Parallel Model Order Reduction for Sparse Electromagnetic/Circuit Models

Giovanni De Luca [1], Giulio Antonini [1], and Peter Benner [2]

[1] Dipartimento di Ingegneria Industriale e dell'Informazione e di Economia
Università degli Studi dell'Aquila, L'Aquila, 67100, Italy
giovanni.deluca84@gmail.com, giulio.antonini@univaq.it

[2] Computational Methods in Systems and Control Theory
Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany
benner@mpi-magdeburg.mpg.de

Abstract ─ This paper describes a parallel Model Order Reduction (MOR) technique for Linear Time Invariant (LTI) electromagnetic/circuit systems with sparse structure. The multi-point Krylov-subspace projection method is adopted as framework for the model order reduction and a parallelization strategy is proposed. More specifically, a multi-point version of the well-known PRIMA algorithm is proposed, which is parallelized with respect to the computation of the error between the original model and the reduced one. The number of moments to be matched for any expansion point is chosen adaptively as well. The numerical results show that the proposed parallelized MOR algorithm is able to preserve the accuracy of the reduced models while providing a significant compression and a satisfactory speedup with respect to the sequential one.

Index Terms ─ Model order reduction, parallel computing, sparse electromagnetic/circuit systems.

I. INTRODUCTION

The increasing demand for performance of ICs pushes operation to higher signal bandwidth and accurate modeling of previously neglected effects, such as crosstalk, reflection, delay, and coupling becomes increasingly important during circuit and system simulation [1].

When modeling complex 3D geometries, Maxwell's equations are discretized in space and reduced to time continuous-space discrete equations. Time dependence can be further removed by using approximation of time derivatives resulting in time and space discrete equations. This task can be addressed by means of numerical discretization methods like Finite Difference (FD) methods and the Finite Element Method (FEM) or integral ones as the Method of Moments (MoM) [2] or the Partial Element Equivalent Circuit (PEEC) method [3]. Independently of the approach, the direct simulation of the resulting model may be computationally expensive in time and memory storage. The generation of a compact model preserving the properties (e.g., stability, passivity and reciprocity for electrical/electronic devices) of the original physical system is an important step because it allows to use the model in a virtual prototyping environment, avoiding to resort to a physical prototype. Thus, it is possible to verify the correct functioning of the designed system at a virtual prototyping level, testing different operating conditions and undesired effects when linked to other devices. Moreover, the creation of a virtual prototype is definitely cheaper and faster than creating a physical one.

Unfortunately, depending on the mesh of the discretization, the extracted model may contain many equations and/or variables ranging from the thousands to the millions. Such complex dynamical systems need to be simplified, while preserving the accuracy of their input-output behavior, in order to perform simulations within an acceptable amount of time and limited storage capacity, but with reliable outcome [4]. In order to speed up the

Submitted On: March 9, 2014
Accepted On: September 28, 2014

simulations and to save memory space, Model Order Reduction (MOR) techniques have been developed and are efficiently used to cope with this problem [4].

Model order reduction techniques aim to quickly capture the essential features of a model while keeping minimal the number of degrees of freedom. The reduced model has to match the frequency response of the original model within a desired error tolerance, in a prescribed frequency range of interest, and at the same time, has to preserve all necessary properties such as stability and passivity. The interested reader may refer to the "MOR Wiki" web site [5] for a list of model order reduction methodologies and benchmarks.

Model reduction methods based on balanced truncation are very efficient for medium to large-scale problems [6]. It is well known, that a global error bound for the reduced model makes the model reduction process automatic. Although, model reduction methods based on Krylov subspace and moment-matching are much simpler to be implemented and also require usually less computational complexity than methods based on balanced truncation; the most difficult task is choosing suitable expansion points, and in many cases using only an expansion point at zero is not sufficient for an accurate and small reduced model. However, how to choose proper nonzero expansion points and how to decide upon the corresponding number of moments accordingly is still an open problem, despite some progress as reported in [6].

An early method called CFH (Complex Frequency Hopping) is proposed in [7]. By using a binary search algorithm, the expansion points are chosen with respect to the common poles contained in both circles of the neighboring expansion points. However, the poles of the transfer function are computed based on explicit moment-matching; i.e., the moments are computed by recursive matrix-vector multiplications, in the same way as the Asymptotic Waveform Evaluation method in [8]. Therefore, the poles computed are actually not accurate because of numerical instability; although, they would represent the actual poles if computed with precise arithmetic. Moreover, in order to compute the actual poles, higher order moments must be computed, and explicit moment computation cannot guarantee that the higher order moments are accurately computed, again because of numerical inaccuracies [6].

In this paper, we focus on the use of a multi-point version of the well-known PRIMA algorithm [9], based on implicit moment-matching (that can maintain numerical stability), which is made adaptive and parallel by exploiting the power of modern multi-core processors. In particular, in order to obtain a reduced model with, say, a minimum order of reduction with respect to the desired accuracy, we just fix the number of frequency test-points in correspondence of which we compute the error between the dynamics of the original model and of the reduced one, employing PRIMA on those points having the maximum error value in a recursive way, and increasing the order of moments to be matched for the selected points when necessary. When the number of frequency test-points is high (which should ensure a full exploration of the frequency range of interest), the computation of the error values may be time-consuming, mostly depending on the original system dimension. For this reason in this paper, such a computation is performed in parallel.

Section II provides a brief review of the PRIMA algorithm, including the theory of deflated Krylov subspaces. Section III describes the parallelization of the multi-point PRIMA algorithm, providing remarks and comments on the adopted technique as well; several numerical results are presented in Section IV using different data sets available in public repositories. Finally, the conclusions are drawn in Section V and future perspectives are pointed out.

II. THE PRIMA ALGORITHM

A. Basic formulation

Let us assume the electromagnetic/circuit system be represented in descriptor form as:

$$\Phi:\begin{cases} C\dot{x}(t) = -Gx(t) + Bu(t) \\ y(t) = Lx(t) \end{cases}, \tag{1}$$

with zero initial condition $x(0) = 0$. The matrices $C, G \in R^{n\times n}$ contain memory and memoryless elements, respectively, $x(t) \in R^n$ denotes the vector of state variables. In a circuit environment, the Modified Nodal Analysis (MNA) [10] naturally leads to a descriptor form assuming as unknowns the node potentials and the currents flowing in inductances and voltage sources of the equivalent circuit. Also, $B \in R^{n\times m}$ and $L \in R^{m\times n}$ are the input and output matrices, respectively.

The corresponding complex-valued, matrix transfer function in the Laplace domain reads:

$$H(s) = L(G + sC)^{-1} B \cdot \tag{2}$$

The PRIMA [9] algorithm provides a unitary projection matrix V (e.g., computed by the one-side Arnoldi method, where $V^H V = I$ with I being identity matrix of appropriate dimension), and then the approximation $\hat{x}(t)$ can be represented as $\hat{x}(t) = Vz(t)$. Therefore, $x(t)$ can be approximated by $x(t) \approx Vz(t)$. Here, $z(t)$ is a vector of length $k \ll n$. Once $z(t)$ is computed, the approximate solution $\hat{x}(t) = Vz(t)$ for $x(t)$ can be obtained. The vector $z(t)$ can be computed from the reduced model:

$$\tilde{\Phi}: \begin{cases} \tilde{C}\dot{z}(t) = -\tilde{G}z(t) + \tilde{B}u(t) \\ \tilde{y}(t) = \tilde{L}z(t) \end{cases} \tag{3}$$

Besides, the reduced model preserves the main properties of the original system (stability, passivity) under some assumptions on the structure of the matrices *C* and *G* [9]. The reduced MNA matrices are:

$$\begin{aligned} \tilde{C} = V^H C V, \quad \tilde{B} = V^H B, \\ \tilde{G} = V^H G V, \quad \tilde{L} = L V. \end{aligned} \tag{4}$$

The unitary projection matrix V of dimension $n \times k$ is obtained using the block-Arnoldi procedures with the modified Gram-Schmidt process, such that its *k* column-vectors span the Krylov subspace $K_l(A, R)$ induced by the *l*-block moments, as the sequel $colspan\{V\} = K_l(A,R) = span\{R, AR, ..., A^{(l-1)}\}$, where $R \equiv (sC + G)^{-1} B$, with $s \in C$ (complex set) a selected expansion point, and $A \equiv -(sC + G)^{-1} C$.

B. Deflation of the Krylov subspace

In the case of multiple input systems, i.e., with $m > 1$, we need to use a block-version of the Arnoldi algorithm to compute the moments of the transfer function with PRIMA. Equivalently, we need to solve linear systems of the form $AX = B$ with multiple right-hand side.

An important aspect of block-Krylov subspace is represented by the possible linear dependence of the basis vectors. Such dependence can arise both in the set of starting vectors and in the following blocks, during the subspace construction phase: in the first case, linearly dependent columns of the input matrix *B* should be eliminated before starting the Arnoldi procedure, so that the solution of the linear system (which is the first moment of the transfer function in this case) is of full column rank, and this is called "initial deflation"; in the second case, linearly dependent vectors are detected within the orthogonalization process of the following blocks, this is reffered to as "Arnoldi deflation" [11]. Exact linear dependence rarely occurs in practical applications, on the contrary approximate deflation, depending on a deflation tolerance tol_{defl}, is more common and may reduce the computational cost of the iterative procedure [11].

As implemented in [11], differently from the standard rank-revealing QR (RRQR) factorization, the deflation is implemented by comparing the norm of the last orthogonalized column with the initial norm of the same column: if the ratio between the former and the latter is smaller than tol_{defl}, then, such column has to be deflated. Clearly, the accuracy in the reconstruction of the deflated systems solution depends on the choice of tol_{defl}, [11]. We set $tol_{defl} = 10^{-10}$.

Remark: Purely imaginary expansion point(s) (for single-point or multi-point PRIMA) is the common choice, since it is the response along the imaginary axis which is of interest for interconnect analysis [12]. Besides, the use of complex expansion points s_i typically results in a significantly smaller state-space dimension *k* of the reduced order model, compared to the case of real expansion point(s), since s_i can be placed closer to the frequency range of interest than any real one(s) [13].

However, using complex expansion points results in complex reduced models. Then, to generate a real ROM (as the original one), one can separate the complex projection matrix *V*, resulting from a MOR method, in real and imaginary parts to generate a unique real matrix *V*, such that:

$$\tilde{V} = \left[Re(V) \;\; Im(V) \right]. \tag{5}$$

One obvious disadvantage of this approach is that the dimension of the resulting reduced-order model is doubled to 2*k*. Furthermore, in general, the projection matrix (5) is not guaranteed to have full column rank, and so before using (5) as a projection matrix, one would need to check for and possibly delete any linearly dependent columns of (5) [13]. To avoid the linear dependent columns, one can check whether tol_{defl} is satisfied at each step of the

orthogonalization.

