Model order reduction of large scale ODE systems : MOR for ANSYS versus ROM workbench

Document status and date:
Published: 01/01/2006

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computer Science

CASA-Report 06-38
November 2006

Model order reduction of large scale ODE systems:
MOR for ANSYS versus ROM workbench

by

A.J. Vollebregt, T. Bechtold, A. Verhoeven, E.J.W. ter Maten

Model Order Reduction of Large Scale ODE Systems:
MOR for ANSYS versus ROM Workbench

A.J. Vollebregt[1], T. Bechtold[2], A. Verhoeven[3], E.J.W. ter Maten[2,3]

[1] Bergische Universität Wuppertal
[2] Philips Semiconductors - NXP, Eindhoven
[3] Technical University of Eindhoven

Summary. In this paper we compare the numerical results obtained by different model order reduction software tools, in order to test their scalability for relevant problems of the microelectronic-industry. MOR for ANSYS [2] is implemented in C++ and ROM Workbench [3] is a MATLAB code. The chosen benchmarks are large scale linear ODE systems, which arise from the finite element discretisation of electro-thermal MEMS models.

1 Introduction

Decreasing size of silicon chips and their increasing integration density require permanently new and more powerful simulation tools and strategies in microelectronics and microsysytem technology. Model order reduction (MOR) approaches [1] are successfully used to considerably reduce both the computational time and the resources. Mathematical development of MOR is an active area of research, which is growing from the reduction of linear ordinary differential equation systems (ODEs) towards the reduction of parameterized and nonlinear differential-algebraic equations (DAEs) and partial differential-algebraic equations (PDAEs). The implementation aspects of model order reduction are advancing as well. Practical MOR has developed from academic prototyping environments to several strong tools that can be easily used as an extension of the commercial simulators like e. g. ANSYS [2].

In today's age of fast computers it is possible to use quick prototyping tools like MATLAB or Mathematica for convenient implementation and testing of new MOR methods. However, the run time for the usually large-scale industry relevant problems enforces the use of programming languages, like for example C++. Such implementations offer better performances, but also demand more time and programming skills from the developer. ROM Workbench [3] also provides a learning environment using MATLAB.

The goal of this paper is to numerically compare two MOR tools, which belong to the described streams: MOR for ANSYS (M4A) [2] and ROM Workbench (RW) [3]. The first was developed at the university of Freiburg, Germany, as an extension to the commercial finite element simulator ANSYS. However, it can be easily coupled to an arbitrary circuit simulator, provided the matrices of the linear dynamical system are exported in the Matrix Market [4] format. It implements block Arnoldi

algorithm from [5] and SOAR from [6]. RW is a MATLAB library of different MOR methods, which has been developed at the University Politehnica of Bucharest, Romania, within the European project CODESTAR. It implements a PRIMA version using block-Arnoldi based on [8]. Both tools are planned for use in the European project COMSON [9], which joins the efforts of the major European semiconductor companies and academic nodes to develop a demonstrator platform in a software code, that could fulfill the demands of the modern microelectronic industry. Such a comparison will give us a clear understanding up to which size and for what structure of the industrial problem the MATLAB code can be used and at which point one should switch to the compiled language implementation.

In section 2 we prove the equivalence of algorithms [5] and [8]. In section 3 we comment on the implementations within two codes and describe two electro-thermal MEMS (micro-electro-mechanical-systems) devices used as case studies for model order reduction. In section 4, the numerical results for order reduction with both tools are presented. Lastly, in section 5, we conclude the paper and describe some future directions.

2 Block Arnoldi Algorithms

In microsystem simulation, the spatial discretisation of computational domain often results in a linear multiple-input multiple-output ODE systems of the form

$$C \cdot \dot{\mathbf{x}} + G \cdot \mathbf{x} = B \cdot \mathbf{u}(t)$$
$$\mathbf{y} = L^T \cdot \mathbf{x} \, , \tag{1}$$

with initial condition $\mathbf{x}(0) = \mathbf{x}_0$. Here, t is the time variable, $\mathbf{x}(t) \in \mathbb{R}^n$ the state vector, $\mathbf{u}(t) \in \mathbb{R}^m$ the input excitation vector and $\mathbf{y}(t) \in \mathbb{R}^p$ the output measurement vector. G, $C \in \mathbb{R}^{n \times n}$ are linear (not depending on $\mathbf{x}$ and t) symmetric and sparse system matrices, $B \in \mathbb{R}^{n \times m}$ and $L \in \mathbb{R}^{n \times p}$ are (constant) input and output distribution arrays, respectively. n is the dimension of the system and m and p are the number of inputs and outputs.

