Improving Accuracy of Parallel SLICOT Model Reduction Routines for Stable Systems*

David Guerrero-López[1] and José E. Román[1]

***Abstract*— This paper shows part of the work carried out to develop parallel versions of the SLICOT routines for model reduction of stable systems. In particular, the routines that have been parallelised are those based on the solution of Lyapunov equations. The goal is to be able to work with larger unreduced models and also to obtain better performance in the reduction process. New routines have been developed using standard libraries to improve portability and efficiency. A preliminary version was released previously by the authors, which achieved high performance. However, accuracy improvements have been necessary in order to make the new routines similar to the sequential ones in this aspect. Routines presented in this paper preserve good performance obtained by the previous parallel implementation while maintaining high accuracy of sequential SLICOT routines.**

Keywords: parallel computing, control linear systems, model reduction, Lyapunov equation

I. INTRODUCTION

In the area of system control, large systems appear frequently. As technology advances, more accuracy is wanted in the solution of problems arising from real world. This desire of increased accuracy usually involves working with larger and larger problems. However, high order systems are obviously harder to manage than lower order ones. That is why model reduction is so important in control theory. By means of model reduction, large systems can be reduced to a size that allows being processed in an easier or faster way.

However, the process of model reduction itself must deal necessarily with the original unreduced system, thus becoming a computationally demanding task. There are multiple algorithms and libraries that implement reduction methods, although most of them are sequential ones. Therefore, the availability of parallel model reduction routines is a valuable asset, enabling to work with larger systems in terms of both memory and time requirements.

In [1], parallel routines for model reduction were presented. However, later work showed that, for some systems, when large distribution block sizes are used, the accuracy of the reduced models is far from that obtained using the equivalent sequential routines of SLICOT [2].

A new approach has been adopted for the main kernel of those parallel routines (the Lyapunov solver that obtains the Cholesky factor of the solution), providing a solution to the problem with high accuracy while maintaining good performance. This new approach is described in this paper.

*This work was partially supported by the Spanish Ministry of Economy and Competitiveness under grant TIN2013-41049-P.

[1]David Guerrero-López and José E. Román are with the D. Information Systems and Computation, Universitat Politècnica de València, Cam de Vera sn, 46022 Valencia, Spain {dguerrer,jroman}@dsic.upv.es

In the next sections, first a brief introduction to model reduction is given. The importance of Lyapunov equations for model reduction is discussed in section III, and various methods are described. Then, the developed parallel algorithms are explained, focusing on the new approach for improved accuracy. In the final sections, experimental results are shown after describing the problems used in the tests.

II. MODEL REDUCTION OF STABLE SYSTEMS

Model reduction of stable systems is a necessary step for model reduction of unstable systems. Most of the methods to reduce unstable systems rely on first computing a reduced model of the stable part of the system. Thus, when the goal is to obtain new parallel implementations for model reduction, the first thing to be accomplished is to obtain parallel algorithms for model reduction of stable systems.

There exist many different methods that can be used for model reduction of stable systems [2]. However, those working with the state-space representation of the system are usually more adequate for a parallel implementation. As they work with matrices representing the system, these methods are commonly based on algebraic operations with these matrices. There are several parallel linear algebra libraries that provide some of the most frequently used matrix operations, thus helping to parallelise those methods.

In this kind of methods, *truncation methods* are specially suitable since they can be applied to many types of systems.

Given a linear-time invariant system in its state space representation (a continuous time system is shown, the process for discrete time systems is similar)

$$\dot{x}(t) = Ax(t) + Bu(t), \quad (1)$$

$$y(t) = Cx(t) + Du(t), \quad (2)$$

truncation methods for model reduction are based on computing a similarity transformation matrix S which leads the system to an equivalent representation

$$\left[\begin{array}{c|c} S^{-1}AS & S^{-1}B \\ \hline CS & D \end{array}\right] = \left[\begin{array}{cc|c} A_{11} & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \\ \hline C_1 & C_2 & D \end{array}\right]. \quad (3)$$