III. PARALLEL, ADAPTIVE MULTI-POINT PRIMA

The PRIMA algorithm approximates the original system just locally, around a selected expansion point. Wide-band modeling would require a large number of moments to be matched to obtain a prescribed accuracy. Also, in some applications, too many derivatives need to be matched to obtain a sufficient accuracy in the frequency range of interest. Without resorting to higher order moments, multiple expansion points have been considered, so that a limited number of moments need to be matched on each frequency point.

There have been many efforts and contributions in this direction: Benner, et al., developed an efficient MOR scheme which chooses the expansion points and the number of moments in a fully adaptive way [6]; an adaptive algorithm to automatically identify the expansion points and set the order of the model has been proposed in [1]; among others, Lee, et al., proposed an Adaptive-Order Rational Arnoldi (AORA) method [14] to be applied to large-scale linear systems. Also, Nakhla, et al., in [1] used a binary search method to find frequency points where the ROM is computed.

However, the choice of expansion points in the rational Krylov-based MOR theory, to reach a desired approximation of the ROM with a minimum order of reduction, while keeping the simulation time as small as possible, is still an open problem.

In order to retain a small number of expansion points and a limited number of moments for each of them, trying to catch just the dominant poles of the transfer function describing the input-output behavior of the original full order model, we propose a new method, with the following motivations.

1. We assume to know the starting and ending points of the frequency range of interest and the matrices describing the LTI, SISO or MIMO system; no information is available about the original system in terms of the exact location of the dominant poles of its transfer function, or about stability properties. Since the approximation of the ROM, with the desired accuracy over the frequency range of interest is the main goal in the MOR field, one can recursively check the error between the dynamics (accounting for modules and phases) of the original model and that of the currently computed ROM for a high number of distributed frequency test-points n_{FS} (FS stands for Frequency Samples), applying PRIMA (thus, extracting the projection matrix) on those points showing the maximum error value among all the others. This way, we would avoid heuristic searching method for the expansion points. Obviously, as the number of distributed points tends to infinity, then there will be retained a sufficient number of points which make sure the accuracy is reached in all the frequency range, matching just one or very few moments on each selected point with the use of PRIMA. But this has the big drawback of the intractable simulation time, when computing the original model's transfer matrix in correspondence of each test-point, and PRIMA (both these operations involve the solution of sparse linear systems), mostly for very large size matrices.
2. To avoid the choice of an intractable, high number of frequency test-points to be checked, we select a reasonable number of linearly distributed points in the range of interest, in correspondence to which the error has to be computed. Then, the frequency sample with the maximum error value is selected as an expansion point and a satisfactory solution of the transfer function is obtained on it, executing PRIMA with matching just one moment. Since an acceptable solution can be achieved only in the retained frequency samples used as expansion points, whilst this is not the case for frequencies belonging to the interval between two consecutive test-points, we can think of increasing the number of the moments in an adaptive way: let us assume the current maximum error of the dynamics is in correspondence to the (angular) frequency point ω_i which lies between ω_{i-1} and ω_{i+1}, such that $[\omega_{i-1}\ \omega_i\ \omega_{i+1}]$ is a subinterval of the whole range and they are linearly spaced. Then, PRIMA will be computed on $s_i = j\omega_i$, with $j = \sqrt{-1}$, matching just the first moment on ω_i. In the next step, we check the error on both

the middle points $\omega_{i-1,i}$ of the left-subinterval $[\omega_{i-1}\ \omega_i]$ and $\omega_{i,i+1}$ of the right one $[\omega_i\ \omega_{i+1}]$: until the error of at least one of the mid points is greater than the desired tolerance, we increase iteratively the number of moments to be matched at ω_i. We choose to check the middle points inside each subinterval $[\omega_{i-1}\ \omega_i\ \omega_{i+1}]$, starting from the idea adopted in [6] for the adaptive choice of the moments, where the increasing of moments depends on the accuracy (error check) on points far away from the currently selected one: in our case, the current point is ω_i and the farthest points (from it) are $\omega_{i-1,i}$ (in the left-subinterval $[\omega_{i-1}\ \omega_i]$) and $\omega_{i,i+1}$ (in the right one $[\omega_i\ \omega_{i+1}]$), respectively. Nevertheless, as pointed out in [6], it is not ensured that the farthest point exhibits the maximum error value, so that the choice of a "sufficiently" high number of linearly distributed points to be checked in the range is justified: as this number increases, the distance between the currently processed frequency point ω_i and the farthest points in the left and right subintervals $\omega_{i-1,i}$ and $\omega_{i,i+1}$, respectively, decreases; thus, these mid points are "close enough" to ω_i such that there may be also a good accuracy on them matching eventually few moments on ω_i, and the error check on them as farthest points is useful in order to accurately reproduce the dynamics on subintervals between consecutive test-point, as well. In Section IV, numerical results will be provided, which qualitatively relates the selected number of linearly distributed points for the error check and the resulting order of reduction.

3. Even with a moderate number of distributed frequency points to be checked, the computational time of the original transfer function for each point may be expensive, since it requires n_{FS} solves with (s_iC+G), according to (2), and the CPU time could increase with the size of the initial model. PRIMA also needs the inverse of the same matrix, to compute the moments. To overcome this limitation, we divide the initial frequency range into a certain number of subintervals, delegating the search for expansion points and the computations of PRIMA to different available parallel processors, as described below.

Briefly, our algorithm is based on the recursive exploration of the frequency range of interest and additional expansion points are identified through the exploration step, by comparing the frequency response of the original system and the reduced one. When the size of the modeled systems exceeds few thousands, the computation of the frequency responses and that related to the moment(s) with PRIMA are typically time consuming.

As a preliminary step, we select the desired tolerance *tol* for the accuracy of the ROM's dynamics and a number of n_{FS} frequency samples, linearly distributed in the range of interest, over which the error has to be checked and PRIMA has to be applied, when necessary; successively, we divide the range in a certain number of subintervals, each of them containing an equal number of expansion points. The number of subintervals is equal to the number of available parallel processors, and we parallelize the executions on the multiple subintervals, exploiting the power of modern multi-core processors. We assign a subinterval to each processor (such that the selected processors will cover all the subintervals, the union of which forms the entire frequency range), resulting in an "embarrassingly" task parallel execution (no communication required between the operations to be computed on each processor).

However, the efficiency of a parallelized method depends on some factors. First of all, the sequential part of the algorithm could slow down the overall performance of the algorithm, dominating over the parallelized one (Amdahl's law [15]). Also, when working with very large data sets, the data transfer time and memory demands could exceed that needed for the real computation, i.e., when the workload for each parallel processor is not "high enough" to overlap the data transfer and memory accesses (to request data from the hierarchical memory and retrieve it from the CPU, where the operations are executed). Besides, another factor causing the slowdown of the parallel performance could be the data bus contention, when each processor requests data from the RAM (mainly, when the data cannot be entirely loaded into fast, private memory blocks, namely the

cache), resulting in a waste of time for the processors waiting for data, which travels in a bus eventually shared by every one of them before proceeding with the computation. We will examine these factors in the section on numerical results.

Assuming initially to work with a single processor, then there is just one interval, which is identical to the original frequency range. As starting point, we compute the frequency response for all the n_{FS} points, storing these values (for SISO systems) or matrices (for MIMO models) in memory. Afterwards, we use PRIMA on the lowest frequency point ω_0 of the range, matching just the first moment, and extracting the first projection matrix. What follows is a further check on the farthest point from ω_0, which lies in the middle of the right subinterval $[\omega_0 \quad \omega_1]$, increasing the order of the moments at ω_0 if the error in the midpoint of this subinterval is larger than the tolerance *tol*. (*Note:* when increasing the moments and the selected point is either the lowest or the highest one in the frequency range, we just consider the right subinterval or the left one, respectively.) We define the (relative) error as:

$$err = \frac{\left|H(s) - \tilde{H}(s)\right|}{\left|H(s)\right|}, \tag{6}$$

where $H(s)$ and $\tilde{H}(s)$ are the transfer functions of the original model (already available since previously computed) and of the ROM (which changes each time a new expansion is done), respectively.

It is worth noticing that the error (6) is a matrix, where the (*i, j*)-th element corresponds to the error of the dynamics of the (*i, j*)-th transfer function; in the multi-point approach followed in [1], the error was computed as a RMS error, an average among all the error values of each ports of the model; thus, in the case that all the ports but one show an error smaller than the desired tolerance, the final RMS error may be smaller, thus the tolerance too, and PRIMA would not be computed on the processed frequency point and consequently the approximation in that point or interval was considered accurate enough but it is not. Instead, we computed the error as (6), accounting for the error value for each (*i, j*)-th transfer function, discarding the processed frequency point just in case all the ports have a satisfactory accuracy. This error formulation is suitable for both SISO and MIMO systems.

The next step is the computation of the error for each distributed frequency sample: the one exhibiting the maximum value among the others is selected, and a projection matrix is extracted using that frequency sample as expansion point. Again, we do a further check on the mid points in its left and right sub-intervals, increasing the order of the moments when required. For each (iteratively) retained frequency point, the corresponding obtained projection matrix is block-column concatenated with the previous ones. The computation of the error is repeated for the remaining points in the frequency interval, and as before, other points may be retained and used as expansion points until for all the distributed points, the error is smaller than the a-priori fixed tolerance *tol*.

Instead, when using n_{proc} processors (parallel workers), thus having n_{proc} subintervals each of the with n_{FS}/n_{proc} sample points, each worker uses the same approach as before and will only care about its own subinterval, initially applying PRIMA at the lowest point of it, eventually increasing the moments to be matched, and iteratively selecting the other points exhibiting the maximum error value repeating the computation of PRIMA until the desired accuracy is satisfied. This parallel approach is based on the independence of each subinterval on the other ones, but we will see that a little drawback could arise, in terms of redundancy of some retained expansion points. The pseudo-code of the parallel and adaptive multi-point PRIMA is shown in Algorithm 1.

The algorithm receives as inputs the matrices of the original system in the general form (1) (G, C, B, L), the starting (*fStart*) and ending (*fStop*) points of the frequency range of interest, the number of points to be linearly distributed (n_{FS}), the desired tolerance *(tol)* for the accuracy of the ROM and the number of processors (n_{proc}) to run in parallel (which will be equal to the number of subintervals). The projection-matrix V_{fin} is produced as output.

