Exploiting Parallelism for Improved Automation of Multidimensional Model Order Reduction

Jorge Fernández Villena, *Member, IEEE,* and Luís Miguel Silveira, *Senior Member, IEEE*

***Abstract*—This paper addresses the issue of automatically generating reduced order models of very large multidimensional systems. To tackle this problem we introduce an efficient parallel projection based model order reduction framework for parameterized linear systems. The underlying methodology is based on an automated multidimensional sample selection procedure that maximizes effectiveness in the generation of the projection basis. The parallel nature of the algorithm is efficiently exploited using both shared and distributed memory architectures. This leads to a highly scalable, automatic, and reliable parallel reduction scheme, able to handle very large systems depending on multiple parameters. In addition, the framework is general enough to provide a good approximation regardless of the model's representation or underlying nature, as will be demonstrated on a variety of benchmark examples. The method provides the potential to tackle, in an automatic fashion, extremely challenging models that would be otherwise difficult to address with existing sequential approaches.**

***Index Terms*—Multidimensional sampling, parallel model order reduction, parameterized systems.**

I. Introduction

MODEL ORDER REDUCTION (MOR) methodologies are a set of techniques aimed at compressing the information contained in detailed models representing physical effects [2], [3] allowing for more efficient simulations. The most relevant linear MOR techniques can be broadly characterized into those based on subspace generation and projection [4], [5], and those based on balancing techniques [6]. MOR methodologies have also been proposed to handle parameterized systems, where the response depends on a set of parameters modeling operating, environmental and process variations that affect the underlying physical system. The goal of these parameterized model order reduction (pMOR) methods is to generate an approximate reduced order model (ROM) with equivalent I/O response for the whole multi-dimensional space of interest. Most of the pMOR methods rely on the generation of a low order subspace that spans the solution of the system, followed by a projection of the original system into that subspace. Different techniques propose different methods for the subspace generation, either based on multidimensional moment matching [7]–[9], or in multipoint sampling approaches [10], [11].

Either way, the application of such approaches still suffers from some drawbacks that prevent a more widespread adoption by the EDA industry. Two of the more relevant stumbling blocks are related to the automation of the procedure and the excessive cost of handling models of very large scale networks. Furthermore, moment matching methods are unable to efficiently handle a large number of parameters, whereas sampling on a large number of dimensions may became too expensive.

Nowadays the availability and extensive use of multicore architectures, GPUs, and distributed environments are revolutionizing the implementation of algorithmic solutions to relevant problems, allowing for widespread low-cost high-performance parallel approaches able to overcome some of the existing bottlenecks in certain applications. In the framework of MOR there has been relatively little work devoted to the usage of such architectures. Moreover, most of these efforts have been directed toward the parallelization of the underlying linear algebra routines [12]–[14]. Little effort has been devoted in particular to the application of such architectures to the pMOR problem, even though it is one of the most challenging tasks, and could readily take advantage of the availability of such parallel environments.

This paper addresses the automatic generation of reduced models of general parameterized linear systems, which can be simulated more efficiently than the original ones while maintaining the same IO response for a wide parameter and frequency range. In particular, we focus on the development of efficient parallel algorithms that will speed up the model build-up and generation step. Unlike most previously proposed approaches, instead of focusing on the parallelization of the underlying linear algebra routines, our target is a parallelization at a higher level, that will prove to be very effective and scalable. The framework proposed relies on multidimensional sample-based methods [10] for the generation of a suitable basis, combined with an automated sample selection approach such as the one in [11]. It will be shown that the procedure is robust and reliable, provides considerable automation and shows good results, regardless of system representation and characteristics, without degrading the efficiency. As a result,

Manuscript received February 6, 2011; revised June 3, 2011; accepted August 10, 2011. Date of current version December 21, 2011. This work was supported by Fundação para Ciencia e Técnologia (FCT) (INESC-ID Multiannual Funding) through the PIDDAC Program Funds and by FCT Grant SFRH/BPD/71632/2010. Preliminary results appeared in [1]. This paper was recommended by Associate Editor P. Li.

J. F. Villena is with the Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa, Lisbon 1000-029, Portugal (e-mail: jorge.fernandez@inesc-id.pt).

L. M. Silveira is with the Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa, Lisbon 1000-029, Portugal, and also with Instituto Superior Técnico, Technical University of Lisbon, Lisbon 1000-029, Portugal (e-mail: lms@inesc-id.pt).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2011.2167510

it enables the otherwise difficult reduction of large challenging models depending upon multiple parameters.

This paper is structured as follows. In Section II an overview of the MOR paradigm is presented, along with a discussion of existing approaches relevant for this paper, their pros and cons and the opportunities for parallelization, at different levels, that they offer. In Section III the initial sequential approach is introduced, and in Section IV the proposed implementation is presented, along with a study and discussion of its complexity and relevant computational issues. In Section V several examples are shown that illustrate the performance of the proposed technique and in Section VI conclusions are drawn.

II. Background

A. Parameterized Systems and Model Order Reduction

A considerable number of MOR techniques are geared toward the reduction of a state space linear time-invariant system representation, where the output y is related to the input u via some inner states x. When parametric variations are taken into account, the system is represented as a parameterized state-space descriptor, with an associated frequency-domain transfer function

$$\begin{array}{c} C(\lambda)\dot{x}(\lambda) + G(\lambda)x(\lambda) = Bu \qquad y(\lambda) = Ex(\lambda) \\ H(s,\lambda) = E(sC(\lambda) + G(\lambda))^{-1}B \end{array} \tag{1}$$

where $C, G \in \mathbb{R}^{n\times n}$ are, respectively, the dynamic and static matrices, $B \in \mathbb{R}^{n\times m}$ is the matrix that relates the input vector $u \in \mathbb{C}^m$ to the inner states $x \in \mathbb{C}^n$ and $E \in \mathbb{R}^{p\times n}$ is the matrix that links those inner states to the outputs $y \in \mathbb{C}^p$, and $H(s,\lambda) \in \mathbb{C}^{p\times m}$ is the transfer function matrix. We assume here, as is common, that the elements of C and G, as well as the states x, depend on a set of Q parameters $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_Q] \in \mathbb{R}^Q$ which model the effects of the uncertainty. Usually, but not always, the input (B) and output (E) matrices do not depend on the parameters.

The representation of parametric dependence is often obtained via first order sensitivity computation of the discretized elements with respect to the parameters [15]–[17]. Therefore, matrices C and G in (1) can be represented in a polynomial form with respect to the parameters.

The most common procedure to obtain an accurate and structurally similar ROM is to use an orthogonal projection on the polynomial matrix representation (as advocated in [7]), leading to a structurally equivalent reduced polynomial formulation. Standard pMOR methodologies rely on the generation of a suitable low order subspace (spanned by the basis $V \in \mathbb{R}^{n\times q}$), where the original system matrices $C(\lambda)$, $G(\lambda)$, B and E are subsequently projected. Then a reduced model such as (3) can be obtained, that captures the behavior of the system under parameter variations

$$\begin{array}{c} \widehat{C}(\lambda) = V^T C(\lambda) V \quad \widehat{G}(\lambda) = V^T G(\lambda) V \\ \widehat{B} = V^T B \quad \widehat{E} = EV \quad x(s,\lambda) = V\widehat{x}(s,\lambda) \end{array} \tag{2}$$

where $V \in \mathbb{R}^{n\times q}$ spans the projection subspace of reduced dimension q, and $\widehat{C}, \widehat{G} \in \mathbb{R}^{q\times q}$, $\widehat{B} \in \mathbb{R}^{q\times m}$, $\widehat{E} \in \mathbb{R}^{q\times p}$, and $\widehat{x} \in \mathbb{C}^q$ define the reduced order model of dimension $q \ll n$ (q is the reduced order). These matrices provide an approximated transfer function

$$\widehat{H}(s,\lambda) = \widehat{E}(s\widehat{C}(\lambda) + \widehat{G}(\lambda))^{-1}\widehat{B} \approx H(s,\lambda) \qquad \forall\{s,\lambda\}. \tag{3}$$

To ensure the accuracy of the ROM, the basis $V \in \mathbb{R}^{n\times q}$ must be able to capture the behavior of $x(s,\lambda)$ for the relevant $\{s,\lambda\}$ space, or in other words

$$x(s,\lambda) \approx \sum_{i=0}^{q} \alpha_i(s,\lambda) V_i \qquad \forall\ \{s,\lambda\} \tag{4}$$

where V_i is the ith column of the projector V, and $\alpha_i \in \mathbb{C}$. Different projection based pMOR approaches have been presented, which differ in how to generate the matrix V.

B. Multidimensional Moment Matching

Most of the methods in the literature choose to extend the standard moment matching paradigm [4] to the multidimensional case [7]–[9]. Therefore they generate a basis V that spans the multidimensional moments of the transfer function around an expansion point, and can be used as a projector. In general, these methods, which rely in local matching around the expansion point, suffer from model oversize when the number of moments to match is high, either because high order is required, or because the number of parameters is large. Also, for efficient implementation, they require the availability of a Taylor series based representation of the system matrices, a feature that is not always present. When this is the case, linearization schemes are necessary, but Taylor series of order higher than one are hard to handle. A different problem faced by these approaches is the lack of automation as it is very difficult to determine the number of moments to match, both in frequency and with respect to the parameters, in order to generate an accurate model.