Model order reduction is based on the projection of (1) onto some low-dimensional subspace. Most MOR methods generate two projection matrices $V, W \in \mathbb{R}^{n \times \nu}$, to construct a reduced system of the order ν as

$$C_r \cdot \dot{\mathbf{z}} + G_r \cdot \mathbf{z} = B_r \cdot \mathbf{u}(t)$$
$$\mathbf{y}_r = L_r^T \cdot \mathbf{z} \, , \tag{2}$$

with $C_r = V^T C W$, $G_r = V^T G W$, $B_r = V^T B$, and $L_r = W^T L$. The ultimate goal of MOR is to find matrices V and W in such a way that $\nu \ll n$, while minimizing the error between the full and the reduced system in either time domain $min\|y - y_r\|$ or Laplace domain. Furthermore, the stability and passivity of the original system should be preserved in 2.

The basic idea behind the Krylov-subspace based block-Arnoldi algorithm is to transfer (1) into the implicit (left-hand side) formulation

$$A\dot{\mathbf{x}} = \mathbf{x} + R\mathbf{u}$$
$$\mathbf{y} = L^T \mathbf{x} \, , \tag{3}$$

with $A = -G^{-1}C$, and $R = -G^{-1}B$, and to write down the transfer function of (3) in the frequency domain, using a Taylor series in s_0 as

$$H(s) = -L^T(I - sA)^{-1}R = \sum_{i=0}^{\infty} m_i(s - s_0)^i, \qquad (4)$$

where $m_i = -L^T(I - sA)^{-1}B$ is called the i-th moment around s_0. One aims to find a reduced system whose transfer function $H_r(s)$ will have the same moments as $H(s)$ up to a degree ν. However, due to numerical instabilities, the moments are not computed explicitly, but via the right-sided Krylov subspace $Kr(A, R, \rho) := span(R, AR, A^2R, \ldots, A^{\rho-1}R)$. Block Arnoldi algorithm generates a single orthonormal basis W for $Kr(A, R, \rho)$ and the system (3) is reduced by projection to

$$A_r \dot{\mathbf{z}} = \mathbf{z} + R_r \mathbf{u}$$
$$\mathbf{y}_r = L_r^T \mathbf{z} , \qquad (5)$$

with $A_r = W^T A W$, $R_r = W^T R$ and $L_r = W^T L$. The order of (5) is $\nu = \rho \cdot m$. The property of the Krylov subspace is such that the first ν moments of $H_r(s) = -L_r^T(I - sA_r)^{-1}R_r$ and of $H(s)$ are indeed identical.

As the reduced system (5) is not necessarily passive (this means that the system generates no energy, which property is important for applications in circuit simulation), two alternatives to "classical" block-Arnoldi have been suggested: PRIMA algorithm [8] and Freund's Arnoldi [5]. Both are described and compared below.

2.1 PRIMA

The PRIMA algorithm was designed in 1998 to guarantee the passivity of the reduced system. PRIMA [8] stands for Passive Reduced-order Interconnect Macromodeling Algorithm. Again, an orthonormal basis, X, is generated such that $span(X) = Kr(A, R, \rho)$, but X is used for an explicit projection of (1), such that $C_r = X^T C X$, $G_r = X^T G X$, $B_r = X^T B$ and $L_r = X^T L$. In [8] is proven that for this reduced system the passivity is preserved if C is positively semi-definite and that the first n moments of the transfer function of the original and the reduced system are matched. Introducing the notation $X_k = [x_{km+1}| \ldots |x_{(k+1)m}]$, an implementation of PRIMA can be found in Algorithm 1.

2.2 Freund's Arnoldi

Freund suggests in [5] that vectors which are almost linearly dependent with other vectors in the span of the orthonormal matrix should be eliminated. He calls this method of eliminating vectors *deflation*. His algorithm is vector based, although there is a block structure visible for multiple input multiple output systems. Instead of generating orthonormal blocks the algorithm generates candidate vectors, and each vector $\hat{\mathbf{v}}_k$ that satisfies

$$||\hat{\mathbf{v}}_k|| < DTOL, \qquad (6)$$

for some appropriate threshold $DTOL$, is removed. Therefore, the number of vectors per orthonormal block m is smaller then, or equal to, the number of vectors of block $m - 1$. If this deflation is omitted we get Algorithm 2.

