3POr - Parallel Projection based Parameterized Order reduction for multi-dimensional linear models

Jorge Fernández Villena*, Luís Miguel Silveira*§
*INESC ID / IST, TU Lisbon. Rua Alves Redol 9, 1000-029 Lisbon, Portugal. *jorge.fernandez@inesc-id.pt*
§Cadence Research Labs. Rua Alves Redol 9, 1000-029 Lisbon, Portugal. *lms@inesc-id.pt*

Abstract— **This paper introduces a distributed and shared memory parallel projection based model order reduction framework for parameterized linear systems. The proposed methodology is based on a sampling scheme followed by a projection to build the reduced model. It exploits the parallel nature of the sampling methods to improve the efficiency of the basis generation. The sample selection scheme uses the residue as a proxy for the model error in order to improve automation and maximize the effectiveness of the sampling step. This yields an automatic and reliable methodology, able to handle large systems depending on the frequency and multiple parameters. The framework can be used in shared and distributed memory architectures separately or in conjunction. It is able to deal with different system representations and models of different characteristics, is highly scalable and the parallelization is very effective, as will be demonstrated on a variety of industrial benchmarks, with super linear speed-ups in certain cases. The methodology provides the potential to tackle large and complex models, depending on multiple parameters in an automatic fashion.**

I. Introduction

Model Order Reduction (MOR) methodologies are a set of techniques aimed at compressing the information contained in detailed models representing physical effects [1]. Within the EDA industry, these techniques have been routinely applied to compress excessively large models obtained after modeling and extraction steps, allowing for efficient simulation. In the case linear models, the most relevant MOR techniques can be broadly characterized into those based on subspace generation and projection [2], [3], and those based on balancing techniques [1]. MOR methodologies have also been proposed to handle parameterized systems, where the response depends on the frequency plus a set of parameters modeling environmental and process variations affecting the underlying physical system. The goal of these Parameterized Model Order Reduction (pMOR) methods is to reduce the order of linear systems depending on multiple dimensions, and generate a Reduced Order Model with equivalent I/O response for the whole frequency and parameter space of interest. Many pMOR methods rely on the generation of a low order subspace spanning the solution of the system, and projection of the original system into that subspace. Different techniques propose different methods for the subspace generation, either based on multi-dimensional moment matching [4], [5], or in multi-point sampling approaches [6], [7].

Nowadays the extensive use of multi-core architectures, GPUs, and distributed environments are revolutionizing the algorithmic implementation, allowing for widespread low-cost high-performance parallel approaches able to overcome some of the existing bottlenecks in some applications. However, there is little work in the use of these architectures to the framework of MOR, and most of it is aimed at parallelizing the linear algebra routines that the algorithms are based upon [8]. A particular limitation of current EDA oriented MOR methods comes from the memory footprint even though storage of these sparse models is not necessarily the main issue. However, the reduction of such models usually relies on factorizations and decompositions of the matrices (e.g. **LU** factorizations for the generation of Krylov subspaces or for solving the system for sampling). Memory requirements in these operations can be very high for multiple expansion points. To deal with this issue, iterative methods with incomplete factors as preconditioners have been presented and successfully applied [9]. Also, some approaches for sparse factorizations [10], [11] are reporting excellent results in terms of fill-in minimization. In this respect, some applications have successfully applied parallel approaches in order to deal with the increasing size of the matrices [12], [13].

In this paper we propose a combined shared and distributed memory parallel Model Order Reduction framework for linear models depending on single and multiple dimensions. The proposed methodology will not be focused on the parallelization of the underlying linear algebra routines. Instead it will be geared towards to a higher level parallelization that will prove to be very effective and scalable, both in terms of initial model and reduced model sizes. The framework relies on multi-dimensional sample-based methods [6] for the generation of a suitable basis, combined with an automated sample selection approach such as the one in [7]. It will be shown that the procedure is robust and reliable, providing good results in many situations, and also very flexible, allowing the handling of models with different representations and characteristics without degrading the efficiency. As a result, it enables the reduction of large models depending upon multiple parameters in a fraction of the original time.