In this new (but still equivalent) representation, the system concentrates its *most valuable* state variables in the leading submatrix, A_{11}. Then a truncation is done, neglecting the effect produced by matrices $A_{12}, A_{21}, A_{22}, B_2, C_2$ and their associated state variables. The remaining system becomes

$$\dot{x}_r(t) = A_{11}x_r(t) + B_1u(t), \quad (4)$$

$$y_r(t) = C_1x_r(t) + Du(t). \quad (5)$$

The criterion by which a state variable is considered to be more/less valuable than others depends on the properties that want to be preserved in the reduced model. This produces several different truncation methods. Typical ones are based on what is known as a *balanced* representation of the system. In this representation, the system is expressed in such a way that its Grammians are equal and diagonal. The Grammians of observability and controllability of a system can be seen as two matrices representing the weights of each state variable over these important properties. The truncation method that uses the balanced realisation of the system tries to preserve these important properties in the reduced model. This truncation method is known as *the square root method* (SR) and is one of the methods implemented in the sequential library SLICOT that have been parallelised in this work.

A. SLICOT routines for model reduction of stable systems

SLICOT[1] is a collection of routines for control theory [3], that was partly developed under a European project with the aim of formalising and extending existent collaboration with respect to robust numerical software for control systems analysis and synthesis.

SLICOT provides Fortran 77 implementations of numerical algorithms for computations in systems and control theory. Based on numerical linear algebra routines from BLAS and LAPACK, SLICOT provides methods for the design and analysis of control systems. The basic ideas behind the library are usefulness of algorithms, robustness, numerical stability and accuracy, performance with respect to speed and memory requirements, portability and reusability, standardisation and benchmarking.

The main routines of the SLICOT library for model reduction of stable systems are:

- `AB09AD` computes reduced (or minimal) order balanced models using either the Square-Root or the Balancing-Free Square-Root Balance & Truncate method.
- `AB09BD` computes reduced order models using the Balancing-Free Square-Root Singular Perturbation Approximation method.
- `AB09CD` computes reduced order models using the optimal Hankel-Norm Approximation method based on Square-Root balancing.
- `AB09DD` computes a reduced order model by using the Singular Perturbation formulas.

We have developed parallel version of routines `AB09AD`, `AB09BD` and `AB09DD`, whose main operations are the solution of Lyapunov equations (obtaining the Cholesky factor of the solution) and the singular value decomposition.

III. SOLUTION OF STANDARD LYAPUNOV EQUATIONS

All these methods have in common that they need to compute the solution of standard Lyapunov equations (Grammians of stable systems can be obtained as the solutions of two Lyapunov equations).

[1]Software Library for Control Theory, `http://slicot.org`

The standard Lyapunov equation is presented in two forms, one for continuous-time systems (6) and one for discrete-time systems (7),

$$A^T X + XA = -E, \tag{6}$$

$$A^T XA - X = -E. \tag{7}$$

Here, A and E are real square matrices of size n. Matrix E is symmetric, as well as the solution matrix X when unique. The most commonly-used method for solving these equations is due to Bartels and Stewart [4]. However, when the solution is going to be used in the model reduction methods mentioned previously, it is the Cholesky factor of the solution that is desired. This Cholesky factor can be obtained in an efficient manner by using the method due to Hammarling [5]. We have focused in this method, because it is the one used in the sequential SLICOT routines. In case the matrices of the equations were sparse, iterative methods could be more appropriate [6].

A. Hammarling's method

Hammarling's method [5] is an alternative for solving Lyapunov equations when their right-hand side is positive semidefinite and matrix A is stable. In these cases, the right-hand side of the equation is usually in the form of the product of a matrix by its transpose, and the Cholesky factor U of the solution matrix X is the desired output. The above equations become

$$A^T U^T U + U^T UA = -B^T B, \tag{8}$$

$$A^T U^T UA - U^T U = -C^T C. \tag{9}$$

This method produces the Cholesky factor of the solution directly without explicitly computing the product in the right-hand side of the equation. It is similar to Bartels-Stewart method in that both of them work by transforming the equation to a reduced form, then solving this reduced equation and later obtaining the solution to the original equation by a back transformation.