1) *Basic Operations and Parallelization Potential:* The moment matching approaches usually rely on Arnoldi iterative methods, which are very computationally efficient (see [4], [9] for actual implementation details). The main operations that these methods require, for matching q moments, are a single point evaluation of the system matrix A (at cost $O(z)$ with z the number of non-zeros of A), a single LU matrix factorization (whose cost is problem dependent, and that we will denote as $O(S)$), and q sparse matrix dense block vectors multiplications ($O(qzm)$, with m the number of columns of B), q solves with m right hand sides (with cost dependent on the problem), and q block orthonormalizations (dense operations with approximate overall cost $O(nq^2m^2)$). Due to the iterative nature of the method, parallelizations of these type of algorithms usually rely on the parallelization of the underlying linear algebra routines. Since the dominant cost comes from the LU factorization, parallelization of this routine seems to be the most critical operation. Shared or distributed memory based parallelization such as [18] and [19] may provide good results. The rest of the operations usually rely on efficient linear algebra operations, and thus efficient parallel versions of BLAS-type operations may improve performance. Due to the iterative nature of the method, where computations for matching a given moment q require information from matching

moment $q-1$, parallelization usually implies considerable data dependencies and seemingly discourages further efforts into higher-level parallelization. But in this paper we seek exactly such high-level concurrency in an algorithm, hopefully leading to a scalable procedure that will allow handling of very large scale problems. For this reason and the issues previously presented in terms of automation and generality, we are not pursuing a parallelization of this family of MOR methods.

C. Multidimensional Sampling

An alternative approach to the moment-matching techniques are the multipoint methods. The goal of multipoint approaches is to generate the projection basis either by generating the transfer function moments from multiple expansion points $\psi_j = \{s_j, \lambda_j\}$, or from solving the system at different sample points on the relevant space

$$x_j = x(\psi_j) = A(\psi_j)^{-1}B = (s_j C(\lambda_j) + G(\lambda_j))^{-1}B \tag{5}$$

where $x(\psi_j) \in \mathbb{C}^n$ is the sample vector generated at the sample point $\psi_j = \{s_j, \lambda_j\}$, and is directly related to the zeroth order moment at ψ_j. Here the most relevant vectors among those generated by such quadrature are selected, for instance, via singular value decomposition, in order to build the projection matrix V. This approach is more reliable as it is less sensitive to the number of parameters, but, on the other hand, depends on a good sampling selection scheme. This leads to the main problems of these approaches: without sampling guidance the ability to reliably generate an accurate ROM may be compromised. Furthermore, sample solution may end up being too expensive, and thus covering the whole domain of interest with sampling points may be out of question. General and automated sample selection schemes are thus absolutely necessary for this technique to be practical.

1) *Basic Operations and Parallelization Potential:* The multipoint methods require the evaluation and solution of the system at each of a set of K sampling points. Here the computational cost is dominated by the solves ($O(KS)$), which depends on the problem, plus an overall orthonormalization of the basis, a dense operation whose cost can be approximated by $O(nK^2m^2)$ and usually represents a small fraction of the total cost. The data independence inherent to the sampling step is very appealing for parallelization and makes this family of methods as ideal for a high level parallelization. Evaluations and solves are independent, and thus theoretically perfect parallelization can be achieved in this step.

D. Automated Sample Selection Schemes

Recently, in [11], an automatic sampling scheme was proposed, which aims at obtaining a minimum number of vectors (thus minimizing the number of solves) so that a good approximation of the states vector can be obtained, i.e., a minimum set q so that (4) holds. The approach seeks to retrieve the "best" samples to solve for among an initial candidate set, with the sample selection being done before actually solving them. The method requires computing the residues obtained at every point in a candidate set and assumes that this residue is a good and cheap proxy for the true error

$$r_j = b(\psi_j) - \sum_{i=1}^{k} \alpha_i(\psi_j) A(\psi_j) v_i \tag{6}$$

where $r_j \in \mathbb{C}^n$ is the residue of the system $b(\psi_j) = A(\psi_j)x(\psi_j)$, evaluated at the multidimensional point ψ_j, when we approximate $x(\psi_j)$ by a basis V of k column vectors ($v_i \in \mathbb{R}^n$ with associated multiplier $\alpha_i \in \mathbb{C}$). The error, on the other hand is defined as

$$e_j = x(\psi_j) - \sum_{i=1}^{k} \alpha_i(\psi_j) v_i \tag{7}$$

where $e_j \in \mathbb{C}^n$ is the error, which is computationally expensive to generate, as we need to solve the system for $x(\psi_j)$.

After computing all the residues, the candidate point ψ_j with largest residue norm (or sum of norms in the case of multiple input multiple output systems) is deemed as the best new sample, i.e., the one that improves accuracy the most. Therefore, only after we have selected the best suited candidate ψ_j, do we solve the system to generate the block vector (see [11] for details).

The advantage is that the number of solves is drastically reduced in order to obtain an accurate model, and the points to solve for are automatically selected from an initial candidate set. On the other hand, if this candidate set is too large, the sample selection, which requires computing the residue at every candidate in every iteration, may become expensive.

1) *Basic Operations and Parallelization Potential:* The methodology described generates, at each iteration, the residue at every point on the candidate set. This requires evaluating the system A (at a cost of $O(z)$, where again z is the number of non-zeros of A) and to compute the sparse matrix dense matrix product AV (at cost $O(zkm)$, with k the iteration number and m the number of columns of B), plus the orthonormalization of B against the basis of AV, which requires dense matrix multiplications and orthogonalizations with approximated overall cost of $O(nk^2m^2)$. Notice that the cost depends on the number of columns in V, km, which increases with every iteration k. These operations must be repeated for every candidate in the set, but independently, which provides the opportunity for efficient parallelization.

III. Sequential Approach

The main goal of this paper is to generate a general and efficient automated projection-based MOR methodology, able to effectively reduce parameterized linear systems regardless of their representation and with minimal human interaction (ideally none at all). Pursuing on the facts presented in the previous section, to achieve this goal we are going to focus on the parallelization of a multipoint methodology combined with the presented sample selection scheme [11]. A detailed depiction can be seen in Algorithm 1.

A. Sequential Algorithm Implementation

The algorithm can be divided into four main tasks. The first task is the initialization (steps 1–3 in Algorithm 1). We set

the tolerance thresholds for the stopping criteria, and, starting from the original system and the (multidimensional) domain of interest, we define an initial candidate set $\Psi = \{\psi_1 \dots \psi_T\}$ with T multidimensional samples ψ_j, covering the whole domain. Each candidate has an associated residue norm $R(j)$. In order to automate the candidate generation and improve the coverage, a logarithmic mesh is defined in complex frequency, with increasing number of samples per decade as the frequency increases, and for each frequency point a number of different parameter perturbations are performed following for instance a low-discrepancy sequence [11], [20]. The number of samples per decade and parameter samples per frequency are defined according to a defined discretization level. The reasoning for such a scheme comes from the fact that the range of variation of frequency is typically much wider than that of the parameters, and frequency has in general a larger impact on the performance of electric circuits than the remaining parameter perturbations. Other schemes can be applied.

The second task (steps 4 and 5 in Algorithm 1) is the system evaluation and solve for the candidate ψ_j to generate the block vector x_j. In the first iteration, a sample is chosen blindly (DC is usually a good choice), whereas in future iterations the sample to evaluate and solve for is already selected. This step requires a direct or iterative solve of the system, and thus is one of the more expensive steps.

Next, the third task (step 6) is the orthonormalization of the new vector x_j with respect to the existing basis V (in the first iteration V is empty, and thus we only need to orthonormalize x_j column-wise). Generally this orthonormalization can be done by an incremental rank revealing QR operation [21]. The maximum norm of the columns of x_j after block orthogonalization w.r.t. V is stored in variable n_v for the stopping criteria. It is important to notice that complex samples generate complex vectors. In these cases, in order to match the equivalent subspace to x_j while maintaining V real, we must work with both the real and imaginary parts of x_j separately.

The forth task (step 7) is the residue generation. The current basis V is used to update the residue at the required (remaining) candidate points in the set. This requires a series of operations detailed at (step 7) of Algorithm 1. Once the maximum residue at the remaining candidates is found, we check the stopping criteria: if the current residue norm $R(k)$ and maximum norm of the vectors n_v fall below a given (relative) threshold, we assume that our basis is a good approximation to the solution for all the candidates, and that the vectors generated are not adding relevant rank to our basis. Thus we can stop the procedure. Otherwise the candidate with maximum residue is selected as next sample, and the procedure repeated (from step 4). The final basis V can be used as a projector in the congruence transformation for the reduction.