The 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 work. In Section III the initial sequential approach is introduced, and in Section IV the proposed implementation is presented, along with a study of its complexity and 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

The main techniques in MOR are geared toward the reduction of a state space linear time-invariant system, obtained by some modeling methodology, and representing a physical system. In such representation, 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 parametric 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

This work was partially supported by FCT through the PIDDAC Program funds. Jorge Fernández Villena was supported by FCT grant SFRH/BD/61213/2009.

parameters $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_Q] \in \mathbb{R}^Q$ which model the effects of the uncertainty. Usually the system is formulated so that 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 [14], [15]. Therefore, matrices C and G in (1) can be represented in a polynomial form via a Taylor Series expansion with respect to the parameters, as advocated in [4]

$$\begin{array}{l} G(\lambda) = G_{0\ldots 0} + \sum_{\phi_1\ldots\phi_Q} \Lambda_{\phi_1\ldots\phi_Q} G_{\psi_1\ldots\psi_Q} \\ C(\lambda) = C_{0\ldots 0} + \sum_{\phi_1\ldots\phi_Q} \Lambda_{\phi_1\ldots\phi_Q} C_{\phi_1\ldots\phi_Q} \end{array} \tag{2}$$

where $G_{0\cdots 0}$ and $C_{0\cdots 0}$ are the nominal values for the matrices, $G_{\phi_1\ldots\phi_Q}$ and $C_{\phi_1\ldots\phi_Q}$ are the joint sensitivities of order $(\phi_1 \ldots \phi_Q)$ w.r.t. to the Q parameters, and $\Lambda_{\phi_1\ldots\phi_Q} = \lambda_1^{\phi_1} \ldots \lambda_Q^{\phi_Q}$.

The most common procedure to obtain an accurate and structurally similar ROM is to use some form of projection scheme on a sensitivity-based Taylor Series Representation, such as (2), combined with an orthogonal projection scheme, as presented in [4]. Standard pMOR methodologies rely on the generation of a suitable low order subspace (spanned by the basis $V \in \mathbb{R}^{n\times q}$), in which the original system matrices $C(\lambda)$, $G(\lambda)$, B and E are projected. Then a reduced model such as (4) can be obtained, that captures the behavior of the system under parameter variations.

$$\begin{array}{lll} \widehat{C}_{\phi_1\ldots\phi_Q} = V^T C_{\phi_1\ldots\phi_Q} V & \widehat{G}_{\phi_1\ldots\phi_Q} = V^T G_{\phi_1\ldots\phi_Q} V & \\ \widehat{B} = V^T B & \widehat{E} = EV & x(s,\lambda) = V\widehat{x}(s,\lambda) \end{array} \tag{3}$$

where $V \in \mathbb{R}^{n\times q}$ spans the projection subspace of reduced dimension q, and $\widehat{C}, \widehat{G} \in \mathbb{R}^{q\times q}$, $\widehat{B} \in \mathbb{R}^{q\times m}$, $\widehat{E} \in \mathbb{R}^{p\times q}$, 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,

$$\widehat{H}(s,\lambda) = \widehat{E}(s\widehat{C}(\lambda) + \widehat{G}(\lambda))^{-1}\widehat{B} \approx H(s,\lambda) \quad \forall \{s,\lambda\}. \tag{4}$$

To ensure the ROM's accuracy 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, e.g.,

$$x(s,\lambda) \approx \sum_{i=0}^{q} \alpha_i(s,\lambda) V_i \quad \forall \{s,\lambda\}, \tag{5}$$

where V_i is the i-th 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. Most of the techniques in the literature choose to extend the moment matching paradigm [2] to the multi-dimensional case [4], [5]. Therefore they generate a basis V that spans the multi-dimensional moments of the transfer function around an expansion point, and can be used as a projection matrix. In general, these methods, which rely in local matching, suffer from oversize of the models when the number of moments to match is high, either because high order is required, or because the number of parameters is large. Also they need a Taylor Series based representation (2) for efficient implementation. Other approaches are based on Multi-Point methods. The goal of Multi-Point approaches is to generate the 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,