Algorithm 1 Block-Arnoldi as in PRIMA [8]

1: $\hat{X}_0 = R$
2: **for** $j = 1, \ldots, m$ **do**
3:      $\mathbf{x}_j = \hat{\mathbf{x}}_j / \|\hat{\mathbf{x}}_j\|$
4:      **for** $i = j + 1, \ldots, m$ **do**
5:          $\hat{\mathbf{x}}_i = \hat{\mathbf{x}}_i - \mathbf{x}_j \mathbf{x}_j^T \hat{\mathbf{x}}_i$
6:      **end for**
7: **end for**
8: **for** $k = 1, 2, \ldots, \rho - 1$ **do**
9:      Determine $\hat{X}_k = A X_{k-1}$
10:     **for** $j = 1, \ldots, k$ **do**
11:         $\hat{X}_k = \hat{X}_k - X_{k-j} X_{k-j}^T \hat{X}_k$
12:     **end for**
13:     **for** $j = km + 1, \ldots, (k+1)m$ **do**
14:         $\mathbf{x}_j = \hat{\mathbf{x}}_j / \|\hat{\mathbf{x}}_j\|$
15:         **for** $i = j + 1, \ldots, (k+1)m$ **do**
16:             $\hat{\mathbf{x}}_i = \hat{\mathbf{x}}_i - \mathbf{x}_j \mathbf{x}_j^T \hat{\mathbf{x}}_i$
17:         **end for**
18:     **end for**
19: **end for**

Algorithm 2 Freund's Arnoldi ignoring deflation

1: **for** $i = 1, \ldots, m$ **do**
2:      $\hat{\mathbf{v}}_i = \mathbf{r}_i$, $\mathbf{r}_i$ being the i -th column of R
3: **end for**
4: **for** $k = 1, 2, \ldots, \nu$ **do**
5:      $\mathbf{v}_k = \hat{\mathbf{v}}_k / \|\hat{\mathbf{v}}_k\|$
6:      Determine $\hat{\mathbf{v}}_{k+m} = A\mathbf{v}_k$
7:      **for** $i = 1, \ldots, k$ **do**
8:          $\hat{\mathbf{v}}_{k+m} = \hat{\mathbf{v}}_{k+m} - \mathbf{v}_i \mathbf{v}_i^T \hat{\mathbf{v}}_{k+m}$
9:      **end for**
10:     **for** $i = k - 1, k - 2, \ldots, k - m + 2, k - m + 1$ **do**
11:         $\hat{\mathbf{v}}_{i+m} = \hat{\mathbf{v}}_{i+m} - \mathbf{v}_k \mathbf{v}_k^T \hat{\mathbf{v}}_{i+m}$
12:     **end for**
13: **end for**

2.3 Comparison between both Algorithms

We suppose that the exact results of Algorithm 1 and Algorithm 2 are the same. In other words, Freund's Arnoldi is PRIMA with deflation. We are now going to prove that this is indeed the case.

Proving that the exact results of both algorithms are the same, is equivalent to proving that $X = V$ and that for V holds that

$$\forall j,\ 0 < j \le \nu: \quad \mathbf{v}_j = \frac{\tilde{\mathbf{v}}_j}{\|\tilde{\mathbf{v}}_j\|}\ ,\quad \tilde{\mathbf{v}}_j = \left(1 - \sum_{i=1}^{j-1} \mathbf{v}_i \mathbf{v}_i^T\right) \mathbf{g}(\mathbf{v}_{j-m}), \tag{7}$$

where $\mathbf{g}(\mathbf{v}_{j-m}) = A\mathbf{v}_{j-m}$ if $j > m$ and $\mathbf{g}(\mathbf{v}_{j-m}) = \mathbf{r}_j$ if $0 < j \le m$, $\mathbf{g} : \mathbb{R}^m \to \mathbb{R}^m$. This is the wanted result of the original Arnoldi algorithm for multiple starting

vectors. So from this point on we will use V as projection matrix for both algorithms. Introducing the notation

$$\mathbf{q}(\mathbf{v}_j, \tau, \varphi) = (1 - \sum_{i=\tau}^{\varphi} \mathbf{v}_i \mathbf{v}_i^T) \mathbf{g}(\mathbf{v}_j), \tag{8}$$

we need to prove that for all j between 0 and q we have $\mathbf{v}_j = \frac{\mathbf{q}(v_{j-m},1,j-1)}{||\mathbf{q}(v_{j-m},1,j-1)||}$. Also, introduce the set of invariants