Algorithm 1 Detailed Sequential Version

Given the system and the domain of interest,

1: Define stopping thresholds for the residue t_r and the vectors t_v
2: Generate a set of T candidate sample points $\Psi = \{\psi_1 \dots \psi_T\}$
3: Initialize: $k = 1$; $V = [\ ]$; $R(i) = 2\,\|B\|\ \ \forall i = 1 : T$;
4: Evaluate the system: $A_k = A(\psi_k)$; $B_k = B(\psi_k)$;
5: Solve the system: $A_k x_k = B_k$ for x_k
6: Orthonormalization: k++
   $v_k = x_k - V(V^H x_k)$; $n_v = \|v_k\|$; $V = \left[V \quad v_k/n_v\right]$;
7: FOR $i = k : T$
   $A_i = A(\psi_i)$; $\quad X = A_i V$; $\quad X = QR(X)$;
   $r_i = B - X(X^H B)$; $\quad R(i) = \|r_i\|$;
   IF $R(i) >= R(i{+}1)$ BREAK
8: Sort R in decreasing order, and Ψ accordingly
9: IF $R(k) > t_r$ and $n_v > t_v$
   GOTO 4
10: Use V in a congruence projection on the system

B. Computational Complexity Analysis

The overall cost of this algorithm can be attributed to three operations: the solve (direct or iterative method), the orthonormalization, and the sample selection.

The cost of the solve for sparse matrices is very dependent on the matrix characteristics in terms of sparsity pattern and number of non-zeros, and varies with the problem. We will denote the cost of these operations $O(S)$, and thus, after K iterations (where K is the number of samples required to generate the appropriate basis), the cost is $\mathcal{X}_{\text{solve}} = O(KS)$.

The orthonormalization of K block vectors of size m (m the number of ports) is a dense matrix operation whose overall cost can be approximated by [11] $\mathcal{X}_{orth} = O(nm^2K^2)$.

For the sample selection, the cost is dominated by the computation of the residue, which requires the orthogonalization of B_j against A_jV for each candidate point. To the authors knowledge there is no incremental approach that can be used to overcome this bottleneck without running into other problems (for example, to store and reuse the factorization at each candidate point would speedup the residue computation but lead to huge memory requirements). The basic operations required are: the sparse dense product $X = A_jV$ (with cost $O(zkm)$ if z is the number of non zeros of A_j and km the columns of V at iteration k), to orthonormalize X (dense operations at cost $O(nk^2m^2)$), to orthogonalize B against X to generate the residue, (dense operations at cost $O(nkm^2)$), and to compute the column-wise norm of the residue ($O(nm)$). The cost is dependent upon the number of columns in V, which increases with the iteration k, and is dominated by the orthonormalization $O(nk^2m^2)$. Therefore, if we have T initial candidate points of a system with n states and m ports, the cost associated with the sample selection is [11]

$$\mathcal{X}_{ss} = O(\sum_{i=1}^{K}(T-i)n(im)^2) \tag{8}$$

where i indicates the iteration, and thus im is the rank of the current basis, and K is the final number of iterations required to generate the basis ($K < T$). After some algebra, (8) can be approximated by

$$\mathcal{X}_{ss} = O(nm^2K^3(T-K)). \tag{9}$$

This cost is highly dependent on the number of points in the initial candidate set, T, and the dimension of the basis required for a good accuracy, which is related to K.

IV. Exploiting Parallelism

Although the methodology outlined in the previous section provides good results in terms of accuracy and reduction for both single and multidimensional systems, it can be expensive for large domains, with multiple dimensions, and large size, both in the original and reduced systems.

To illustrate the main costs related to Algorithm 1, and help to understand the selected parallelization scheme, Table I shows a comparison of the times of the main linear algebra routines in which the MOR algorithms are based upon. Let us take a single iteration, for example $k = 25$ of Algorithm 1, in which we have 50 vectors on the current basis V. Step 5 requires a LU and back solve, which according to Table I would elapse $1107''$. Step 6 requires a QR, elapsing $3''$. Step 7 requires to perform, for each candidate left, a complex sparse matrix dense matrix product ($3''$), a dense complex QR ($6''$), and to compute the residue and its norm, operations with a cost similar to a QR factorization ($6''$). Therefore, if we have 100 candidates left, the overall elapsed time will be approximately $1500''$. It is important to recall that the cost of step 7 depends on multiple factors, such as the number of vectors in the basis and number of candidates, and thus varies from iteration to iteration. With these numbers in mind, efficiently speeding up the algorithm implies reducing the number of iterations (which also reduces the number of solves) more so than speeding up the individual operations themselves. But in addition, the parallelization efforts should focus on the reduction of the cost of the sample selection (critical for multidimensional models that require a large candidate set).

We propose a simple yet highly effective parallelization of the algorithm that will overcome the main drawbacks of the serial version, while keeping the same accuracy, robustness and reliability. It is important to point out once more that the goal of this paper is not related to efficient parallelization of the underlying linear algebra routines (which can nevertheless be combined with the proposed methodology to provide further speedups). Instead, we will focus on parallelization at a higher, more abstract level. With the advent of multicore processors and fast network connections, most computational environments are now hybrid shared and distributed memory architectures. The kind of parallelization we seek to apply can take advantage of both architectures, and although each can be pursued independently, we propose in fact an hybrid framework exploiting both. In the following we will assume an hybrid architecture is available, denote P as the number of cores per machine (processors with shared memory), and M as the number of nodes or machines in a distributed environment (processors with distributed memory). For the parallelization routines we will use the standard nomenclature of the OpenMP [22] and OpenMPI [23] languages, which are the most commonly used libraries and the ones we used in our implementation.

A. Shared Memory Parallelization Opportunities

Shared memory are uniform memory access architectures in which a common, global memory is accessed by all the processors. Usually the memory is local, even if multiple memory levels are present, such as in the case of multicore CPUs, and thus the access is very fast. This allows for a finer granularity on the parallelization, since the communication time is rather small. On the other hand, some attention must be paid to data-races or concurrent access to the same memory. Shared memory based parallelization achieves its best performance in vectorial operations, i.e., operations that are independent and with small data dependencies.

TABLE I
Comparison of Linear Algebra Routines: A Is Sparse with Size 179 272 and 1 898 327 Nonzeros, and X Is Dense with 50 Columns

	AX	(Lapack) QR(X)	(UMFPack) LU and Solve
Real	$1''$	$3''$	$404''$
Complex	$3''$	$6''$	$1107''$

Looking at (9), searching for a suitable sampling point can be costly if there is a large number of vectors in the basis and a large candidate set. This is usually the case for systems where many parameters have a critical impact on the behavior of the model. A first parallelization can be efficiently applied to this sample selection procedure where computation of the residue at each remaining point in the candidate set can be done independently at each point, as there are no data races or concurrency issues, and it has relatively small memory requirements. Therefore, shared memory parallelization is appropriate and a simple distribution of the candidates among the available cores, with negligible overhead, can provide a theoretical near perfect speed-up for this sample selection step. Note that the parallelization is done at a high level, avoiding the parallelization of the underlying routines. Each one of the P cores perform the same operations with different data input. We thus propose a shared memory parallelization of the loop in step 7 of Algorithm 1.

B. Distributed Memory Parallelization Opportunities

A different kind of parallelization which can be easily combined with the shared memory approach above, is to use the distributed environment. Distributed memory environments are a set of connected machines or processors, named nodes, each one with its own memory, so an explicit communication takes place whenever remote data is required. Typical architectures are clusters or grids, i.e., sets of machines connected by a (dedicated) fast network. In this scenario, the communication time may be relevant in the performance, and needs to be minimized. Coarse granularity parallelization, with operations as independent as possible, is prone to be applied here. On the other hand, the total amount of memory increases, thus larger problems or higher level parallelization becomes amenable.

We propose to exploit the distributed memory capabilities in two steps of the algorithm: the sample selection and the factorization and solve. With respect to the sample selection, we can divide the remaining candidate points among the different nodes, and at each node generate the respective residues. However, since this is a distributed environment, care must be paid to the communication between nodes. In our case, the residue generation is independent for each candidate set, and thus, as long as the basis and the system matrices are in

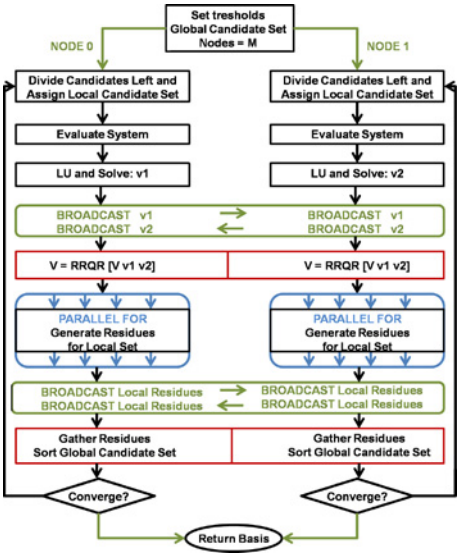

Fig. 1. Distributed multicore algorithm graph.

the node, we only need the point coordinates to generate the residue, with no communication with the rest of the nodes. Therefore, in the case of identical machines, we distribute the remaining points evenly among the nodes, in order to obtain the residues in parallel. When the distributed environment is composed of machines with different characteristics, a different strategy may be used, in order to maintain a good balancing and avoid idle computational resources.