Algorithm 1 Function V_{subInt}= MultiPts$(G, C, B, L, D, fStart, fStop, n_{FS}, tol, n_{proc})$

create $subInt(n_{proc}, n_{FS}/n_{proc})$, where each row is a subinterval and contains n_{FS}/n_{proc} test-points;

compute in parallel the frequency responses of the original model for all the n_{FS} test-points, according to (2), and store the data into $freqResp(n_{proc}, n_{FS}/n_{proc})$;

set $dist_{farPts}$ as the half distance between a point and its consecutive;

for j=1:n_{proc} **do**
  V_{subInt}=parSubInt$(subInt(j,:), freqResp(j,:),$
  $G, C, B, L, D, fStart, fStop, tol)$;
end for

block-column concatenate each V_{subInt} to form V_{fin};
re-orthogonalization of V_{fin};

It is worth noticing, that in order to decrease the time needed for generating higher-order moments (higher than 1), we estimate the increased-order moment (with the *higherMomPRIMA* function in Algorithm 1) only with respect to the previously computed one (selecting from the current projection matrix V_i the moment corresponding to *lastMom*).

Regarding the final projection matrix V_{fin}, it is not unique, as a basis for the rational Krylov-subspace. In fact, we start to approximate the original transfer function from the initial point $\omega_0 = \min(subInt)$, that in the case of a unique interval (sequential execution with one processor), it is the closest to the DC component, but it can be chosen differently. The choice $\omega_0 = 0$ (or $\omega_0 \approx 0$) is widely used as it often delivers good results in a large neighborhood of the low-frequency part of the spectrum, including the steady state [16]. However, numerous simulations have shown that a random selection of the initial value for each subinterval neither affects the convergence nor the overall CPU time of the algorithm, nor the accuracy of the ROM's dynamics.

Figure 1 may better clarify the search for and selection of expansion points, for both the sequential and the parallel executions, as the algorithm proceeds.

For instance, assuming we linearly distribute $n_{FS} = 7$ frequency test-points (from ω_0 to ω_6), the sequential algorithm (top) computes in step 1 the first moment with PRIMA (filled circle) on the lowest frequency point ω_0. Then, it calculates the error in the point $\omega_{0,1}$ (the midpoint in the interval $[\omega_0\ \omega_1]$), where ω_0 and ω_1 are two consecutive points, and employs PRIMA to catch higher-order moments if the error in $\omega_{0,1}$ is greater than the tolerance (in this picture we do not show the adaptivity for the computation of higher-order moments, for simplicity). In step 2, the algorithm evaluates the error values on each test-point and select that with the maximum value (empty circle, in this case in correspondence to ω_4), and in step 3 PRIMA is executed matching the first moment at ω_4, and eventually computes those with higher order. The algorithm stops when the error values on all the test-points are less than the desired tolerance. Clearly, the method will not make the expansion on all the test-points, since a global accuracy in the entire range will be obtained by matching the moments of the original transfer function at different, selected expansion points.

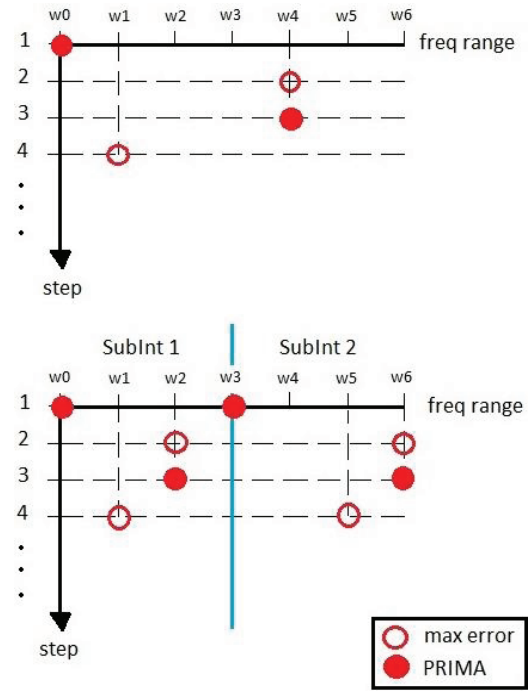


Fig. 1. Example of the sequential execution of the search method within the entire frequency range (top) and the parallel version with two processors/subintervals (bottom).

Instead, for the parallel version with two processors (bottom of Fig. 1), the method divides the frequency range into two subintervals (*SubInt1*

and *SubInt2*), and then proceeds with the same criterion of the sequential one. One can easily note, that there may be redundancies in executing PRIMA at those points that are "shared" between two consecutive subintervals (in Fig. 1 it is ω_3), but this redundancy can be eliminated with the deflation rule presented in Section II. However, other redundancies could not be deleted by the deflation, e.g., when the parallel algorithm (relying on the independence of the subintervals) retains two distinct points, which are sufficiently close in the entire range but are parts of two (distinct) consecutive subintervals. We will see that increasing the number of parallel workers (thus, the number of subintervals) could result in a slight augmentation of the final order of reduction of the ROM, but also with a better accuracy, which becomes much more satisfactory than that fixed a priori, and most important, with a gain in the CPU time execution. Nevertheless, the problem of obtaining a ROM with the desired accuracy is resolved in any case.

As a measure of the parallelization, we refer to the speedup factor:

$$SP\left(n_{proc}\right)=\frac{t_{seq}^{*}}{t_{par}\left(n_{proc}\right)}, \tag{7}$$

where t_{seq}^{*} is the execution time of the fastest sequential program solving the same problem. Moreover, as a measure of utility of the selected number of processors that work in parallel, we define the *efficiency* of the parallelization:

$$Eff\left(n_{proc}\right)=\frac{t_{seq}^{*}}{n_{proc}t_{par}\left(n_{proc}\right)}, \tag{8}$$

where n_{proc} is the number of parallel processes [17].

Another common factor used in the field of parallelization of the execution is the *strong scalability*. Fixing the problem size (i.e., the dimension *n* of any data set), a program scales linearly if the speedup factor is equal to the number of parallel processes used, n_{proc}.

Evidently, optimal (linear) scaling is attained when the speedup factor is equal or close to n_{proc} (or, stated in another way, the efficiency stays close to one) and therefore, strong scaling results can be visually inspected by plotting the speedup factor (or the efficiency) versus the number of parallel processes n_{proc}.

A. Parallelizing the computation of multiple expansion points

There are some different ways to parallelize the computation of time-expensive routines. From the hardware point of view, one can use either a local multi-core desktop or a cluster of computers, called nodes, linked in a computer network or a combination of both of them (hybrid distributed/ shared memory-based implementations). On the other hand, there are several software environments allowing high performance, parallel programming, such as the MATLAB(R) Parallel Computing Toolbox (PCT) for local desktop, and the MATLAB Distributed Computing Server for cluster workstations. Another very popular platform is the OpenMP, an API that supports multiplatform shared memory multiprocessing, which together with MPI (Message Passing Interface, a computer communications protocol for parallel computation) can be used for cluster computers (an example of the use of this latter combination was done by Ciuprina, et al., in [18], where a MOR technique for multiple expansion points is presented).

For our numerical results, we used the MATLAB Parallel Computing Toolbox. It allows to solve computationally and data-intensive problems using multi-core processors, GPUs, and computer clusters. High-level constructs-like parallel for-loops, special array types, and parallelized numerical algorithms let one parallelize MATLAB applications without CUDA(R) (Compute Unified Device Architecture, a parallel computing platform and programming model created by NVIDIA) or MPI programming. The toolbox provides a maximum of twelve parallel "workers" (MATLAB computational engines) to execute applications locally on a multi-core desktop. With slightly changing the code, one can run the same application on a computer cluster or on a grid computing service.

More in detail, we used the parallel for-loop (*parfor*), instead of the standard *for* loop, in the computation of the frequency responses of the original system (in Algorithm 1), as well as in the generation of the projection matrices for each subinterval (with the function *parSubInt* described in Algorithm 2). It runs loop iterations in parallel on

a pool of parallel workers using the parfor language construct, allowing several processors to execute individual loop iterations simultaneously. Restriction on parallel loops is that no iterations be allowed to depend on any others (in our case, each processor computes the original frequency responses at a certain number of frequency samples and works on its own subinterval, resulting in what is called an embarrassingly parallel problem, since there is no communication between each worker); besides, the body of the parfor has to be a computationally intensive routine, such that its simulation time far exceeds the one needed for the transfer of very-large data from the client to each workers, and vice versa.

Algorithm 2 Function V_{subInt}= parSubInt($subInt, freqResp, G, C, B, L, D, fStart, fStop, tol$)

$\omega_0 = \min(subInt)$;
$V_i = firstMomPRIMA(G, C, B, \omega_0, nMom = 1)$;
compute the frequency responses of the original model in $\omega_{0,1} = (\omega_0 + dist_{farPts})$ and store it;
compute the error in $\omega_{0,1}$;
while $err(\omega_{0,1}) > tol$ **do**
  $V = higherMomPRIMA(G, C, B, \omega_0, nMom + +, lastMom)$;
  $V_i = [V_i, V]$;
  compute the current ROM;
  compute the error in $\omega_{0,1}$;
end while
$V_{subInt} = V_i$;
$cycle = 1$;
while $cycle$ **do**
  for $i = 1 : size(subInt, 2)$ **do**
    compute the error for each test-points in the subinterval, using $freqResp$;
  end for
  select that frequency point ω_i exhibiting the maximum error value ($maxVal$) among the others;
  if $maxVal > tol$ **then**
    $V_i = firstMomPRIMA(G, C, B, \omega_i, nMom = 1)$;
    compute the frequency responses of the original model in $\omega_{i-1,i} = (\omega_i - dist_{farPts})$ and $\omega_{i,i+1} = (\omega_i + dist_{farPts})$ and store it;
    compute the errors in $\omega_{i-1,i}$ and $\omega_{i,i+1}$;
    while $err(\omega_{i-1,i}) > tol$ OR $err(\omega_{i,i+1}) > tol$ **do**
      $V = higherMomPRIMA(G, C, B, \omega_i, nMom + +, lastMom)$;
      $V_i = [V_i, V]$;
      re-orthogonalization of V_i;
      compute the current ROM;
      compute the errors in $\omega_{i-1,i}$ and $\omega_{i,i+1}$;
    end while
    $V_{subInt} = [V_{subInt}, V_i]$;
  else
    $cycle = 0$;
  end if
end while
re-orthogonalization of V_{subInt};
return V_{subInt};

The first step is to run a pool of n_{proc} MATLAB sessions for parallel computation, with the command *matlabpool open* n_{proc}, which connects the pool to the client. Then, to see the benefit of the parallelization, we can run the sequential algorithm, i.e., calculating the frequency responses for all the n_{FS} samples in a sequential fashion and working on the original frequency range and compare the obtained simulation time with that from the parallel version, evaluating the speedup factor with (7) and the efficiency of the parallelization with (8), as the number of available parallel processors n_{proc} increases, to analyze the scalability of the method.

