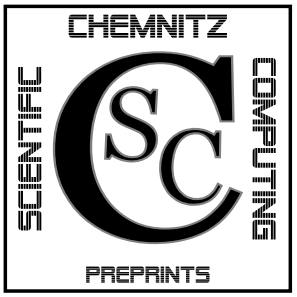
TECHNISCHE UNIVERSITÄT CHEMNITZ

José M. Badía · Peter Benner · Rafael Mayo
Enrique S. Quintana-Ortí · Gregorio Quintana-Ortí
Alfredo Remón

Balanced Truncation Model Reduction of Large and Sparse Generalized Linear Systems

CSC/06-04



Chemnitz Scientific Computing Preprints

Impressum:

Chemnitz Scientific Computing Preprints — ISSN 1864-0087
(1995–2005: Preprintreihe des Chemnitzer SFB393)

Herausgeber:
Professuren für
Numerische und Angewandte Mathematik
an der Fakultät für Mathematik
der Technischen Universität Chemnitz

Postanschrift:
TU Chemnitz, Fakultät für Mathematik
09107 Chemnitz
Sitz:
Reichenhainer Str. 41, 09126 Chemnitz

`http://www.tu-chemnitz.de/mathematik/csc/`

TECHNISCHE UNIVERSITÄT CHEMNITZ

Chemnitz Scientific Computing Preprints

José M. Badía · Peter Benner · Rafael Mayo
Enrique S. Quintana-Ortí · Gregorio Quintana-Ortí
Alfredo Remón

Balanced Truncation Model Reduction of Large and Sparse Generalized Linear Systems

CSC/06-04

CSC/06-04 ISSN 1864-0087 **November 2006**

Contents

Balanced Truncation Model Reduction of Large and Sparse Generalized Linear Systems

José M. Badía [a,1], Peter Benner [b,2], Rafael Mayo [a,1], Enrique S. Quintana-Ortí [a,1], Gregorio Quintana-Ortí [a,1] Alfredo Remón [a,1]

[a] *Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071-Castellón, Spain; e-mails: {badia,mayo,quintana,gquintan,remon}@icc.uji.es*

[b] *Fakultät für Mathematik, Technische Universität Chemnitz, D-09107 Chemnitz, Germany; e-mail:* benner@mathematik.tu-chemnitz.de

Abstract

We investigate model reduction of large-scale linear time-invariant systems in generalized state-space form. We consider sparse state matrix pencils, including pencils with banded structure. The balancing-based methods employed here are composed of well-known linear algebra operations and have been recently shown to be applicable to large models by exploiting the structure of the matrices defining the dynamics of the system.

In this paper we propose a modification of the LR-ADI iteration to solve large-scale generalized Lyapunov equations together with a practical convergence criterion, and several other implementation refinements. Using kernels from several serial and parallel linear algebra libraries, we have developed a parallel package for model reduction, SpaRed, extending the applicability of balanced truncation to sparse systems with up to $\mathcal{O}(10^5)$ states. Experiments on an SMP parallel architecture consisting of Intel Itanium 2 processors illustrate the numerical performance of this approach and the potential of the parallel algorithms for model reduction of large-scale sparse systems.

Key words: Model reduction, balanced truncation, generalized linear time-invariant systems, sparse linear algebra, generalized Lyapunov equations, SMP, parallel computation.

[1] Supported by the project *Acciones Integradas* HA2005-0081, and the CICYT project TIN2005-09037-C02-02 and FEDER.

[2] Supported by DFG grant BE 2174/7-1 *Automatic parameter-preserving model reduction for microsystems technology* and the DAAD project D/05/25675.

1 Introduction

We address model reduction of continuous linear time-invariant (LTI) systems, defined in generalized state-space form by

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t), \quad t > 0, \quad x(0) = x^0, \\ y(t) &= Cx(t) + Du(t), \quad t \geq 0, \end{aligned} \tag{1}$$

where $E, A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times m}$, and $x^0 \in \mathbb{R}^n$ is the initial state of the system. Throughout this paper, we assume E to be nonsingular (a usual case as, in general, E will represent the mass matrix corresponding to a finite-element discretization). The number of states, n, is known as the state-space dimension or the order of the system and, in practice, is often much larger than its numbers of inputs and outputs, m and p, respectively. The corresponding transfer function matrix (TFM) of the system is given by

$$G(s) := C(sE - A)^{-1}B + D,$$

which defines the relation between the inputs and the outputs in the frequency domain, as $y(s) = G(s)u(s)$. Hereafter, we assume that the generalized spectrum of the state matrix pencil $A - \lambda E$ is contained in the open left half plane, implying that the system is stable. Despite the fact that under the given assumptions (1) is equivalent to the standard state-space realization given by

$$\begin{aligned} \dot{x}(t) &= E^{-1}Ax(t) + E^{-1}Bu(t) \;=\; A_s x(t) + B_s u(t), \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{2}$$

performing this transformation explicitly is often undesirable as, in general, it will destroy the sparsity structure of the state matrix pencil, and it may also introduce large rounding errors at an early stage of the computation due to possible ill-conditioning of E.

The model reduction problem consists in finding a reduced-order realization

$$\begin{aligned} E_r\dot{x}_r(t) &= A_r x_r(t) + B_r u(t), \quad t > 0 \qquad x_r(0) = x_r^0, \\ y_r(t) &= C_r x_r(t) + D_r u(t), \quad t \geq 0, \end{aligned} \tag{3}$$

of order $r \ll n$, so that the output error $y - y_r$ is "small". Now, assuming the input of the system is bounded, this is equivalent to requiring that the TFM of the reduced-order realization (3),

$$G_r(s) := C_r(sE_r - A_r)^{-1}B_r + D_r,$$

"approximates" $G(s)$, as $y - y_r = Gu - G_r u = (G - G_r)u$.

Model reduction is an important task in control design because real-time control is only possible using controllers of low complexity and the fragility of such devices increases with the complexity. In particular, control and optimization of the large-scale systems arising from spatial finite element discretization of systems governed by partial differential equations vary from difficult to impossible without model reduction. Also, simulation of many physical phenomena greatly benefits from important reductions in time when using small, reduced-order realizations. Model reduction of large-scale generalized LTI systems is employed in control of multibody (mechanical) systems, manipulation of fluid flow, (e.g., Navier-Stokes equations), circuit simulation, VLSI chip design, in particular when modeling the interconnections via RLC networks, simulation of MEMS and NEMS (micro- and nano-electro-mechanical systems), etc.; see, e.g., [2,7,21] and references therein. State-space dimensions n of order 10^2–10^4 are common in these applications. Very large systems, with state-space dimension as high as $\mathcal{O}(10^5)$–$\mathcal{O}(10^6)$, arise in weather forecast, circuit simulation, VLSI design, and air quality simulation among others (see, e.g., [2,7]). It is quite frequent that the state matrix pencils in these problems contain a small number of nonzero entries, i.e., they can be defined as *sparse* (Hereafter, we use the term sparse to refer to both unstructured sparse matrices and banded matrices.) It is this particular type of systems, with large and sparse state matrix pencils, that we address in this paper.

The methods for model reduction can be classified into two different families: moment matching-based methods and SVD-based methods. (For a thorough analysis of these two families of methods, see [2]). The efficacy of model reduction methods strongly relies on the problem and there is no technique that can be considered optimal in an overall sense. In general, moment matching methods employ numerically stable and efficient Arnoldi and Lanczos procedures in order to compute the reduced-order realizations. These methods, however, are specialized for certain problem classes and often do not preserve important properties of the system such as stability or passivity.

On the other hand, SVD-based methods are appealing in that they usually preserve these properties, and also provide bounds on the approximation error. However, SVD-based methods present a higher computational cost. In particular, all SVD-based methods require, as the major computational stage, the solution of two Lyapunov (or other matrix) equations. The Lyapunov equations arising in balanced truncation when applied to (1) are

$$AW_cE^T + EW_cA^T + BB^T = 0, \tag{4}$$

$$A^T\hat{W}_oE + E^T\hat{W}_oA + C^TC = 0, \tag{5}$$

where due to the assumed stability of $A - \lambda E$, matrices $W_c,\ W_o$ are symmetric, positive semi-definite. Unfortunately, $W_c,\ W_o$ are dense, square $n \times n$ matrices even if A, E are sparse. These equations can for instance be solved

by codes from the SLICOT library [3] using "direct" Lyapunov solvers as those in [4,18], which allows the reduction of small LTI systems (roughly speaking, $n \leq 5000$ on current desktop computers with 32-bit architecture). Larger problems, with tens of thousands of state-space variables, can be reduced using the sign function-based methods in PLiCMR [4] [9,10] on parallel computers [10]. However, the difficulties of exploiting the usual sparse structure of the Lyapunov equations using direct or sign function-based solvers ultimately limits the applicability of the SVD-based algorithms in these two libraries. Most recently, a new approach has been proposed for the solution of large Lyapunov equations arising in special classes of dense problems via the sign function. The method features a linear-polylogarithmic complexity, achieved by employing hierarchical matrix structures and the related formatted arithmetic [5].

In the last years, with the formulation of the *low-rank alternating direction implicit* (LR-ADI) iteration for the Lyapunov equation [16,17,20,24,25], SVD-based methods have regained interest. This iteration exploits the sparse structure of the coefficient matrices of the Lyapunov equation and, therefore, can be employed to construct SVD-based model reduction algorithms for large LTI systems. Using this approach, standard LTI systems ($E = I_n$, the identity matrix of order n) with tens of thousands of state-space variables can be reduced using desktop computers.

In this paper we present a variant of the LR-ADI iteration adapted for the solution of generalized Lyapunov equations. In order to extend the applicability of the algorithms to larger problems, we have used existing efficient dense and sparse linear algebra libraries to develop a parallel library SpaRed [5] for model reduction of large-scale (standard and generalized) sparse systems, including those with state matrix pencils with banded structure. Note that the parallelization approach considered here can be based on processes or threads. The latter variant can be used to efficiently exploit the parallelism in modern shared-memory architectures so that SpaRed may become a useful software tool for model reduction on next generation desktop multi-core computers. Experiments on a SMP system with 16 Intel Itanium 2 processors demonstrate that balanced truncation can be an efficient approach for model reduction of large-scale systems.

The paper is structured as follows: In Section 2 we briefly review how to reduce generalized LTI systems efficiently. There we also describe a simple generalization of the LR-ADI iteration for generalized Lyapunov equations that preserves the sparse structure of the coefficient matrices of the equation; we discuss strategies to detect convergence of the iteration; and we give hints on how to select the shifts for the iteration depending on the structure of the matrices. Details of the parallel implementation of the corresponding algorithms are then given in Section 3, and the numerical and parallel performances of the

[3] Available from `http://www.slicot.org`.
[4] Available from `http://www.pscom.uji.es/software`.
[5] Available from `http://www.pscom.uji.es/software`.

new algorithms are reported in Section 4. Finally, some concluding remarks follow in Section 5.

2 Balanced Truncation for Model Reduction

Balanced truncation (BT) [22] belongs to the class of absolute error methods and is part of the family of SVD-based methods [2,23]. Absolute error methods attempt to minimize

$$\|\Delta_a\|_\infty := \|G - G_r\|_\infty,$$

where $\|G\|_\infty$ denotes the $\mathcal{L}_\infty$- or $\mathcal{H}_\infty$-norm of a stable, rational matrix function which is defined for proper transfer functions as

$$\|G\|_\infty := \sup_{\omega\in\mathbb{R}} \sigma_{\max}(G(\jmath\omega)),$$

with $\jmath := \sqrt{-1}$; here, $\sigma_{\max}(M)$ stands for the largest singular value of the matrix M.
We next review two BT approaches for model reduction of generalized LTI systems and then present a modification of the LR-ADI iteration [20,24] that can be used to solve generalized Lyapunov equations.

2.1 Obtaining the reduced-order system

BT methods are strongly related to the controllability and observability Gramians the system, W_c and W_o, respectively. These Gramians are given by the solutions of the two dual *generalized Lyapunov equations* from (4), (5),

$$AW_cE^T + EW_cA^T + BB^T = 0, \quad A^T\hat{W}_oE + E^T\hat{W}_oA + C^TC = 0,$$

and $W_o = E^T\hat{W}_oE$. As $A-\lambda E$ is assumed to be stable, W_c and W_o are positive semidefinite and therefore there exist factorizations $W_c = S^TS$ and $W_o = R^TR$. Matrices S and R are called the *"Cholesky" factors* of the Gramians (even if they are not Cholesky factors in a strict sense).
Consider now the singular value decomposition (SVD) of the product

$$SR^T = U\Sigma V^T = [U_L\ U_R]\begin{bmatrix}\Sigma_L & 0\\ 0 & \Sigma_R\end{bmatrix}\begin{bmatrix}V_L^T\\ V_R^T\end{bmatrix}, \tag{6}$$

where the matrices Σ, U, and V are conformally partitioned at a given dimension r such that $\Sigma_L = \mathrm{diag}\,(\sigma_1,\ldots,\sigma_r)$, $\Sigma_R = \mathrm{diag}\,(\sigma_{r+1},\ldots,\sigma_n)$, $\sigma_j \geq 0$ for all j, and $\sigma_r > \sigma_{r+1}$. Here, $\sigma_1,\ldots,\sigma_n$ are the *Hankel singular values* (HSV) of the system. In case $\sigma_r > \sigma_{r+1} = 0$, then r is the *McMillan* degree of the

system. That is, r is the state-space dimension of a minimal realization of the system. The HSV contain important information about the system dynamics as they measure "how much" a state is involved in the energy transfer from a given input to a certain output.