With respect to the solve step, from the analysis in Section III-B and previous sections, it is clear that there are two factors to consider as cost: one is the cost of a single solve $O(S)$, and another is the fact that we must perform K solves. Improving the complexity of the system solve $O(S)$ has been exploited by parallel implementations of linear algebra routines [19], also with application to MOR [13]. This reduces both the cost in CPU and in memory, and a similar approach could be combined with the parallelization proposed so far.

However, we will assume that the solve can be performed in a single machine without memory issues, and take advantage of the multipoint methods to focus on reducing the cost related to the factor K. Since each node in a distributed environment has its own memory, the solves can be done in parallel. If the system is already loaded in the node, it only needs the point coordinates to solve (the communication overhead is minimum). Once we have selected a suitable point, each solve can be done independently. In order to cope with the memory limitations, sparse factorizations or iterative methods can be applied. Therefore, at each iteration we are solving M points, one on each node, with a potential reduction of the overall solve time by a factor of M.

C. Combined Proposed Implementation

After having identified the algorithmic parts that are prone to be parallelized, let us discuss the mapping and implementation. The methodology is detailed in Algorithm 2, and the main steps illustrated for $M = 2$ in Fig. 1.

Algorithm 2 Parallel Version-3P0r

Given the system and the domain of interest,

1: Define stopping thresholds for the residue t_r and the vectors t_v
2: Generate a set of candidate sample points $\Psi = \{\psi_1 \dots \psi_T\}$
3: Generate an array with candidate indexes $\Upsilon = [1, \dots T]$
4: Generate a residue array $R = [r_1, \dots r_T]$, $r_i = 2\,\|B\|$
5: MPI initialization, M nodes: N = node number; $\Delta = T/M$;
6: Initialize: $V = [\ ]$; $k = 1$
7: Mapping Local set: $i = kM$, $T_N = 0$, While($i < T$)
$\Upsilon_N = [\Upsilon_N \Upsilon(i : i + \Delta)]$; i+=Δ; T_N+=Δ
8: Select first local sample: $j = \Upsilon_N(1)$, ψ_j
9: Evaluate system matrix: $A_j = A(\psi_j)$
10: Solve the system: $A_j x_j = B$ for x_j; $X(:, N) = x_j$
11: MPI BROADCAST X
12: Orthonormalization:
$v_k = X - V(V^H X)$; $n_v = \|v_k\|$; $V = \begin{bmatrix} V & v_k/n_v \end{bmatrix}$;
13: OMP PARALLEL $i = 1 : T_N$
$j = \Upsilon_N(i)$; $A_i = A(\psi_j)$; $X = A_i V$; $X = QR(X)$;
$r_i = B - X(X^H B)$; $R_N(i) = \|r_i\|$;
14: MPI BROADCAST R_N so that $R = \begin{bmatrix} R_0 \dots R_{M-1} \end{bmatrix}$
15: MPI BROADCAST Υ_N so that $\Upsilon = \begin{bmatrix} \Upsilon_0 \dots \Upsilon_{M-1} \end{bmatrix}$
16: Sort R in decreasing order, and Global Υ accordingly; k++
17: IF $R(0) > t_r$ and $n_v > t_v$
Set Δ to desired size, $\Delta < T - kM$, and GOTO 7
18: Use V in a congruence projection on the system

1) *Initialization, Load and First Solve (Steps 1–10):* The first step is the system load and the generation of the global candidate set Ψ that covers the region of interest. The thresholds for the stopping criteria must also be set. A global residue array R and index array Υ are initialized, as well as an array X that will contain the generated vectors. These are redundant in all nodes to avoid unnecessary communications. The element $R(i)$ will contain the residue of the $\Upsilon(i)$ candidate, $\psi_{\Upsilon(i)}$. These arrays will be used for communication among the different nodes.

Each node N will define a local residue and index array, R_N and Υ_N, which are related to the global candidates. However, at each iteration, each node only works with a subset of T_N candidates (the workload distribution among the nodes will be discussed later). Each node will select the first candidate in its local set, $\Upsilon_N(0)$, generate the system matrix and solve the system at that point. This step is done in parallel, and since each node has a disjoint subset of the global candidates, M samples are generated at each iteration.

2) *Vector Communication and Orthonormalization (Steps 11, 12):* Once the samples are computed, in order to use global information in the generation of the residues, we need to gather all the vectors at each node. To this end, each node N copies the solution vector to the Nth column of X (if $m > 1$, a block vector is generated and the columns for each node start in the position Nm), and broadcasts the vector, i.e., an array of nm elements starting from the first element of the column Nm. Broadcast is a global communication routine in MPI, and thus a very efficient way to transmit data among all nodes. Broadcasts are blocking in MPI, and thus the communication cannot be overlapped with any other computation. However, the communication time is small in comparison with other linear algebra operations. In addition, the blocking operation works as a barrier, allowing for a synchronization of the nodes.

Now all the nodes have all the sample vectors generated in the previous stage. The next step is an incremental orthonormalization of the new vectors with respect to the ones already stored in the basis V. In the general case where the samples are complex, each vector is broken into its real and imaginary parts, in order to span the same subspace as the original complex vector, avoiding using complex algebra, and maintaining real system matrices after projection. Recall that all the nodes have the same information, and thus each node does the same operation in parallel. This redundancy avoids further communications. Since the orthonormalization is done on a relative small set of vectors, and it is a linear algebra routine that relies on efficient cache reuse, this option is preferred to a distributed implementation. However, shared memory or GPU based implementations could be applied here.

3) *Residue Generation and Communication (Steps 13–15):* The next step is the generation of the residues with the new upgraded basis, and selection of the sample to solve next. In order to speed up the procedure, each node generates the residues for its subset of candidates. Therefore, each processor will have to compute the residues at $T_N = (T - kM)/M$ candidates, with k the iteration number, T the overall number of candidates, and M the number of machines. Notice that prior to this step, each node N has all the global information: the basis and the global candidate set, plus two local arrays R_N and Υ_N that indicates its subset of candidates. Here the multicore parallelization proposed in Section IV-A can again be applied for independent residue generation at each node. Each node stores the corresponding residue of the candidate with index $\Upsilon_N(i)$ in the element $R_N(i)$, for all its elements with the exception of the first one (recall that the first one was used as the assigned sample to solve). Once the residues are generated, each node copies its local array R_N and Υ_N to the global arrays R and Υ starting at position $N(T-kM)/M$. Once the arrays R and Υ have the updated information, a broadcast is performed transmitting T_N starting in their first element $N(T - kM)/M$. Again, this broadcast is a blocking operation that works as a synchronization barrier, and the iteration counter is increased.

4) *Candidate Mapping and Sample Selection (Steps 16, 17):* Once the broadcast is completed, at each node we have all the global information, with R containing the residues of the candidates indexed by Υ. To find the best candidates for the next iteration, we sort the elements in R in descending order, and those in Υ accordingly. Since all the nodes have the same data, the sorting is done locally, avoiding further communications and providing data coherence. The maximum residue of the candidates left and the norm of the orthogonalized vectors at this iteration are checked against the convergence thresholds. If convergence is not achieved, the remaining $T-kM$ candidates are distributed among the nodes to perform the next iteration. Notice that since all the candidates are stored in each node, no communication is required in this step: a simple assignment of the indexes Υ of the candidates left to the nodes is enough.

An issue not yet discussed is how to distribute the candidates among the different nodes. This mapping can have a critical effect on the results of the algorithm, as the first sample of the local subset is used to solve the system, and thus to generate the global sample. In the initial iteration, there is no indication of how to select the samples, and thus we have to make a guess. For the type of linear models we are interested in, frequency is often the most relevant parameter, the one with the more relevant effect on the behavior of the system. In order to obtain vectors as different as possible, a solution is to use samples that are separated in frequency. If the candidates are sorted in frequency, the initial set can be split into M subsets of size $\Delta = T/M$, and each assigned to a node. For the remainder of the iterations $k > 1$, we already have the residues and indexes sorted in the arrays R and Υ, with the initial kM elements related to solved samples and the indexes. We divide the array Υ in chunks of size Δ, and assign chunks alternatively to the nodes (i.e., chunks $0, M, 2M, \ldots$ are assigned to the first node, chunks $1, M{+}1, 2M{+}1, \ldots$ to the second, and so forth).

If we divided Υ into M chunks, $\Delta = (T - kM)/M$, the node $N = M - 1$ will be loaded with the candidates with small residue, and thus may generate a vector that will add "small" or no rank to the current basis. On the other hand, if we take $\Delta = 1$, the first M samples indexed by Υ (and with the larger residue) will be assigned to the M nodes and solved, one at each node (recall the first sample of the subset is the one to solve). This may not be the best option in every scenario. Imagining a fine discretization of the space of interest, the candidates are "close" in the discretized space. Linear systems are characterized for their continuity, and thus if the candidates are "close," the behavior (the vector solution) of the system may be similar (and thus it may not be a good idea to solve for both vectors), and they will also have similar residue. Therefore, if we solve the first M samples it may happen that they span a rank-deficient set of vectors (note that the serial implementation has no such problem since at each iteration only a sample is solved for, and thus the residue information about the remaining candidates is updated before the next choice). The optimal value for Δ depends upon multiple factors, such as the model behavior or the discretization of the space to sample, and, in addition, may vary at each iteration. We propose an intermediate option, and to take a relatively small value for Δ, yet larger than one, in order to maximize the probability of generating relevant yet different vectors. This has direct consequences on the optimality of the sample selection, since we are no longer taking the "best" sample (according to the residue) at each iteration. There is one node that solves that "best" sample, whereas the remainder choose probable good samples. This may be translated in an optimality loss with respect to the sequential approach.