- $\Gamma(p) \equiv \{\forall j, \ 0 < j \le p - 1 : \ \mathbf{v}_j = \frac{\mathbf{q}(v_{j-m},1,j-1)}{||\mathbf{q}(v_{j-m},1,j-1)||}\}, \Gamma : \mathbb{N} \to \mathbb{B} = \{false, true\}$.
- $\Lambda(p, \tau, \varphi) \equiv \{\hat{\mathbf{v}}_p = \mathbf{q}(\mathbf{v}_{p-m}, \tau, \varphi - 1)\}, \Lambda : \mathbb{N}^3 \to \mathbb{B}$.
- $\Omega(p, \tau, \varphi, \omega) \equiv \Lambda(p, \tau, \varphi) \wedge \ldots \wedge \Lambda(p + \omega - 1, \tau, \varphi), \Omega : \mathbb{N}^4 \to \mathbb{B}$.

If we can prove that $\Gamma(\nu + 1)$ holds at the end of Algorithm 1 and Algorithm 2 we can conclude that $V = X$. We use Lemma 1 in both proofs.

Lemma 1. *If for the set parameters (α, β, γ) the invariant $\Lambda(\alpha, \beta, \gamma)$ holds, then after*

$$\hat{\mathbf{v}}_\alpha = \hat{\mathbf{v}}_\alpha - \mathbf{v}_\gamma \mathbf{v}_\gamma^T \hat{\mathbf{v}}_\alpha, \tag{9}$$

is executed, $\Lambda(\alpha, \beta, \gamma + 1)$ holds.

Proof. Assume $\Lambda(\alpha, \beta, \gamma)$ holds. Then

$$\hat{\mathbf{v}}_\alpha = \mathbf{q}(\mathbf{v}_{\alpha-m}, \beta, \gamma - 1). \tag{10}$$

Substitute this into (9) we get

$$\begin{aligned}
\hat{\mathbf{v}}_\alpha &:= (1 - \mathbf{v}_\gamma \mathbf{v}_\gamma^T) \mathbf{q}(\mathbf{v}_{\alpha-m}, \beta, \gamma - 1) \\
&= (1 - \sum_{j=\beta}^{\gamma-1} \mathbf{v}_j \mathbf{v}_j^T) \mathbf{g}(\mathbf{v}_{\alpha-m}) - \mathbf{v}_\gamma \mathbf{v}_\gamma^T [(1 - \sum_{j=\beta}^{\gamma-1} \mathbf{v}_j \mathbf{v}_j^T) \mathbf{g}(\mathbf{v}_{\alpha-m})] \\
&= (1 - \sum_{j=\beta}^{\gamma} \mathbf{v}_j \mathbf{v}_j^T) \mathbf{g}(\mathbf{v}_{\alpha-m}), \tag{11}
\end{aligned}$$

since $v_i^T v_j = 0$ for all i and j, $i \ne j$. □

Now we will prove that $\Gamma(\nu + 1)$ holds at the end of both algorithms. We start with Algorithm 2 since it is less complicated.

Theorem 1. *$\Gamma(\nu + 1)$ holds at the end of Algorithm 2.*

Proof. $\Gamma(1) \wedge \Omega(1, 1, 1, m)$ holds after line 3 in Algorithm 2, hence we are left with proving that $\Gamma(k) \wedge \Omega(k, 1, k, m)$ implies $\Gamma(k + 1) \wedge \Omega(k + 1, 1, k + 1, m)$ after the loop that starts in line 4. So assume we have we have $\Gamma(k) \wedge \Omega(k, 1, k, m)$ after line 4. In line 5 the current vector is normalized, and since $\Lambda(k, 1, k)$ holds we see that after this line we have $\Gamma(k + 1) \wedge \Omega(k, 1, k, m)$. In line 6 a new candidate vector is created, this adds $\Lambda(k + m, 1, 1)$. This implies that after line 6 we have $\Gamma(k + 1) \wedge \Omega(k, 1, k, m) \wedge \Lambda(k + m, 1, 1)$. Applying Lemma 1 wields that after line 9 we have $\Gamma(k + 1) \wedge \Omega(k, 1, k, m) \wedge \Lambda(k + m, 1, k + 1)$. This is equivalent with $\Gamma(k+1) \wedge \Lambda(k, 1, k) \wedge \ldots \wedge \Lambda(k+m-1, 1, k) \wedge \Lambda(k+m, 1, k+1)$. Again, using lemma