$$z_j = z(\psi_j) = A(\psi_j)^{-1}B = (s_j C(\lambda_j) + G(\lambda_j))^{-1}B, \tag{6}$$

where $z(\psi_j) \in \mathbb{C}^n$ is the sample vector generated at the sample point $\psi_j = \{s_j, \lambda_j\}$, and is directly related to the zero-th order moment at ψ_j. The most relevant vectors among those generated by such quadrature are selected, for instance, via *Singular Value Decomposition* (SVD), 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. Recently, in [7], an automatic sampling scheme was presented, 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 (5) holds. The approach seeks to retrieve the "best" samples to solve for among an initial candidate set, with the selection done before actually solving them.

III. Automated Sequential Approach

The underlying algorithm that we will use is the one presented in [7]. The sample selection is done by finding which point from an initial candidate set Ψ, has an associated vector that is less similar to the vectors already computed. This is done by using the residue r_j as a proxy for the error e_j,

$$r_j = A_j e_j = B - \sum_{i=0}^{k} \alpha_i(\psi_j) A_j V_i, \tag{7}$$

where $A_j = A(\psi_j)$ is the system matrix evaluated at the candidate point ψ_j, and V_i is the i-th vector of the current basis $V \in \mathbb{R}^{n\times k}$ at the current refinement iteration of the algorithm. Therefore, to see if the system $\{A_j, B\}$ is well approximated by a set of vectors V, we simply orthogonalize vector B against the set of vectors $A_j V \in \mathbb{C}^{n\times k}$, and generate the norm

$$\|r_j\| = \|B \perp A_j V\|, \tag{8}$$

where r_j is the residue after the orthogonalization ($\perp$) of B against the set of vectors $A_j V$. The advantage of this procedure is that (7) entails a cheap computation, whereas (6) is expensive. Next we describe some details of the implementation proposed.

A. Sequential Implementation

The initial candidate set $\Psi = \{\psi_1 \ldots \psi_T\}$ is generated, with T samples, where each ψ_j is a point in the space of interest. Ψ is a set that covers such space of interest (joint frequency and parameter space). In order to improve the coverage of this initial set, a logarithmic mesh is defined in the complex frequency, and for each frequency point different parameter perturbations are performed following a Low-Discrepancy sequence [16]. The reason for the differentiated treatment for frequency comes from the fact that the range of variation of frequency is much wider, and frequency has in general a larger impact in the performance of electric circuits than the remaining parameter perturbations. Low-Discrepancy sequences are routinely used in numerical integration of large dimensional problems. Since we are doing a numerical quadrature in a large dimensional space, the points generated by such schemes provide a better alternative to linear or pure random schemes, and ensure that each sample is different.

From this initial set, a candidate sample is taken and solved to generate the first vector. It is orthonormalized and its real and imaginary parts are added to the real basis V (recall that the samples are complex, and thus we need the real and imaginary part to span the same subspace). We then perform the operations in (8) to generate the residues at each remaining candidate point. From these residues, we select the point with maximum residue norm, and solve it. The vector generated is orthonormalized against the previous ones (via an incremental Rank Revealing QR operation **RRQR** [17]), and added to the basis V. The procedure is then iteratively repeated: residue generation, sample selection, sample solve and orthonormalization. In order to stop the procedure, a tolerance for the norm of the vectors and the residue is set. When both the norm of the generated vector after the **RRQR** and the maximum norm of the residue fall below a pre-defined threshold, the procedure is stopped: the small residue indicates that the approximation in the remaining samples is good enough, and the small norm of the vector indicates that the generated vector does not add rank to the current basis. A more clear depiction can be seen in Algorithm 1.

```
Algorithm 1 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 candidate sample points $\Psi = \{\psi_1 \ldots \psi_T\}$
 3: Initialize: $V$=[ ], $k$=0
 4: Evaluate system matrix: $A_k$=$A(\psi_k)$
 5: Solve the system: $z_k$=$A_k^{-1}B$
 6: Orthonormalization: $V$= RRQR([$V z_k$]), $k$++
 7: For $i$=$k$ : $T$
        $A_i$=$A(\psi_i)$, $r_i$=$B \perp A_i V$, $R(i)$ = $\|r_i\|$
 8: Sort $R$ in decreasing order, and $\Psi$ accordingly
 9: IF $R(k) > t_r$ and $\|v_k\| > t_v$
        GOTO 4
10: Use $V$ in a congruence projection on the system
```