Transformation to reduced form

The transformation to reduced form is performed to obtain the equation (working in the continuous case)

$$A_s^T X_s + X_s A_s = -B_s^T B_s, \tag{10}$$

where B_s is an upper triangular matrix of size $n \times n$, and A_s is an upper quasi-triangular matrix of the same dimensions (that is, upper triangular with the exception of some nonzero elements in the first subdiagonal). This form of the equation is obtained by reducing matrix A to the real Schur form A_s, via computing the orthogonal matrix Z that verifies $A_s = Z^T AZ$. A_s can be seen as an upper block triangular matrix, whose diagonal is formed by 1×1 or 2×2 blocks corresponding to real or complex eigenvalues of A, respectively.

In this transformation to reduced form (10), matrix B_s is obtained as the upper triangular matrix R from the QR factorisation of the product BZ.

Solution of the reduced equation

To solve equation (10) for U_s, the Cholesky factor of the solution ($X_s = U_s^T U_s$), the involved matrices A_s, B_s and U_s are partitioned in a way that yields a 2 by 2 or 1 by 1 Lyapunov equation and two other equations. This small Lyapunov equation is solved as a linear system of equations (by Kronecker products). Then the other two equations are updated. One of them is solved as a reduced Sylvester equation in which the involved matrices are upper quasi-triangular. The other equation is transformed by a QR decomposition and matrix products to a reduced Lyapunov equation that can be solved by this same procedure [5], [7].

The solution of this reduced equation in parallel has been treated previously for the generalised case [8] and for the standard case [9].

Back transformation

Once the solution of the reduced equation has been computed, another transformation is required to obtain the solution of the original equation. In Hammarling's method, this transformation consists in obtaining the QR factorisation of the product of U_s, the solution of the reduced equation, and the transpose of matrix Z coming from the transformation to real Schur form, that is, computing $U_s Z^T = Q_U U$.

With this transformation, the upper triangular matrix U, the Cholesky factor of the solution X of equation (6), is obtained.

IV. PARALLEL IMPLEMENTATIONS

In order to obtain parallel versions of SLICOT routines for model reduction of stable systems, sequential routines have been taken as the starting point. Each routine has been parallelised, including also parallel versions for other routines used by it.

In the best case, a sequential routine makes use of routines for which a parallel version is available. In these cases, a simple modification has been needed. Calls to sequential routines from the standard libraries BLAS and LAPACK have been replaced by calls to the equivalent parallel routines from PBLAS and ScaLAPACK, respectively.

However, there have been several routines in which this process is not adequate. Some sequential routines did not have the equivalent ScaLAPACK counterpart, thus requiring parallelisation. Moreover, some routines made operations over very small matrices, making it a bad idea just replacing calls to sequential routines by calls to parallel ones. Some of these routines have required a complete re-design in order to be efficient. This re-design has usually consisted in a block orientation of the algorithm before parallelising.

The process of parallelisation has generated these new routines:

- Parallel driver routines for model reduction, including routines that work with the original system: `PAB09AD`, `PAB09BD`, `PAB09DDS`, and those working with the system in reduced form: `PAB09AY`, `PAB09BY`.
- The kernel dedicated to solve Lyapunov equations: `PSB03OU`, `PSB03OT`, `PSB03OR`, `PMB04OD`, `PMB04ND`.
- Parallel versions of some other SLICOT routines used by the main ones: `PTB01IDS`, `PTB01WD`, `PMB01UD`, `PMB03UD`.
- Some parallel versions of LAPACK routines which are not yet present in ScaLAPACK: `PDGEES`, `PDLANV` (related to Schur decomposition), `PDORGHR`, `PDORGBR` (to generate orthogonal matrices from elementary reflectors), `PDBDSQR` (related to the SVD problem).
- Parallel implementations of some basic matrix operations, not present in PBLAS: `PDGEMM2`, `PDTTMM`.
- Some other operations which are trivial in the sequential case, but require careful treatment in their parallel versions: `PDSHIFT`, `PDSHIFT1`, `PDTRSCAL`, `PDDIAGZ`.