D. Computational Complexity Analysis

Let us study the efficiency of this parallelized approach. We have T initial samples, and we need a total number of K samples to achieve a good model in the sequential case. Again, let us suppose we have M machines with P cores each, and the cost of the communication at each iteration is $O(t_x)$. At each iteration we solve for M samples, and thus, we can potentially reduce the number of iterations by a factor of M. Thus the

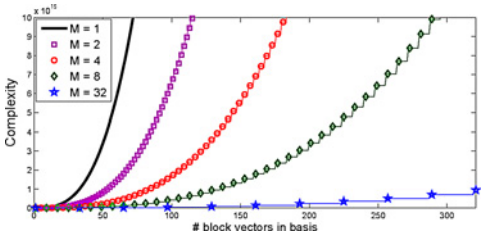

Fig. 2. Theoretical complexity growth of the sample step (12) with the number of block vectors in basis, for $M = 1, 2, 4, 8$ and 32 nodes and $P = 1$, with $n = 1e6$, $m = 2$, and $T = 5000$. Markers indicate when a new sample is solved (i.e., a new iteration k++).

overall cost of factorization and solve is

$$\mathcal{X}_{MPsolve} = O(\lceil K/M \rceil S). \tag{10}$$

Potentially, if the cost of the solves dominates, a speedup close to M can be achieved in these stages. Notice that this performance can be further improved if shared memory parallelized approaches of the solve and orthonormalization are applied (i.e., trying to reduce $O(S)$). We have however not pursued that. Since we are not parallelizing the orthonormalization and it is done on the same number of vectors, the cost is essentially the same as in the sequential case (all the processors perform this operation simultaneously)

$$\mathcal{X}_{MPorth} = O(nm^2K^2). \tag{11}$$

With respect to the sample selection, at each iteration the task of residue generation is divided by the number of nodes plus the number of cores at each machine. In addition we have reduced the number of iterations to $\lceil \frac{K}{M} \rceil$, although at each iteration Mm new vectors are generated. Therefore the cost of this step is

$$\mathcal{X}_{MPss} = O(\sum_{i=1}^{\lceil K/M \rceil} \frac{(T - iM)}{MP} n(iMm)^2) \tag{12}$$

which can be approximated by

$$\mathcal{X}_{MPss} = O(\frac{nm^2M}{P} \lceil \tfrac{K}{M} \rceil^3 (T - M\lceil \tfrac{K}{M} \rceil). \tag{13}$$

Notice that we reduce the number of residues generated by each core by a factor of MP. Since these operations are independent, the speedup should be close to that value. But in addition, we are reducing the number of iterations by a factor M, which is translated into a potential speedup larger than MP. To illustrate this, Fig. 2 shows the (theoretical) complexity growth with the increase of the number of vectors in the basis (and thus with the number of iterations K) by evaluation of (12) with $n = 1e6$, $m = 2$, and $T = 5000$, for $M = 1, 2, 4, 8$, and 32, with a single core ($P = 1$). Notice that as M increases the number of iterations quickly decreases (the markers for each line are further apart) leading to a strong complexity reduction. A quick computation shows this reduction to be superlinear.

This effect is thus not surprising, since, as mentioned, we solve for M samples at a time, and thus perform the sample selection once per M solves, whereas in the sequential approach we recompute the residues after the solve at every iteration. In a sense we are trading optimality for efficiency (only one sample is optimal in terms of maximum residue), which is translated into slightly larger reduced orders, but as we shall see, this optimality loss is small in comparison with the efficiency gains. Clearly, the same scheme could be applied on a single machine, performing multiple sequential solves at each iteration. While this can be pursued in the sequential version, the advantages are debatable: the computational cost is slightly reduced (as less residue recomputations are performed), but not the overall solve time, and would incur in the same optimality loss. On the parallel version however, this is a natural option: the availability of M processors makes it inefficient if less than M samples are computed at a time, and the consequent optimality loss may be compensated by the large speedup achieved in the model generation.

With respect to the communication overhead in distributed environments, this cost is very small in comparison with the overall elapsed time. Notice that the communication only involves a broadcast of the solution vector (Mnm complex numbers, with M the number of nodes, n the size of the system and m the number of ports), and a broadcast of the residue norms (T doubles) for each iteration.

The overall cost can be approximated by the sum of the costs in (10), (11) and (12), plus the communication costs. The parallelization is more effective in cases where the sample selection elapsed time is a large fraction of the overall time and cases that require a large subspace for a good approximation (i.e., the number of vectors in the basis, and thus the iterations required for the convergence).

E. Potential Improvements

We have opted for the implementation of a general algorithm, whose results will be presented in the next sections, and which is able to tackle fairly complex and challenging models with very different characteristics and representations, and difficult to handle with existing sequential approaches. However, the proposed framework is very flexible, allowing to include small modifications that allow to handle specific models or special cases more efficiently. In the next we present some of these possibilities.

A simple modification can help reduce the optimality loss. In the general proposed scheme each node performs an independent solve, reducing the cost related to the number of iterations K. As already mentioned, and as will be shown in the results, in cases when the number of iterations K required for good accuracy is close to the number of nodes M, there is a high probability to generate redundancies when solving M points at a time (recall that only one is the best suited according to our criteria). This is translated in larger ROMs for the same accuracy. A solution is to divide the distributed resources in order to minimize both the cost related to number of iterations K and the cost related to the solve $O(S)$. This can be done by grouping the M nodes in sets of D nodes and applying a distributed solve routine (e.g., [19]) on each of them. Therefore we are solving $\lfloor M/D \rfloor$ points at a time, but with reduced cost (that depends, of course, of the performance of the distributed solve). This option, which is best suited when the number of available nodes M is very large, also allows

TABLE II
BENCHMARKS CHARACTERISTICS

Example	D-Spiral	K-Spiral	pRLC	Heat
Modeling	FIT	K-element	MNA	FD
Size/#IO	179 272/4	89 134/1	7551/1	3481/1
nnz (A)	1 898 327	613 102	22 531	17 169
State Space	First order	Second order	First order	First order
Parameterized	No	Fourth TS	First TS	Fourth HP
# λ/Terms	0/1	1/5	45/46	16/4845
Freqs. (GHz)	0.5–60	40–60	0–100	0–0.001
# Validation	750	6000	116 000	100 000
# Candidate	400	700	4800	2300

to deal with very large models that cannot be solved with the memory constraints of a single node. This is a particular case of combination of high level parallelization with an efficient parallelization of linear algebra routines. As already mentioned, there is nothing that prevents both approaches to be combined in any form that may take advantage of the underlying architecture or of some prior knowledge on the original model characteristics. Efficiently parallelized linear algebra subroutines can be easily adopted within the proposed framework and may largely improve its performance. The possibilities are vast and represent a very interesting field of research. An optimal combination is likely to require an exact tuning that will depend on the underlying architecture and resources available, as well as on the problem.

Another option is related to the initial set and sample selection. In the proposed approach, even though dynamic candidate sets can be readily handled, we have chosen to use a static initial set fine enough to cover the complete domain. The algorithm is greedy, treating all points independently with no particular regard for spatial distribution and with similar weight. The added computational power of multiple nodes would allow us to look upon residue minimization region-wise in addition to pointwise. This requires maintaining hierarchical information for the mapping, tying points in nearby regions in order to estimate an average residue error in a given region. If accuracy in certain regions appear to be slowly improving, we can refine the candidate set in that region, and map it into a set of nodes. This allows to identify and focus in certain complex regions to improve accuracy and convergence. This can of course be done in the serial implementation, but the cost of such a procedure could be overwhelming, specially because the benefit is unknown, potentially nil. The ability to do these computations in parallel makes the extra cost almost negligible and the potential benefits more enticing.