Square-root (SR) BT algorithms use the products of the SVD in (6) to obtain the reduced-order model as

$$E_r := T_L E T_R, \quad A_r := T_L A T_R, \quad B_r := T_L B, \quad C_r := C T_R, \quad D_r := D, \tag{7}$$

where the truncation (or projection) matrices $T_L \in \mathbb{R}^{r\times n}$ and $T_R \in \mathbb{R}^{n\times r}$ are given by

$$T_L := \Sigma_L^{-1/2} V_L^T R E^{-1} \quad \text{and} \quad T_R := S^T U_L \Sigma_L^{-1/2}. \tag{8}$$

Note that $E_r = I_r$ and thus need not be computed.

The *balancing-free square-root* (BFSR) BT algorithms can provide more accurate reduced-order models in the presence of rounding errors [28,33]. These algorithms share the first two stages (solving the coupled equations and computing the SVD of SR^T) with the SR methods, but differ in the procedure to obtain T_L and T_R. Specifically, the following two QR factorizations are computed,

$$S^T U_L = [P_1\ P_2] \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}, \qquad R^T V_L = [Q_1\ Q_2] \begin{bmatrix} \bar{R} \\ 0 \end{bmatrix}, \tag{9}$$

where P_1, $Q_1 \in \mathbb{R}^{n\times r}$ have orthonormal columns, and $\hat{R}$, $\bar{R} \in \mathbb{R}^{r\times r}$ are upper triangular. The reduced-order system in (7) is then given by the truncation matrices

$$T_L := (Q_1^T P_1)^{-1} Q_1^T E^{-1} \quad \text{and} \quad T_R := P_1.$$

In this version, we again obtain $E_r = I_r$. As this may not be desirable or necessary if we start out with an LTI system in generalized state-space form, one can also use

$$(E_r, A_r, B_r, C_r) = \left(Q_1^T P_1, Q_1^T E^{-1} A P_1, Q_1^T E^{-1} B, C P_1\right).$$

Both SR and BFSR BT algorithms provide a realization G_r which satisfies the theoretical error bound

$$\|\Delta_a\|_\infty = \|G - G_r\|_\infty \leq 2 \sum_{j=r+1}^{n} \sigma_j. \tag{10}$$

This allows an adaptive choice of the state-space dimension r of the reduced-order model once the HSV are known. Further details on model reduction of

generalized LTI systems using BT methods, in particular when E is singular, are given in [29].

2.2 Computing low-rank solutions of generalized Lyapunov equations

In this section we re-elaborate the Lyapunov solvers introduced in [20,24] in order to deal with the generalized Lyapunov equation. This is a fundamental part of the model reduction methods described in the previous subsection. The software package LYAPACK [6] [25] provides MATLAB scripts for solving generalized Lyapunov equations as in (4), (5) by using an implicit transformation to standard state-space form based on a sparse Cholesky (or LU) factorization of E. Here, we will follow an approach discussed in [6,19] which avoids this factorization. Another variation of this idea can be found in [30].

The methods considered here benefit from (4)–(5) sharing A and E as coefficient matrices of the Lyapunov equation. Part of their efficiency comes also from the constant terms in the equations, BB^T and C^TC, being formed as rank-k products, with $k \in \{m, p\}$ in general much smaller than n. The proposed iterative algorithms exploit the frequent low-rank property of the constant terms in (4)–(5) to provide low-rank approximations to the Cholesky or full-rank factors of the solution matrices. These approximations can reliably substitute S and R in the computation of (6), (8), and (9); see [9].

From the relation between the standard and generalized systems in (2) and (1), the controllability Gramian W_c is also given by the solution of the Lyapunov equation

$$A_s W_c + W_c A_s^T + B_s B_s^T = 0.$$

Now, given a set of shift parameters $\tau = \{\tau_1, \tau_2, \ldots\}$ with negative real part such that $\tau_j = \tau_{j+t_s}$, $j = 1, 2, \ldots$ (that is, the set is cyclic of period t_s), the LR-ADI iteration [20,24] delivers a pair of sequences of matrices, $\{U_j\}_{j=1}^{\infty}$ and $\{Y_j\}_{j=1}^{\infty}$, as follows. Initially,

$$\begin{aligned} U_1 &:= \gamma_1 \left(A_s + \tau_1 I_n\right)^{-1} B_s, \\ Y_1 &:= U_1, \end{aligned}$$

where $\gamma_1 = \sqrt{-2\,\mathrm{Re}(\tau_1)}$ and $\mathrm{Re}(\tau_k)$ stands for the real part of τ_k. From then on ($j > 1$),

$$\begin{aligned} U_j &:= \gamma_j \left(I_n - (\tau_j + \overline{\tau_{j-1}})(A_s + \tau_j I_n)^{-1}\right) U_{j-1}, \\ Y_j &:= \left[Y_{j-1},\ U_j\right]. \end{aligned}$$

Here, $\overline{\tau_j}$ denotes the conjugate of τ_j, and $\gamma_j = \sqrt{\frac{\mathrm{Re}(\tau_j)}{\mathrm{Re}(\tau_{j-1})}}$, $j > 1$. On con-

[6] Available from `http://www.slicot.org`.

vergence, after l_c iterations, the procedure provides an approximation matrix $Y_{l_c} \in \mathbb{R}^{n\times(m\cdot l_c)}$ such that $Y_{l_c}Y_{l_c}^T \approx S^TS = W_c$.
Let us rewrite the LR-ADI iteration in order to avoid explicit references to A_s or B_s. Thus, for the first iteration,

$$\begin{aligned}U_1 :=& \gamma_1 \left(A_s + \tau_1 I_n\right)^{-1} B_s = \gamma_1 \left(E^{-1}A + \tau_1 I_n\right)^{-1} E^{-1}B \\ &= \gamma_1 \left(E^{-1}(A + \tau_1 E)\right)^{-1} E^{-1}B = \gamma_1 \left(A + \tau_1 E\right)^{-1} E E^{-1}B \\ &= \gamma_1 \left(A + \tau_1 E\right)^{-1} B,\end{aligned}$$

while, for the remaining ones ($j > 1$),

$$\begin{aligned}U_j :=& \gamma_j \left(I_n - (\tau_j + \overline{\tau_{j-1}})(A_s + \tau_j I_n)^{-1}\right) U_{j-1} \\ &= \gamma_j \left(I_n - (\tau_j + \overline{\tau_{j-1}})(E^{-1}A + \tau_j I_n)^{-1}\right) U_{j-1} \\ &= \gamma_j \left(I_n - (\tau_j + \overline{\tau_{j-1}})(E^{-1}(A + \tau_j E)^{-1}\right) U_{j-1} \\ &= \gamma_j \left(I_n - (\tau_j + \overline{\tau_{j-1}})(A + \tau_j E)^{-1}E\right) U_{j-1} \\ &= \gamma_j \left(U_{j-1} - (\tau_j + \overline{\tau_{j-1}})(A + \tau_j E)^{-1}EU_{j-1}\right).\end{aligned}$$

The situation is slightly different for the observability Gramian. Rather than working with (5), we use here

$$A_s^TW_o + W_oA_s + C^TC = 0.$$

Following the same idea as above, the LR-ADI iteration can be re-formulated to produce the sequence $\{V_j\}_{j=1}^{\infty}$ as

$$V_1 := \gamma_1 \left(A_s^T + \overline{\tau_1} I_n\right)^{-1} C^T = \gamma_1 \, E^T(A + \overline{\tau_1}E)^{-T}C^T,$$

and for $j > 1$

$$\begin{aligned}V_j :=& \gamma_j \left(I_n - (\tau_j + \overline{\tau_{j-1}})(A_s^T + \overline{\tau_j} I_n)^{-1}\right) V_{j-1} \\ &= \gamma_j \left(I_n - (\tau_j + \overline{\tau_{j-1}})((E^{-1}A)^T + \overline{\tau_j} I_n)^{-1}\right) V_{j-1} \\ &= \gamma_j \left(V_{j-1} - (\tau_j + \overline{\tau_{j-1}})E^T(A + \overline{\tau_j}E)^{-T}V_{j-1}\right).\end{aligned}$$

Similarly to the previous case, during the iteration we construct a sequence $\{Z_j\}_{j=1}^{\infty}$ using

$$\begin{aligned}Z_1 &:= V_1, \\ Z_j &:= [Z_{j-1},\ V_j], \quad j > 1,\end{aligned}$$

so that, on convergence after l_o iterations, we obtain a matrix $Z_{l_o} \in \mathbb{R}^{n\times(p\cdot l_o)}$ such that $Z_{l_o}Z_{l_o}^T \approx R^TR = W_o$. Note that we do not compute an approximation to the solution of (5), but we go for R directly without the detour to solve

(5) for a solution factor $\hat{R}$. Working with the latter and noting that $\hat{R}E = R$, in (6) we would need to compute the SVD of $SE^T\hat{R}^T$. On the other hand, in (8) we could work with $T_L := \Sigma_L^{-1/2} V_L^T \hat{R}$ and thus save the inversion with E there. The question here is which version is to be preferred. We chose to work with R instead of $\hat{R}$ as this delays, as long as possible, the application of E^{-1} and rounding errors introduced by a possibly ill-conditioned E. In both cases, a (Cholesky or LU) factorization of E has to be computed. But if $\hat{R}$ were computed, we would have to use this factorization to solve $p \cdot l_o$ linear systems of equations while in (8), only $r \leq p \cdot l_o$ linear systems have to be solved! As usually, $r \ll p \cdot l_o$, this can yield a significant savings of computational time.

Now, consider that both $l_c, l_o > t_s$. Then, provided sufficient storage space is available, a significative computational cost can be saved by using direct methods to solve the linear systems (of equations) that appear in the previous iterations. In particular, we only need to factorize matrices $(A + \tau_j E)$ and $(A^T + \overline{\tau_j} E^T)$ once, but we can then use the same factors in iterations $j + kt_s$, $k = 0, 1, \ldots$. Actually, only one factorization is needed here as the LU factorization (with row pivoting) $(A + \tau_j E) = PLU$ yields a factorization of $(A^T + \overline{\tau_j} E^T)$ as well:

$$A^T + \overline{\tau_j} E^T = \overline{(A + \tau_j E)^T} = \overline{U}^T \overline{L}^T P.$$

Moreover, the shifts are always chosen to yield a set that is closed under complex conjugation. Let $\tau_i = \overline{\tau_j}$; then iterations j and i involve linear systems with coefficient matrices that are complex conjugates of each other. Hence, only one factorization is required to perform these two iterations. In summary, if $\{\tau_1, \ldots, \tau_{t_s}\}$ is composed of t_s^r real and t_s^c complex shifts, then we need to compute (and store) t_s^r factorizations with real arithmetic and $t_s^c/2$ factorizations with complex arithmetic.

Current direct methods for the solution of unstructured sparse linear systems are usually composed of three stages: analysis, factorization, and resolution, with a significative portion of the time spent in the first one [1,14]. As all matrices $(A + \tau_j E)$ and $(A^T + \overline{\tau_j} E^T)$, $j = 1, 2 \ldots, t_s$, share the same sparsity pattern, a major part of the analysis stage needs to be performed only once.

The previous discussion justifies that we prefer direct methods for the solution of sparse linear systems over iterative ones [1]. Nevertheless the use of direct methods is not compulsory at all. As a matter of fact, some sparse systems cannot be solved using direct methods because of the fill-in that occurs during the factorization. In such case, we can still rely on iterative methods.

The LR-ADI iterations are also easily adapted to exploit a banded structure in the state matrix pencils. In this case, the fill-in during the factorization is bounded so it is unlikely that iterative methods are needed.

2.2.1 Symmetric-definite state-matrix pencils

In case both A and E are symmetric, and one of them is definite, the corresponding matrix pencil only has real eigenvalues and is said to be symmetric-definite. In fact, because of the stability of $A - \lambda E$, if A is negative definite then E must be positive definite (and vice versa). In such case all shifts must be chosen to be real, and the iterations for U_j and V_j boil down to

$$\begin{aligned} U_1 &:= \gamma_1 (A + \tau_1 E)^{-1} B, \\ U_j &:= \gamma_j \left(U_{j-1} - (\tau_j + \tau_{j-1})(A + \tau_j E)^{-1} E U_{j-1} \right), \quad j > 1, \end{aligned}$$

and

$$\begin{aligned} V_1 &:= \gamma_1 E (A + \tau_1 E)^{-1} C^T, \\ V_j &:= \gamma_j \left(V_{j-1} - (\tau_j + \tau_{j-1}) E (A + \tau_j E)^{-1} V_{j-1} \right), \quad j > 1, \end{aligned}$$

where $\gamma_j = \sqrt{\frac{\tau_j}{\tau_{j-1}}}$.