B. Computational Complexity Analysis

The cost of this algorithm can be divided into three main parts: the cost of solve (the **LU** or the methodology used for solving the system, if an iterative method is used), the cost of the orthonormalization and basis expansion, and the cost of the sample selection. The cost of the (**LU** and) solve for sparse matrices is very dependent on the matrix characteristics in terms of sparsity pattern and number of non-zeros. For the case of banded matrices arising for linear circuits, its cost is close to linear, but it may be different for matrices generated by **EM** modeling methodologies, or if long inductive effects are taken into account. 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 $\mathcal{X}_{solve} = O(KS)$.

The cost of the orthonormalization, a **RRQR** of a rectangular matrix of dimensions $n \times q$ is $O(nq^2)$, and thus, after generating K block vectors of size m (m the number of ports), the overall cost of orthonormalization is $\mathcal{X}_{orth} = O(nm^2K^2)$.

For the sample selection, the cost is dominated by the computation of the residue, which requires the orthogonalization of B against AV_j 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 speed-up the residue computation but lead to huge memory requirements). 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

$$\mathcal{X}_{ss} = O(\textstyle\sum_{i=1}^{K}(T-i)n(im)^2), \tag{9}$$

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 we arrive at the following expression for the right-hand side of (9)

$$O(\frac{Tnm^2}{6}(2K^3 + 3K^2 + K) - \frac{nm^2}{4}(K^4 + 2K^3 + K^2)), \tag{10}$$

which can be simplified to

$$\mathcal{X}_{ss} = O(nm^2K^3(T-K)). \tag{11}$$

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. The overall final cost is the sum of solve, RRQR and sample selection costs.

IV. Parallel Implementations

Although the methodology outlined in the previous section provides goods results in terms of accuracy and reduction for both single and multi-dimensional systems, it can be expensive under certain circumstances, namely for large domains, with multiple dimensions, and large size, both in the original and reduced systems.

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 that the goal of this work is not related to efficient parallelization of the underlying linear algebra routines (which can nevertheless be combined with the proposed methodology to provide further speed-ups). Instead, we will focus on parallelization at a higher, more abstract level, that provides both high performance in terms of speed-up and scalability.

With the advent of multi-core 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. However, the characteristics of these architectures lead to different parallelization strategies. 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 multi-core 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. Distributed memory environments are a set of connected machines or processors, named nodes, each one with its own memory, so whenever remote data is required, an explicit communication must be performed. 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 a relevant factor in the performance, and thus needs to be minimized. Coarse granularity parallelization is prone to be applied here, with operations as independent as possible. On the other hand, since each node has its own memory, the total amount of memory is increased, thus larger problems or higher level parallelization becomes amenable.

A. Parallelization Opportunities

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** [18] and **OpenMPI** [19] languages, which are the most commonly used libraries and the ones we used in our implementation.

Looking at (11), 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

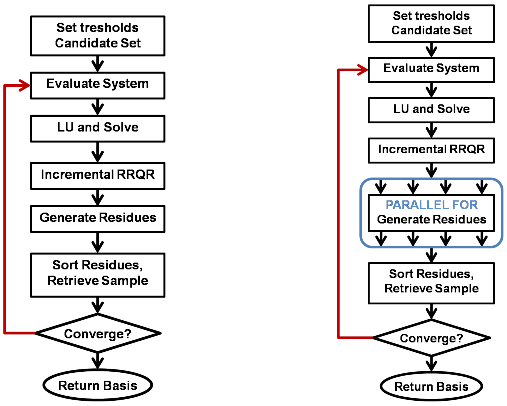

Fig. 1. (Left) Serial and (Right) Multi-Core Algorithm Graphs

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 an almost perfect speed up for this sample selection step. Here we have used a **parallel for** cycle with **dynamic schedule**, which reduces the cost of the sample selection step by a factor of P. An illustration of this flow and comparison with a sequential approach is shown in Figure 1.

