A Parallel Direct Solver for the Simulation of Large-scale Power/Ground Networks

Stephen Cauley, Venkataramanan Balakrishnan, and Cheng-Kok Koh School of Electrical and Computer Engineering 465 Northwestern Ave. Purdue University West Lafayette, IN 47907-2035 {stcauley,ragu,chengkok}@ecn.purdue.edu

December 7, 2009

Abstract

We present an algorithm for the fast and accurate simulation of power/ground mesh structures. Our method is a direct (non-iterative) approach for simulation based upon a parallel matrix inversion algorithm. Through the use of additional computational resources, this distributed computing technique facilitates the simulation of large-scale power/ground networks. In addition, the new dimension of flexibility provided by our algorithm allows for a more accurate analysis of power/ground mesh structures using RLC interconnect models. Specifically, we offer a method that employs a sparse approximate inverse technique to consider more reluctance coupling terms for increased accuracy of simulation. The inclusion of additional coupling terms, however, does not lead to an increase in either time or memory requirements associated with the primary computational task in transient simulation, thus making the simulation procedure *scalable*. The parallel matrix inversion algorithm shows substantial computational improvement over the best known direct and iterative numerical techniques that are applicable to these large-scale simulation problems.

1 Introduction

The accurate and efficient modeling and simulation of power/ground networks has become a difficult problem for modern design. Increases to integration density have necessitated the use of large-scale power mesh structures, and with the scaling of voltages the need for accurate simulation of these structures is crucial. Previously employed direct methods for simulation of this problem have become impractical due to both extraordinary memory requirements and prohibitive simulation times. This has prompted several variations of iterative schemes [1-6] that attempt to meet these rising computational challenges. In [1], the authors employed a Random-Walk technique where a stochastic analysis of the supply network allowed for a trade off between accuracy and simulation time (the Random-Walk based approach of [1] has been adapted to provide a preconditioner to be used with an iterative linear solver for quadratic placement [7]; this approach can be directly applied toward the transient simulation of RC mesh structures). There has also been research into applying standard domain decomposition techniques known as "alternating" procedures, which are iterative schemes used to relax the stringent memory requirements associated with power mesh simulation [2]. A similar domain decomposition based method which used the random walk procedure to specifically perform computation along the boundary nodes for the domains, within a global iterative scheme, was shown in [3]. Alternatively, a method with the same goal of facilitating large-scale mesh simulation only through a hierarchical framework was presented in [4]. In [5], a different iterative scheme, based upon the classical SOR approach, was used to iterate between either rows of the power supply grid or across groups of nodes. In addition, techniques such as the multigrid-like [6] method aim to improve simulation time by extrapolating information from reduced order systems in order to simulate the global system efficiently. The convergence for each of these methods, and therefore the simulation time, is problem dependent (i.e. both switching activity within the network and branch coupling will affect the simulation time).

Although most of these methods have been shown to be quite successful for largescale simulations (millions of nodes) of RC mesh structures, none have clearly demonstrated an efficient and scalable approach to deal with inductive coupling effects. This can be largely attributed to the fact that with the inclusion of inductive coupling, much of the locality for the problem is lost. Specifically, an iterative method that uses small independent or slightly overlapped subsets of the network in order to infer information about the global system dynamics will not converge quickly if there is significant coupling across different regions of the network. In addition, as was alluded to by the authors of [4], the conditioning of the underlying system matrices would degrade if the interconnects, which constitute the mesh structure, are modeled as RLC. By employing a parallel direct technique for simulation, we offer a stable alternative that allows for the efficient simulation of networks with a large amount of branch coupling.

In this work, we present a direct method for parallel matrix inversion that uses additional computing resources in order to stably relax both computational and memory requirements typically associated with direct techniques for simulation. Specifically, the parallel method for solving block tridiagonal systems presented in this work *scales well with the inclusion of additional reluctive coupling effects*, within an assumed block tridiagonal structure. In addition, a computationally efficient method based upon sparse



Figure 1: 4x4 RLC mesh structure.

approximate inverses will be demonstrated in order to flexibly construct the coefficient matrices for simulation. The effects on the accuracy for simulation, given these additional reluctive coupling terms, as well as the overall impact for simulation time are analyzed. Finally, we conclude that the parallel direct technique for simulation demonstrates computational advantages over existing direct and iterative methods. Our method extends the computational efficiency afforded by direct methods to consider large simulation problems. These simulation problems have typically only been analyzed through the use of iterative methods, which in general suffer from poor scalability as the amount of branch coupling being considered increases. The method presented in this work is specifically designed to avoid a computational dependence on the amount of branch coupling considered, making it ideally suited for accurate large-scale transient simulation.