IV. NUMERICAL EXPERIMENTS

The proposed algorithm is implemented in MATLAB r2012b, running on a shared-memory local desktop equipped with an Intel i7 Quad-Core Processor, CPUs operating at 3.50 GHz, with 16 GB of RAM available, on a Windows7 OS and setting the priority of the MATLAB process(es) to Real-Time, the highest one (in the Task Manager). We provide numerical results of eight sparse data sets, some coming from the Max Planck Institute of Magdeburg, others free downloadable from the SLICOT benchmarks ([19]) and from the Joost Rommes' homepage ([20]). We set a tolerance $tol = 10^{-2}$ for all the data sets, which is considered reasonable for engineering applications (it can be changed); the number of linearly distributed frequency test points was set to $n_{FS} = 96$.

Figure 2 shows magnitude, phase and error spectra of the transfer function $H_{1,4}$ of an interconnect model with $n = 980$ internal state and $m = 4$ inputs and outputs, of both the original system evaluated in 2000 frequency points (as for all the other data sets) and the reduced model of order $k = 32$, obtained with the sequential multipoint PRIMA algorithm. For this data set, the algorithm retained five expansion points (which are plotted in the module picture, translated in angular frequencies). Figure 3 shows the same dynamics of the previous model (we do not provide the phase dynamic, it is identical to that of Fig. 2), obtained from the parallel execution with two processors. As said in the previous section, one can note that the

number of retained expansion points is increased (eight in this case), resulting in a slightly bigger order of reduction $k = 44$. For this model, this has also resulted in a higher accuracy of the final ROM, as one can see in the picture of the error.

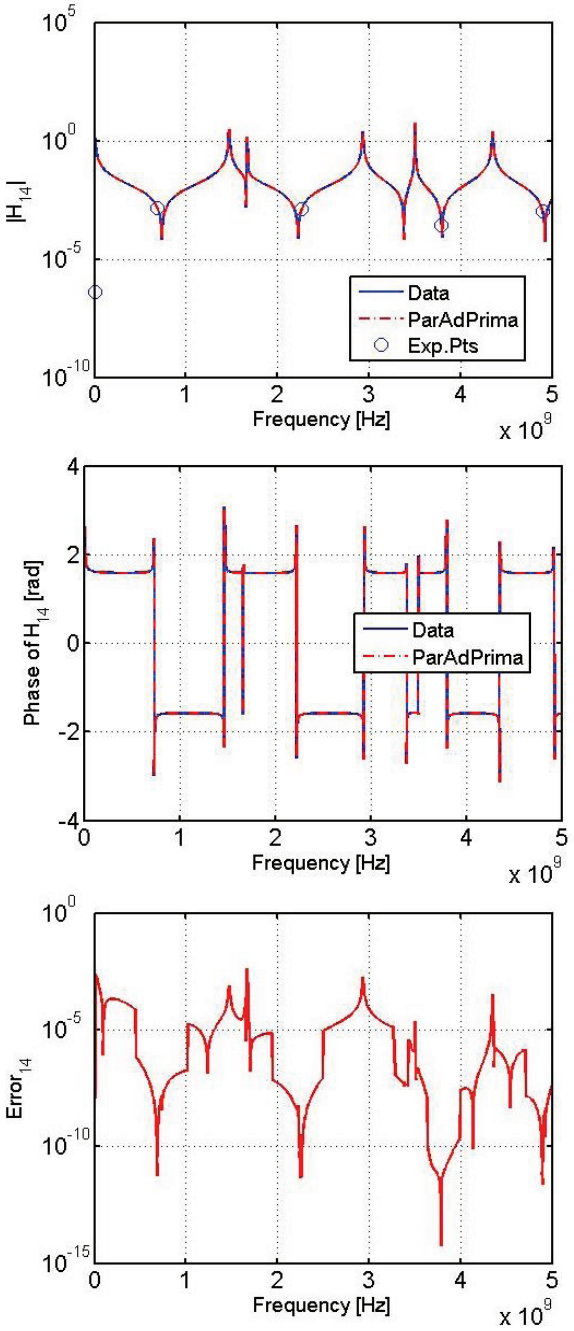


Fig. 2. Magnitude (top), error (middle) and phase (bottom) spectra of the transfer impedance of $H_{1,4}$ an interconnect model with dimension $n = 980$, running the sequential method.

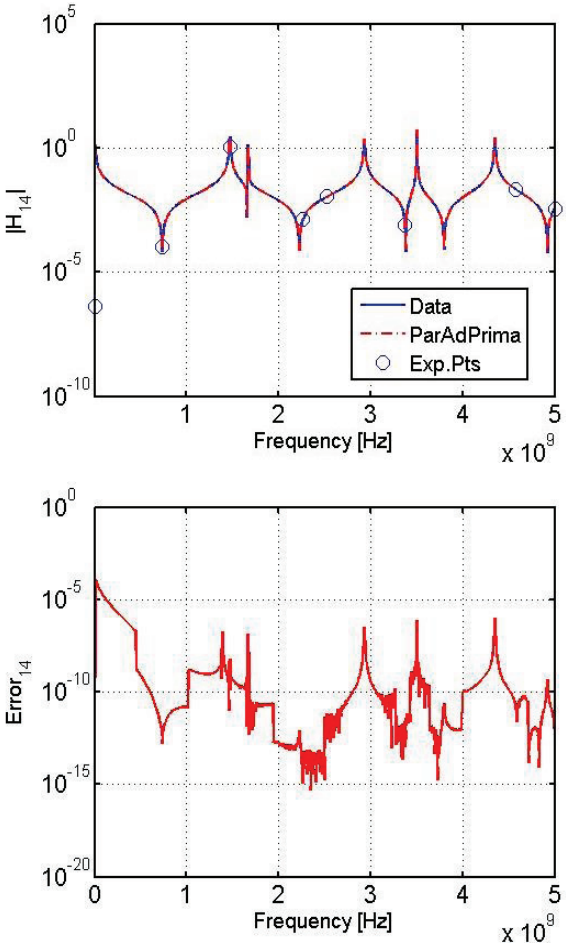


Fig. 3. Magnitude (top) and error (bottom) spectra of the transfer impedance $H_{1,4}$ of an interconnect model with dimension $n = 980$, running the parallel method with two processors.

Figure 4 plots magnitude, phase and error spectra of the transfer function $H_{4,3}$ of an interconnect model of dimension $n = 13309$ and with $m = 8$ inputs and outputs, with a ROM of dimension $k = 246$, retaining thirteen expansion points with the sequential execution. Figure 5 shows magnitude (top) and error spectra (bottom) of the same interconnection of Fig. 4, obtained with two parallel processors, resulting in an order of reduction $k = 256$ with sixteen retained expansion points. Comparing the error plots, one can note that for this interconnection, the parallel method did not result in an improvement of the accuracy, since there were few redundancies of the retained expansion points as compared to the sequential case. Indeed, in the sequential execution, the

algorithm selected points quite scattered on the frequency axis, such that with the parallel version each subinterval showed a bigger degree of independence of the other ones, compared to the model with $n = 980$.

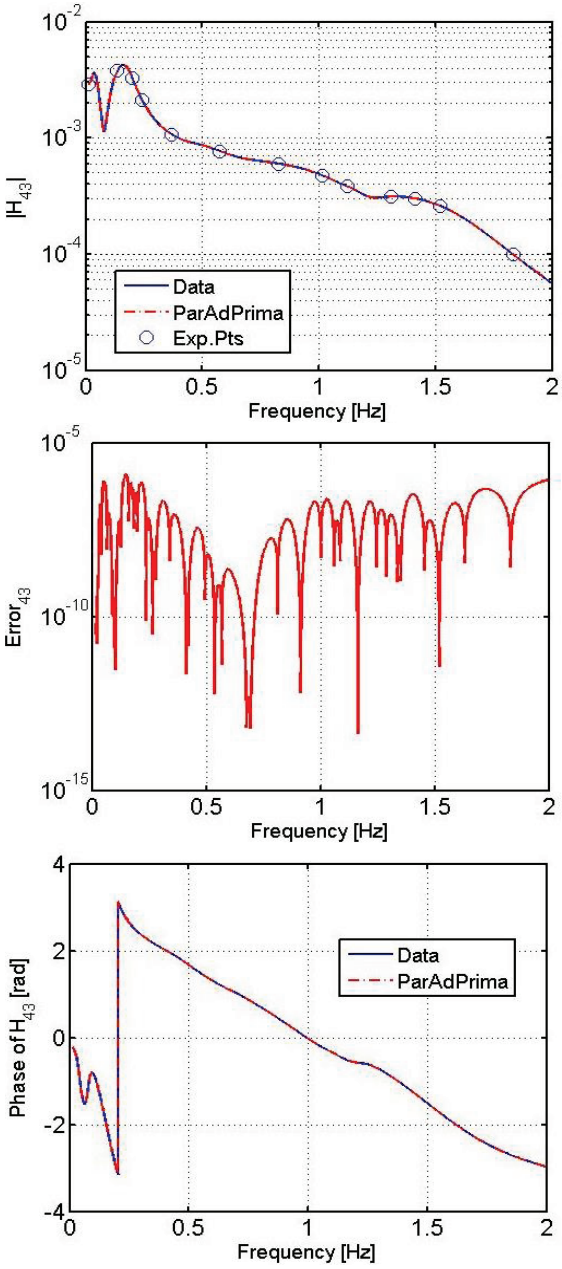


Fig. 4. Magnitude (top), error (middle) and phase (bottom) spectra of the transfer impedance $H_{4,3}$ of an interconnect model with dimension $n = 13309$, running the sequential method.

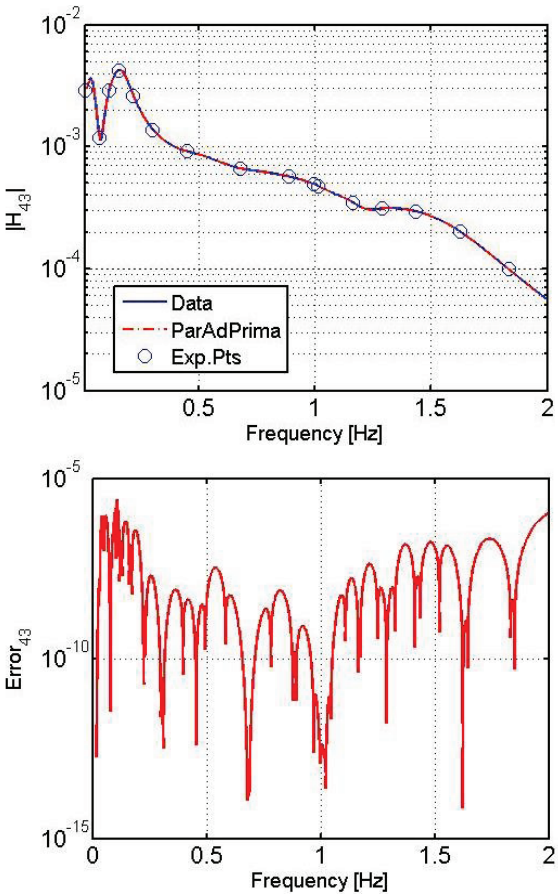


Fig. 5. Magnitude (top) and error (bottom) spectra of the transfer impedance $H_{4,3}$ of an interconnect model with dimension $n = 13309$, running the parallel method with two processors.