As $\tau_j < 0$ and $A - \lambda E$ is symmetric-definite stable, the matrix $-(A + \tau_j E)^{-1}$ is symmetric positive definite and the linear systems that appear in the iteration can be solved via a (sparse) Cholesky factorization.

2.2.2 Selection of the shifts parameters

The performance of the previous iterations strongly depends on the selection of the shift parameters.

For A or E unsymmetric, we propose to employ a modification of the heuristic proposed in [24] for standard LTI systems. This procedure delivers approximations for the generalized eigenvalues of $A - \lambda E$ and $(A - \lambda E)^{-1}$ of largest magnitude using the shift-and-invert Arnoldi iteration [15]. In such a case it is important that the set of selected shifts is closed under complex conjugation.

In case $A - \lambda E$ is symmetric definite, we utilize a procedure to compute the optimal shifts which requires estimators of both the largest and smallest magnitude generalized eigenvalue of the pencil. Efficient codes to compute these approximations are based on generalizations of the Lanczos iteration. The shifts are then computed from these estimators with a negligible cost [8].

For further details on the convergence of the LR-ADI iteration and the properties of the heuristic selection procedure, see [24,35].

2.2.3 Convergence criteria

The LR-ADI iterations present (at best) a superlinear convergence. A practical stopping criterion is to halt the iteration when the *contribution of the norm* of the columns that are added to the solution is relatively "small". This is equivalent, e.g., to stop the computation of the sequences when

$$\|U_j\|_F < \varepsilon \cdot \gamma \cdot \|Y_j\|_F \quad \text{and} \quad \|V_j\|_F < \varepsilon \cdot \gamma \cdot \|Z_j\|_F, \tag{11}$$

where ε denotes the machine precision and γ is a tolerance threshold for the iteration; in our experiments we set $\gamma := 100 \cdot n$.

A different approach requires computation of the relative *residuals* for the approximations Y_j and Z_j to the solutions of the Lyapunov equations. The idea here is to stop when the residuals

$$\mathcal{R}_{W_c}(Y_j) := \frac{\|A(Y_jY_j^T)E^T + E(Y_jY_j^T)A^T + BB^T\|_F}{2\|A\|_F\|E\|_F\|Y_jY_j^T\|_F + \|BB^T\|_F}, \tag{12}$$

$$\mathcal{R}_{W_o}(Z_j) := \frac{\|A^TE^{-T}(Z_jZ_j^T) + (Z_jZ_j^T)E^{-1}A + C^TC\|_F}{2\|A\|_F\|E\|_F\|E^{-T}Z_jZ_j^TE^{-1}\|_F + \|C^TC\|_F}, \tag{13}$$

are smaller than a given tolerance threshold.

In practice, neither of these criteria (nor their combination) is completely satisfactory and a "trial-and-error" approach together with careful monitoring of the iteration is necessary.

As the cost of computing the previous convergence criteria can be quite large (specially for the one based on the residuals), we propose to reduce it by using the following techniques:

1.) The norms of $\|Y_j\|_F$ and $\|Z_j\|_F$ can be computed incrementally. For example, as $Y_j = [Y_{j-1},\ U_j]$, by construction $\|Y_j\|_F = \sqrt{\|Y_{j-1}\|_F^2 + \|U_j\|_F^2}$ so that only the norm of the $n \times m$ matrix U_j is required at each iteration step.

2.) In several norm computations, we can avoid forming full dense $n \times n$ matrices by using $\|M^TM\|_F = \|MM^T\|_F$. This is the case, e.g., when computing $\|Y_jY_j^T\|_F$ or $\|BB^T\|_F$ and implies a significant reduction both in storage and computational costs.

3.) We can also generalize the approach for computing the residual norms in [26] as follows. Assume E is invertible; then $Y_jY_j^T$ is an approximation of the solution of

$$A(Y_jY_j^T)E^T + E(Y_jY_j^T)A^T + BB^T = 0.$$

Let

$$M_c^j = [AY_j,\ EY_j,\ B] = Q_c^jR_c^j = Q_c^j\,[R_1,\ R_2,\ R_3] \tag{14}$$

be a (skinny) QR factorization of M_c^j, where $Q_c^j \in \mathbb{R}^{n\times r_q}$ has $r_q = 2(j+1)m$ orthonormal columns, and $R_c^j \in \mathbb{R}^{r_q\times r_q}$ is an upper triangular matrix partitioned into blocks R_1, R_2, and R_3 of $j \cdot m$, $j \cdot m$, and m columns, respectively. Then, we can use that

$$\|A(Y_jY_j^T)E^T + E(Y_jY_j^T)A^T + BB^T\|_F$$

$$= \| \, [AY_j, \; EY_j, \; B] \, [EY_j, \; AY_j, \; B]^T \, \|_F = \|M_c^j \begin{bmatrix} 0 & I_{r_q} & 0 \\ I_{r_q} & 0 & 0 \\ 0 & 0 & I_m \end{bmatrix} (M_c^j)^T \|_F$$

$$= \|(M_c^j)^T M_c^j\|_F = \|(R_c^j)^T R_c^j\|_F = \|R_1^T R_1 + R_2^T R_2 + R_3^T R_3\|_F.$$

As the orthogonal factor need not be computed, the cost for this procedure is $2(nr_q - \frac{1}{3}r_q^2 + (2j^2+1)m^2)r_q$ floating-point arithmetic operations (flops) and thus increases in each step. Thus, if many iterations are required, the cost for evaluating the residual dominates the cost of the iteration even if no full $n \times n$ matrices need be formed.

The idea is analogously derived for the residual (12).

4.) Instead of evaluating the residual convergence criteria at each iteration, we can do it periodically every few iterations.

5.) The iterations for Y_j and Z_j can be dealt with independently so that one of them can be stopped earlier.

2.2.4 Keeping the storage needs within limits

The LR-ADI iterations add m and p columns per step to the approximation factors Y_j and Z_j. As these are dense matrices (with possibly complex entries), the storage needs of the algorithm increase by $n(m+p)$ entries per iteration. In order to keep the required workspace within reasonable limits, we can compress the matrices Y_j and Z_j during the iteration. Specifically, it is possible to reduce the number of columns in the factors by computing a rank-revealing QR (RRQR) factorizations [15]. For this purpose, we proceed to compute the RRQR factorization

$$Y_j^T = Q_s R_s \Pi_s, \quad r_s := \operatorname{rank}(Y_j) = \operatorname{rank}(R_s),$$

where $Q_s \in \mathbb{R}^{n \times r_s}$ has orthonormal columns, $R_s \in \mathbb{R}^{r_s \times r_s}$ is an upper triangular matrix, and Π_s is a square permutation matrix of order r_s. Then, we can substitute Y_j by its full-rank factor

$$Y_j := R_s \Pi_s^T.$$

A similar strategy delivers a compressed matrix for Z_j. Whether the savings in storage introduced by this column compression technique are worth the cost of the procedure depends on the rank of the computed factors. Also, the procedure for evaluating the residuals (12) and (13) greatly benefits from a reduction in the number of columns of Y_j, Z_j.

Traditionally the QR factorization with column pivoting [15] is employed for rank-revealing purposes because of its low computational cost and high reliability. (Although the SVD usually provides better accuracy, it does so at the expense of a considerable higher cost. An SVD-based compression technique

is discussed in [17].) In [27] a BLAS-3 variant is proposed for this algorithm which maintains the same numerical behavior.

2.2.5 Computational savings using the LR-ADI solvers

It should be emphasized that the methods just described for solving (4)–(5) and computing (6) significantly differ from standard methods used in the MATLAB toolboxes or SLICOT [34]. First, the proposed LR-ADI iteration for the solution of the coupled Lyapunov equation has the potential to exploit the sparsity of the coefficient matrices A and E. Besides, as we are using low-rank approximations to the full-rank or Cholesky factors, the computation of the SVD in (6) is usually less expensive: instead of a computational cost of $\mathcal{O}(n^3)$ when using the Cholesky factors, this approach leads to an $\mathcal{O}(l_c m \cdot l_o p \cdot n)$ cost where, in model reduction, often $\max\{m, p\} \ll n$; see [10]. If column compression is applied, the cost of the SVD can be even more reduced. However then we need to pay for the cost of the column compression procedure.

The reduction of the cost of the SVD is an advantage shared by the routines in our dense parallel model reduction library PLiCMR [10,3]. However, the routines in PLiCMR cannot exploit the sparsity of the coefficient matrices of the Lyapunov equation.

2.3 Computational cost

For simplicity, we make the following assumptions in the evaluation of the computational cost of a serial implementation of the SR-BT model reduction algorithm:

- Matrices A and E are unsymmetric. We refer to the number of nonzeros in these matrices as $\bar{z}(A)$ and $\bar{z}(E)$, respectively.
- In case one of the state matrices $M \in \{A, E\}$ presents a banded structure, we denote by $k_u(M)$ and $k_l(M)$ the dimensions of its upper and lower bandwidths, respectively.
- All the shifts $\{\tau_1, \ldots, \tau_{t_s}\}$ are considered to be real.
- Only computational costs, measured in flops, are considered. Minor order terms are neglected in the expressions.

With the previous premises, the costs of a few basic sparse linear algebra operations are introduced in Table 1.

The algorithm for model reduction is composed of four main stages: computation of shifts, LR-ADI iteration, SVD (of SR^T), and application of the SR formulae. Table 2 reports the operations involved in these stages together with approximations of their computational costs. Only the cost for $\mathcal{R}_{W_c}(Y_j)$ is given, the cost for $\mathcal{R}_{W_o}(Z_j)$ being analogous.

Operation	Notation	Structure	
		Sparse	Banded
Matrix-vector product with matrix M	$C_{MV}(M)$	$2\bar{z}(M)$	$2n(k_u(M)+k_l(M)+1)$
Factor $M \rightarrow L_M U_M$; general matrix M	$C_{LU}(M)$	Unknown a priori	$2n\,k_u(M)\,kl(M)$
Solve from $L_M U_M$ single right-hand side	$C_{SV}(M)$	$2(\bar{z}(L_M)$ $+\bar{z}(U_M))$	$2n(k_u(L_M)+k_l(L_U)$ $+k_u(U_M)+k_l(U_M))$

Table 1
Computational cost of sparse linear algebra operations appearing in the model reduction algorithm.

3 Moving the Frontier Further: Parallel Implementation

Parallelism can be exploited in the SR-BT algorithm at two different levels. In our "fine-grain" variant, parallelism is exploited at the level of linear algebra kernels. The idea here is that all "computational resources" (CRs), usually processes or threads, cooperate in the solution of each one of the linear algebra operations in the SR-BT algorithm: factorizations, triangular solves, matrix-vector products, etc. On the other hand, in most stages of the algorithm, different linear algebra operations can be executed concurrently using different CRs, which results in a new opportunity to exploit parallelism at a higher, "coarse-grain" level. A simple example are the two iterations for the computations of the shifts, which are completely independent, and can be computed in parallel using two separate CRs. Both approaches can be combined in a two-level (hybrid) parallel SR-BT algorithm.