1.1 Simulation of RLC Mesh Structures

When RLC interconnect models are used for the simulation of mesh structures (see Figure 1) the typical Modified Nodal Analysis (MNA) representation yields equations of the form:

$$\mathcal{G}x + \mathcal{C}\dot{x} = \mathcal{B},\tag{1}$$

where

$$\mathcal{G} = \begin{pmatrix} G & A_l^T \\ -A_l & 0 \end{pmatrix}, \quad \mathcal{C} = \begin{pmatrix} C & 0 \\ 0 & L \end{pmatrix}, \quad x = \begin{pmatrix} v_n \\ i_l \end{pmatrix},$$
$$\mathcal{B} = \begin{pmatrix} A_l^T I_s \\ 0 \end{pmatrix}, \quad G = A_g^T R^{-1} A_g, \text{ and } C = A_c^T \hat{C} A_c.$$

Here, R, \hat{C} , and L are the resistance, capacitance, and inductance matrices respectively. The matrices A_g and A_c transform the conductances and capacitances into node based relationships. The matrices A_l and A_i link the node voltages and branch currents described by the state variable x. In addition, I_s is the current vector that dictates, through the matrix A_i , the relationship of the current sinks onto the nodes of the mesh.

A classical algorithm for the numerical integration of ordinary differential equations such as (1) is the trapezoidal method [8]. Consider a uniform discretization of the time axis with resolution *h*. Then, using the notation $v_n^k = v_n(kh)$ to denote the voltage at the *n*th node we may then solve for v^{k+1} in terms of v^k through the Nodal Analysis (NA) equations [9]:

$$\underbrace{\left(G + \frac{2}{h}C + \frac{h}{2}S\right)}_{K} v_{n}^{k+1} = \underbrace{\left(-G + \frac{2}{h}C - \frac{h}{2}S\right)}_{H} v_{n}^{k} + A_{i}^{T} \left(I_{s}^{k+1} + I_{s}^{k}\right) - 2A_{l}^{T} i_{l}^{k}, \qquad (2)$$
$$2A_{l}^{T} i_{l}^{k+1} = 2A_{l}^{T} i_{l}^{k} + hS \left(v_{n}^{k+1} + v_{n}^{k}\right),$$

where $S = A_l^T L^{-1} A_l$. It is important to note that with the inclusion of inductance, for the modeling of the interconnects, we must now account for the effect of this additional susceptance term *S*.

For the transient simulation of RC mesh structures, the coefficient matrix K defined in (2) was symmetric, positive-definite, and sparse, which allowed for the use of fast and numerical stable direct techniques for simulation, e.g. the Cholesky factorization [10], multi-frontal methods [11], and other general sparse LU decompositions. In addition, the diagonally dominance property of the matrix K, along with the small number of non-zero entries, made the problem ideally suited for iterative schemes such as [1–6], including the conjugate gradient method [10] and the generalized minimal residual method [12]. As the problem size of interest has grown, the memory requirements of the direct approaches has restricted their use. Typically they are only utilized in conjunction with decomposition based iterative schemes such as [2,4], or as part of a hybrid iterative and direct scheme [3].

When considering RLC mesh structures, the simulation methods described above are restricted by the use of iterative methods, which in general suffer from poor scalability as the amount of branch coupling being considered increases. For example, the presence of the susceptance term in (2) precludes the use of the original random walk [1] and the new hybrid random walk [7] directly for the matrix K. These algorithms rely on the formulation of a stochastic game which at its foundation places a requirement that the coefficient matrix K have positive diagonal entries and non-positive off-diagonal entries. As was shown in [1] the random walk based methods could only be used if inductive coupling effects were captured through the inclusion of an additional self-consistent iterative scheme.