A different kind of parallelization which can be easily combined with the shared memory approach above, is to use the distributed environment. In this case, the communication cost may be larger, but the possibility of using several machines provides more CPU capabilities, as well as more memory. This fact has been exploited by parallel implementations of linear algebra routines [13], also with application to MOR [8]. In the proposed method, we can exploit these 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 among 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 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 should be used, in order to maintain a good balancing and avoid idle computational resources. In such cases, an initial distribution of a subset of the candidates can be done, and the rest can be assigned later according to the workload at each node. In order to reduce communication delays, non-blocking communications can be used and overlapped with the computations. With respect to the solve step, from the analysis in III-B, 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. As mentioned, in this paper we are not addressing the parallelization of the solve, and thus not trying to improve the complexity $O(S)$ (which can however be done and taken advantage of here). On the other hand, the cost related to the factor K can be efficiently addressed. The solve operation is expensive both in terms of CPU and memory. Therefore, to perform several solves in parallel is not an option in shared memory architectures (note that each solve relates to a different sample point and essentially uses a different matrix). On the other hand, each node in a distributed environment has its own memory, so in this case the solves can be done in parallel. If the system is already loaded in the node, it only needs the point to solve, and thus the communication overhead is minimum. Once we have selected a suitable point for solving, each solve can be done independently on each separate machine. In order to cope with the memory limitations, sparse factorizations or iterative methods can be applied. Since this operation is done in parallel, at each iteration we are solving M points, with a potential reduction of the overall solve time by a factor of M.