A. New approach to improve accuracy

All these new parallel routines have a good degree of parallelism, that complements their block orientation, thus achieving good performance. However, the Lyapunov solver is critical from the point of view of accuracy. In the cases where the Grammians are not markedly positive definite, accuracy of the results is affected by the block size used. The critical part is the treatment of auxiliary matrices $M_1 = U_{11} A_{11} U_{11}^{-1}$ and $M_2 = B_{11} U_{11}^{-1}$ that appear in the solution of the reduced equation of Hammarling's method [5], [7]. In the sequential routines (those present in SLICOT), the algorithm is arranged by 1×1 or 2×2 blocks. This means that auxiliary matrices M_1 and M_2 are of size 2×2 at most. A very well designed algorithm deals with the problem of computing these matrices when the solution is close to singular.

However, for the parallel algorithms to be efficient, they have to be oriented to larger block sizes. Generalizing the computation of these matrices to a larger size is difficult or not even possible. Thus, in the first parallel implementations [1], the algorithms to compute these auxiliary matrices are not as accurate as those used in the sequential routines. This means that when working with systems in which Grammians are close to singular, accuracy decreases. This does not usually happen with small block sizes, since then matrices are smaller and the process to compute them goes better. But in the parallel case, large block sizes are frequently used in order to increase performance by decreasing the number of communications. In these cases, accuracy of the reduced model could be affected.

After evaluating this problem, if the parallel algorithm fails to be as accurate as the SLICOT sequential routines, it is not a good algorithm. It does not matter if the algorithm can reduce the time to compute the reduced model if it is not as good as the one that can be obtained using the sequential routines.

So, a special effort has been made in order to evaluate how to improve accuracy of the parallel algorithms. After some initial failed attempts, a new idea has been taken into account, namely performing exactly the same operations (except for their order) as in the sequential algorithm. This

means working with auxiliary matrices M_1 and M_2 of size 2×2 at most, exactly as in the sequential algorithms, thus allowing to use the same good algorithm to compute them.

When first considering this new idea, it seems that it will make the algorithm perform the same operations as the sequential ones, thus serialising the process. This would be terrible for parallel performance. Moreover, working with such small matrix sizes would spoil the performance gain obtained by block orientation.

However, in the proposed algorithm operations involving matrices of sizes 1×1 or 2×2 are blocked in larger sizes making it possible to obtain a special block oriented algorithm. This algorithm is similar to the initial block oriented algorithm in the sense that it continues to maximize locality of operations to improve cache reutilization.

In the new approach, operations inside each block are not homogeneous. Several different operations are grouped together in a new sense of block oriented operations, thus benefiting from good performance of block oriented algorithms (both in sequential and parallel) and good accuracy of the sequential original algorithms.

A significant part of the parallel routines developed in the first versions are preserved. Only routines directly involved in the main kernel in charge of solving Lyapunov equations have been changed in this new approach. New versions of those routines have been developed again:

- `PSB03OT`, in charge of solving Lyapunov equations by Hammarling's method, has been re-designed.
- `SB03OT2` is a new sequential routine for solving Lyapunov equations by Hammarling's method. It is very similar to `SB03OT` SLICOT routine, but it returns auxiliary matrices M_1 and M_2 (apart from other auxiliary data), which are needed outside the routine in this new approach.
- `PSB03OR2` is a re-designed version of `PSB03OR` from the previous release. It solves the reduced Sylvester equation that appears in solving the reduced Lyapunov equation. But now it does so in a way more similar to the sequential unblocked algorithm, using auxiliary matrices M_1 and M_2 of small sizes.

The new codes are available from the authors under demand, and will eventually be included in the SLICOT repository.

V. EXPERIMENTAL RESULTS

The platform used for the performance evaluation is a PC cluster with 6 nodes. Each node is a biprocessor computer with two AMD Opteron processors (16 cores) at 2.1 GHz, with 32 GB of RAM memory and Linux operating system. Different nodes are interconnected by using an InfiniBand QDR network.

When possible, processors of different nodes have been used. This is done to avoid having different communications among computing processes depending on whether they are in the same node (communications via share memory) or not (communications through the physical network).

A. Problems

Several test problems have been considered to evaluate the new routines, in terms of performance as well as accuracy.

In order to see if the new algorithms can achieve good performance in parallel, several problems have been generated synthetically. System matrices are generated in a similar way as in [7] to test Lyapunov solvers, but specialised for the standard equation as in [1].