The scalable algorithm presented in this work avoids the need for any iteration during transient simulation. In addition, this technique allows for the dominant computational task associated with (2) to be performed efficiently and exactly by exploiting

a distributed computing environment. Specifically, we offer a parallel matrix inversion algorithm for the transient simulation of RLC mesh structures. Our algorithm fully exploits the structure of the coefficient matrix K, during each solution of linear equations $Kx_i = c_i$, across the time steps i. This inherently parallel algorithm facilitates a fast and distributed matrix-vector multiplication to evaluate $x_i = K^{-1}c_i$, for each time step of the simulation. This matrix-vector multiplication, and therefore the largest portion of the transient simulation time, is not affected by increases in simulation accuracy through the inclusion of additional reluctance terms.

2 Coefficient Matrices for Simulation

We begin first with the construction of the coefficient matrix K from (2), given a regular power mesh topology. The 4 × 4 mesh shown in Figure 1 has a total of 40 nodes. Each direction of the mesh has three groups of four parallel wires and if all mutual inductive couplings are considered both the reluctance matrix L^{-1} and coefficient matrix K will be dense. Thus, in this work we investigate efficiency and accuracy for the simulation of power mesh structures when considering approximations to the reluctance matrix that will finally result in a block tridiagonal susceptance matrix. A matrix Y is block tridiagonal if it has the form

$$Y = \begin{pmatrix} A_1 & -B_1 & & \\ -B_1^T & A_2 & -B_2 & & \\ & \ddots & \ddots & \ddots & \\ & & -B_{N_y-2}^T & A_{N_y-1} & -B_{N_y-1} \\ & & & -B_{N_y-1}^T & A_{N_y} \end{pmatrix},$$
(3)

where each $A_i, B_i \in \mathbb{R}^{N_x \times N_x}$. Thus $Y \in \mathbb{R}^{N_y N_x \times N_y N_x}$, with N_y diagonal blocks of size N_x each. The notation $Y = \text{tri}(A_{1:N_y}, B_{1:N_y-1})$ can be used to compactly represent such a block tridiagonal matrix. We now offer a new flexible method by which reluctance values can be approximated to produce a sparse block tridiagonal susceptance matrix, allowing for efficient and accurate simulation of power mesh structures.

2.1 Inductance Approximation Methods and the Formulation of the Susceptance Matrix

In [9, 13] the accuracy for simulation with interconnects was explored using *window* based techniques. In those works reluctive coupling was considered only to exist between neighbors in a given layer of parallel wires. Consider the 4×4 mesh shown in Figure 1, the inductor with terminal nodes (13, 14) could be assumed to interact only with the inductors between nodes (2,3) and (24,25) in the reluctance matrix. The capacitance matrix in this case is considered to be a diagonal matrix with non-zero entries only in the 16 node positions from which the capacitors branch. Therefore, for a $m \times m = 4 \times 4$ mesh structure, we can form a block tridiagonal matrix with $N_y = m = 4$ blocks of size $N_x = 3m - 2 = 10$. Thus, the block size N_x is the result of an even distribution of the nodes in the RLC mesh structure into *m* groups. This block decomposition



Figure 2: Accuracy for transient of m = 16 power mesh using windowing.

is illustrated in Figure 1, where all nodes enclosed together are considered to be part of the same block. The fact that the nearest neighbor windowing technique results in a block tridiagonal coefficient matrix can be justified by noticing that no coupling (i.e. resistive, capacitive, and reluctive) can bridge more than one block separation.

In order to first motivate the necessity of a more accurate method for inductance modeling over the traditional windowing based technique, we consider a slightly larger example. Figure 2 shows a comparison of the transient behavior for a 16×16 power mesh. The solid line corresponds to all mutual inductance terms being included in the analysis and the dotted line to the window based technique. In this case an exaggerated undershoot can be seen when the switching activity subsides and the voltage begins to return to V_{dd} . In addition, the window based approximation assumes that the voltage will return more quickly to that of the supply than is evident from the exact case.

In this work we consider the use of a sparse approximate inverse technique (SPAI) [14]. Although this technique was originally developed as a method to form a preconditioner for iterative linear solvers, the metric used by the authors in order to construct the preconditioner directly translates into improved accuracy in the case of power mesh simulation. Specifically, given an inductance matrix L the SPAI method can be used to form another matrix M that is constructed in an attempt to match the inverse of the inductance matrix under the Frobenius norm:

$$\|LM - I\|_F^2 = \sum_{i=1}^n \|(LM - I)e_i\|_2^2, \tag{4}$$

where *n* in the number of columns of *L* and e_i is the *i*th euclidean basis vector. Therefore, we can solve *n* independent least squares problems:

$$\min \|Lm_i - e_i\|_2^2, \quad i = 1, 2, \dots, n \tag{5}$$

in order to construct the columns m_i of the approximate inverse matrix M. Ref. [14]