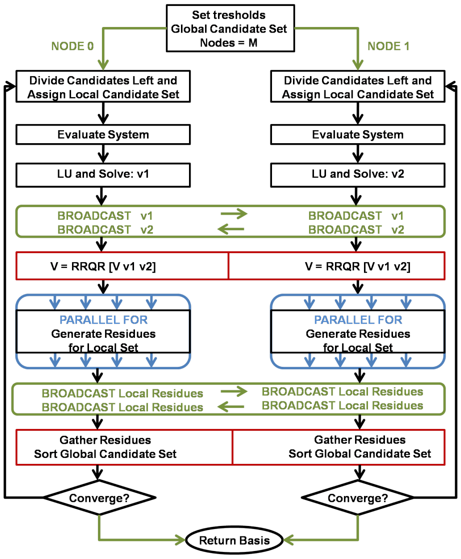

Fig. 2. Distributed Multi-Core Algorithm Graph

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: $V$=[ ], $k$=0
 7: Mapping Local set: $i$=$kM$, $T_N$=0, While($i < T$)
      $\Upsilon_N$=$[\Upsilon_N \Upsilon(i : i + \Delta)]$, $i$+=$\Delta$, $T_N$+=$\Delta$
 8: Select first local sample: $\psi_j$, $j$=$\Upsilon_N(0)$
 9: Evaluate system matrix: $A_j$=$A(\psi_j)$
10: Solve the system: $z_j$=$A_j^{-1}B$, $X(:, N)$=$z_j$
11: BROADCAST $X$
12: Orthonormalization: $V$=RRQR($[V \quad X]$)
13: PARALLEL FOR DYNAMIC $i$=$1 : T_N$
      $j$=$\Upsilon_N(i)$, $r_i$=$B \perp A_j V$, $R_N(i)$=$\|r_i\|$
14: BROADCAST $R_N$ so that $R$=$[R_0 \dots R_{M-1}]$
15: BROADCAST $\Upsilon_N$ so that $\Upsilon$=$[\Upsilon_0 \dots \Upsilon_{M-1}]$
16: Sort $R$ in decreasing order, and Global $\Upsilon$ accordingly, $k$++
17: IF $R(0) > t_r$ and $\|v_k\| > t_v$
      Set $\Delta$ to desired size, $\Delta < T - kM$, and GOTO 7
18: Use $V$ in a congruence projection on the system
```

B. 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 Figure 2.

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.

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 N-th 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 RRQR 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 cache hits for efficiency, this option is preferred to a distributed implementation. However, shared memory or GPU based implementations could also be applied here.

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 multi-core 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.

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, k=0, 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 $k > 0$, 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

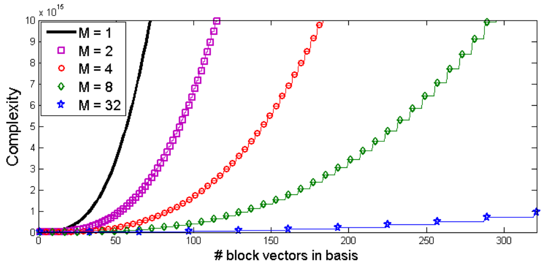

Fig. 3. Theoretical complexity growth of the sample step (14) with the number of block vectors in basis, for M=$1, 2, 4, 8$ and 32 nodes and P=1, with n=$1e6$, m=2, and T=5000. Markers indicate when a new sample is solved (i.e. a new iteration: K++).

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.

C. 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 overall cost of factorization and solve is

$$\mathcal{X}_{MPsolve} = O(\lceil K/M \rceil\, S). \qquad (12)$$

Potentially, if the cost of the solves dominates, a speed-up 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),

$$\mathcal{X}_{MPorth} = O(nm^2K^2). \qquad (13)$$

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

$$\mathcal{X}_{MPss} = O(\textstyle\sum_{i=1}^{\lceil K/M \rceil} \frac{(T-iM)}{MP} n(iMm)^2), \qquad (14)$$

which after some algebra, similarly to (10) can be approximated by

$$\mathcal{X}_{MPss} = O(\tfrac{nm^2M}{P} \lceil \tfrac{K}{M} \rceil^3 (T - M\lceil \tfrac{K}{M} \rceil). \qquad (15)$$

Notice that we reduce the number of residues to generate by each core by a factor of MP. Since these operations are independent, the speed-up 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 speed-up larger than MP.

To illustrate this, Figure 3 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 (14) with n=$1e6$, m=2, and T=5000, for M=$1, 2, 4, 8,$ and 32, with a single core (P=1). These results show the potential of the proposed parallelization, that can achieve improvements larger than MP (close to M^2P for a large number of iterations). However, these are theoretical results, and in general, as the number of machines increases, the optimality of the sample selection is decreased (only one sample is optimal in terms of maximum residue). This is translated into slightly larger reduced orders. In addition, communication costs can slow down the overall performance.

TABLE I
BENCHMARKS CHARACTERISTICS AND SEQUENTIAL REDUCTION

	Example	Sp-Nwell	K-Spiral	pRLC
Model Features	Modeling	FIT	K-method	MNA
	Order	1^{st}	2^{nd}	1^{st}
	Size	39759	89134	30011
	λ / Terms	–/–	1/4	15/15
	ω (GHz)	0.5–60	40–60	0–200
	# Validation	10000	6000	50000
	# Candidate	1179	700	6048
Serial Times	Sample Selection	$17'40''$	$605'41''$	*
	LU and Solve	$37'33''$	$99'22''$	*
	RRQR	$0'01''$	$0'18''$	*
	Overall	$55'16''$	$706'17''$	*