The matrix A of size $n \times n$, $n = 3q$, is generated as $A = W_n^{-1}\mathrm{diag}(A_1, \ldots, A_q)W_n$, with

$$A_i = \begin{pmatrix} s_i & 0 & 0 \\ 0 & t_i & t_i \\ 0 & -t_i & t_i \end{pmatrix}. \tag{11}$$

W_n is a matrix of size $n \times n$ whose elements are all one except those of the main diagonal which are zero. The parameters s_i and t_i determine the eigenvalues of the generated matrix. They are chosen (in the continuous case) as $s_i = t_i = t^i$, for a given value of t.

Matrices B and C of the system are chosen as square randomly generated matrices, since no special features are needed on these matrices by the implemented methods (this is not very realistic but it is useful to have problems of large size that help test performance of developed routines).

The order of the system used to test parallel routines has been chosen to be 3000, which takes several minutes to solve in the parallel platform used. Parameter t has been assigned a value of 1.01.

When the goal of the test is to evaluate accuracy, things change. In this case, it is better to work with problems with a few number of inputs/outputs, so they can be easily checked for time response. For this purpose, several real problems taken from the SLICOT collection of benchmarks [10] have been used. All of them have behaved similarly, so here only one of them is shown. The chosen problem is known as "eady" [10], and corresponds to the model of an atmospheric storm track. It has 1 input, 1 output and 598 states. This problem is an example of the kind of systems that produced loss of accuracy when working with large block sizes in the first parallel routines developed. This is the reason why it is a good problem to test precision of the newly developed routines. (Moreover, accuracy has also been assessed in the large problems by observing a small number of randomly selected inputs and outputs).

The routines `PAB09AD` and `PAB09AY`, that implement the square-root model reduction method in parallel, have been used in all the tests. Results obtained with these routines can illustrate what can be expected with the other.

B. Timing results

The random system described above has been used in different runs where it has been reduced by using `PAB09AD` and `PAB09AY` routines. The measured parameter has been the execution time.

`PAB09AD` is a driver routine that does all the necessary operations to obtain a reduced model starting from an stable system. Showing results of this routine would be enough

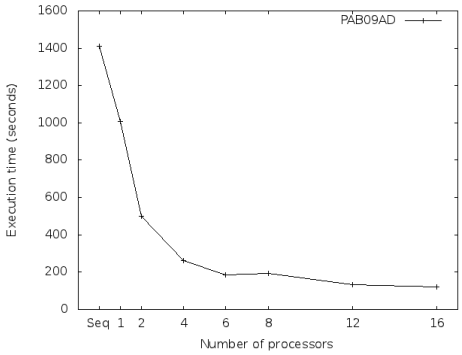

Fig. 1. Execution times of PAB09AD

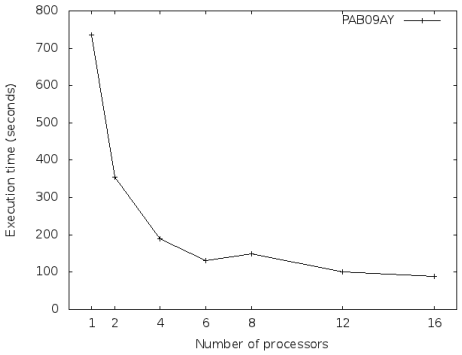

Fig. 2. Execution times of PAB09AY

to have an idea of the performance of the new parallel routines. However, it is also interesting to show the results of its auxiliary routine PAB09AY, that makes the same but working with a system in a reduced form. This reduced form is obtained by computing the Schur decomposition of the matrix A of the system and applying it to the rest. PAB09AD includes the computation of this transformation. However, the main part of this transformation (obtaining the Schur form of a matrix) spends an important part of the execution time of the whole process, while not being a newly developed routine. Thus, showing results without taking into account that part of the computation is interesting in order to see how the new routines perform.

Before testing the parallel routines, a block size has to be chosen to be used in the parallel executions. For doing this, the routines have been executed in only one processor using block sizes varying from 32 to 150. The block size which has required less execution time in the available platform (50) has been used in the rest of parallel runs (although differences have been small). This block size is not guaranteed to be the best block size for the parallel algorithm (when running with several processors), since in that case the block size also affects the number and length of messages to be exchanged. However, it can be considered to be a good approximation, and it also allows to obtain the best sequential times for that problem in the platform used.

