# 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 is implemented in C++ and ROM Workbench is a MATLAB code. We further compare two Arnoldi-based reduction algorithms, which seems to be the most promising for microsystem design applications. 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 microsystem 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.
The goal of this paper is to numerically compare two MOR tools, which belong to the mentioned 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 format [4]. Back coupling of the reduced model with the rest of the circuitry might be done by either converting a reduced ODE/DAE system into equivalent electrical circuit, or by enabling a simulator to incorporate a reduced model as a black-box. M4A 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 of block-Arnoldi based on [7]. Both tools are planned for use in the European project COMSON [8], 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 [7]. 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 block Arnoldi-based order reduction with both tools are presented. In section 5, we conclude the paper.

## 2 Block Arnoldi Algorithms

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

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

with initial condition $\mathbf{x}(\mathbf{0}) = \mathbf{x}_0$. Here, $t$ is the time variable, $\mathbf{x}(t) \in R^n$ the state vector, $\mathbf{u}(t) \in R^m$ the input excitation vector and $\mathbf{y}(t) \in R^p$ the output measurement vector. $G$, $C \in R^{n\times n}$ are linear (not depending on $\mathbf{x}$ and $t$) symmetric and sparse system matrices, $B \in R^{n\times m}$ and $L \in R^{n\times p}$ 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 $V, W \in R^{n\times\nu}$, to construct a reduced system of the order $\nu$ as

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

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

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

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

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

where $m_i = -L^T A^i B$ 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 $\nu$. 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

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

with $A_r = W^TAW$, $R_r = W^TR$ and $L_r = W^TL$. The order of (5) is $\nu = \rho \cdot m$. The property of the Krylov subspace is such that the first $\nu$ moments of $H_r(s) = -L_r^T(I - (s - s_0)A_r)^{-1}R_r$ and of $H(s)$ are 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 [7] 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 [7] stands for Passive Reduced-order Interconnect Macromodeling Algorithm. An orthonormal basis, $X$, is generated such that span($X$) = $Kr(A, R, \rho)$, but $X$ is used for an explicit projection of (1), which means that $C_r = X^TCX$, $G_r = X^TGX$, $B_r = X^TB$ and $L_r = X^TL$. In [7] 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

$$||\hat{\mathbf{v}}_k|| < DTOL, \tag{6}$$

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

**Algorithm 1**
Block-Arnoldi as in PRIMA [7]

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

**Algorithm 2**
Freund's Arnoldi ignoring deflation

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

## 2.3 Comparison between both Algorithms

We state 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 $X = V$ ($V$ is the orthonormal projection basis of Algorithm 2), i. e. that both methods produce the same space and that for $V$ holds that

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

with $\mathbf{g}(\mathbf{v}_{j-m}) = A\mathbf{v}_{j-m}$ if $j > m$ and $\mathbf{g}(\mathbf{v}_{j-m}) = \mathbf{r}_j$ if $0 < j \leq m$, $\mathbf{g} : R^m \rightarrow R^m$. (7) states that the columns of $V$ indeed form the wanted Krylov space with orthogonalization, which is the wanted result of the original Arnoldi algorithm for multiple starting vectors. In the following we will call the projection matrix for both algorithms $V$. 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

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

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 the proof.

**Lemma 1.** *If for the set parameters $(p, \tau, \varphi)$ the invariant $\Lambda(p, \tau, \varphi)$ holds, then after*

$$\hat{\mathbf{v}}_p = \hat{\mathbf{v}}_p - \mathbf{v}_\varphi \mathbf{v}_\varphi^T \hat{\mathbf{v}}_p, \tag{9}$$

*is executed, $\Lambda(p, \tau, \varphi + 1)$ holds.*

*Proof.* Assume $\Lambda(p, \tau, \varphi)$ holds. Then

$$\hat{\mathbf{v}}_p = \mathbf{q}(\mathbf{v}_{p-m}, \tau, \varphi - 1). \tag{10}$$

Substitute this into (9) we get

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

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

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

*Proof.* We start with the observation that $\Gamma(1) \wedge \Omega(1, 1, 1, m)$ holds after line 3 in Algorithm 2 and after line 1 in Algorithm 1. Afterwards, we prove that $\Gamma(k) \wedge \Omega(k, 1, k, m)$ implies $\Gamma(k+1) \wedge \Omega(k+1, 1, k+1, m)$ within the loop. This can be done by using Lemma 1. Then it indeed follows by induction that $\Gamma(\nu+1)$ holds at the end of both algorithms. The complete proof can be found in [9]. □

## 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 [10].

### 3.2 ROM Workbench (RW) and its Modification (symRW)

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 [8] 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. Please note, that it is further possible to generate C code directly from MATLAB function. This would speed up the loops, but one would still need the quick solvers from the TAUCS-library.