V. SIMULATION RESULTS

The algorithms were implemented in C/C++, with double precision and using OpenMP [22] and OpenMPI [23] for the parallelization. Tests were run on a cluster of 8 Intel $Q6600$ (4-core, 2×4 MB L2 Cache) at 2.4 GHz machines with $8G$ RAM, connected by a dedicated gigabit net. In all cases sparse algebra routines were use when possible to to minimize the memory requirements. For the LU and solve, the sequential UMFPack of the SuiteSparse package [24]–[26] was

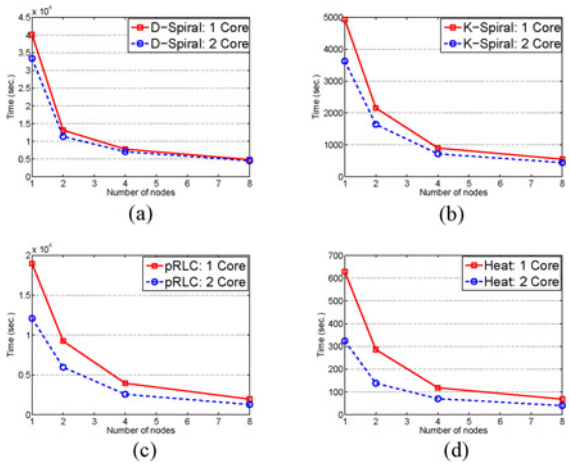

Fig. 3. Performance evolution with the number of nodes for (a) D-Spiral, (b) K-Spiral, (c) pRLC, and (d) Heat examples.

used. COLAMD [27], [28] and AMD [29] were used as pre-ordering algorithms to minimize fill in for the non-symmetric and symmetric matrices, respectively. The rest of the algebra routines were not particularly tuned for the kind of systems under study. LAPACK [30] functions were used for dense matrix operations, such as dense matrix multiplications and QR factorizations. For the sampling, no prior knowledge of the system was assumed, other than the frequency and parameter variation range. The initial candidate set was automated with different levels of discretization, which fixed a number of frequency samples per decade, and a number of parameter samples per frequency sample. Notice that prior knowledge of the system behavior could help reducing the number of initial candidates, and thus improving the efficiency.

Table II presents the characteristics of the set of mid-large benchmarks. The description includes the technique used for the original model generation (Modeling), the original number of states and the number of ports (Size/#IO), the number of non-zeros of the system matrix [nnz(A)], the order of the state space representation (State Space) and the parameterized representation (Parameterized), either Taylor series (TS) or Hermite polynomial (HP) expansions, the number of parameters and the number of matrix terms of the polynomial parameter representation (λ/Terms), the frequency range (Freqs.), the number of points used as candidates (# Candidate), and the number of points used to compute the error and validate the ROMs (# Validation). The number of candidates for each example represents a fine enough discretization and has been chosen in order to ensure the complete coverage of the multidimensional space of interest. Finer discretization would not report changes in the ROM size or accuracy, nor in the parallelization speedup. However, the number of candidates will have a relevant impact on the overall elapsed time, as presented in (13). The number of points to validate the ROMs, on the other hand, represent an extremely fine grid, in order to show that a small fraction of candidates can generate an accurate response in the complete domain, an indication of the method's robustness. Fig. 3 shows the evolution of the elapsed times with the number of nodes, for 1 and 2 cores, for all the

TABLE III
CACHE INTERFERENCE EFFECT (SAMPLE SELECTION TIMES)

	Proc. Specs.	1 Core	2 Core	4 Core	8 Core
K-Spiral	Quad Q6600	3682″	2386″	2552″	–
	(4-core, L2 2 × 4 M)	1.0X	1.54X	1.44X	–
	2× Xeon E5410	7846″	4465″	2278″	1840″
	2×(4-core, L2 2 × 6 M)	1.0X	1.75X	3.44X	3.26X
Heat	Quad Q6600	619″	315″	325″	–
	(4-core, L2 2 × 4 M)	1.0X	1.96X	1.90X	–
	2× Xeon E5410	741″	385″	200″	197″
	2×(4-core, L2 2 × 6 M)	1.0X	1.92X	3.70X	3.76X

TABLE IV
D-SPIRAL ARNOLDI MOMENT MATCHING PERFORMANCE

Size/Moments	Abs. Err.	LU	Solve	Orth.	Total
269/80	41.5 (>100%)	401″	988″	271″	1724″

examples, where the speedup is defined as the ratio of the time elapsed in the parallel procedure with respect to the time elapsed in the sequential algorithm. Details and discussion on the performances will be presented in the next subsections, where the routines related to the sample selection are labeled as *sample*, the routines of factorization and system solve as *solve*, and the routines related to basis orthonormalization as *orth.*

A. Cache Limitations in Multicore Architectures

An interesting aspect that we verified when implementing the algorithm and believe is interesting to report has to do with architectural cache limitations. The performance of the multicore parallelization of the sample selection was good with two cores, but disappointing when applied on all the four cores of the Q6600 machines. Little speed up is achieved in comparison with what was expected. We realized that the degradation of the performance was caused by cache interference of the highly efficient LAPACK routines at the core of this step. These routines are based on BLAS, which rely on cache hit for efficiency improvement. The Q6600 is a quad-core processor formed by two dual-core in the same die. Each pair of dual-core share a L2 cache block, and thus when independent routines are executed at the same time, cache hit ratio drops. In order to validate this assumption, we performed tests in a different machine, with two Xeon E5410 processors at 2.33 GHz, each with four cores and 2 × 6 MB of L2 cache. The cores are also grouped in pairs, and share the L2 memory. Table III shows a comparison of the times elapsed in the sample selection for two of the benchmarks, and how do the performance evolve with the number of processes running. It is clear that the performance drops when the number of processes is larger than the number of L2 blocks. The achievable speedup is therefore reduced in such cases, a point that should be noted.

B. Integrated Double Spiral

The first example (D-Spiral) is a non-parameterized EM model of two integrated spiral inductors, obtained with FIT [31], [32], and taking into account substrate and upper air.

TABLE V
D-SPIRAL: PROPOSED PARALLELIZATION RESULTS

Nodes Cores	Iter. Size	Abs. Err. Rel. Err.	Sample	Solve	Orth.	Total
1	12	5.2e-2	26 457″	13 590″	45″	40 093″
1	96	0.18%	1X	1X	1X	1X
1	12	5.2e-2	19 689″	13 626″	45″	33 365″
2	96	0.18%	1.34X	1.00X	1.00X	1.20X
2	6	6.9e-2	5901″	6778″	46″	13 134″
1	96	0.74%	4.48X	2.00X	0.98X	3.05X
2	6	6.9e-2	4372″	6778″	46″	11 329″
2	96	0.74%	6.05X	2.00X	0.98X	3.54X
4	4	5.7e-2	3081″	4534″	75″	7776″
1	120	0.18%	8.57X	3.00X	0.60X	5.16X
4	4	5.7e-2	2354″	4534″	75″	7057″
2	120	0.18%	11.24X	3.00X	0.60X	5.68X
8	3	9.4e-3	1139″	3398″	109″	4796″
1	137	0.08%	23.23X	4.00X	0.41X	8.36X
8	3	9.4e-3	872″	3398″	109″	4494″
2	137	0.08%	30.34X	4.00X	0.41X	8.92X

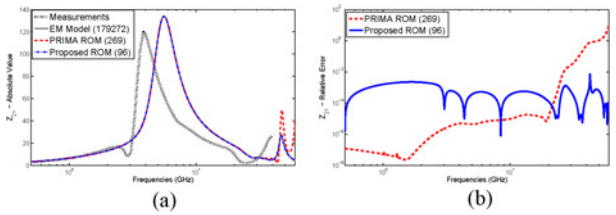

Fig. 4. D-Spiral example. (a) $|Z_{21}(s)|$ from the measurements (up to 40 GHz) and the EM model and PRIMA and proposed ROMs. (b) Relative error in $|Z_{21}(s)|$ for the ROMs w.r.t. the EM model.

This realistic benchmark was also manufactured and measured. For comparisons in this example, we present in Table IV the results of an Arnoldi based moment matching approach [4]. Table V shows the details on the performance of the proposed methodology. Both sample selection and system solve are expensive in this example, and thus the distributed memory parallelization is more effective. It can also be seen that performance, in terms of speedup, degrades as the number of machines increases. This is caused by the ROM size: at one point the number of samples required for good accuracy is almost the same as the number of machines. This limits the scalability of the methodology. In any case the proposed approach, although more CPU expensive, automatically generates a much smaller ROM with better accuracy than the moment matching approach, which, even for a large number of moments, is unable to match the original response at high frequencies. Fig. 4 shows the frequency domain impedance $|Z_{21}(s)|$, for the original EM model, and for the Arnoldi and proposed reduced models. We have also included the physical measurements. The figure also shows the relative error of the ROMs with respect to the original EM model.