While the coarse-grain approach only employs serial implementations of linear algebra operations, the fine-grain approach requires the use of parallel codes for these operations. Parallel routines for the solution of sparse linear systems are available in packages such as MUMPS and SuperLU [1,14]. A parallel implementation of the sparse matrix-vector product Mx is easily obtained by computing the operation row-wise, as a series of dot (inner) products among rows of the matrix and vector x, and distributing the task so that each CR performs some of the dot products. Provided x is accessible to all CRs, the operation can be performed fully in parallel. The efficiency of the procedure depends on how the computational load is balanced among the CRs which, in turn, is dependent on the sparsity pattern of M and the specific mapping of dot products to CRs. Finally, two other major linear algebra operations in the SR-BT algorithm are the dense matrix product and the SVD. Parallel codes for these two operations are provided in the libraries ScaLAPACK and PLAPACK [12,32]. Note that the SVD routine in ScaLAPACK is used despite the fact that we are computing only a small size SVD here: the reason is that the

Phase	Procedure	Cost (flops)
Comp. shift	Factor A, E	$C_{LU}(A) + C_{LU}(E)$
	Solve with single right-hand side	$(C_{SV}(A) + C_{SV}(E))\, t_s/2$
	MatVec with A, E	$(C_{MV}(A) + C_{MV}(E))\, t_s/2$
LR-ADI. Iter.:		
$j = 1$	Factor $(A + \tau_1 E)$	$C_{LU}(A + \tau_1 E)$
	Solve with right-hand side B, C^T	$C_{SV}(A + \tau_1 E)\,(m + p)$
	MatVec with E	$C_{MV}(E)\, p$
$1 < j \leq t_s$	Factor $(A + \tau_j E)$	$C_{LU}(A + \tau_j E)$
	Solve with right-hand side U_j, V_j	$C_{SV}(A + \tau_j E)\,(m + p)$
	MatVec with E	$C_{MV}(E)\,(m + p)$
$t_s < j$	Solve with right-hand side U_j, V_j	$C_{SV}(A + \tau_j E)\,(m + p)$
	MatVec with E	$C_{MV}(E)\,(m + p)$
All j	$\|U_j\|_F, \|V_j\|_F$	$2n(m + p)$
$\mathcal{R}_{W_c}(Y_j)$	Incremental $AY_j = A[Y_{j-1}, U_j]$	$C_{MV}(A)\ m$
	Incremental $EY_j = E[Y_{j-1}, U_j]$	$C_{MV}(E)\ m$
	QR factorization of M_c^j	$2(n - r_q/3)r_q^2$
	$\|R_1^T R_1 + R_2^T R_2 + R_3^T R_3\|_F$	$2m^2(2j^2 + 1)r_q$
SVD	MatMat $Y_{l_c}^T Z_{l_o} \approx SR^T$	$2l_c\, l_o\, m\, p\, n$
	$(Y_{l_c}^T Z_{l_o}) = U\Sigma V^T$	$4l_c^2 l_o\, m^2 p + 22 l_o^3 p^3$
SR formulae	$T_L := \Sigma_L^{-1/2}(V_L^T Z_{l_o}^T)E^{-1}$	$2l_o^2\, n\, p^2 r + C_{SV}(E)\ r$
	$T_R := Y_{l_c}^T U_1 \Sigma_L^{-1/2}$	$2l_c\, m\, n\, r$
	$A_r := T_L A T_R$	$2C_{MV}(A)\, r$
	$B_r := T_L B$	$2m\, n\, r$
	$C_r := C T_R$	$2n\, p\, r$

Table 2
Operations during the different stages of the SR-BT model reduction algorithm in SpaRed. MatMat and MatVec stand for the matrix-matrix and matrix-vector products, respectively.

data required by the SVD is distributed due to the preceding computations. We next describe how to parallelize each one of the stages in the SR-BT model reduction algorithm. We consider that the premises made for the presentation of the computational cost in the previous section still hold.

We will use the following notation hereafter: given a sparse matrix M, $T_{MV}(M)$,

$T_{LU}(M)$, and $T_{SV}(M)$ stand, respectively, for the (sequential) times of computing a matrix-vector product involving M, factorizing the matrix, and solving the corresponding triangular linear systems with a single RHS. These times are directly proportional to the costs reflected in Table 1. (More specifically, $T_{LU}(M)$ and $T_{SV}(M)$ depend on the platform, the sparse solver that is employed, and the structure of the coefficient matrix of the linear system.) Given dense matrices M_1 and M_2, $T_{MM}(M_1M_2)$ will be used for the (sequential) time to compute the dense matrix-matrix product M_1M_2, while $T_{SD}(M_1)$ will denote the (sequential) time for computing the SVD of M_1. When the routine is executed in parallel, we will denote its time using a "P" superscript; thus, e.g., $T^P_{MV}(M)$ is the time to compute a (sparse) matrix-vector product with coefficient matrix M using a certain parallel code. The superscripts "CP" and "FP" will denote coarse-grain and fine grain parallelism, respectively.

3.1 Computation of shifts

The two iterations in this stage require sparse matrix-vector products with AE^{-1} and EA^{-1}, where the inversions are performed as two triangular solves each given factorizations of A and E. The coefficient matrices only need to be factorized once, and the triangular factors can be used as many times as needed during the iteration. However, there is a strict dependence between consecutive steps so that iteration $j+1$ requires the results from iteration j. The minimum execution time for the fine-grain parallelization of this stage is

$$\begin{aligned} T^{FP}_{SHC} = & T^P_{LU}(A) + T^P_{LU}(E) \\ & + (T^P_{SV}(A) + T^P_{SV}(E) + T^P_{MV}(A) + T^P_{MV}(E))\, t_s/2, \end{aligned}$$

which corresponds to having computed all operations in the stage, one by one, in parallel.
The coarse-grain approach to parallelizing the computation of shifts employs two CRs to compute the two iterations independently. This results in the coarse-grain parallel execution time

$$\begin{aligned} T^{CP}_{SHC} = \max(& T_{LU}(A) + (T_{SV}(A) + T_{MV}(E))\, t_s/2, \\ & T_{LU}(E) + (T_{SV}(E) + T_{MV}(A))\, t_s/2). \end{aligned}$$

Clearly, the maximum speed-up in this case is 2.

3.2 LR-ADI iteration

In order to better illustrate this stage, we will consider the following simplifications:

- The first one performs the same operations as all the subsequent ones.
- Both sequences perform the same number of iterations, $l_c = l_o$.

- The number of iterations is larger than the number of shifts, $l_c > t_s$.
- The number of inputs equals the number of outputs of the system, $m = p$.
- We will obviate the computation of the Frobenius norm for the sequences $\{U_j\}_{j=1}^{\infty}$, $\{V_j\}_{j=1}^{\infty}$, as well as the procedure to compute the residual norms associated with the convergence criterion.

The fine-grain variant of this stage results in the parallel execution time

$$T_{ADI}^{FP} = T_{LU}^{P}(F_j)\ t_s + \left(T_{SV}^{P}(F_j) + T_{MV}^{P}(E)\right)\ l_c\,(m+p),$$

where $F_j = A + \tau_j E$.

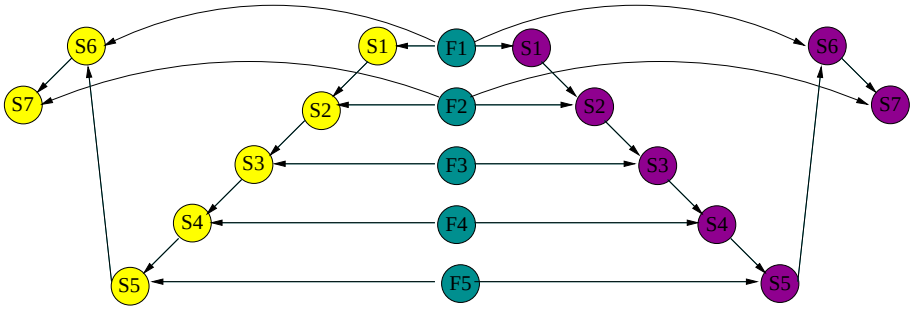


Fig. 1. Data dependencies graph among factorizations ($F1, F2, \ldots$) and triangular solves/matrix-vector products (two sequences $S1, S2, \ldots$, for $\{U_j\}_{j=1}^{\infty}$ and $\{V_j\}_{j=1}^{\infty}$) with t_s=5 shifts and l_c=7 iterations.

The coarse-grain variant of the LR-ADI iteration is more elaborated due to the dependency between the j-th factorization and the solutions of the triangular linear systems/matrix-vector products in iterations j, $j+t_s$, $j+2t_s$, ... (see Figure 1). As all factorizations are independent, we can compute them in parallel using n_p CRs. On the other hand, the triangular solves/matrix-vector products for each one of the sequences need to be computed sequentially, as the result of iteration j is needed during iteration $j+1$. Therefore, we can overlap factorizations, factorizations and triangular solves/matrix-vector products, but not triangular solves/matrix-vector products. We have devised two different ways of exploiting the parallelism that will referred to as heterogeneous and homogeneous. In the heterogeneous scheme, $n_p - 2$ "producers" (CRs) are in charge of computing the factorizations while two "consumers", one per iteration sequence, is dedicated to the solution of triangular linear systems and computing the matrix-vector products. In the homogenous scheme, all n_p CRs perform analogous tasks, combining factorizations and solutions/matrix-vector products. Figure 2 illustrates the functional differences between the two schemes.

In both these schemes, at best, we can expect the time for all the factorizations, except the first one, to be overlapped with the triangular solves/matrix-vector products. Now, as the two iterations are themselves independent, we could

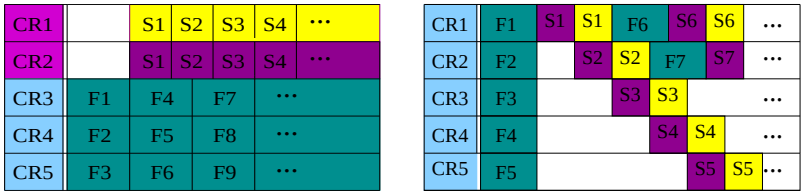


Fig. 2. Two different schemes to organize the computations ($F1, F2, \ldots$ for the factorizations, and $S1, S2, \ldots$ for the triangular solves/matrix-vector products) in the coarse-grain approach, heterogeneous (left) and homogeneous (right); the number of CRs is n_p=5 in both cases.

theoretically obtain a parallel execution time for this coarse-grain approach

$$T_{ADI}^{CP} = T_{LU}(F_1) + (T_{SV}(F_j) + T_{MV}(E))\ l_c\ \max(m, p). \tag{15}$$

However, due to the need to synchronize factorizations and solutions, the execution time of the heterogeneous and homogeneous coarse-grain parallel schemes will, in general, be higher, as is analyzed next.
Consider the left-hand side plot in Figure 3, corresponding to the heterogenous scheme. The gap δ_{HT}, which exposes a deviation from the optimum due to the factorizations not being computed fast enough, is given by

$$\delta_{HT} = \max\left(0,\ T_{LU}(F_j) - (T_{SV}(F_j) + T_{MV}(E))\ m\,(n_p - 2)\right). \tag{16}$$

This overhead is incurred till the last factorization is computed, at $\lceil \frac{t_s - n_p - 2}{n_p - 2} \rceil$, after which the remaining triangular solves/matrix-vector products are computed sequentially. Thus, the actual time for the heterogeneous scheme is given by

$$T_{ADI}^{HTCP} = T_{ADI}^{CP} + \delta_{HT} \left\lceil \frac{t_s - n_p - 2}{n_p - 2} \right\rceil. \tag{17}$$

The analysis for the homogenous scheme is similar. The "gap" in the right-hand side plot in Figure 3, δ_{HM}, is now given by

$$\delta_{HM} = \max\left(0,\ T_{LU}(F_j) - (T_{SV}(F_j) + T_{MV}(E))\ m\,(n_p - 1)\right), \tag{18}$$

which is now due to the solutions being delayed too long. This overhead is incurred till the last factorization is computed, resulting in a time for the homogenous scheme

$$T_{ADI}^{HMCP} = T_{ADI}^{CP} + \delta_{HM} \left\lceil \frac{t_s - n_p}{n_p} \right\rceil. \tag{19}$$

The two analyses result in similar expressions which converge asymptotically as n_p is increased. The main difference is encountered for very small values

of n_p (2 or 3), where the homogenous scheme is in theory better than the heterogenous one. These formulae also yield an expression for the minimum execution times, attained for

$$n_p \geq \lceil \frac{T_{LU}(F_j)}{T_{SV}(F_j) + T_{MV}(E)} \rceil + 2 \quad \text{and} \quad n_p \geq \lceil \frac{T_{LU}(F_j)}{T_{SV}(F_j) + T_{MV}(E)} \rceil + 1 \quad (20)$$

in the heterogeneous and homogeneous schemes, respectively.

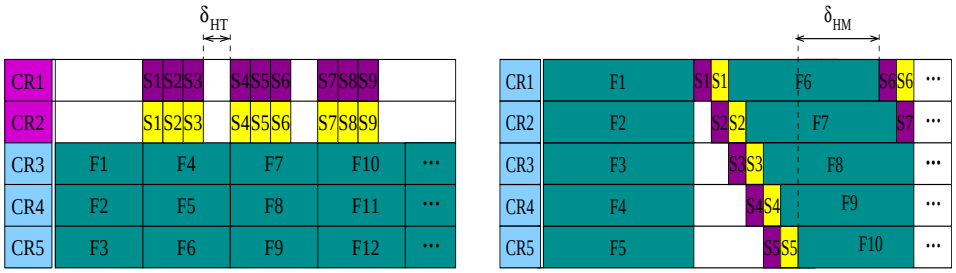


Fig. 3. "Idle" times, or gaps, that result in deviations from the minimum in the two different schemes to organize the computations ($F1, F2, \ldots$ for the factorizations, and $S1, S2, \ldots$ for the triangular solves/matrix-vector products) in the coarse-grain approach, heterogeneous (left) and homogeneous (right); the number of CRs is n_p=5 in both cases.

3.3 SVD

As a result of the LR-ADI iteration we obtain two dense "skinny" matrices, Y_{l_c} and Z_{l_o}, that need to be multiplied together, resulting in a dense, small matrix whose SVD is to be computed next. We can approximate the execution time of a fine-grain parallelization of this stage as

$$T^P_{SVD} = T^P_{MM}\left(Y^T_{l_c} Z_{l_o}\right) + T^P_{SD}\left((Y^T_{l_c} Z_{l_o})\right).$$

There is a strict dependency between these two operations that does not allow a coarse-grain parallelization of the stage. Nevertheless, the bulk of the computations here comes from the product $Y^T_{l_c} Z_{l_o}$, while the cost for the SVD is almost negligible.