Once the block size has been chosen, these routines have been tested from 1 to 16 computing processors. For each number of processors all possible configurations of the processors mesh have been used, showing only the best execution times obtained for a given number of processors. Generally, square meshes (those in which the number of rows and columns are similar) have worked better with both algorithms (e.g., for 6 processors, possible meshes are 1×6, 2×3, 3×2 and 6×1, the mesh of 3×2 giving the lowest execution time).

Figure 1 shows execution times obtained when reducing the system via the PAB09AD routine.

Figure 2 shows execution times obtained when reducing the system via the PAB09AY routine (starting from the same system, but previously reduced to Schur form outside the routine).

As it can be seen, execution times reduce considerably as the number of processors increases, but differences are larger in the left portion of the figures corresponding to less processors. This is usual in parallel algorithms. With increasing number of processors, if the problem size remains constant, the computational work to be done by each processor is reduced thus decreasing the efficiency of the whole process. It is expected that by increasing the problem size, efficiencies will go up.

Moreover, in the platform used, when using more than 6 processors some of them will be in the same node, thus communicating via shared memory and not through the physical network. Our currently installed message-passing library is optimised for communications through the physical network, but has a poorer performance when communicating inside one node. This can be the reason of the decay of performance from that number of processors up, added to the usual previously commented one. And moreover, the first case with this decay is with 8 processors which suffers also of a poorly square mesh (2×4 and 4×2 are the most "square" meshes that can be used for this).

Table I shows speed-ups and efficiencies of both routines. This parameters have been computed using the execution time of the parallel algorithm in one processor as the 'best sequential time'. This can be done, since this time is much smaller than the execution time of the original SLICOT routines. The reason for this is that the corresponding SLICOT routines are not block oriented, as opposed to the parallel algorithm. Moreover, it should be noted that the block size has been chosen trying to obtain the best execution time in 1 processor.

C. Accuracy results

The control system referred to as "eady" has been reduced using PAB09AD routine and compared to the reduced system obtained by its equivalent sequential SLICOT routine AB09AD.

As explained previously, the first implementations of these parallel model reduction routines presented loss of accuracy for some problems when large block sizes were used. Thus, an interesting test is to look at the accuracy results obtained for increasing block sizes.

TABLE I

PARALLEL SPEED-UP ($S_p = t_1/t_p$) AND EFFICIENCY ($E_p = S_p/p$) FOR VARYING NUMBER OF PROCESSES (p) FOR THE COMPLETE MODEL REDUCTION (PAB09AD) AND STARTING FROM SCHUR FORM (PAB09AY)

PAB09AD	p=2	p=4	p=6	p=8	p=12	p=16
S_p	2.02	3.85	5.45	5.22	7.55	8.33
E_p	101.19	96.26	90.84	65.21	62.95	52.03
PAB09AY	p=2	p=4	p=6	p=8	p=12	p=16
S_p	2.07	3.87	5.60	4.95	7.26	8.35
E_p	103.44	96.63	93.41	61.83	60.47	52.21

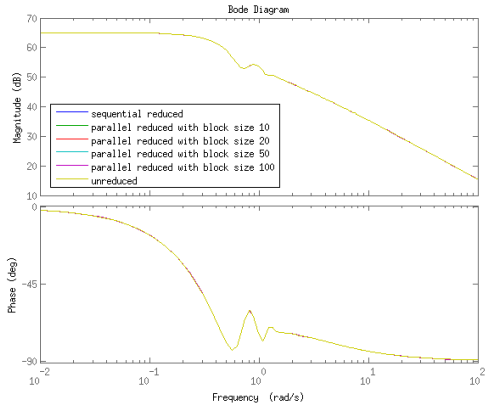

Fig. 3. Bode plot for "eady" problem reduced using different block sizes

Since reducing a model is a complex process involving several operations, some of them iterative (e.g. Schur decomposition), executions with different number of processors or different block sizes lead to different final results, while being equivalent representations of the same system. This makes it difficult to verify the correctness of the results. Comparing the matrices of the resulting systems is not adequate.