C. Parameterized Integrated Spiral

The second benchmark (K-Spiral) is an RCK model of an integrated spiral inductor, whose side length varies up to 37 μm around the nominal value of 187 μm. It is modeled with a parameterized second order system (i.e., the system

TABLE VI
K-SPIRAL: PROPOSED PARALLELIZATION RESULTS

Nodes Cores	Iter. Size	Abs. Err. Rel. Err.	Sample	Solve	Orth.	Total
1	23	8.3e-2	3682″	1237″	11″	4931″
1	46	3.1e-3%	1X	1X	1X	1X
1	23	8.3e-2	2386″	1237″	11″	3634″
2	46	3.1e-3%	1.54X	1.00X	1.00X	1.36X
2	12	1.4e-1	1496″	644″	13″	2157″
1	48	2.2e-3%	2.46X	1.92X	0.85X	2.27X
2	12	1.4e-1	984″	644″	13″	1641″
2	48	2.2e-3%	3.74X	1.92X	0.85X	3.00X
4	6	6.8e-2	537″	325″	14″	899″
1	47	4.4e-3%	6.86X	3.80X	0.78X	5.48X
4	6	6.8e-2	357″	325″	14″	717″
2	47	4.4e-3%	10.31X	3.80X	0.78X	6.88X
8	4	9.2e-2	296″	218″	19″	547″
1	56	1.3e-3%	12.44X	5.67X	0.58X	9.01X
8	4	9.2e-2	204″	218″	19″	442″
2	56	1.3e-3%	18.05X	5.67X	0.58X	11.16X

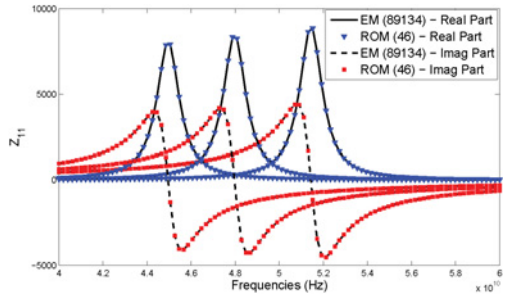

Fig. 5. K-Spiral example: real and imaginary parts of Z_{11} for the original and ROM for three parameter values.

equations have a $(Y(\lambda) + sC(\lambda) + K(\lambda)/s)x = Bu$ structure), in which each matrix is represented as a fourth order Taylor Series (i.e., we have five matrix terms) to model the parameter dependence. The model has 5124 resistors, 20 736 capacitors and 381 120 susceptances. In this case both the sampling and the solves are critical in the execution time, and thus the parallelization is very effective, achieving good speedups. Table VI shows the details on the performance of the parallelization, and it is clear that the distributed environment approach generates excellent results. However, since the number of samples is relatively small, the performance slightly degrades for a lot of machines. Fig. 5 shows the frequency domain real and imaginary parts of the impedance for three different parameter settings. It can be seen that the parameter largely affects the system behavior, which is perfectly matched by the ROM.

D. Parameterized RLC

The third benchmark (pRLC) is a parameterized MNA formulated set of five connected lossy lines. Each line is divided into three parts modeled with 100 RLC segments in which the p.u.l. values of each part depend locally on three parameters, with a total of 45 parameters. The artificial parameters modify the R, L, and C values of each part up to 20% of their nominal value. The system is represented as a 46-term Taylor series formulation for the static and dynamic system matrices.

TABLE VII
PRLC: PROPOSED PARALLELIZATION RESULTS

Nodes Cores	Iter. Size	Abs. Err. Rel. Err.	Sample	Solve	Orth.	Total
1	100	1.2e-1	18 895″	1″	53″	18 952″
1	198	6.0%	1X	1X	1X	1X
1	100	1.2e-1	12 017″	1″	53″	12 074″
2	198	6.0%	1.57X	1X	1X	1.57X
2	52	1.1e-1	9181″	< 1″	55″	9239″
1	204	5.6%	2.05X	1X	0.96X	2.05X
2	52	1.1e-1	5893″	< 1″	55″	5955″
2	204	5.6%	3.20X	1X	0.96X	3.18X
4	27	1.6e-1	3874″	< 1″	62″	3938″
1	209	8.2%	4.88X	1X	0.85X	4.81X
4	27	1.6e-1	2494″	< 1″	62″	2553″
2	209	8.2%	7.57X	1X	0.85X	7.42X
8	15	1.4e-1	1880″	< 1″	68″	1948″
1	225	7.4%	10.05X	1X	0.78X	9.73X
8	15	1.4e-1	1202″	< 1″	68″	1275″
2	225	7.4%	15.72X	1X	0.78X	14.86X

Although relatively simple, this example is very challenging in terms of sample selection, due to the large frequency range (0–100 GHz) and the flat response at low frequencies and large resonances at high frequencies decreasing with small ripples until the transfer function falls to zero. In addition, the large number of parameters modify the amplitude and location of the resonances, as can be seen in Fig. 6.

Table VII presents the performance details of the proposed methodology. In this case the sample selection is by far the most expensive task, since the a large amount of samples are required to refine the accuracy in the areas of small resonances. The method scales well, and the parallelization, in particular the one based on distributed environments, is very efficient.

Fig. 7 shows the absolute error distribution for the ROMs generated with 1, 2, 4 and 8 machines. All ROMs have a similar worst case accuracy, which is < 0.2 in a validation set of 116 000 points, but it can be seen that the optimality is degraded as the number of machines increases. This effect is clear in the deviation of the error. For one node the deviation is small, indicating that the algorithm makes good choices in the sample selection in order to minimize the maximum. As the number of machines increases, the ROM order and deviation slightly increases, indicating that some samples are not placed in the best choice (oversampling). This was expected, since we are performing a number of extra solves at each iteration that were not the optimal (as defined) point: in a sense we are trading optimality for efficiency.

E. Parameterized Heat Model

The last benchmark (Heat) is a parameterized first order model obtained from the discretization of a heat equation with a log normal diffusivity field using finite differences [33]. The system and input matrices depend on 16 parameters, and the parameter effect is modeled via a fourth order multivariate Hermite polynomials (HP), which generates 4845 matrix terms. This means that the system is fully parameterized, represented as $\dot{x} = A(\lambda)x + B(\lambda)u$, where the matrix $A(\lambda)$ can be represented as a 4845-term polynomial matrix function,

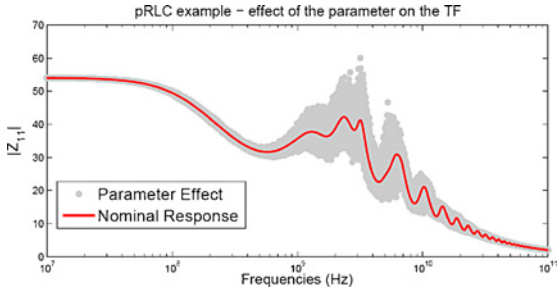

Fig. 6. pRLC example frequency response. Red line is the nominal $|Z_{11}(s)|$, whereas the gray area shows the possible values form the 116 000 parameter settings considered in the validation.

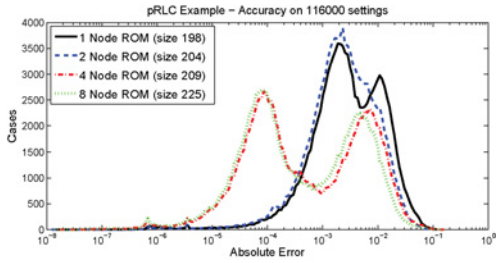

Fig. 7. pRLC: distribution of absolute errors of ROMs obtained with different number of machines. On the y-axis are the number of the occurrences out of the 116 000 validation settings, whereas on the x-axis are the absolute errors. Note that although the maximum error is similar for all ROMs, the optimality reduces with the number of machines (ROM size increases for equivalent accuracy).

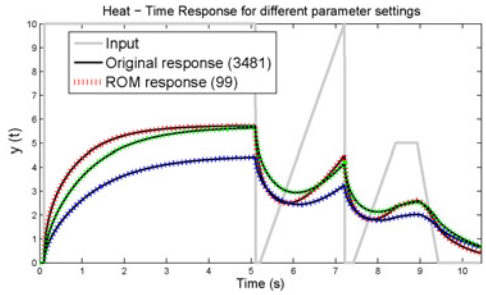

Fig. 8. Heat time domain simulation: original (line) and ROM (dashed) time response for three random parameter settings (each color represents a parameter setting).

and the IO matrix $B(\lambda)$ also depends on the parameters. This representation is hard to handle with multidimensional moment matching approaches. On the other hand, the proposed multidimensional sampling framework can handle it with minimum modifications. Instead of using the explicit HP, we use a compressed form with 65 matrix terms for the evaluation of the matrices A and B. The rest of the procedure remains exactly the same. Table VIII shows the details on the performance. In this case the solve time is almost negligible, and the main cost is driven by the sample selection. Parallelization is very effective and exhibits a good scalability in this case, since the ROM requires a relative large amount of samples for the given accuracy. Fig. 8 shows a time domain simulation of the original and reduced models, for three different parameter settings and different input signals. It can be seen that the original and ROM responses are indistinguishable.