3.4 SR formulae

A fine-grain parallelization of this stage yields an execution time

$$\begin{aligned} T^{FP}_{SRF} = & T^P_{MM}\left(V^T_L Z^T_{l_o}\right) + T^P_{SV}(E)\ r + T^P_{MM}\left(Y^T_{l_c} U_L\right) \\ & + T^P_{MV}(A)\ r + T^P_{MM}((T_L A) T_R) + T^P_{MM}(T_L B) + T^P_{MM}(C T_R). \end{aligned}$$

Alternatively, we could calculate T_L and T_R in two different CRs and, once these projectors are available, obtain the matrices of the reduced-order system using three CRs. This yields the parallel execution time

$$T_{SRF}^{CP} = \max\left(T_{MM}\left(V_L^T Z_{l_o}^T\right) + T_{SV}(E)\ r,\ T_{MM}\left(Y_{l_c}^T U_L\right)\right) + \max(T_{MV}(A)\ r + T_{MM}((T_L A)T_R)\,,\ T_{MM}(T_L B)\,,\ T_{MM}(CT_R)).$$

3.5 Overlapping stages

There is still some more parallelism that can be extracted from part of the SR-BT model reduction algorithm. Consider the product $Y_{l_c}^T Z_{l_o}$ needed to compute the SVD. Assume we are just about to start step j in the LR-ADI iteration so that Y_{j-1}/Z_{j-1}, composed of $(j-1)m/(j-1)p$ columns, are known, and consider we have already computed the product $Y_{j-1}^T Z_{j-1}$. Then, during iteration j, U_j/V_j are computed so that $Y_j = [Y_{j-1},\ U_j]$ and $Z_j = [Z_{j-1},\ V_j]$. Therefore, using one additional CR, we can overlap the computations corresponding to step $j+1$ of the LR-ADI iteration with the products $Y_{j-1}^T U_j$, $V_j^T Z_{j-1}$, and $U_j^T V_j$ that are necessary as part of the construction of the overall product $Y_{l_c}^T Z_{l_o}$. Proceeding in this manner, a major bulk of the computation of the product can be overlapped with the LR-ADI iteration. Note that forming this product is the major part of the SVD stage, so that overlapping can here yield a significant reduction in the total computational time.

A second overlap can be attained by rearranging the computation of the projector $T_L := \Sigma_L^{-1/2}(V_L^T Z_{l_o}^T)E^{-1}$ so that the linear system $Z_{l_o}^T E^{-1}$ is solved first, and then the matrix product of V_L^T with this partial result is computed. Although this increases the overall cost of the procedure, the solution of the linear system can be then overlapped with the LR-ADI iteration resulting in a lower execution time.

3.6 Threads vs. processes as CRs

The fine-grain variants require parallel implementations of several linear algebra operations. Thread-level parallelism can be exploited by simply linking in multithreaded implementations of BLAS (e.g., Intel MKL [7] or GotoBLAS [8]), both for sparse and dense linear algebra operations. To exploit process-level parallelism, we can use message-passing packages as, e.g., MUMPS or SuperLU for sparse linear algebra, or ScaLAPACK, the message-passing version of LAPACK, for dense linear algebra.

The coarse-grain variants can be implemented using higher level tools for the shared-memory and the message-passing programming models as OpenMP and MPI. Naturally, thread-level parallelism is only appropriate for shared-

[7] Available from `http://www.intel.org`.

[8] Available from `http://www.tacc.utexas.edu/resources/software`.

memory multiprocessors (SMM). The process-level/message passing combination is possible both in SMM and distributed-memory platforms (multicomputers).
Consider now the coarse-grain heterogeneous and homogeneous schemes for the LR-ADI iteration. The former computes the solutions in two CRs (the consumers) so that factorizations computed at any other CRs need to be streamlined through these. This can result in a high volume of data movement (This is clearly true in multicomputers, but also in shared-memory platforms with a ccNUMA or NUMA organization). In the homogeneous scheme, it is the solutions that are to be passed between "neighbour" CRs. Therefore, we can expect the amount of data movement to be lower in this scheme.
The hybrid two-level approach could be easily implemented by combining processes at the coarse-level and threads at the fine-grain level. However, this implementation is not considered further as thread-safe implementations of the linear algebra libraries are needed and, unfortunately, as to date, MUMPS does not satisfy this property.

4 Numerical Experiments

All the experiments presented in this section were performed on a ccNUMA SGI Altix 350 platform with 8 nodes using IEEE double-precision floating-point arithmetic ($\varepsilon \approx 2.2204\text{e}{-16}$). Each node consists of two Intel Itanium 2 processors@1.5 GHz with 2 GBytes of RAM. We employ a BLAS library specially tuned for this processor that achieves around 5300 Mflops (millions of flops per second) for the matrix product (routine `DGEMM` in MKL 8.0). The nodes are connected via an SGI NUMAlink network and the MPI communication library is specially developed and tuned for this network.
All results were obtained using the SR-BT algorithm in our library SpaRed. We found no significative difference when using the BFSR-BT code for the examples reported in the following.

4.1 Model reduction benchmarks

In the evaluation we employ several examples coming from two different benchmarks: the first one is part of the NICONET project [9], which led to the development of the SLICOT library [10], and includes several small-scale examples for standard realizations ($E = I_n$) with unsymmetric sparse state matrix A [13]. The second benchmark is the result of a recent effort to assemble large-scale examples in the Oberwolfach model reduction collection coordinated at the University of Freiburg [11]. Table 3 and the following brief description feature

[9] `http://www.icm.tu-bs.de/NICONET`
[10] `http://www.slicot.org`
[11] `http://www.imtek.de/simulation/benchmark/`

some of the relevant aspects of the models.

CD player (C_DISC). This example is frequently used as a benchmark for model reduction. The system describes the dynamic behavior of the mechanisms of the swing arm and the lens of a portable Compact Disc player. The goal is to obtain a low-cost controller which is both fast and robust to external shocks.

Random model (RAND). This is a randomly generated single-input, single-output model. There is no real application behind this example.

Hospital building (BUILD_I). This example comes from modeling vibrations of a building at *Los Angeles University Hospital*. The model is discretized with 8 floors and 3 degrees of freedom each, which correspond to the displacements in the x and y directions, and the rotation. A second-order differential equation is thus obtained which is then transformed into an standard continuous-time LTI system.

Clamped beam (BEAM). This system results from the discretization of a hyperbolic PDE modeling a clamped beam with proportional damping, resulting in a second-order system. Here the input contains the force applied to the structure while the output represents the corresponding displacement.

Russian service module ISS (ISS_I). This is a (linearized second-order) system modeling component 1R of the International Space Station (ISS). The challenge is to control a flexible structure in space in real-time so that it becomes necessary to reduce the flex models in order to complete the analysis in a timely manner.

Extended service module ISS (ISS_II). This model corresponds to a second assembly stage of the ISS, also known as module 12A. The goal of model reduction here is similar to that of the previous example.

Optimal cooling of steel profiles (STEEL_I and STEEL_II). This model arises in a manufacturing method for steel profiles [11,31]. The goal is to design a control that achieves moderate temperature gradients when the rail is cooled down. The mathematical model corresponds to the boundary control for a 2-D heat equation. A finite element discretization, followed by adaptive refinement of the mesh result in the two examples in this benchmark.

Micropyros thruster (T3DL). This is a model of a microthruster array based on the co-integration of solid fuel with a silicon micromachined system [21]. The design problem is to reach the critical temperature within the fuel but, at the same time, do not reach the critical temperature at the neighboring microthrusters. The model for the device was constructed and meshed using ANSYS.

Chip cooling model (CHIP). This is a design corresponding to a 3-D model of a chip cooled by forced convection that is used in the thermal simulation of heat exchange between a solid body (the chip) and a fluid flow.

SLICOT benchmark collection					
Example	n	m	p	$\bar{z}(A)$	$\bar{z}(E)$
C_DISC	120	2	2	240	$= \bar{z}(I_n)$
RAND	200	1	1	2,132	$= \bar{z}(I_n)$
BUILD_I	48	1	1	1,176	$= \bar{z}(I_n)$
BEAM	348	1	1	60,726	$= \bar{z}(I_n)$
ISS_I	270	3	3	405	$= \bar{z}(I_n)$
ISS_II	1,412	3	3	2,118	$= \bar{z}(I_n)$
Oberwolfach benchmark collection					
Example	n	m	p	$\bar{z}(A)$	$\bar{z}(E)$
STEEL_I	20,209	7	6	139,233	139,473
STEEL_II	79,841	7	6	553,921	554,913
T3DL	20,360	1	7	265,113	20,360
CHIP	20,082	1	5	281,150	20,082

Table 3
Examples included in the evaluation of the SpaRed model reduction routine.

4.2 Numerical performance of the Lyapunov solver

We first study the convergence rate and the numerical performance of the LR-ADI Lyapunov solvers. Table 4 reports, for each example, the number of shifts used for the iteration, t_s, the number of iterations required for convergence, l_c, the relative contribution of the last columns added to the solution (11), and the residuals (12)–(13) after l_c iterations. As the convergence is strongly dependent on the selection and the number of shifts, repeated executions were performed in order to choose those values reported in the table.

A few comments are worth about the results in the table:

- The iteration did not converge according to any of the criteria for examples C_DISC, BUILD_I, ISS_I, ISS_II, and T3DL. (Due to the scale of problem T3DL, we only allowed 140 iterations for this case.) Although the residuals $\mathcal{R}_{W_c}(Y_{l_c})$ and $\mathcal{R}_{W_o}(Z_{l_o})$ can be considered small for the former two examples, both ISS examples offers poor residuals. Example T3DL also offer unsatisfactory residuals for one of the equations. Notice however that the quality of the solutions should be judged by the soundness of the reduced-order systems, to be evaluated in short.
- The convergence criterion based on the contribution of the columns added to the solution is appropriate for examples BEAM, STEEL_I, STEEL_II, and CHIP.

Example	t_s	l_c	$\frac{\|U_{l_c}\|_F}{\|Y_{l_c}\|_F}$	$\frac{\|V_{l_c}\|_F}{\|Z_{l_o}\|_F}$	$\mathcal{R}_{W_c}(Y_{l_c})$	$\mathcal{R}_{W_o}(Z_{l_o}))$
C_DISC	25	1,000	1.24e−05	1.24e−05	1.36e−11	1.33e−11
RAND	20	40	6.29e−08	6.94e−08	1.47e−16	1.20e−16
BUILD_I	40	1,000	9.87e−06	4.40e−05	7.30e−12	4.29e−11
BEAM	20	40	1.96e−03	5.90e−04	2.06e−08	2.17e−09
ISS_I	20	1,000	1.24e−03	1.41e−03	6.02e−07	2.11e−05
ISS_II	20	1,000	1.17e−02	1.74e−03	9.36e−06	3.35e−06
STEEL_I	25	76	4.25e−10	4.21e−11	2.94e−09	4.89e−20
STEEL_II	26	78	1.63e−10	1.59e−09	1.80e−08	4.54e−21
T3DL	29	140	1.45e−02	1.49e−02	4.04e+00	1.32e−12
CHIP	20	59	1.43e−10	5.00e−11	4.45e−09	1.28e−25

Table 4
Convergence and numerical performance of LR-ADI iteration routine.

- The LR-ADI iteration for the RAND example was stopped after only 40 iterations with residuals as low as one could expect. The convergence criterion based on the residuals, combined with a careful monitoring, served its purpose for this example as we detected that no improvement in the residuals was obtained by iterating further.

4.3 Numerical performance of the SR-BT model reduction algorithm

We next compare the frequency response of the original systems with that of the reduced-order realizations obtained using SpaRed. For the small-scale problems in SLICOT, we include a reliable BT model reduction routine from PLiCMR in the experiments. The kernels in PLiCMR, though parallel, do not preserve nor exploit the sparsity of E or A, and therefore cannot deal with the large examples from the Oberwolfach benchmark.
Both the SpaRed and the PLiCMR routines select the order r automatically so that

$$\sigma_r > \max(\gamma_1, n \cdot \varepsilon \cdot \sigma_1) > \sigma_{r+1},$$

where ε is the machine precision and γ_1 is a user-specified tolerance threshold. In our case, we set $\gamma_1 = \eta \cdot \sigma_1$, with the value η adjusted for each particular example. Obviously, a larger order provides a more accurate model, but also increases the cost of latter computations involving the reduced-order model. For each example, Table 5 shows the values of r and σ_1, and the theoretical error bound Δ_a the reduced-order realization must satisfy (see (10)). As a measure of the quality of the reduced-order systems, in the table we also

report the difference

$$\|G - G_r\|_\infty = \sigma_{\max}(G - G_r);$$

here, the TFMs are evaluated at $\jmath w$, with w composed of 1,000 frequency samples logarithmically distributed in the interval $[f_{\min}, f_{\max}]$, and $f_{\min}$ and $f_{\max}$ chosen specifically for each different example.