	16×16		32×32		48×48			64×64			
	Matrix Size: 736		Matrix Size: 3008		Matrix Size: 6816			Matrix Size: 12160			
Data	NNZ	S %	RMSE	NNZ	S %	RMSE	NNZ	S %	RMSE	NNZ	S %
window	6256	98.9	6.75E-04	26320	99.7	1.30E-03	60208	99.9	2.29E-03	107920	99.9
$\tau = 0.94$	14040	97.4	3.48E-04	60280	99.3	5.83E-04	138776	99.7	1.14E-03	249528	99.8
$\tau = 0.95$	19612	96.4	3.31E-04	84956	99.1	4.47E-04	196124	99.6	1.10E-03	353116	99.8
$\tau = 0.96$	30032	94.5	2.76E-04	132736	98.5	3.92E-04	308400	99.3	5.26E-04	557024	99.6
$\tau = 0.97$	45564	91.6	2.42E-04	206156	97.7	3.76E-04	482460	99.0	1.74E-04	874476	99.4
$\tau = 0.98$	73114	86.5	2.01E-04	346558	96.2	1.36E-04	817778	98.2	2.29E-04	1499314	99.0
$\tau = 0.99$	119188	78.0	1.82E-04	649208	92.8	1.24E-04	1598196	96.6	1.84E-04	1758736	98.8

Table 1: Accuracy comparison for reluctance approximation methods against full inductance, mesh sizes m = 16, 32, 48, 64, with associated number of unkowns given as "Matrix Size".

offers several heuristics in an attempt to satisfy (5) by iteratively filling dominant entries of the inverse matrix M. In this work the threshold based approach is employed to create a matrix whose entries must be significant with respect to the absolute maximum in a column:

$$|M_{i,j}| > (1 - \tau) \max_{j} |M_{i,j}|, \tag{6}$$

where the diagonal entries $M_{i,i}$ are always included. If τ is close to zero, this criterion would prevent fill-in and result in a matrix that is very sparse. The value $\tau = 1$ would correspond to a matrix M where the entire pattern of L^{-1} will be considered.

Finally, we are interested in having the structure of the matrix K, defined in (2), be block tridiagonal in order to allow for efficient simulation through the use of our divide-and-conquer approach. This in turn would imply that the structure of the susceptance matrix: $S = A_l^T L^{-1}A_l$ be block tridiagonal as well (note that the window based technique referred to above will produce a sparse block diagonal approximation to the reluctance matrix, which also corresponds to a block tridiagonal structure for the matrix K). Therefore, we construct a framework where the sparsity of the reluctance matrix L^{-1} is governed via the matrix A_l to produce the final desired form for the susceptance matrix. Through the use of the SPAI procedure described above, assuming a fixed threshold value of τ , we can form an approximation for the reluctance matrix M which aims to satisfy (5). Next, a symmetric truncated version of this matrix is formed so that the susceptance matrix S will be block tridiagonal, given some assumed block size N_x .

Given this framework for approximation we can examine the accuracy of simulation considering increases to the drop tolerance parameter τ . Figure 3(a) demonstrates that using a tolerance: $\tau = 0.94$ for the SPAI approximation method we can already see the waveform more closely track the return to supply voltage with a less exaggerated undershoot. This level of approximation results in approximately a factor of $2.25 \times$ increase to the number of non-zero entries (refer to Table 1) of the coefficient matrix, where all inductive couplings considered where determined based upon the metric described in (5). The column labeled "NNZ" is the number of non-zero entries in the coefficient matrix and the column labeled "S %" is the percentage of the matrix whose entries are zero.



Figure 3: Accuracy for transient behavior of m = 16 power mesh using SPAI.

Figure 3(b) shows the same comparison for the tolerance: $\tau = 0.99$. It can be seen that for this power mesh simulation we are able to track every inflection of the waveform using strictly a block tridiagonal representation for the coefficient matrix. Finally, the root mean-squared error (RMSE) for the case $\tau = 0.99$, when compared to the waveform corresponding to the full inductance matrix being considered, was less than 27% of that using the window based technique. From Table 1 we can observe that for a larger 48 × 48 mesh the SPAI based inductance approximation procedure resulted in a reduction of more than 12× the RMSE achieved using the nearest neighbor windowing technique. The case m = 64 provided in the summary tables does not include an accuracy comparison due to memory limitations as a result of the greater than four thousand inductors associated with each direction of the mesh topology.