1 we find that for $j = 1 \ldots m - 1$ we can replace $\Lambda(k+j, 1, k)$ with $\Lambda(k+j, 1, k+1)$. This gives

$$\Gamma(k+1) \wedge \Lambda(k, 1, k) \wedge \Lambda(k+1, 1, k+1) \wedge \ldots \wedge \Lambda(k+m, 1, k+1)$$
$$\Rightarrow \Gamma(k+1) \wedge \Omega(k+1, 1, k+1, m).$$

This concludes our proof.   □

Theorem 2. $\Gamma(\nu + 1)$ *holds at the end of Algorithm 1.*

Proof. Again, we start with $\Gamma(1)$ and after the initialization of line 1 we have $\Gamma(1) \wedge \Omega(1, 1, 1, m)$. Using Lemma 1 like in Theorem 1 we find after line 7 that we have $\Gamma(m + 1)$. Now assume we have $\Gamma(km + 1)$ after line 8. This is valid for $k = 1$, so for induction arguments we should find $\Gamma((k + 1)m + 1)$ after line 18. In line 9, new candidate vectors are created. So for i is equal $km + 1$ to $(k + 1)m$ we get $\hat{x}_i = g(x_{i-m})$. This gives $\Gamma(km + 1) \wedge \Omega(km + 1, 1, 1, m) \equiv \Gamma(km + 1) \wedge \Omega(km + 1, km + 1, km + 1, m)$. Now assume after line 10 we have $\Gamma(km+1) \wedge \Omega(km+1, (k-j+1)m+1, km+1, m)$. Apply $\hat{X}_k = \hat{X}_k - X_{k-j}X_{k-j}^T \hat{X}_k$. In vector terms this means that $[\hat{x}_{km+1} | \ldots | \hat{x}_{(k+1)m}] = [\hat{x}_{km+1} | \ldots | \hat{x}_{(k+1)m}] - [x_{(k-j)m+1}x_{(k-j)m+1}^T \hat{x}_{km+1} - \ldots - x_{(k-j+1)m}x_{(k-j+1)m}^T \hat{x}_{km+1} | \ldots | x_{(k-j)m+1}x_{(k-j)m+1}^T \hat{x}_{(k+1)m} - \ldots - x_{(k-j+1)m}x_{(k-j+1)m}^T \hat{x}_{(k+1)m}] = [Ax_{(k-1)m+1} - \sum_{i=(k-j)m+1}^{km} x_i x_i^T Ax_{(k-1)m+1} | \ldots$

$|Ax_{km} - \sum_{i=(k-j)m+1}^{km} x_i x_i^T Ax_{km}] \Rightarrow \Omega(km+1, (k-j)m+1, km+1, m)$ if $j \le k$. This is the case in line 11. Therefore, after line 11 we have $\Gamma(km+1) \wedge \Omega(km+1, (k-j)m+1, km+1, m)$, and the result of this loop will be $\Gamma(km+1) \wedge \Omega(km+1, 1, km+1, m)$. Again, a modified Gramm-Schmidt step is done for the lines 13 until 18. As we saw before, the result after line 18 is $\Gamma((k+1)m+1)$, which was the result we needed for our assumption to be correct. We may now substitute the $\rho - 1$ for the final result to get $\Gamma(\rho m + 1)$. With $\rho m + 1 = \nu + 1$ we may conclude the proof.   □

3 Implementation and Case Studies

In the previous section we have proved the mathematical equivalence of the generated subspaces, while neglecting the numerical errors. In this section we will comment on the implementation of both algorithms within the software tools MOR for ANSYS and RW and will point out what adjustments we have made to the RW function in order to improve the performance for chosen case-studies.

3.1 MOR for ANSYS

MOR for ANSYS is an extension to the commercial finite element simulator ANSYS. It takes as input a linear ANSYS model (file.full), reduces it and gives as output the matrices of the reduced system (2) in MatrixMarket format. However, for the purpose of the COMSON project it has been adjusted to also take as input the matrices of the arbitrary linear dynamical system (1). The code is a C++ implementation of Algorithm 2. The solve step in line 6 can be done with several forward-backward substitution methods (like LU- and Cholesky decomposition), which are available via the TAUCS-library. The reordering is done with METIS [7].