Example	r	σ_1	Δ_a	$[f_{\min}, f_{\max}]$	SpaRed $\|G - G_r\|_\infty$	PLiCMR $\|G - G_r\|_\infty$
C_DISC	42	1.17e+6	2.35e−01	[1e−1,1e+5]	1.64e−2	1.64e−2
RAND	7	8.19e+6	1.79e+01	[1e+1,1e+5]	6.29e+0	6.29e+0
BUILD_I	30	2.50e−3	2.69e−05	[1e−1,1e+3]	4.92e−6	4.92e−6
BEAM	12	2.38e+3	1.24e+01	[1e−2,1e+3]	2.98e+0	2.37e+0
ISS_I	36	5.79e−3	1.83e−03	[1e−2,1e+3]	2.03e−3	8.61e−5
ISS_II	66	5.84e−3	4.21e−02	[1e−2,1e+3]	7.06e−3	7.38e−4
STEEL_I	45	2.55e−2	1.47e−04	[1e−2,1e+6]	1.54e−5	–
STEEL_II	60	2.57e−2	4.56e−06	[1e−2,1e+6]	5.29e−6	–
T3DL	30	2.81e+2	2.16e−06	[1e+0,1e+8]	6.74e−4	–
CHIP	40	5.27e+1	6.71e−10	[1e+0,1e+5]	3.35e−10	–

Table 5
Numerical performance of SpaRed model reduction routine.

A close inspection of the results in the table reveals the following:

- All the examples satisfy the theoretical bound, except for the ISS_I and T3DL examples computed using the SpaRed routine. We can impute this on the low quality of the approximations to the solutions of the Gramians produced by the LR-ADI iteration for these particular examples.
- A comparison between the SpaRed and PLiCMR codes reports close results except for the two ISS examples, where PLiCMR performs better.
- It is quite remarkable that models of order 45 and 60 are enough to capture the dynamics of the large systems of the STEEL examples.

The measure $\|G-G_r\|_\infty$ compresses all the information relative to the absolute error of the reduced-order system into a single number. However, in general it is more enlightening to analyze the frequency response dynamics and its error over the relevant frequency spectrum. Figures 4–7 report next the frequency responses

$$|G_{ik}(\jmath\omega)|, \quad 1 \le k \le p, \quad 1 \le i \le m,$$

of the transfer function from input i to output k for the examples in the SLI-

COT benchmark. These figures show the virtues of the reduced-order models computed using SpaRed and PLiCMR over a large portion of the frequency spectrum, and also offer an idea of the complexity of the model reduction task. In particular, those systems with a large number of peaks in a transfer function are expected to present major difficulties for the reduced-order systems.

The plots in Figure 4 report the dynamics of the rotating arm for each input-output pair of the CD player example. Two main difficulties here are the lack of a significative gap in the distribution of the Hankel singular values and the "shaky" dynamics of the transfer functions, in particular for G_{12} and G_{21}, which requires a considerable number of states in the reduced-order model in order to capture all the significant peaks in the frequency response. The results report no visible difference in the reduced-order models computed by PLiCMR or SpaRed.

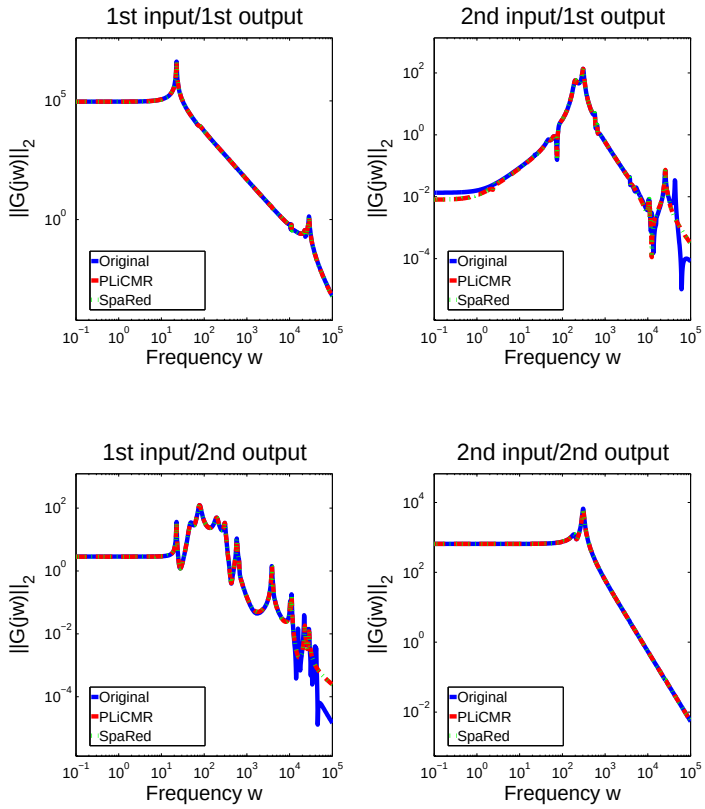


Fig. 4. Frequency responses of the CD player example (C_DISC).

The left-hand side plot in Figure 5, corresponding to the frequency response for the random example, shows that reduced-order models with only 7 states are sufficient to match the dynamical behavior of the original system smoothly, both for PLiCMR and SpaRed.

The plot in the middle of Figure 5 reports the frequency response of the building example. A good approximation is obtained for both reduced-order models, computed using PLiCMR and SpaRed, but the achieved reduction is not quite satisfactory (from 48 to 30 states). The problem here is that

all poles have imaginary parts, resulting in a dynamics which is difficult to approximate (just counting peaks shows that the reduced-order model needs at least 20 states to match all of them).

The right-hand side plot in Figure 5 illustrates the frequency response of the beam example. Though most of the poles of the system are slightly undamped, a good approximation at low frequencies (which are the relevant ones for vibration analysis) is achieved with a quite reduced-order by both methods.

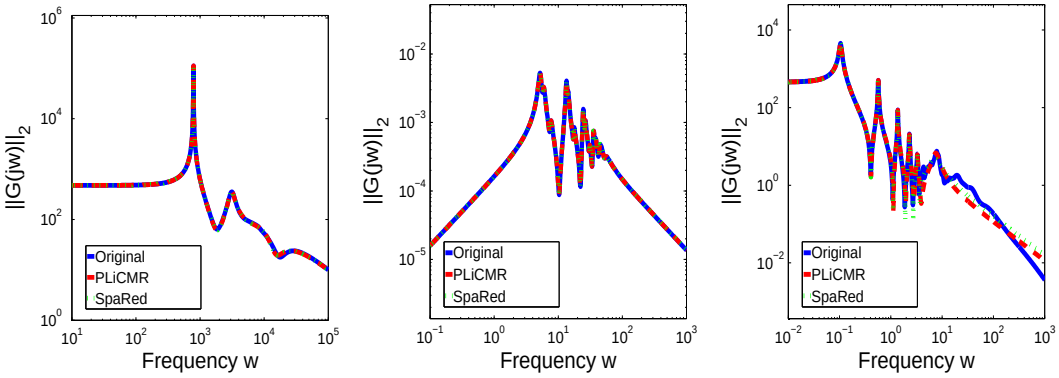


Fig. 5. Frequency responses of the random (left), the building (center), and the beam examples (RAND, BUILD_I, and BEAM respectively).

Figures 6 and 7 report the frequency responses associated with each input/output pair for the ISS models. The large deviations in some of the cases are due to the nature of absolute error methods: as the gain in the corresponding input-output channels is much smaller than, e.g., in the $(1,1)$-component, and hence much smaller than the H_∞-norm of the system, a better approximation cannot be expected as the absolute error is small enough; the plots give a different impression, though.

As in the large-scale examples the number of input/outputs in the system is considerable, in order to limit the number of figures we only present for those the behavior of the error in the frequency response itself; that is, the figures show $\|G - G_r\|_2$.

Figure 8 reports the absolute error in frequency response for both STEEL examples. The figure shows that the absolute error is well below the theoretical bound in both cases.

The plots in Figure 9 presents the behavior of the absolute error for the T3DL and CHIP examples. The large error for the T3DL example at small frequencies comes from a severe ill-conditioning of $A - j\omega E$ for small ω which makes an approximation of $G(j\omega)$ there very difficult and on the other hand leads to huge errors in the evaluation of the frequency response.

4.4 Parallel performance

In this subsection we evaluate the parallelism of the solutions proposed for each one of the stages in the SR-BT algorithm as well as the overall performance,

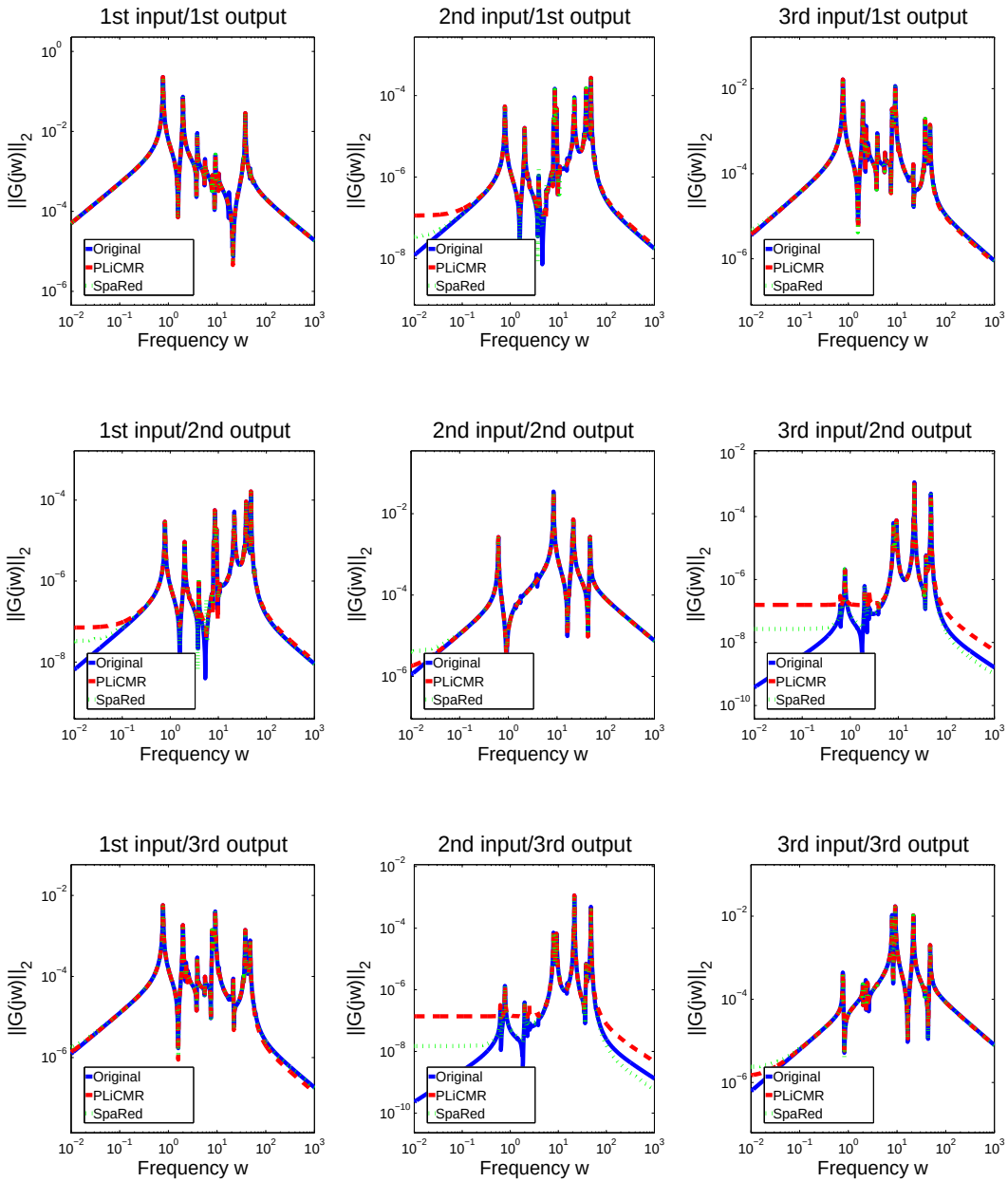


Fig. 6. Frequency responses of the Russian service module example (ISS_I).

using the large-scale examples from the Oberwolfach model reduction benchmark. In order to reduce the number of results, we will only consider a parallel implementation based on message-passing (parallelism using processes), both for the fine-grain and coarse-grain approaches. A different possibility would be to exploit parallelism at thread-level.