TABLE VIII
HEAT: PROPOSED PARALLELIZATION RESULTS

Nodes Cores	Iter. Size	Abs. Err. Rel. Err.	Sample	Solve	Orth.	Total
1 1	50 99	4.2e-3 0.7%	619″ 1X	2″ 1X	7″ 1X	628″ 1X
1 2	50 99	4.2e-3 0.7%	315″ 1.96X	2″ 1.0X	7″ 1.0X	324″ 1.94X
2 1	27 101	4.1e-3 0.7%	276″ 2.24X	1″ 2.0X	9″ 0.77X	286″ 2.19X
2 2	27 101	4.1e-3 0.7%	126″ 4.91X	1″ 2.0X	9″ 0.77X	138″ 4.55X
4 1	14 106	6.2e-3 1.0%	108″ 5.73X	1″ 2.0X	9″ 0.77X	118″ 5.32X
4 2	14 106	6.2e-3 1.0%	60″ 10.32X	1″ 2.0X	9″ 0.777X	70″ 8.97X
8 1	8 116	3.4e-3 0.5%	58″ 10.67X	< 1″ 2.0X	9″ 0.77X	67″ 9.23X
8 2	8 116	3.4e-3 0.5%	29″ 21.34X	< 1″ 2.0X	9″ 0.77X	40″ 15.7X

VI. CONCLUSION

A distributed and shared memory parallel framework for parameterized linear model order reduction has been presented. The underlying method relies on multidimensional sampling, providing robustness and reliability, and the combination with the sample selection approach provides a high level of automation. In addition, it is general enough to be applied to different scenarios and model representations, and can efficiently handle single and multidimensional problems with large sets of parameters. The proposed parallelization is done at the algorithmic level and has been proven to be very effective regardless of the model's original size and characteristics. The major benefit from the proposed framework is the potential to tackle large and complex models depending on multiple parameters in an automatic fashion in a reasonable time. Large processing farms are available that can be used to reduce extremely challenging models, otherwise difficult to address with sequential approaches.

ACKNOWLEDGMENT

The authors would like to thank K. J. van der Kolk and N. van der Meijs (T. U. Delft, Delft, The Netherlands) for providing the K-Spiral example, T. El-Moselhy (Massachusetts Institute of Technology, Cambridge) for the Heat example, and G. Ciuprina and D. Ioan (P. U. Bucharest, Bucharest, Romania) for the D-Spiral example. They would also like to thank R. Janssen (NXP, Eindhoven, The Netherlands) and W. H. A. Schilders (T. U. Eindhoven, Eindhoven).

REFERENCES

[1] J. F. Villena and L. M. Silveira, "3POr: Parallel projection based parameterized order reduction for multidimensional linear models," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2010, pp. 536–542.

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

[3] W. H. A. Schilders, H. A. van der Vorst, and J. Rommes, Eds., *Model Order Reduction: Theory, Research Aspects and Applications* (Mathematics in Industry, The European Consortium for Mathematics in Industry, vol. 13). Berlin, Germany: Springer, 2008.
[4] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: Passive reduced order interconnect macromodeling algorithm," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 8, pp. 645–654, Aug. 1998.
[5] J. R. Phillips and L. M. Silveira, "Poor man's TBR: A simple model reduction scheme," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 1, pp. 43–55, Jan. 2005.
[6] B. Moore, "Principal component analysis in linear systems: Controllability, observability, and model reduction," *IEEE Trans. Autom. Control*, vol. 26, no. 1, pp. 17–32, Feb. 1981.
[7] L. Daniel, O. C. Siong, S. C. Low, K. H. Lee, and J. K. White, "A multiparameter moment-matching model-reduction approach for generating geometrically parametrized interconnect performance models," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 5, pp. 678–693, May 2004.
[8] Z. Zhu and J. Phillips, "Random sampling of moment graph: A stochastic Krylov-reduction algorithm," in *Proc. Des. Autom. Test Eur. Conf. Exhibit.*, Apr. 2007, pp. 1502–1507.
[9] Y. Li, Z. Bai, Y. Su, and X. Zeng, "Model order reduction of parameterized interconnect networks via a two-directional Arnoldi process," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 9, pp. 1571–1582, Sep. 2008.
[10] J. Phillips, "Variational interconnect analysis via PMTBR," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2004, pp. 872–879.
[11] J. F. Villena and L. M. Silveira, "Multidimensional automatic sampling schemes for multipoint modeling methodologies," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 30, no. 8, pp. 1141–1151, Aug. 2011.
[12] P. Benner, E. S. Quintana-Ortí, and G. Quintana-Ortí, "Balanced truncation model reduction of large-scale dense systems on parallel computers," *Math. Comput. Modeling Dyn. Syst.*, vol. 6, no. 4, pp. 383–405, 2000.
[13] P. Benner, E. S. Quintana-Ortí, and G. Quintana-Ortí, "State-space truncation methods for parallel model reduction of large-scale systems," *Parallel Comput.*, vol. 29, nos. 11–12, pp. 1701–1722, Nov.–Dec. 2003.
[14] J. M. Badía, P. Benner, R. Mayo, and E. S. Quintana-Ortí, "Parallel algorithms for balanced truncation model reduction of sparse systems," in *Lecture Notes in Computer Science*, vol. 3732. Berlin, Germany: Springer-Verlag, 2006, pp. 267–275.
[15] D. Ioan, G. Ciuprina, and W. H. A. Schilders, "Parametric models based on the adjoint field technique for RF passive integrated components," *IEEE Trans. Magn.*, vol. 44, no. 6, pp. 1658–1661, Jun. 2008.
[16] T. El-Moselhy and L. Daniel, "Stochastic integral equation solver for efficient variation-aware interconnect extraction," in *Proc. ACM/IEEE DAC*, Jun. 2008, pp. 415–420.
[17] Y. Bi, K.-J. van der Kolk, and N. van der Meijs, "Sensitivity computation using domain-decomposition for boundary element method based capacitance extractors," in *Proc. IEEE CICC*, Sep. 2009, pp. 423–426.
[18] J. W. Demmel, J. R. Gilbert, and X. S. Li, "An asynchronous parallel supernodal algorithm for sparse gaussian elimination," *SIAM J. Matrix Anal. Applicat.*, vol. 20, no. 4, pp. 915–952, 1999.
[19] X. S. Li and J. W. Demmel, "SuperLU DIST: A scalable distributed memory sparse direct solver for unsymmetric linear systems," *ACM Trans. Math. Softw.*, vol. 29, no. 2, pp. 110–140, Jun. 2003.
[20] P. Bratley, B. L. Fox, and H. Niederreiter, "Implementation and tests of low-discrepancy sequences," *ACM Trans. Modeling Comput. Simul.*, vol. 2, no. 3, pp. 195–213, Jul. 1992.
[21] C. H. Bischof and G. Quintana-Ortí, "Computing rank-revealing QR factorizations of dense matrices," *ACM Trans. Math. Softw.*, vol. 24, no. 2, pp. 226–253, 1998.
[22] *The OpenMP API Specification for Parallel Programming* [Online]. Available: http://openmp.org/wp
[23] *The Open MPI Project* [Online]. Available: http://www.open-mpi.org
[24] T. A. Davis, "Direct methods for sparse linear systems," in *The Fundamentals of Algorithms*. Philadelphia, PA: SIAM, Sep. 2006.
[25] T. A. Davis, "A column pre-ordering strategy for the unsymmetric-pattern multifrontal method," *ACM Trans. Math. Softw.*, vol. 30, no. 2, pp. 165–195, Jun. 2004.
[26] T. A. Davis, "Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method," *ACM Trans. Math. Softw.*, vol. 30, no. 2, pp. 196–199, Jun. 2004.
[27] T. A. Davis, J. R. Gilbert, S. Larimore, and E. Ng, "A column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 353–376, Sep. 2004.
[28] T. A. Davis, J. R. Gilbert, S. Larimore, and E. Ng, "Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 377–380, Sep. 2004.
[29] P. Amestoy, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, an approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 381–380, Sep. 2004.
[30] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users Guide*, 3rd ed. Philadelphia, PA: SIAM, 1999.
[31] T. Weiland, "A discretization method for the solution of Maxwell's equations for six-component fields," *Electron. Commun. AEUE*, vol. 31, no. 3, pp. 116–120, 1977.
[32] G. Ciuprina, D. Ioan, and D. Mihalache, "Magnetic hooks in the finite integration technique: A way toward domain decomposition," in *Proc. IEEE Conf. Electromagn. Field Comput.*, May 2008, pp. 391–394.
[33] T. E. Moselhy and Y. Marzouk, "An adaptive iterative method for high dimensional stochastic PDEs," 2011, to be published.

Jorge Fernández Villena (S'06–M'11) was born in Avilés, Spain. He received the Engineers degree in telecommunications from Cantabria University, Santander, Spain, in 2005, and the Ph.D. degree in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 2010.

He is currently a Researcher with the Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa, Lisbon. His current research interests include integrated circuit interconnect modeling and simulation, with emphasis in parameterized model order reduction.

Luís Miguel Silveira (S'85–M'95–SM'00) was born in Lisbon, Portugal. He received the Engineers (summa cum laude) and Masters degrees in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, in 1986 and 1989, respectively, and the M.S., E.E., and Ph.D. degrees in electrical and computer engineering from the Massachusetts Institute of Technology, Cambridge, in 1990, 1991, and 1994, respectively.

He is currently a Professor of electrical and computer engineering with the Instituto Superior Técnico, Technical University of Lisbon, a Senior Researcher with the Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa, Lisbon, and a Founding Member of the Lisbon Center of the Cadence Research Laboratories, Lisbon. He has received notable paper citations at various conferences, including ECTC, ICCAD, DAC, DATE, VLSI-SoC, and SBCCI. His current research interests include various aspects of computer-aided design of integrated circuits, with emphasis on interconnect modeling and simulation, parallel computer algorithms, and the theoretical and practical issues concerning numerical simulation methods for circuit design problems.

Dr. Silveira is a member of Sigma Xi.