The overall cost can be approximated by the sum of the costs in (12), (13) and (14), plus the communication costs. In cases where the sample selection elapsed time is a large fraction of the overall time, potential super linear speed-ups can be achieved, with good scalability. In addition, the parallelization effectiveness in terms of speed-up is increased with the size of the subspace required for a good approximation (i.e. the number of vectors in the basis, and thus the iterations required for the convergence). This comes from the fact that we are reducing the number of iterations by a factor of M, but also the residue generation at each machine is done on a smaller number of candidates.

V. SIMULATION RESULTS

The algorithms discussed were implemented in **C/C++**, with double precision and using **OpenMP** and **OpenMPI** for the parallelization. As mentioned parallelization was not applied to the underlying algebra routines, and thus the results could still be considerably improved. For the **LU** and solve, the sequential **KLU** algorithm of the SuiteSparse package [10], [11] was used, in order to minimize memory requirements. Tests were run on a cluster of 8 Intel $Q6600$ (4-core) @2.4GHz machines with $8G$ RAM, connected by a dedicated gigabit net. In the **KLU** routines we used **AMD** for the symmetric second order systems, and **COLAMD** for the non-symmetric first order systems. For the sampling, no prior knowledge of the system was assumed, other than frequency and parameter variation ranges.

As benchmarks we will use an example depending solely on the frequency, and two parameterized systems. **Sp-Nwell** is a non-parameterized EM model of an integrated spiral inductor over an N-Well, obtained with FIT, and taking into account substrate and upper air. **K-spiral** is an RCK second order EM model of a integrated spiral inductor, whose side length varies up to 37um around the nominal value of 187um, and is modeled with a 4^{th} order Taylor Series. **pRLC** is an distributed RLC model of a interconnect with 5 metal lines and 3 ports, in its MNA formulation. Each line depends on 3 parameters, with total of 15 parameters (plus the frequency). Table I presents the characteristics of the benchmarks, including the number of candidate sets for the procedure and the number of points used to compute the error in the validation of the ROMs. Also shown are detailed reduction times for each benchmark, divided among the main

TABLE II
PARALLELIZATION AND REDUCTION RESULTS

	# of	ROM size	1	4	Abs.Err.
	Nodes	Iterations	core	cores	Rel.Err.
SP-Nwell	M = 1	24	$55'16''$	$41'56''$	$8.8e-4$
		12	$1X$	$1.318X$	$7.3e-6$
	M = 2	24	$24'21''$	$20'23''$	$1.5e-4$
		6	$2.269X$	$2.711X$	$4.6e-6$
	M = 4	32	$15'48''$	$13'37''$	$2.4e-3$
		4	$3.497X$	$4.058X$	$2.6e-5$
	M = 8	44	$11'32''$	$10'13''$	$6.1e-4$
		3	$4.791X$	$5.409X$	$4.9e-6$
K-Spiral	M = 1	102	$706'17''$	$280'6''$	$1.12e-2$
		51	$1X$	$2.521X$	$1.7e-6$
	M = 2	110	$323'54''$	$138'11''$	$1.00e-2$
		28	$2.180X$	$5.111X$	$1.5e-6$
	M = 4	117	$144'40''$	$67'40''$	$1.12e-2$
		15	$4.882X$	$10.437X$	$1.7e-6$
	M = 8	124	$65'15''$	$33'21''$	$1.04e-2$
		8	$10.824X$	$21.177X$	$1.6e-6$

steps (for the **pRLC** model, * indicates the simulation was aborted due to excessive time). Although mid-size models, it can be seen that in some cases the cost of solving the system is relatively large. The proposed framework is able to maintain, and even increase, its effectiveness for larger models.

For the **Sp-Nwell**, most of the time is devoted to the sampling (solves) in the sequential algorithm. Therefore, in this case, the parallelization gain obtained is mostly due to the fact that we perform M solves at each step. Table II shows the performance of the algorithms, for different number of nodes, and for single and 4 core machines. Since the parallelization of the solves is driven by the number of nodes, the speed-up is close to M. The use of the multi-core approach here provides some extra gain, but is not critical since the number of iterations required is small (and so is the ROM size), and the parallelizable fraction in the shared memory parallelization is small (the sample selection takes $\approx 32\%$ of the overall elapsed time in the sequential case). This of course limits the scalability of the method (at some point we are generating more vectors than required for the basis, so we can no longer reduce the number of iterations).