4.4.1 Computation of shifts

Table 6 reports the execution times and speed-ups of the parallelizations of this first stage of the algorithm. To note in the table are the speed-ups lower than 1 (decelerations or slowdowns) when the fine-grain parallel algorithm is

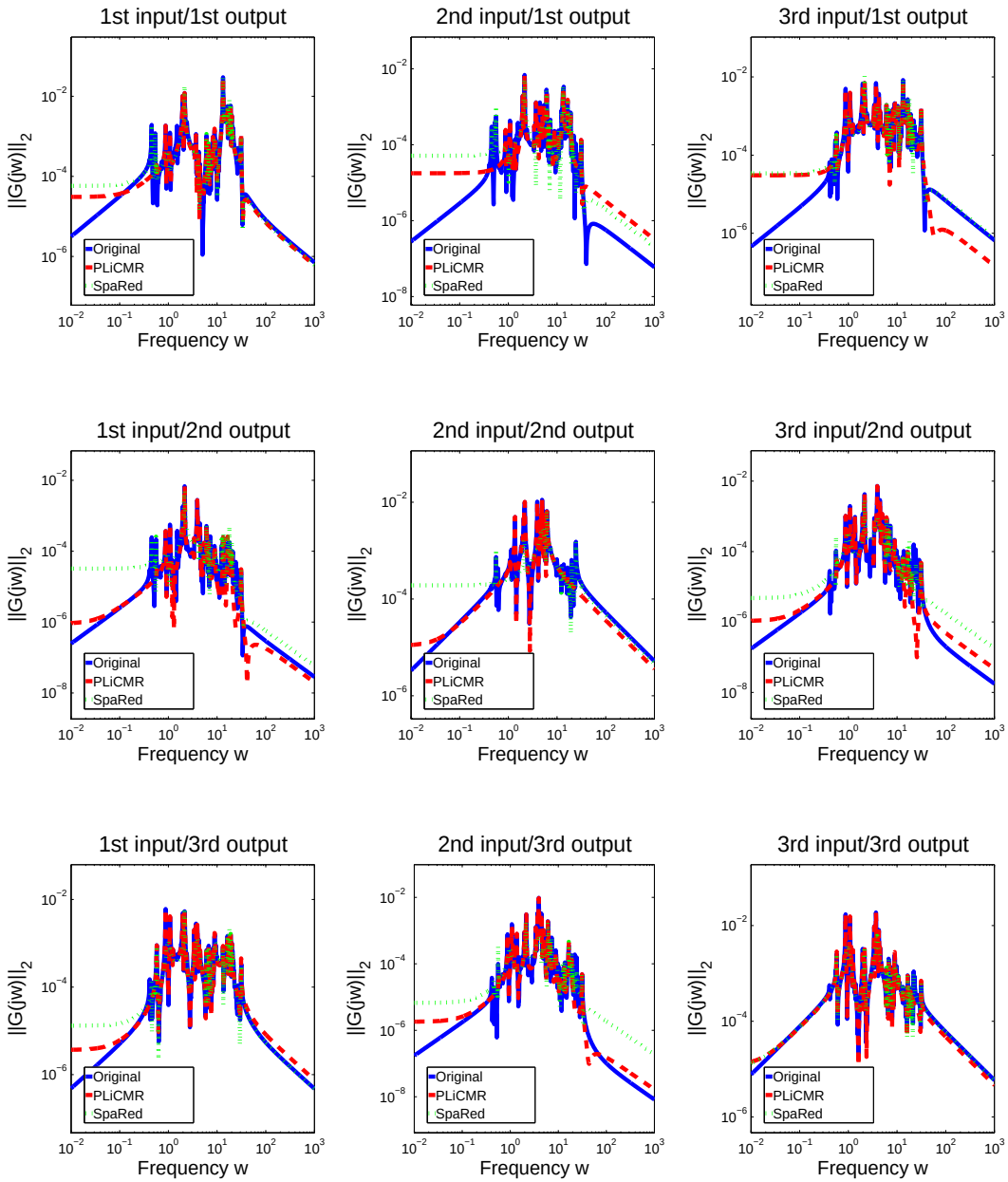


Fig. 7. Frequency responses of the extended service module example (ISS_II).

applied in the three smaller examples: STEEL_I, T3DL, and CHIP. Only for example STEEL_II we obtain a positive speed-up from this approach. Further experiments showed that it is the factorization stage in MUMPS that is responsible for this poor results, while the triangular solves and the matrix-vector products offer a reasonable parallel efficiency.

The coarse-grain approach delivers higher speed-ups on two processors for the three small examples, and similar to those of the fine-grain approach for example STEEL_II. The speed-ups of this second parallel alternative using two processors vary from 1.27 to 1.72 depending on how well balanced are the costs of the two (Arnoldi) iterations.

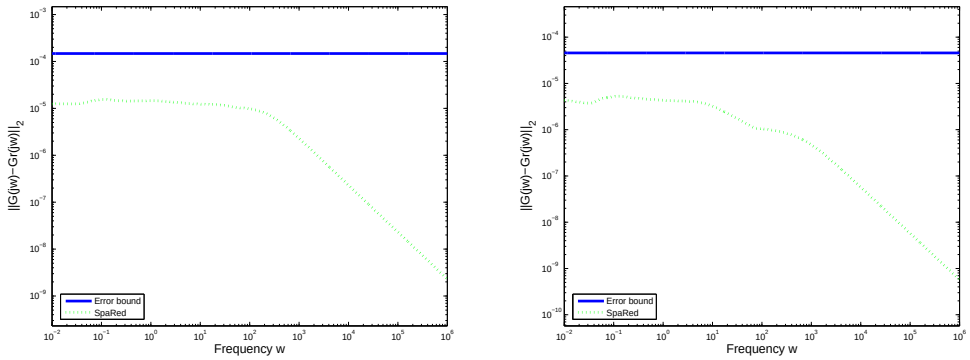


Fig. 8. Absolute error in the frequency responses of the optimal cooling examples STEEL_I (left) and STEEL_II (right).

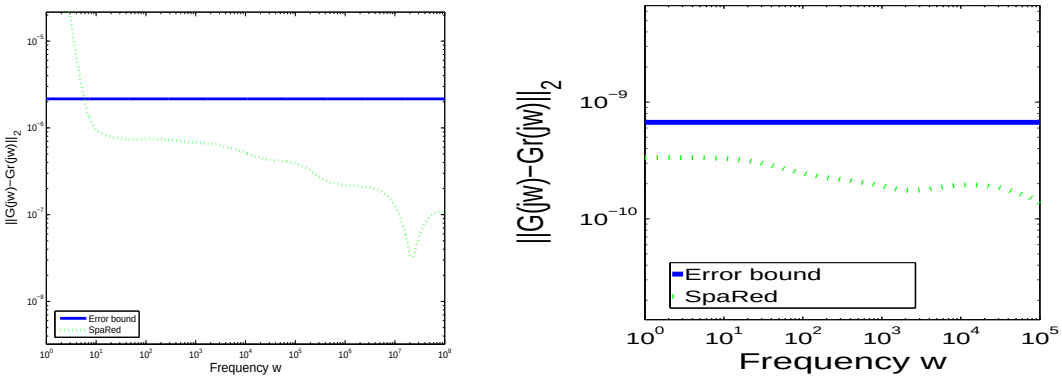


Fig. 9. Absolute error in the frequency responses of the T3DL (left) and CHIP (right) examples.

Example	Sequential	Fine-grain		Coarse-grain
	(1 Proc.)	2 Proc.	4 Proc.	2 Proc.
STEEL_I	5.44	6.98 (0.78)	7.82 (0.69)	3.86 (1.40)
STEEL_II	35.94	28.29 (1.27)	28.30 (1.27)	28.38 (1.27)
T3DL	7.55	9.25 (0.81)	9.84 (0.76)	4.33 (1.74)
CHIP	6.07	7.63 (0.79)	8.13 (0.74)	3.51 (1.72)

Table 6
Execution time (in secs.) and speed-ups (inside parenthesis) for the two parallel approaches, fine-grain and coarse-grain, considered for the computations of shifts.

4.4.2 LR-ADI iteration

The previous experimental analysis showed that very little parallelism is extracted during the factorizations of the state matrices using MUMPS. Our experiments with the LR-ADI iteration reported that the same problem persists for the factorizations of the sequences of matrices $F_j = A + \tau_j E$, $j = 1, 2, \ldots$, in the LR-ADI iteration. Therefore, we will not consider the fine-grain parallel implementation of this stage.

Two different variants were proposed for the coarse-grain approach, corresponding to the CRs playing heterogeneous or homogeneous roles. As we are using processes as CRs with nonshared memory, the heterogenous variant requires the communication of the factorizations between processes while in the homogeneous variant it is the solutions U_j and V_j that are transferred. Now, MUMPS (as well as most other sparse direct solvers) keeps the factorized matrices in internal data structures, hidden from the user, in order to offer an object-oriented interface. As a consequence, transferring the factors becomes too complex and we therefore rely only on the homogeneous approach. (Nevertheless, experiments similar to those presented next, using the band direct solvers in LAPACK, where the data structures for the factors are explicitly available to the user, showed close results.)

Figure 10 compares the theoretical and experimental execution times (see equations (18) and (19)) of the parallel homogeneous coarse-grain approach for the four examples, as the number of processors is increased. In all four examples the theoretical models capture the behavior of the actual parallel codes remarkably well. The expression in the right-hand side of equation (20) also offers the number of processors for which the minimum execution time is obtained: $n_p \approx 3.7$, 2.8, 19.8, and 22.1 for the STEEL_I, STEEL_II, T3DL, and CHIP examples. This roughly corresponds to the moment the curves in the first two plots flatten, while, for the last two examples, not enough processors are available to contrast this result.

Figure 11 reports the acceleration obtained in each example using the homogeneous coarse-grain parallel code. Although the speed-ups are only moderate when the number of processes is large, it is necessary to realize that for these problems, using up to $n_p = 14$ nodes can be excessive. Indeed, the speed-ups are 1.47/1.62 for examples STEEL_I/STEEL_II on 2 processors, and 3.13/3.69 for examples T3DL/CHIP on 5 processors, which can be considered as notable.

4.4.3 SVD

Table 7 lists the execution times and speed-ups of the parallelization of this stage using the only option that is available: the fine-grain approach. The two operations in this stage, the computation of the product $Y_{l_c}^T Z_{l_o}$ and its SVD, operate on dense matrices using parallel kernels from ScaLAPACK. The overall performance of the stage is thus determined by the parallel efficiency of routines `pdgemm` and `pdgesvd` in this library. Again we note that the bulk of the computations here usually comes from the product $Y_{l_c}^T Z_{l_o}$ rather than the small-size SVD.

4.4.4 SR formulae

In subsection 3.5 we discussed two orders in which the computation of T_L could be arranged, as $T_L := \Sigma_L^{-1/2}(V_L^T Z_{l_o}^T)E^{-1}$ or $T_L := \Sigma_L^{-1/2} V_L^T (Z_{l_o}^T E^{-1})$. As we aim at partially overlapping this stage with the LR-ADI iteration, we only investigate the second choice.

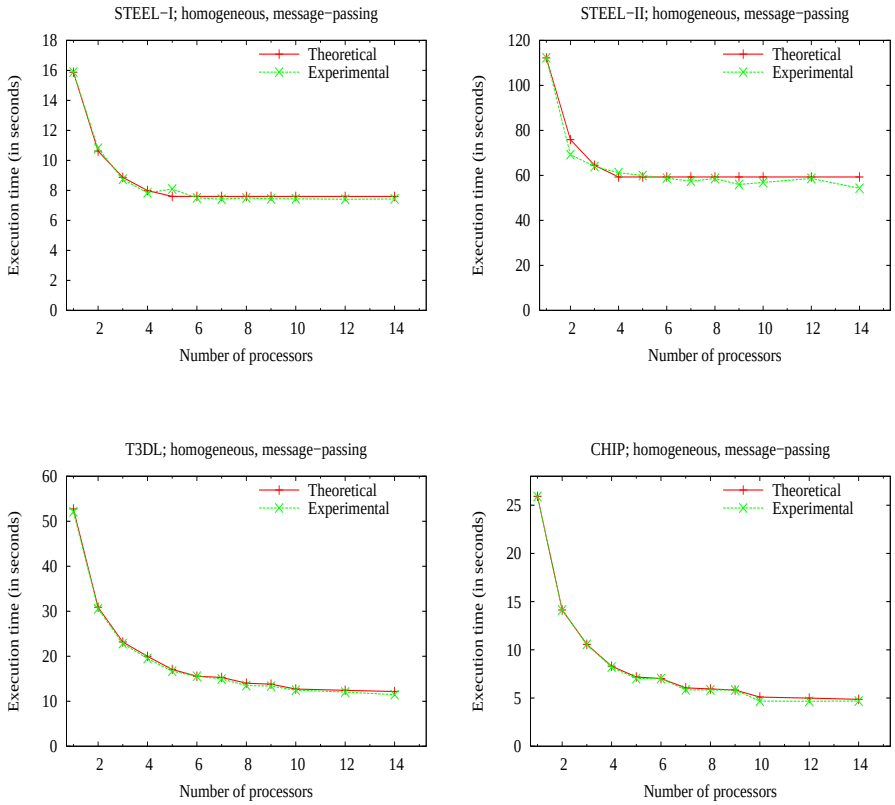


Fig. 10. Execution time (in secs.) the homogeneous variant of the coarse-grain approach considered for the LR-ADI iteration.

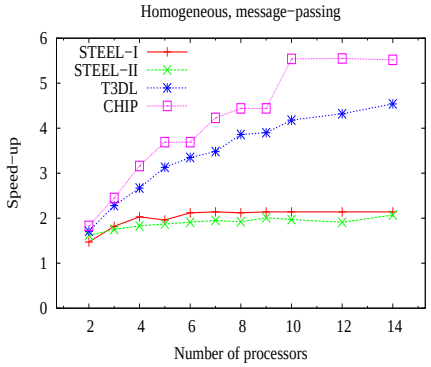


Fig. 11. Speed-ups of the homogeneous variant of the coarse-grain approach considered for the LR-ADI iteration.

