	60,04	21,20
2048	768	71,17	21,93
2304	1024	68,60	22,68

d. AliasClass

n	OMC - 7745		Dymola 7.4		JModelica 1.4	
	T	S	T	S	T	S
10	0,99	0,08	0,04	0,10	3,63	0,29
20	1,00	0,10	0,41	0,15	3,36	0,18
40	0,94	0,14	0,45	0,11	4,39	0,32
80	1,09	0,21	0,44	0,19	6,73	0,61
160	1,39	0,49	0,48	0,31	15,26	2,00
320	2,11	0,66	0,54	0,55	38,80	6,06
640	5,10	1,29	0,69	0,99	107,41	25,03
1280	18,53	2,59	0,96	1,95		
2560	128,74	5,06	2,00	3,71		
5120	820,39	10,45	5,42	7,41		

e. LinSysClass

n	OMC - 7745		Dymola 7.4		JModelica 1.4	
	T	S	T	S	T	S
10	1,05	0,15	0,34	0,08	5,64	0,18
20	1,45	0,20	0,39	0,06	4,70	0,16
40	5,33	0,41	0,42	0,09	15,31	0,25
80	66,56	2,06	0,68	0,09	131,26	0,49
160	1363,40	10,92	2,24	0,12		
320			13,64	0,11		

f. MixedLinSysClass

n	OMC - 7745		Dymola 7.4	
	T	S	T	S
5	1,05	0,06	0,36	0,02
10	1,22	0,08	0,54	0,02
20			0,44	0,06
40			0,54	0,08
80			0,73	0,18
160			2,39	0,26
320			14,55	0,34

g. NonLinSysClass

n	OMC - 7745		Dymola 7.4		JModelica 1.4	
	T	S	T	S	T	S
10	1,03	0,11	0,45	0,02	8,76	1,24
20	1,43	0,25	0,38	0,06	4,70	0,14
40	5,37	1,02	0,47	0,09	21,50	0,67
80	69,56	6,38	0,76	0,11	181,44	0,42
160	1402,46	45,11	2,36	0,24		
320			14,24	0,38		

h. MixedNonLinSysClass

n	Dymola 7.4	
	T	S
10	0,60	0,23
20	0,41	0,11
40	0,45	0,09
80	0,90	0,13
160	2,44	0,30
320	14,93	0,61