The **K-spiral** example is a large second order model with 5124 Resistors, 20736 Capacitors and 381120 Susceptances; each of the 3 system matrices is represented by the nominal plus 4 Taylor Series terms with respect to a design parameter that strongly affects the system behavior. The proposed framework is able to handle it with minimum modifications in the flow (evaluation of the system matrix is done with 3 matrices instead of 2). In this case both the sampling and the solves are critical in the execution time, and thus the parallelization is very effective, achieving good speed-ups. Table II shows the details on the performance. It can be seen that the distributed parallelization with single core reports super linear speed up. This is due to the fact that the small redundancy achieved is compensated by the super linear speed-up of the sample selection step (which comprises 85.76% of the sequential time). With 4 cores the speed-up is increased by a factor larger than 2, which is a reasonable speed-up if we take into account the parallelizable fraction we are using in the shared memory approach.

The **pRLC** example is shown as a qualitative case. It is a simple yet very challenging model in terms of sample selection: many parameters and a large frequency range require a large initial candidate set (6048 points), with a behavior almost "flat" at low frequencies, but with multiple resonances at high frequencies. A large number of vectors is required for aceptable accuracy. The sequential approach is unable to reduce the model in a reasonable time. On the other hand, the proposed parallelization with M=8 and 4 cores generates a 573-states ROM with 12 iterations in $326'35''$, with the desired accuracy checked in a validation set of 50000 points.

VI. CONCLUSIONS

A distributed and shared memory parallel framework for parameterized linear model order reduction has been presented. The underlying method relies on multi-dimensional 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 multi dimensional 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.

REFERENCES

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

[2] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: passive reduced-order interconnect macromodeling algorithm," *IEEE TCAD*, vol. 17, no. 8, pp. 645–654, August 1998.

[3] J. R. Phillips and L. M. Silveira, "Poor Man's TBR: A simple model reduction scheme," *IEEE TCAD*, vol. 24, no. 1, pp. 43–55, Jan. 2005.

[4] 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 TCAD*, vol. 23, no. 5, pp. 678–693, May 2004.

[5] Z. Zhu and J. Phillips, "Random sampling of moment graph: a stochastic krylov-reduction algorithm," in *Proc. DATE*, April 2007, pp. 1502–1507.

[6] J. Phillips, "Variational interconnect analysis via PMTBR," in *ICCAD*, San Jose, CA, USA, November 2004, pp. 872–879.

[7] J. F. Villena and L. M. Silveira, "ARMS - automatic residue-minimization based sampling for multi-point modeling techniques," in *Proc. DAC*, San Francisco, CA., USA, 26-31 July 2009, pp. 951–956.

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

[9] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Pws Publishing Co., 1996.

[10] T. A. Davis, "Direct methods for sparse linear systems," ser. The Fundamentals of Algorithms. SIAM, Philadelphia, September 2006.

[11] T. A. Davis and E. P. Natarajan, "Algorithm 8xx: Klu, a direct sparse solver for circuit simulation problems," *Submitted to ACM Transactions on Mathematical Software*.

[12] J. W. Demmel, J. R. Gilbert, and X. S. Li, "An asynchronous parallel supernodal algorithm for sparse gaussian elimination," *SIAM J. Matrix Analysis and Applications*, vol. 20, no. 4, pp. 915–952, 1999.

[13] X. S. Li and J. W. Demmel, "SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems," *ACM Trans. Mathematical Software*, vol. 29, no. 2, pp. 110–140, June 2003.

[14] Y. Bi, K.-J. van der Kolk, D. Ioan, and N. van der Meijs, "Sensitivity computation of interconnect capacitances with respect to geometric parameters," in *IEEE EPEP*, San Jose, CA, October 2008.

[15] B. Dewey, I. Elfadel, and T. El-Moselhy, "An efficient resistance sensitivity extraction algorithm for conductors of arbitrary shapes," in *Proc. DAC*, San Francisco, CA., USA, 26-31 July 2009, pp. 770–775.

[16] P. Bratley, B. L. Fox, and H. Niederreiter, "Implementation and tests of low-discrepancy sequences," *ACM Transactions on Modeling and Computer Simulation*, vol. 2, no. 3, pp. 195–213, July 1992.

[17] 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.

[18] The OpenMP API specification, "http://openmp.org/wp/."

[19] The Open MPI Project, "http://www.open-mpi.org/."