Finally, Table 1 provides for each data set the size n of the original model and the number of inputs/outputs m (the data set with dimension $n = 40337$ has two inputs and just one output), the simulation time results t_{seq} of the sequential version and t_{par2}, t_{par4} and t_{par8} of the parallel one with 2, 4 and 8 parallel processes, respectively, all in seconds (to analyze the strong scalability of the algorithm), Speedup (*SP*) and Efficiency (*Eff*) of the parallelization for each number of used processes; the last columns reports the order k_{seq}, k_2, k_4 and k_8 of the reduced models resulting from executing the sequential algorithm and the parallel with 2, 4 and 8 processors, respectively.

Table 1. Table of results: $tol = 0.01$, $n_{FS} = 96$, $n_{proc} = 2$, 4 and 8, respectively (all times are in seconds)

n/m	t_{seq}	t_{par2}- SP-Eff	t_{par4}- SP-Eff	t_{par8}- SP-Eff	$k_{seq,2,4,8}$
980/4	12.35	7.32- 1.69- 0.84	5.28- 2.34- 0.58	6.15- 2.01- 0.25	32-44- 66-85
6134/1	1.21	0.90- 1.34- 0.67	0.75- 1.61- 0.40	0.85- 1.43- 0.18	14-18- 23-26
4863/22	$8.00 \cdot 10^2$	$5.08 \cdot 10^2$- 1.57- 0.79	$3.78 \cdot 10^2$- 2.12- 0.53	$3.91 \cdot 10^2$- 2.05- 0.26	66-94- 149-212
9223/18	$3.48 \cdot 10^2$	$2.02 \cdot 10^2$- 1.72- 0.86	$1.45 \cdot 10^2$- 2.40- 0.60	$1.72 \cdot 10^2$- 2.02- 0.25	36-54- 68-82
13309/8	$1.48 \cdot 10^2$	$0.83 \cdot 10^2$- 1.78- 0.89	$0.44 \cdot 10^2$- 3.36- 0.84	$0.44 \cdot 10^2$- 3.36- 0.42	246- 256- 295-346
21128/4	$1.30 \cdot 10^2$	$0.53 \cdot 10^2$- 2.45- 1.23	$0.35 \cdot 10^2$- 3.71- 0.92	$0.34 \cdot 10^2$- 3.71- 0.46	204- 242- 301-371
24566/1	5.16	3.40- 1.52- 0.76	2.67- 2.31- 0.58	2.96- 1.74- 0.22	16-20- 25-29
40337/2	$2.51 \cdot 10^2$	$2.60 \cdot 10^2$- 0.96-0	$1.41 \cdot 10^2$- 1.57- 0.39	$0.79 \cdot 10^2$- 3.17- 0.40	164- 184- 218-272

A. Consideration on the parallelization

In this subsection, we describe the CPU simulation times obtained from the parallelization of the execution done on the multiple subintervals, using respectively 2, 4 and 8 cores, compared to the sequential one.

In Table 1, it can be observed that the speedup factor (and then the efficiency, too) in the case of two parallel processes is significant for almost all the data sets, but decreases as the number of involved parallel processes rises; thus, it seems the strong scalability of the algorithm is not satisfied, i.e., when increasing the number of available parallel processors, the efficiency of the parallelization being close to 1. The slowdown mostly characterized the cases with four and eight parallel processes. The cause was not the (unavoidable) sequential part of the algorithm (since each subinterval works independently of the others, the sequential part mainly characterizes the final re-orthogonalization in Algorithm 1), but an unbalanced workload issue and the way the data are made available to each parallel workers before starting the computation on them.

More in detail, using a shared-memory multi-core platform, when calling for, say, two (physical) workers core0 and core1 for the parallelization ($n_{proc} = 2$), then having two subintervals, the memory demand increases (each core needs to retrieve data to/from the hierarchical memories), then there is a data-bus contention: the data from the main memory pass through the bus and core0 (or core1) receives them, and thus starts the computation. After core0 (or core1) receives the data, the other worker can then receive the same data, and starts its processing. The waiting time for the second processors to receive the large size data can slowdown the overall parallel execution. And the higher the number of parallel workers competing for the data-bus is, the more the time is wasted in the waiting process. Mostly, during the computation of a parallel task, if the amount of arithmetic operations cannot be entirely performed in one pass (i.e., when the cache available for each core cannot accommodate all the needed data), then the memory demands can be more expensive. However, we experienced this problem mostly when analyzing dense matrices, for which the data movement was more dramatic, as the size of the original problem increases.

Regarding the simulations with eight parallel processes, efficiency-diminishing was the lack of physical computational resources, namely the cores. Indeed, we ran simulations with eight cores thanks to the HyperThreading(R) technology, which our computer is equipped with. It allows half of the cores to only exist in a virtual way (there are eight logical cores, but just four are physical); thus, there are not sufficient physical hardware resources on our computer and the memory bandwidth is insufficient to provide all required data on time.

Even though we experienced this loss of efficiency, there are remedies. To limit the data-bus contention, one can use distributed-memory local desktop, where each core has its own main memory, and thus, a private bus which links together these two components. Of course, the communication between processes is more sophisticated, but for this embarrassingly parallel problem, which does not need inter-processes exchange of data, it may be suitable.

Using such a distributed architecture, we can previously send just once the data describing the original model (1), and the related portion of frequency test-points (the subinterval). After the desired tolerance is satisfied in each subinterval,

each core needs only to send the computed projection-matrix to the client, which gathers these pieces of information and performs the last orthogonalization. One can also use clusters and network workstation, which nowadays are quite available to access to.

The last data set, instead, shows a zero value for the speedup with two processes, meaning the sequential version was at least as fast as the parallel one. This was due to an unbalanced workload issue. In Figs. 6 and 7, we can see how the retained expansion points are located along the angular frequency axis (horizontal axis). From the sequential algorithm (leftmost picture), most of the points were located on the left-half of the entire frequency range, and in those points a high number of moments were matched.

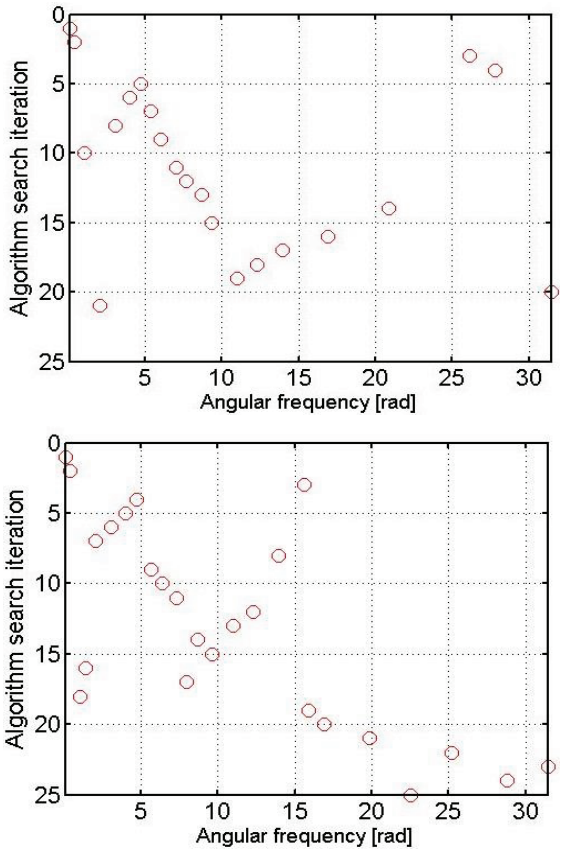


Fig. 6. Retained expansion points by the sequential algorithm (top) and by the parallel one with 2 processors (bottom), respectively, for the model with dimension $n = 40337$.

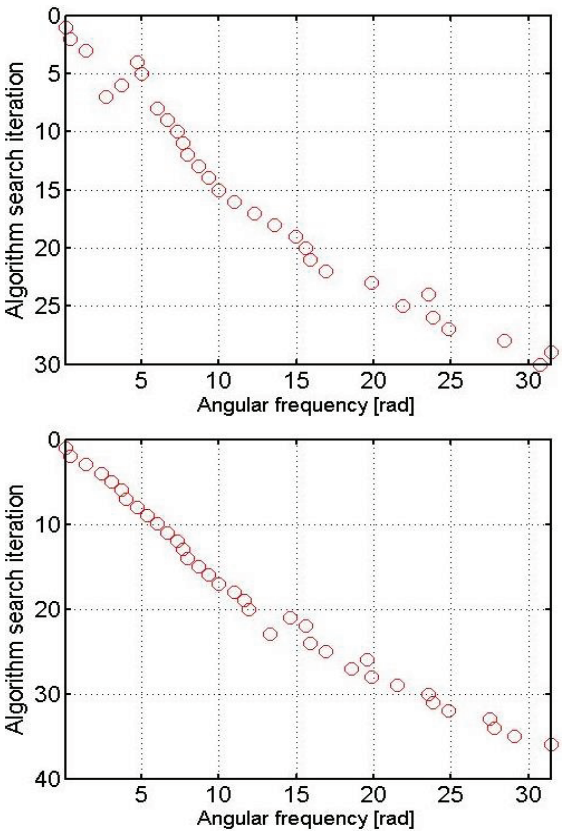


Fig. 7. Retained expansion points by the parallel algorithm with 4 processors (top) and with 8 (bottom), respectively, for the model with dimension $n = 40337$.

Then, when we divide the range into two subintervals, operating on them in parallel (second plot from left in Figs. 6 and 7), the processor responsible for the left subinterval had the biggest workload, in terms of the number of retained expansion points and that of matched moments for each point, such that the other processor, which ended before its execution, remained in the idle state until the operations on the other subinterval stopped. Thus, the slowdown was also caused by the slowest processor execution (with the biggest workload).