Table 8 displays execution times and speed-ups of the fine- and coarse-grain parallelization approaches considered for the application of the SR formulae. Except for the STEEL_II example, none of the approaches achieves a significant reduction in the execution time so that the best option here basically corresponds to a serial execution. A more detailed analysis revealed that the low efficacy of the parallel fine-grain algorithms is due to the poor performance of the sparse linear system solvers during the computation of the system $Z_{l_o}^T E^{-1}$.

Example	Sequential (1 Proc.)	Fine-grain		
		2 Proc.	4 Proc.	6 Proc.
STEEL_I	8.27	5.03 (1.64)	2.31 (3.58)	1.58 (5.23)
STEEL_II	379.27	219.75 (1.72)	109.62 (3.45)	69.09 (5.48)
T3DL	1.64	0.92 (1.78)	0.53 (3.09)	0.37 (4.43)
CHIP	0.22	0.13 (1.69)	0.08 (2.75)	0.06 (3.43)

Table 7
Execution time (in secs.) and speed-ups (inside parenthesis) for the two parallel approaches, fine-grain and coarse-grain, considered for the SVD stage.

For the coarse-grain algorithm, the reason is different. This approach proposed to overlap the computations of T_L and T_R using 2 processes, and then those of A_r, B_r, and C_r using three processes. However, the computations that are performed concurrently are highly unbalanced, with most of the time being spent in the computation of T_L and A_r, so that no benefit results from the parallel execution.

Example	Sequential (1 Proc.)	Fine-grain		Coarse-grain
		2 Proc.	4 Proc.	3 Proc.
STEEL_I	10.05	10.85 (0.92)	11.32 (0.88)	9.86 (1.01)
STEEL_II	90.13	64.73 (1.39)	52.27 (1.72)	79.93 (1.12)
T3DL	14.20	23.96 (0.59)	24.07 (0.58)	14.15 (1.00)
CHIP	6.37	9.06 (0.70)	9.34 (0.68)	6.34 (1.00)

Table 8
Execution time (in secs.) and speed-ups (inside parenthesis) for the two parallel approaches, fine-grain and coarse-grain, considered for the SR formulae.

4.4.5 SR-BT algorithm

Table 9 collects the serial time for the four stages and the total execution time. A parallel algorithm results from a combination of the best parallelization approach for each stage, together with the necessary communication of data between consecutive stages. We selected the coarse-grain approaches for the first two stages; the fine-grain approach for the third stage; and the sequential algorithm in the last stage, except to the STEEL_II example, where the fine-grain approach is employed. The number of processes employed in each stage of the algorithm is given in Table 10; the goal here is to reduce the total execution time significantly by employing all the platform resources. This objective will surely yield a low efficiency in the form of resources that are not fully exploited, but on the other hand produces a solution fast.

The results from the parallel SR-BT algorithm are given in Table 11. When

there is no overlapping of stages, the low parallelism of the first and last stage determine the low speed-up of the algorithm. By overlapping, the influence of the last stage is significantly reduced, so that higher speed-ups are obtained.

Example	Computation of shifts	LR-ADI iteration	SVD	SR formulae	Total
STEEL_I	5.44 (13.7%)	15.89 (40.0%)	8.27 (20.8%)	10.05 (25.3%)	39.65
STEEL_II	35.94 (6.0%)	112.22 (18.8%)	379.27 (63.8%)	90.13 (15.1%)	594.42
T3DL	7.55 (10.0%)	52.10 (69.0%)	1.64 (2.1%)	14.20 (18.8%)	75.49
CHIP	6.07 (15.7%)	25.91 (67.1%)	0.22 (0.05%)	6.37 (16.5%)	38.57

Table 9
Execution time (in secs.) and percentage of time (inside parenthesis) for the four stages executed in the sequential SR-BT algorithm.

	W/out overlapping			
Example	Computation of shifts	LR-ADI iteration	SVD	SR formulae
STEEL_I	2	6	6	1
STEEL_II	2	6	6	4
T3DL	2	10	6	1
CHIP	2	10	6	1
	With overlapping			
Example	Computation of shifts	LR-ADI it. $+Z_{l_o}E^{-1}$ $+Y_{l_c}^T Z_{l_o}$	Remainder SVD	Remainder SR formulae
STEEL_I	2	6+1+2	1	1
STEEL_II	2	6+4+6	1	4
T3DL	2	10+1+4	1	1
CHIP	2	10+1+4	1	1

Table 10
Number of processes employed in each stage of the parallel SR-BT algorithm without and with overlapping. In the second case, the figures in the column corresponding to the LR-ADI iteration correspond to the numbers of processors used in each one of the overlapped processes. Thus, e.g., for the STEEL_I example, 6, 1, and 2 processors are employed, respectively, for the LR-ADI iteration, and the computations of $Z_{l_o}E^{-1}$ and $Y_{l_c}^T Z_{l_o}$.

	W/out overlapping				
Example	Computation of shifts	LR-ADI iteration	SVD	SR formulae	Total (Speed-up)
STEEL_I	3.86	7.50	1.58	10.05	22.99 (1.72)
STEEL_II	28.38	58.76	69.09	52.27	208.50 (2.85)
T3DL	4.33	12.74	0.37	14.20	31.64 (2.38)
CHIP	3.51	4.68	0.08	6.37	14.64 (2.63)
	With overlapping				
Example	Computation of shifts	LR-ADI it. $+Z_{l_o}E^{-1}$ $+Y_{l_c}^T Z_{l_o}$	Remainder SVD	Remainder SR formulae	Total (Speed-up)
STEEL_I	3.86	7.50	0.48	4.15	15.99 (2.48)
STEEL_II	28.38	68.69	0.58	23.60	121.25 (4.90)
T3DL	4.33	12.47	0.09	2.99	19.88 (3.79)
CHIP	3.51	4.68	0.03	2.76	10.98 (3.51)

Table 11
Execution time (in secs.) and speed-up (inside parenthesis) for the parallel SR-BT algorithm without and with overlapping.

5 Conclusions

We have described a method for model reduction of large-scale continuous generalized LTI systems via BT. A simple extension of the LR-ADI iteration is used to solve large-scale generalized Lyapunov equations, which is the major computational task in the method. These equations and all other major computational stages in the procedure are solved (in parallel) using kernels from well-known linear algebra libraries. The methods are collected in a parallel library for model reduction of sparse LTI systems, SpaRed, on architectures with multiple processors. SVD-based methods can thus be of application to systems with sparse state matrix pencils with up to $\mathcal{O}(10^5)$ states.
The efficiency and parallelism of our algorithms is strongly determined by the efficacy of the underlying computational and communication libraries and the problem data.

References

[1] P.R. Amestoy, I.S. Duff, J. Koster, and J.-Y. L'Excellent. MUMPS: a general purpose distributed memory sparse solver. In *Proc. PARA2000, 5th*

International Workshop on Applied Parallel Computing, pages 122–131, 2000.

[2] A.C. Antoulas. *Approximation of Large-Scale Dynamical Systems*. SIAM Publications, Philadelphia, PA, 2005.

[3] J. M. Badía, P. Benner, R. Mayo, and E. S. Quintana-Ortí. Parallel algorithms for balanced truncation model reduction of sparse systems. In J. Dongarra, K. Madsen, and J. Wasniewski, editors, *Applied Parallel Computing: 7th International Conference, PARA 2004, Lyngby, Denmark, June 20-23, 2004. Revised Selected Papers*, volume 3732 of *Lecture Notes in Computer Science*, pages 267–275. Springer-Verlag, Berlin, Heidelberg, New York, 2006.

[4] R.H. Bartels and G.W. Stewart. Solution of the matrix equation $AX+XB = C$: Algorithm 432. *Comm. ACM*, 15:820–826, 1972.

[5] U. Baur and P. Benner. Factorized solution of Lyapunov equations based on hierarchical matrix arithmetic. *Computing*, to appear.

[6] P. Benner. Solving large-scale control problems. *IEEE Control Systems Magazine*, 14(1):44–59, 2004.

[7] P. Benner, V. Mehrmann, and D.C. Sorensen, editors. *Dimension Reduction of Large-Scale Systems*, volume 45 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, Berlin/Heidelberg, Germany, 2005.

[8] P. Benner, H. Mena, and J. Saak. On the parameter section problem in the Newton-ADI iteration for large scale riccati equations. Preprint, Fakultät für Mathematik, TU Chemnitz, D-09107 Chemnitz, Germany, October 2006.

[9] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. Balanced truncation model reduction of large-scale dense systems on parallel computers. *Math. Comput. Model. Dyn. Syst.*, 6(4):383–405, 2000.

[10] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. State-space truncation methods for parallel model reduction of large-scale systems. *Parallel Comput.*, 29:1701–1722, 2003.

[11] P. Benner and J. Saak. A semi-discretized heat transfer model for optimal cooling of steel profiles. Chapter 19 (pages 353–356) of [7].

[12] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997.

[13] Y. Chahlaoui and P. Van Dooren. A collection of benchmark examples for model reduction of linear time invariant dynamical systems. SLICOT Working Note 2002–2, February 2002. Available from `www.slicot.org`.

[14] J.W. Demmel, J.R. Gilbert, and X.S. Li. *SuperLU Users' Guide*.

[15] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, third edition, 1996.

[16] S. Gugercin and J.-R. Li. Smith-type methods for balanced truncation of large systems. Chapter 2 (pages 49–82) of [7].

[17] S. Gugercin, D.C. Sorensen, and A.C. Antoulas. A modified low-rank Smith method for large-scale Lyapunov equations. *Numer. Algorithms*, 32(1):27–55, 2003.

[18] S.J. Hammarling. Numerical solution of the stable, non-negative definite Lyapunov equation. *IMA J. Numer. Anal.*, 2:303–323, 1982.

[19] J.-R. Li and T. Penzl. Square root method via Cholesky factor ADI. In *Proc. MTNS 2000, Perpignan*, 2000. (CD Rom).

[20] J.-R. Li and J. White. Low rank solution of Lyapunov equations. *SIAM J. Matrix Anal. Appl.*, 24(1):260–280, 2002.

[21] J. Lienemann, E. B. Rudnyi, and J. G. Korvink. MST MEMS model order reduction: Requirements and benchmarks. *Linear Algebra Appl.*, 415(2–3):469–498, 2006.

[22] B.C. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Trans. Automat. Control*, AC-26:17–32, 1981.

[23] G. Obinata and B.D.O. Anderson. *Model Reduction for Control System Design*. Communications and Control Engineering Series. Springer-Verlag, London, UK, 2001.

[24] T. Penzl. A cyclic low rank Smith method for large sparse Lyapunov equations. *SIAM J. Sci. Comput.*, 21(4):1401–1418, 2000.

[25] T. Penzl. LYAPACK Users Guide. Technical Report SFB393/00-33, Sonderforschungsbereich 393 *Numerische Simulation auf massiv parallelen Rechnern*, TU Chemnitz, 09107 Chemnitz, FRG, 2000. Available from `http://www.tu-chemnitz.de/sfb393/sfb00pr.html`.

[26] T. Penzl. Algorithms for model reduction of large dynamical systems. *Linear Algebra Appl.*, 415(2–3):322–343, 2006. (Reprint of Technical Report SFB393/99-40, TU Chemnitz, 1999.).

[27] G. Quintana-Ortí, X. Sun, and C.H. Bischof. A BLAS-3 version of the QR factorization with column pivoting. *SIAM J. Sci. Comput.*, 19:1486–1494, 1998.

[28] M.G. Safonov and R.Y. Chiang. A Schur method for balanced-truncation model reduction. *IEEE Trans. Automat. Control*, AC–34:729–733, 1989.

[29] T. Stykel. Gramian-based model reduction for descriptor systems. *Math. Control, Signals, Sys.*, 16:297–319, 2004.

[30] T. Stykel. Low rank iterative methods for projected generalized Lyapunov equations. Preprint 198, DFG Research Center MATHEON, TU Berlin, 2005. Available from `http://www.math.tu-berlin.de/~stykel/Publications/pr_04_198.pdf`.

[31] F. Tröltzsch and A. Unger. Fast solution of optimal control problems in the selective cooling of steel. *Z. Angew. Math. Mech.*, 81:447–456, 2001.

[32] R.A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. MIT Press, Cambridge, MA, 1997.

[33] A. Varga. Efficient minimal realization procedure based on balancing. In *Prepr. of the IMACS Symp. on Modelling and Control of Technological Systems*, volume 2, pages 42–47, 1991.

[34] A. Varga. Model reduction software in the SLICOT library. In B.N. Datta, editor, *Applied and Computational Control, Signals, and Circuits*, volume 629 of *The Kluwer International Series in Engineering and Computer Science*, pages 239–282. Kluwer Academic Publishers, Boston, MA, 2001.

[35] E.L. Wachspress. ADI iteration parameters for the Sylvester equation, 2000. Available from the author.

Chemnitz Scientific Computing Preprints – ISSN 1864-0087