3.2 Rom Workbench (RW)

Rom Workbench is written in MATLAB. It implements several MOR methods including the Algorithm 1. The PRIMA function takes as input the matrices of the linear system (1) in MATLAB-(sparse)array format. Unfortunately, the efficiency of this implementation is limited because it has no special treatment for symmetric C and G, as the only available factorization is the LU decomposition of MATLAB with *colamd* as re-ordering scheme. However, the dynamical systems which arise from technical applications, as MEMS or electrical circuits, usually do have symmetric system matrices. As the COMSON Demonstrator Platform [9] should be able to handle a wide variation of industry-relevant problems, we have adjusted the PRIMA function of RW in such a way, that for symmetric matrices the performance is increased. We have implemented Cholesky-decomposition (as Cholesky is at least two times faster than LU decomposition) with *symamd* as re-ordering scheme. In the following we will call our adjusted version symRW.

3.3 Performance Phases

To analyse the bottlenecks of different implementations we divide the algorithm courses into several phases. These are

- Phase 1: Reading the original matrices into the memory from file and writing the reduced matrices to file.
- Phase 2: Reordering the matrix G.
- Phase 3: Factoring G and constructing the first basis vector.
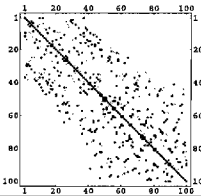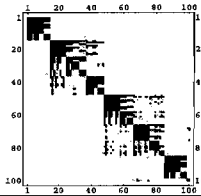- Phase 4: Constructing the rest basis vectors via the back substitution in each iteration.
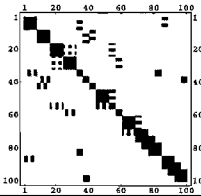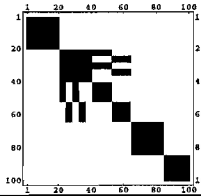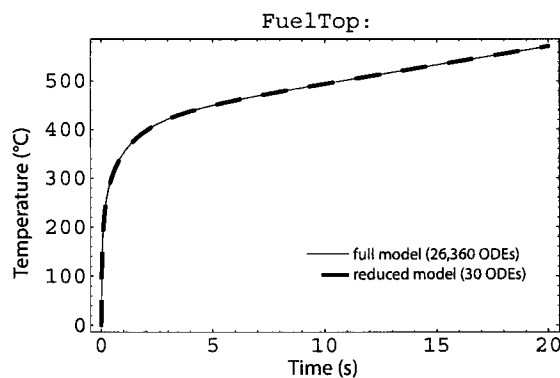
3.4 Case Studies

In order to test the presented MOR tools on industry-relevant problems, we have chosen two electro-thermal MEMS devices [10]. The pyrotechnical microthruster is based on the integration of the solid fuel with a silicon micro-machined structure. The thermally tunable optical filter is a Fabry-Perot interferometer fabricated as a free-standing membrane. Both models have been made and meshed in ANSYS (using low and high-order elements. In Table 1 the dimension of each test model and the matrix structure of the matrix G are shown.

4 Numerical Results

We have reduced the described case studies using MOR for ANSYS, RW and it's ajusted version symRW. In Fig. 1 a good match between the step response of the full-scale and that of the reduced order model at a single output node of the pyrotechnical microthruster are displayed. The difference between the reduced model computed with MOR for ANSYS and RW/symRW is of the order of rounding errors, as expected. In Tables 2 and 3 we compare the reduction time (down to order 30) of MOR for ANSYS, RW and symRW. CPU time of RW is up to 60 times longer than the CPU time of MOR for ANSYS. Due to our improvement, this difference has been reduced to 12 times for the largest case study. As expected the main speed up was achieved by introducing Cholesky factorization for the symmetric G and a more effective ordering. The remaining CPU time difference is mainly due to the interpretation overhead in MATLAB.

Table 1. Characteristics of MEMS benchmark

Model	Dimension	Structure of G (beginning part)	Model	Dimension	Structure of G (beginning part)
A	1 668		B	106 437	
Tunable optical filter (low order elements)			Tunable optical filter (high order elements)		
C	26 360		D	79 171	
Pyrotechnical microthruster (low order elements)			Pyrotechnical microthruster (high order elements)		

FuelTop:

Fig. 1. Step response of the full scale and reduced models (computed with MOR for ANSYS and RW) in a single output node of microthruster.