Indeed, the retained expansion points on the leftmost subinterval was characterized by the highest number of matched moments (computed by PRIMA), such that the processors working on the other subintervals had to wait until the slowest execution was ended, which constituted an upper

bound for the speedup. However, as the number of parallel processor increases (second picture from right and the rightmost one), it seems that the final retained points become distributed on the entire range in a more "balanced" way, as the workload of each processor, which determines a better time performance. With four parallel processes, the biggest waiting-time, characterizing the processor with the biggest workload which is also the slowest one, becomes smaller than that obtained from two processors, due to a decreasing of the workload for each core (fewer test-points to analyze), resulting in a greater speedup and efficiency factors (see the fourth last, third last and last row of Table 1), and most important in a significant reduction of the simulation time. With such a parallel method, it is difficult to statically balance the workload to each processor, since we have initially assumed we do not know how the dynamics of the interconnections evolve in the frequency domain, then where the expansions with PRIMA will be executed. Concluding, we recall that as the number of workers increases (thus, the number of independent subintervals), the redundancies, the number of retained expansion points, and thus, the final order of reduction increase, even with a greater speedup of the execution; but the obtained ROM may be a bit more expensive to be simulated by a further analysis. As previously said, there is a compromise between the CPU generation time of the ROM and its order of reduction. However, when the data set involved in the operations has a very-large size, the work-load increases, then working with parallel processes provides benefits in time performance.

Besides, when running the algorithm in the sequential mode, some operations (e.g., the backslash operator in solving a linear system of equations) exploit the implicit parallelism which the common multi-threaded computers are equipped with; instead, when calling the pool of parallel processes with MATLAB, with an explicit parallelism, each of them run in single-thread mode, giving rise to another factor which slightly limited the execution time of the parallel algorithm. But it is the case just for the factorization of large, sparse matrices with small dense sub-matrices, which takes advantage of the multi-threaded BLAS and LAPACK routines, e.g., with the multi-frontal method, originally developed by Davis [21]. This is the reason why we decided to dedicate our attention to sparse datasets, which BLAS is not optimized for. In fact, we also tested our parallel algorithm for dense models, and the time for the decomposition almost doubled in all the cases because of the single-thread mode. Again, this single-thread limitation can be avoided working with a parallel computing hardware/software environment, which supports a hybrid distributed/shared-memory architecture. Thus, very-large sparse matrices can benefit from this explicit and implicit parallelism, as well as one can use this method for full matrices too.

Another viable option is represented by Graphics Process Units (GPUs), but unfortunately MATLAB does not allow to work on it with sparse datasets. Besides, the use of a GPU seems to be not useful in this work, since it is characterized by having hundreds of dedicated processing units, which translates into hundreds of frequency subintervals. Then, as said before, this would imply more redundant, expansion points, resulting in a useless big size of the reduced model. Of course, scalability is an important requisite to be satisfied in a parallel algorithm, but it strictly depends on the application and in many cases it is difficult to achieve because of the problems one can easily meet, such as data communication and the diminishing of the workload for each processor, as the number of parallel processes increases.

It is also true that, since time performance was an equally important requisite in this work, together with the way the method searches for expansion points, we should had implemented the codes with a more suitable programming language, such as MPI or Pthread, which have more control on the parallel processes and allow different level of parallelization with distributed memories.

We would remark that, since there are no established methods to search for suitable expansion points in Krylov-based MOR in an adaptive way, the other strong point of this work was the full exploration of the frequency range, which is obviously time-consuming for very-large size datasets; thus, using a moderate number of parallel processors, one can achieve speedup with respect to a sequential search, as one can infer from Table 1, avoiding too many redundancies and a high order of reduction.

Lastly, from empirical results, we experienced that the selection of $n_{FS} = 96$ test-points, for the check of the error, was high enough to reach the desired accuracy on the whole frequency range.

Considering the sequential version of the algorithm, then with one interval, we noted a further tradeoff between the selection of the number of frequency test-points n_{FS} and the order of reduction *k* of the final ROM, keeping constant the desired accuracy *tol*.

When increasing n_{FS}, a higher number of test-points are available. It translates into a smaller distance between two consecutive points. Thus, higher-order moments may be no longer needed to be computed for a selected frequency point ω_i (exhibiting the maximum error in the current step of the algorithm), since both its neighbor $\omega_{i-1,i}$ and $\omega_{i,i+1}$ are close enough to it and their error values are likely less than *tol*, even with few moments matched when making the expansion in ω_i with PRIMA. Besides, it translates into an eventual smaller order of reduction *k*, since just the necessary moments are computed for each point, thanks to a more complete exploration of the entire frequency range of interest, without the need to recur in the computation of higher-order moments. However, we have seen that with the parallel execution, a greater accuracy (smaller error magnitudes) may be reached due to eventual redundancies, so that there may be no need to further increase n_{FS}.

On the other hand, decreasing n_{FS} below a certain threshold may not ensure that the desired accuracy will be reached in all the frequency range, since the adaptivity of the moments, computed at a frequency point ω_i, could fail in reproducing the evolution of the dynamics far away from ω_i. We experienced that for the case with $n_{FS} = 48$ selected (linearly distributed) frequency test-points, for the data set with $n = 40337$ the sequential algorithm, running on the entire interval, was not able to satisfy the accuracy in all the frequency range; i.e., for how we structured the search for the frequency points to be retained for the expansion with PRIMA, when choosing a small n_{FS}, clearly the dinstance between two consecutive test-points ω_{i-1} and ω_i increases, compared to the case $n_{FS} = 96$. Assuming ω_i error value among the others, and considering just the left subinterval $[\omega_{i-1}\ \omega_i]$, if we increase the order of moments until the error in the mid-point $\omega_{i-1,i}$ is less than the tolerance, the desired accuracy is not ensured to be reached in the whole subinterval $[\omega_{i-1,i}\ \omega_i]$ (e.g., in the case of presence of module peaks in that range), as stated in the motivation of the selection of a "relatively" high number of n_{FS}. But as said before, as we increase the processors and then the number of independent subintervals, redundancies occurred and the accuracy increased, satisfying the tolerance. From all our simulations, $n_{FS} = 96$ was always high enough to reach the desired approximation.

B. On the usefulness of multiple expansion points

In this section, we want to show that the higher the number of involved expansion points is, the higher the ensured accuracy in a wide range is. We recall that, the task of reducing a dynamical model while matching a number of moments (and/or the Markov parameters) about a point s_0 can be directly interpreted from a system theoretical point of view and employed to describe the similarity between the original and reduced models based on the following facts [16]:

- With s_0, the reduced and original model have the same DC gain, and steady state accuracy is achieved.
- Small values of s_0 result in a reduced model with a good approximation of the slow dynamics.
- Large values of s_0 (and/or matching the Markov parameters) result in a reduced model approximating the system frequency response at high frequencies.
- When matching some of the moments about different frequency points, a better approximation on a wider frequency band or on a specific frequency band of interest can be achieved.

Besides, choosing a purely imaginary expansion point leads to very good local approximation and to a very slow convergence at all frequencies away from s_0. Even though these facts give an idea on the choice of the expansion points, no specific value of s_0 can be derived based on them [16].

Here, we show that, for a test data set, it is not possible to reach the desired accuracy in all the frequency range of interest, when making the expansion on a single frequency point and increasing the order of moments to match. We

executed PRIMA on a dense data set with dimension $n = 980$ and number of inputs $m = 8$, making the expansion in the middle point ($\omega_0 = \left(\omega_{stop} + \omega_{start}\right)/2$) of the frequency range, and increasing the number of moments until the frequency errors, computed in both the starting and ending frequency points (farthest points from ω_0), are smaller than the fixed tolerance $tol = 0.01$: even matching 123 moments on the mid point, obtaining a ROM of order $k = 984$, which is greater than the initial dimension, the accuracy on the lowest frequency point was still unacceptable (whereas, with our multipoint algorithm, we satisfied the tolerance with an order of reduction $k = 576$ in the whole range. The order of reduction was relatively high since the frequency response of the model is highly resonating). Figure 8 shows a zoom of the module and error spectra of the data sets (8-1) interconnection.

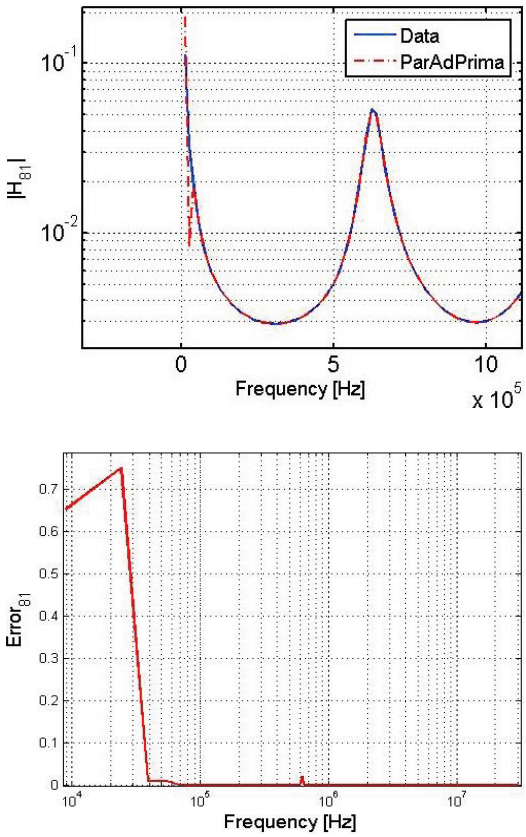


Fig. 8. Zoom of magnitude (top) and error (bottom) spectra of the transfer function $H_{8,1}$ of an interconnect model with dimension $n = 980$.

C. Accuracy evolution plots for a test data set

Lastly, in Figs. 9-22 we show how the accuracy of the ROM evolves according to each step of the algorithm, when it runs sequentially (avoiding to retain redundant expansion points), thus, only considering the entire frequency range without splitting it in subintervals, while a new expansion point (exhibiting the maximum error value) is adaptively retained and added to the previous ones and higher-order moments are eventually matched as well. This was done to provide a correlation between a, say, minimum number of retained expansion points and the location of the peaks of the module dynamics, related to the dominant poles [22] of the original system. We show the plots of the (1-1) interconnection of the SISO model with dimension $n = 6134$.

As one can see from Fig. 22, the retained expansion points ("Exp.Pts" in the legend) for this SISO model are those in proximity of most of the module peaks (e.g., resonances and anti-resonances characterizing the impedance or admittance of an RLC interconnection in the related Bode diagram), which can be related to the dominant poles of the dynamical system [22]. However, for MIMO systems this relationship is less explicit (see the module plot of Fig. 2), due to (eventual) different behaviors among all the interconnections of the transfer function, in the case an expansion point could be retained to reach a local accuracy for an interconnect, whereas it was useless for another one.