### 3.3 Case Studies

In order to test the presented MOR tools on industry-relevant problems, we have chosen two electro-thermal MEMS devices [11]. 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 finite elements.

## 4 Numerical Results

We have reduced the described case studies using MOR for ANSYS, RW and its adjusted version symRW. We have limited opurselfs to the single-input case, so that deflation had no impact and no error control has been used. 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 neglectable, as expected. In Tables 2 and 1 we compare the reduction time (down to order 30) of M4A, RW and symRW.To analyze 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.

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.

## 5 Conclusion

We have compared two software tools (a single MOR routine), 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.

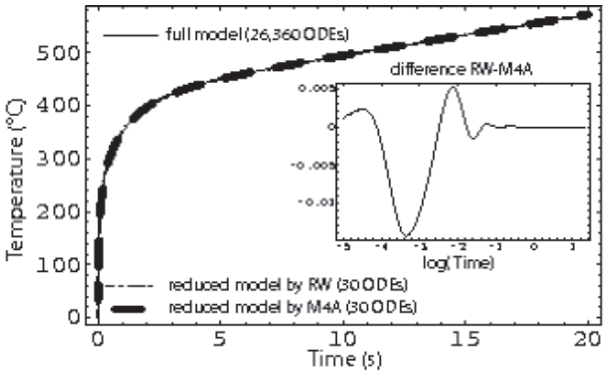


Fig. 1: Step response of the full scale and reduced models (computed with MOR for ANSYS and RW) in a single output node of microthruster (model C).

Table 1: Complete reduction times in s for all the case studies on AMD Opteron with 2.4 GHz and 16 Gb RAM. nnz is the number of nonzero matrix elements of $G$ and its factor $l$ (from the LU decomposition).

| Model | | | M4A | | RW | | symRW | |
|---|---|---|---|---|---|---|---|---|
| | $n$ | nnz($G$) | time | nnz($l$) | time | nnz($l$) | time | nnz($l$) |
| A | 1668 | 6.21e3 | 0.19 | 2.46e4 | 0.2 | 3.39e4 | 0.2 | 2.32e4 |
| B | 106437 | 1.41e6 | 45.5 | 1.89e7 | 402 | 4.82e7 | 114 | 2.84e7 |
| C | 26360 | 2.65e5 | 7.34 | 5.00e6 | 216 | 1.68e7 | 53.8 | 1.06e7 |
| D | 79171 | 2.22e6 | 98.5 | 4.56e7 | 6600 | 1.88e8 | 1660 | 1.24e8 |

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

| Model A | M4A | RW | symRW |
|---|---|---|---|
| Phase 1 | 0.08 | 0.11 | 0.06 |
| Phase 2 | 0.02 | 0.01 | 0.03 |
| Phase 3 | 0.05 | 0.02 | 0.01 |
| Phase 4 | 0.02 | 0.06 | 0.01 |

| Model B | M4A | RW | symRW |
|---|---|---|---|
| Phase 1 | 22.4 | 11 | 10.4 |
| Phase 2 | 3.42 | 2.35 | 2.21 |
| Phase 3 | 10.5 | 349 | 73 |
| Phase 4 | 9.22 | 39.8 | 28.5 |

| Model C | M4A | RW | symRW |
|---|---|---|---|
| Phase 1 | 2.34 | 1.09 | 1.07 |
| Phase 2 | 0.465 | 1.31 | 0.28 |
| Phase 3 | 2.52 | 202 | 44.8 |
| Phase 4 | 2.02 | 11.3 | 7.65 |

| Model D | M4A | RW | symRW |
|---|---|---|---|
| Phase 1 | 3.54 | 16.8 | 16.6 |
| Phase 2 | 4.04 | 7.63 | 2.50 |
| Phase 3 | 41.9 | 6460 | 1560 |
| Phase 4 | 17.1 | 113 | 76.8 |

## Acknowledgment

We would like to thank Dr. Evgenii B. Rudnyi from the University of Freiburg for helping us with MOR for ANSYS, Prof. Dr. Gabriela Ciuprina from the University Politehnica of Bucharest for her help with ROM Workbench and to acknowledge the EU support through the COMSON project.

## References

1. A.C. Antoulas: *Approximation of Large-Scale Dynamical Systems*, SIAM, 2005.
2. http://www.imtek.de/simulation/mor4ansys
3. http://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 Comp. and Appl. Math., 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. 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.
8. http://www.comson.org
9. S. Vollebregt, T. Bechtold, A. Verhoeven, E.J.W. ter Maten: *Model Order Reduction of Large Scale ODE Systems: MOR for ANSYS versus ROM Workbench*, CASA-Report 06-38, 2006, ftp://ftp.win.tue.nl/pub/rana/rana06-38.pdf.
10. 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.
11. Oberwolfach Model Reduction Benchmark Collection, http://www.imtek.de/simulation