5 Conclusion

We have compared two software tools, which are meant to be integrated into the COMSON demonstrator platform. They belong to the two main implementation streams, fast prototyping in the interpreter environment and the compiled language implementation in C++. We have proven that both algorithms generate the same reduced basis and that the most important bottleneck for MATLAB is the decomposition phase. We have implemented Cholesky factorization for the symmetric problems in RW and have switched to a *symamd* re-ordering. Hence, the present run times in MATLAB allow for testing moderate-size industry-relevant problems within the COMSON demonstrator platform.

Table 2. Computational times in seconds on AMD Opteron with 2.4 GHz and 16 Gb RAM.

Model A	M4A	RW	symRW	Model B	M4A	RW	symRW
Phase 1	8.39e-2	1.10e-1	6.00e-2	Phase 1	2.24e1	1.10e1	1.04e1
Phase 2	2.11e-2	1.00e-2	3.00e-2	Phase 2	3.42e0	2.35e0	2.21e0
Phase 3	5.63e-2	2.00e-2	1.00e-2	Phase 3	1.05e1	3.49e2	7.30e1
Phase 4	2.99e-2	6.00e-2	1.00e-1	Phase 4	9.22e0	3.98e1	2.85e1

Model C	M4A	RW	symRW	Model D	M4A	RW	symRW
Phase 1	2.34e0	1.09e0	1.07e0	Phase 1	3.54	1.68e1	1.66e1
Phase 2	4.65e-1	1.31e0	2.80e-1	Phase 2	4.04e0	7.63e0	2.50e0
Phase 3	2.52e0	2.02e2	4.48e1	Phase 3	4.19e1	6.46e3	1.56e3
Phase 4	2.02e0	1.13e1	7.65e0	Phase 4	1.71e1	1.13e2	7.68e1

Table 3. Complete reduction times in s for all the case studies on AMD Opteron with 2.4 GHz and 16 Gb RAM. nnz ist the number of nonzero matrix elements of G and it's factor L.

Model	n	$nnz(G)$	M4A time	$nnz(L)$	RW time	$nnz(L)$	symRW time	$nnz(L)$
A	1668	6.21e3	1.91e-1	2.46e4	2.00e-1	3.39e4	2.00e-1	2.32e4
B	106437	1.41e6	4.55e1	1.89e7	4.02e2	4.82e7	1.14e2	2.84e7
C	26360	2.65e5	7.34e0	5.00e6	2.16e2	1.68e7	5.38e1	1.06e7
D	79171	2.22e6	9.85e1	4.56e7	6.60e3	1.88e8	1.66e3	1.24e8

Acknowledgment

References

1. A.C. Antoulas: *Approximation of Large-Scale Dynamical Systems*, Society for Industrial and Applied Mathematics, 2005.
2. www.imtek.de/simulation/mor4ansys
3. www.imek.be/codestar
4. http://math.nist.gov/MatrixMarket/
5. R.W. Freund: *Krylov-subspace methods for reduced order modeling in circuit simulation*, Journal of Computational and Applied Mathematics, Vol. 123, pp. 395–421, 2000.

6. Z.J. Bai, K. Meerbergen, Y. F. Su: *Arnoldi methods for structure-preserving dimension reduction of second-order dynamical systems, in P. Benner, V. Mehrmann, D. Sorensen (eds), Dimension Reduction of Large-Scale Systems, Lecture Notes in Computational Science and Engineering*, Springer-Verlag, Berlin/Heidelberg, Germany, 2005

7. G. Karypis, V. Kumar: *A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs*, Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995.

8. A. Odabasioglu, M. Celik, T. Pileggi: *PRIMA: Passive Reduced-order Interconnect Macromodeling Algorithm*, IEEE Trans. Comp. Aid. Design Int. Circ. Syst., Vol. 17, pp. 645-654, 1998.

9. www.comson.org

10. Oberwolfach   Model   Reduction   Benchmark   Collection, www.imtek.de/simulation

11. T. Voss: *Model reduction for nonlinear differential algebraic equations*, MSc. Thesis University of Wuppertal, 2005; Unclassified Report PR-TN-2005/00919, Philips Research Laboratories, 2005.