In this cases, an adequate method to compare the systems obtained from different executions is to look at their time response. Of course, this can only be done when working with problems with a few number of inputs/outputs. This is not a problem in this test.

Thus, Bode diagrams have been used to check visually for the similarity of the different obtained systems.

In Figure 3, Bode diagrams of the original unreduced system are shown together with the same diagrams for the reduced systems obtained by the sequential SLICOT routine and the new parallel routines executed with different block sizes: 10, 20, 50 and 100.

It can be seen that all of them are very similar. In fact, the reader will see only one plot with a very small noise in some parts of it. But there are six different plots, corresponding to the original unreduced system, reduced system obtained by the sequential SLICOT routine and four reduced systems obtained by the parallel routines with four different block sizes.

When magnifying the image, it can be seen that all reduced systems have coincident Bode plots, which are very similar to the unreduced system (this did not happen with the previous version of parallel routines, that presented differences at high frequencies for large block sizes). All the runs have obtained the same order for the reduced system (91).

VI. CONCLUSIONS

Model reduction is an important problem in control theory. However, there are not many parallel algorithms for reducing systems. Parallel implementations for SLICOT model reduction routines of stable systems have been previously developed [1]. However, precision of the parallel routines for some problems could be affected by the block size used in the process, making the result not as good as that obtained by the sequential SLICOT routines.

As presented in this paper, the main kernel of those parallel implementations, related to solving Lyapunov equations computing the Cholesky factor of their solutions, has been re-designed. A new parallel algorithm has been developed, which keeps both block orientation, which is needed to have good parallel performance, and performs the same operations as in the sequential routines, which allows preserving good accuracy of SLICOT routines independently of the block size used in the computations. Experimental results show that these two desired features have been achieved.

Execution times are considerably reduced, making it possible to work with larger systems more quickly. Note that even the execution time in 1 processor is reduced, thanks to the block orientation of the parallel algorithms.

Moreover, precision of the new algorithm is similar to that obtained by the original sequential SLICOT routines.

REFERENCES

[1] D. Guerrero, V. Hernández, and J. E. Román, "Parallel SLICOT model reduction routines: the Cholesky factor of Grammians," in *15th IFAC World Congress on Automatic Control, Barcelona, 21-26 July*, 2002.

[2] A. Varga, "Model reduction software in the SLICOT," in *Applied and Computational Control, Signals, and Circuits*. Springer, 2001, vol. 2, pp. 239–282.

[3] P. Benner, V. Mehrmann, V. Sima, S. V. Huffel, and A. Varga, "Slicot–a subroutine library in systems and control theory," in *Applied and Computational Control, Signals, and Circuits*. Springer, 1999, vol. 1, pp. 499–539.

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

[5] S. J. Hammarling, "Numerical solution of the stable, non-negative definite Lyapunov equation," *IMA J. of Numerical Analysis*, vol. 2, pp. 303–323, 1982.

[6] T. Stykel and V. Simoncini, "Krylov subspace methods for projected Lyapunov equations," *Applied Numerical Mathematics*, vol. 62, no. 1, pp. 35 – 50, 2012.

[7] T. Penzl, "Numerical solution of generalized Lyapunov equations," Numerische Simulation auf massiv parallelen Rechnern, Technische Universität Chemnitz-Zwickau, Tech. Rep. SFB393/96-02, May 1996.

[8] D. Guerrero, V. Hernández, J. E. Román, and A. M. Vidal, "Parallel Algorithms for the Cholesky Factor of Generalized Lyapunov Equations," in *5th IFAC Workshop on Algorithms and Architectures for Real-Time Control*, Cancun, Mexico, April 1998, pp. 237–242.

[9] D. Guerrero, V. Hernández, and J. E. Román, "Parallel Solution of the Standard Lyapunov Equation by Hammarling's Method," Third NICONET Workshop on Numerical Control Software, Louvain-la-Neuve (Belgium), Tech. Rep., January 2001.

[10] Y. Chahlaoui and P. V. Dooren, "A collection of benchmark examples for model reduction of linear time invariant dynamical systems," SLICOT Working Note 2002-2, Tech. Rep., February 2002.