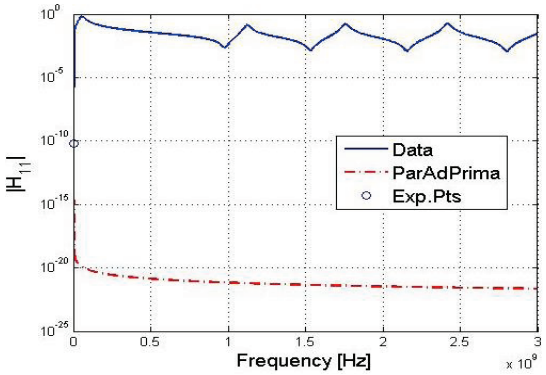


Fig. 9. Accuracy evolution as additional moments, computed in different retained expansion points (s_i), are added to the currently approximated transfer function for the model with dimension $n = 6134$: first moment matched at the expansion point $s_1 = j2\pi 10^{-2}$.

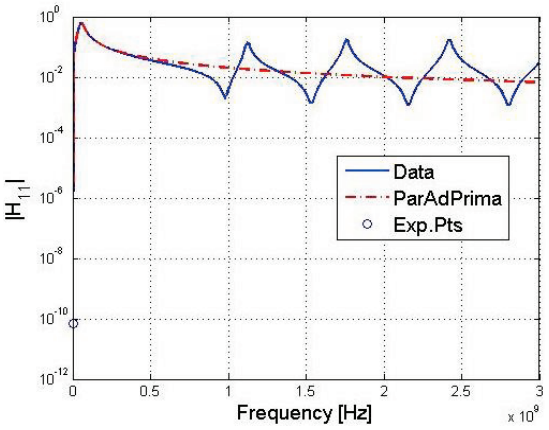


Fig. 10. Second moment matched on the expansion point $s_1 = j2\pi 10^{-2}$.

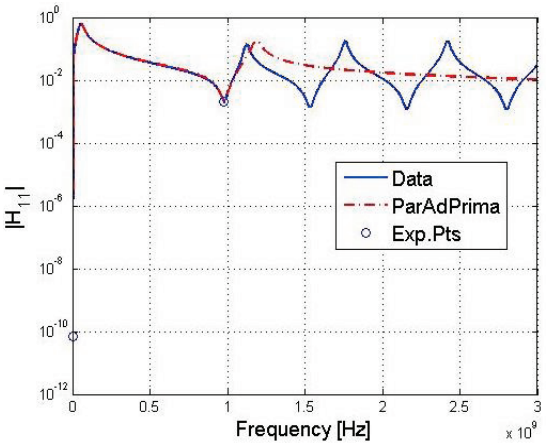


Fig. 11. First moment matched on the expansion point $s_2 = j2\pi 9.7910^{-2}$.

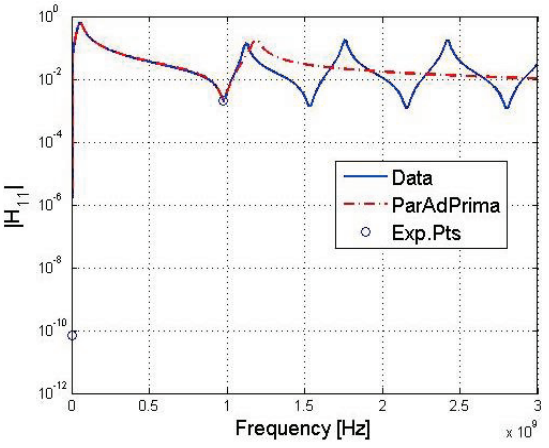
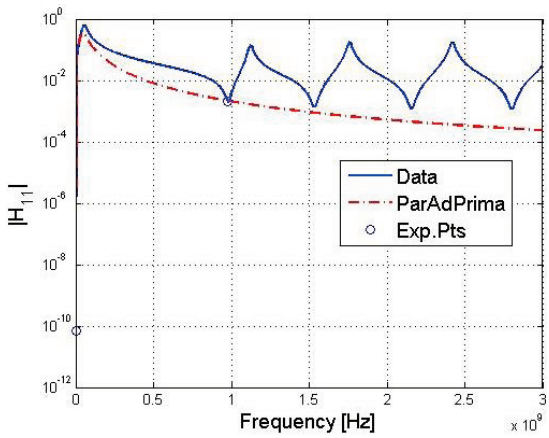


Fig. 12. Second moment matched on the expansion point $s_2 = j2\pi 9.7910^{-2}$.



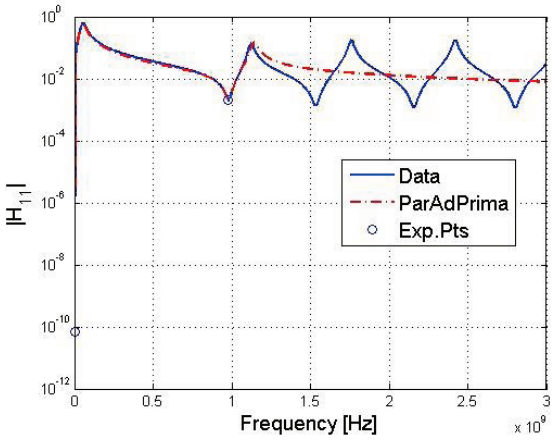


Fig. 13. Third moment matched on the expansion point $s_2 = j2\pi 9.7910^{-2}$.

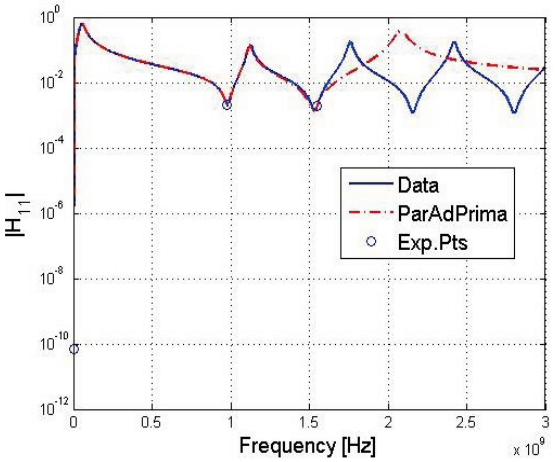


Fig. 14. First moment matched on the expansion point $s_3 = j2\pi 1.5510^{9}$.

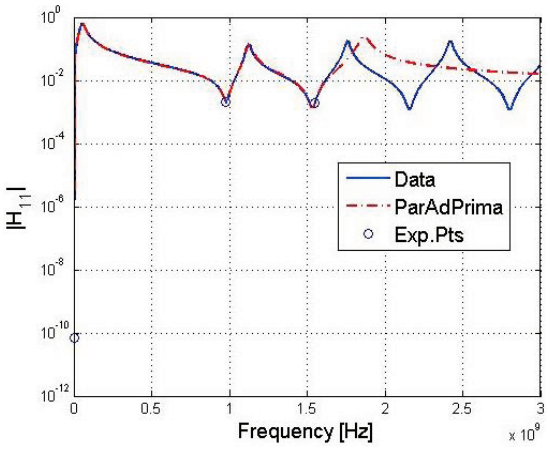


Fig. 15. Second moment matched on the expansion point $s_3 = j2\pi 1.5510^9$.

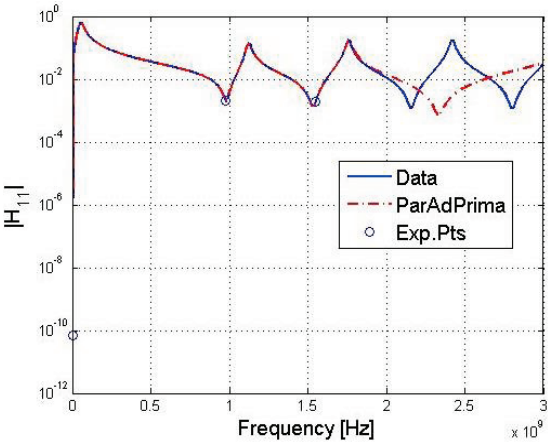


Fig. 16. Third moment matched on the expansion point $s_3 = j2\pi 1.5510^9$.

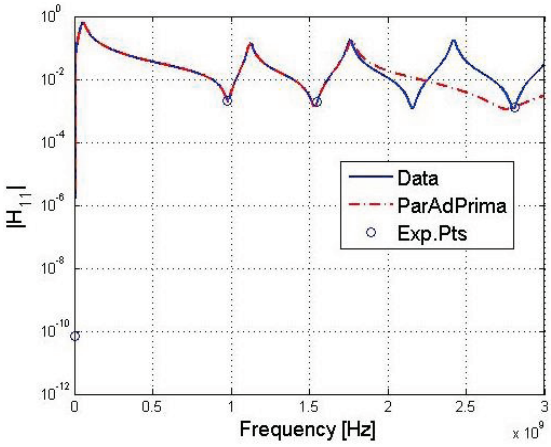


Fig. 17. First moment matched on the expansion point $s_4 = j2\pi 2.8110^9$.

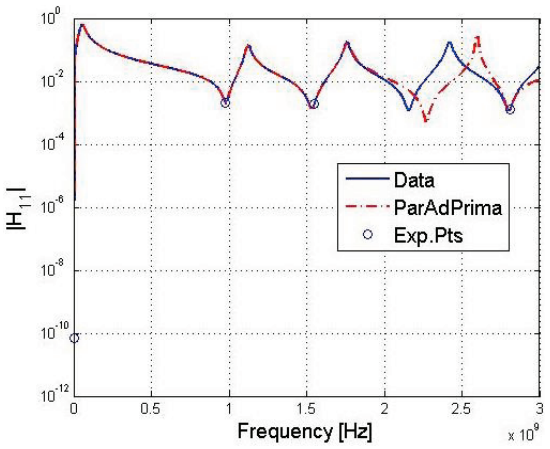


Fig. 18. Second moment matched on the expansion point $s_4 = j2\pi 2.8110^9$.

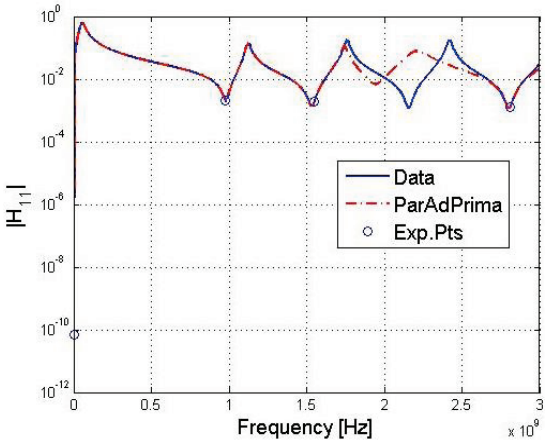


Fig. 19. Third moment matched on the expansion point $s_4 = j2\pi 2.8110^9$.

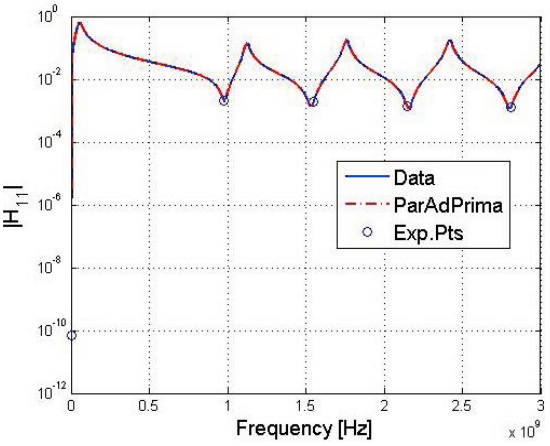


Fig. 20. First moment matched on the expansion point $s_5 = j2\pi 2.1510^9$.

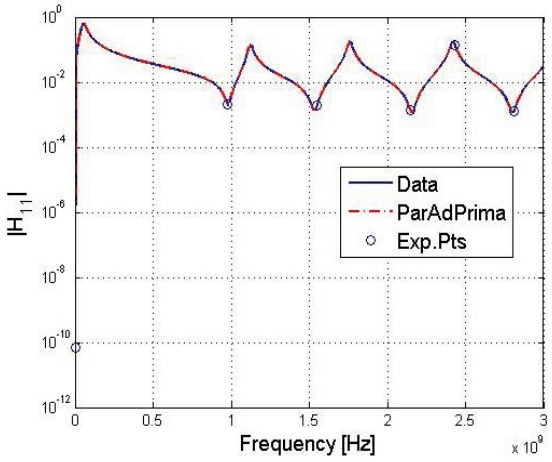


Fig. 21. First moment matched on the expansion point $s_6 = j2\pi 2.4310^9$.

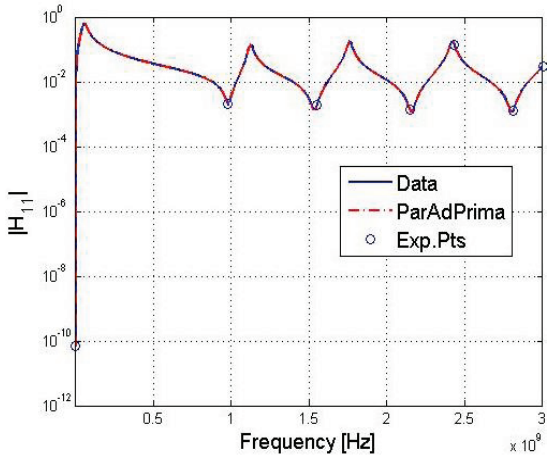


Fig. 22. First moment matched on the expansion point $s_7 = j2\pi 310^9$.

V. CONCLUSION

In this paper, a parallel adaptive multi-point model order algorithm is proposed, based on a full exploration of the frequency range of interest, trying to avoid heuristic searching methods for the expansion points, matching the most important moments of the original model transfer function in order to accurately reproduce its dynamics, without explicitly computing its dominant poles, and reducing the generation time of the ROM by exploiting the architectures of modern multi-core processors. The proposed method is able to achieve a very good approximation of the original transfer function, selecting as expansion points the frequencies in correspondence to which the error, between the response of the original model and that of the ROM, is larger than the desired tolerance, and increasing the number of moments to be matched for each point when necessary.

The numerical experiments have demonstrated that the speedup can be limited by the bus contention in the data transfer between the client and each parallel processor, in a shared-memory environment, but this issue can be handled by using distributed-memory architectures. The unbalanced loads, delegated to each processor, also may cause the performance degradation, but this can be alleviated by increasing the number of processors.

As shown, this method exhibits a slight compromise between the requirements of the ROM's generation time and its order of reduction, i.e., ROM evaluation can be obtained very quickly by slightly increasing the order of the ROM retaining redundant expansion points. However, the time performance will increase, mainly for very large size models.

In the next future, the application of the algorithm to originally-dense data sets like those obtained using integral-equation based methods (e.g., Method of Moments (MoM) or Partial Element Equivalent Circuit (PEEC) method) will be investigated. It will be presented in forthcoming reports.

ACKNOWLEDGMENT

We want to thank some persons for their tips, technical support and exchanged ideas, who contributed to this work. Especially, we are grateful to Lihong Feng, Jens Saak, Martin Köhler and Daniele Romano.

REFERENCES

[1] F. Ferranti, M. Nakhla, G. Antonini, T. Dhaene, L. Knockaert, and A. Ruehli, "Multipoint full-wave model order reduction for delayed PEEC models with large delays," *IEEE Transactions on Electromagnetic Compatibility*, 2011.

[2] R. F. Harrington, "Field computation by moment methods," *Malabar: Krieger*, 1982.

[3] A. E. Ruehli, "Equivalent circuit models for three dimensional multiconductor systems," *IEEE Transactions on Microwave Theory and Techniques*, vol. MTT-22, no. 3, pp. 216-221, March 1974.

[4] W. Schilders, "Introduction to model order reduction," *In Model Order Reduction: Theory, Research Aspects and Applications*, pp. 3-32, 2008.

[5] "The model order reduction wiki," *http://morwiki.mpi-magdeburg.mpg.de/morwiki/index.php/Main Page*, 2013.

[6] L. Feng, J. G. Korvink, and P. Benner, "A fully adaptive scheme for model order reduction based on moment-matching," *Max Planck Institute Magdeburg Preprint MPIMD/12-14*, September 2012, available from http://www.mpi-magdeburg.mpg.de/preprints/.

[7] E. Chiprout and M. Nakhla, "Analysis of interconnect networks using complex frequency hopping (CFH)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 2, pp. 186-200, 1995.

[8] L. Pillage and R. Rohrer, "Asymptotic waveform evaluation for timing analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 4, pp. 352-366, 1990.

[9] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: passive reduced-order interconnect macromodeling algorithm," *IEEE Transactions on Computer-Aided Design*, vol. 17, no. 8, pp. 645-654, August 1998.

[10] C. Ho, A. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Transactions on Circuits and Systems*, vol. 22, no. 6, pp. 504-509, June 1975.

[11] G. Barbella, F. Perotti, and V. Simoncini, "Block krylov subspace methods for the computation of structural response to turbulent wind," *Computer Methods in Applied Mechanics and Engineering*, vol. 200, no. 23-24, pp. 2067-2082, 2011.

[12] M. Kamon, N. Marques, L. Silveira, and J. White, "Automatic generation of accurate circuit models of 3-D interconnect," *IEEE Transactions on Comp. Packag. Manufact. Technol. B*, vol. 21, no. 3, pp. 225-240, 1998.

[13] R. Freund, "Recent advances in structure-preserving model order reduction," *In Simulation and Verification of Electronic and Biological Systems*, P. Li, L. M. Silveira, and P. Feldmann, Eds., *Springer Netherlands*, pp. 43-70, 2011.

[14] H. J. Lee, C. C. Chu, and W. S. Feng, "An adaptive-order rational arnoldi method for model-order reductions of linear time-invariant systems," *Linear Algebra and its Applications*, vol. 415, no. 2-3, pp. 235-261, 2006.

[15] M. Creeland and W. L. Goffe, "Multi-core cpus, clusters, and grid computing: a tutorial," *Computational Economics*, vol. 32, no. 4, pp. 353-382, 2008.

[16] R. Eid, H. Panzer, and B. Lohmann, "How to choose a single expansion point in krylov-based model reduction?," *Lehrstuhl f¨ur Regelungstechnik*, TU M¨*unchen*, Tech. Rep., November 2009.

[17] D. P. Bertsekas and J. N. Tsitsiklis, "Parallel and distributed computation: numerical methods," Upper Saddle River, NJ, *USA: Prentice-Hall, Inc*., 1989.

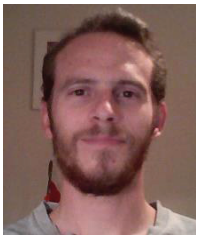
[18] I. A. Lazar, G. Ciuprina, and D. Ioan, "Effective extraction of accurate reduced order models for hf-ics using multi-cpu architectures," *Inverse Problems in Science and Engineering*, vol. 20, no. 1, pp. 15-27, 2012.

[19] "SLICOT: benchmark examples for model order reduction," http://www.slicot.org/index.php?site=benchmodred.

[20] "Homepage of Joost Rommes," https://sites.google.com/site/rommes/software.

[21] T. A. Davis, "Algorithm 832: umfpack v4.3-an unsymmetric pattern multifrontal method," *ACM Trans. Math. Softw*., 2004.

[22] J. Rommes and N. Martins, "Efficient computation of transfer function dominant poles using subspace acceleration," *IEEE Transactions on Power Systems*, vol. 21, no. 3, pp. 1218-1226, 2006.



Giovanni De Luca received his M.Sc. degree in Computer Science and Control System Theory Engineering in 2012 from the Univesity of L'Aquila. He worked in the Dept. of Geniuses of Automated Production, Ecole de Technologie Superieure (ETS) in Montréal (CA), winning the scholarship for the preparation of the final thesis abroad 2010/2011. In 2013, he was a Visiting Researcher in the Max Planck Institute in Magdeburg (GE), for a period of seven months, working on Model Order Reduction techniques and parallelization theoretical aspects. Since March 2014, he is a Ph.D. candidate at the Technical University of Eindhoven (NL), with the goal of accelerating the transient simulation of complex circuits.

Giulio Antonini received his Laurea degree (summa cum laude) in Electrical Engineering in 1994 from the University of L'Aquila and the Ph.D. degree in Electrical Engineering in 1998 from the University of Rome "La Sapienza." Since 1998, he has been with the *UAq EMC Laboratory*, Dept. of Industrial Engineering and Computer Science of the University of L'Aquila, where he is currently Full Professor. His research interests focus on EMC analysis, numerical modeling and in the field of signal integrity for high-speed digital

systems. He has authored or co-authored more than 250 technical papers and 3 book chapters. Furthermore, he has given keynote lectures and chaired several special session at international conferences. He holds one European Patent.

Peter Benner is one of the Directors at the Max Planck Institute for Dynamics of Complex Technical Systems in Magdeburg and Head of the Computational Methods in System and Control group there. His research activities include numerical linear algebra, model reduction and systems approximation, parallel algorithm, linear-quadratic optimization, robust stabilization of linear and non-linear systems, and control of instationary PDEs.