2.2 Inverses of Block Tridiagonal Matrices

Given the flexibility afforded by the SPAI based approach for constructing the susceptance matrix, we can now formulate the coefficient matrices needed for accurate RLC mesh simulation. The problem then turns to addressing the numerical challenges associated with using a direct (non-iterative) approach for the simulation of these structures. The inverse of a symmetric block tridiagonal matrix can be computed explicitly, as demonstrated in [15–17]. However, the mathematical representations presented in these works suffer numerical instability issues and are not directly applicable for large scale problems such as power mesh simulation. A numerically stable mathematical representation for producing the inverse of a symmetric block tridiagonal matrix has been demonstrated in [18]. Specifically, there exists two sequences of "ratio" matrices $\{R_i\}, \{S_i\}$ so that the inverse of a block tridiagonal matrix *K* can be written as:

$$K^{-1} = \begin{pmatrix} D_1 & D_1 S_1 & \cdots & D_1 \prod_{\substack{k=1 \ N_y - 1}} S_k \\ R_1 D_1 & D_2 & \cdots & D_2 \prod_{\substack{k=2 \ N_y - 1}} S_k \\ \vdots & \vdots & \ddots & \vdots \\ (\prod_{\substack{k=N_y - 1}}^1 R_k) D_1 & (\prod_{\substack{k=N_y - 1}}^2 R_k) D_2 & \cdots & D_{N_y} \end{pmatrix}.$$
 (7)

Here, the diagonal blocks of the inverse, D_i , and the ratio sequences satisfy the follow-

ing relationships:

$$R_{1} = A_{1}^{-1}B_{1},$$

$$R_{i} = (A_{i} - B_{i-1}^{T}R_{i-1})^{-1}B_{i}, \quad i = 2, ..., N_{y} - 1,$$

$$S_{N_{y}-1} = B_{N_{y}-1}A_{N_{y}}^{-1},$$

$$S_{i} = B_{i} (A_{i+1} - S_{i+1}B_{i+1}^{T})^{-1}, \quad i = N_{y} - 2, ..., 1,$$

$$D_{1} = (A_{1} - S_{1}B_{1}^{T})^{-1},$$

$$D_{i+1} = (A_{i+1} - S_{i+1}B_{i+1}^{T})^{-1} (I + B_{i}^{T}D_{i}S_{i}), \quad i = 1, ..., N_{y} - 2,$$

$$D_{N_{y}} = A_{N_{y}}^{-1} (I + B_{N_{y}-1}^{T}D_{N_{y}-1}S_{N_{y}-1}).$$
(8)

The time complexity associated with determining the parametrization of K^{-1} by the above approach is $O(N_x^3 N_y)$, with a memory requirement of $O(N_x^2 N_y)$.

2.2.1 Alternative Approach for Determining the Compact Representation

It is important to note that if the block tridiagonal portion of K^{-1} is known, the ratio sequences *R* and *S* can be extracted directly, i.e. without the use of entries from *K* through the ratio expressions (8). Examining closely the block tridiagonal portion of K^{-1} :

$$\operatorname{tri}(K^{-1}) = \begin{pmatrix} D_1 & D_1 S_1 & & \\ R_1 D_1 & D_2 & & \\ & \ddots & \ddots & \\ & & D_{N_y - 1} & D_{N_y - 1} S_{N_y - 1} \\ & & & R_{N_y - 1} D_{N_y - 1} & D_{N_y} \end{pmatrix}$$

we find the following relations:

$$D_i S_i = Z_i \implies S_i = D_i^{-1} Z_i, \quad i = 1, \dots, N_y - 1,$$

$$R_i D_i = Z_i^T \implies R_i = Z_i^T D_i^{-1}, \quad i = 1, \dots, N_y - 1,$$
(9)

where Z_i denotes the (i, i + 1) block entry of K^{-1} . Therefore, by being able to produce the block tridiagonal portion of K^{-1} we have all the information that is necessary to compute the compact representation.

As was alluded to in Section 1, direct techniques for simulation of realistic power mesh topologies often require prohibitive memory and computational requirements due to the large number of distributed RLC segments involved. To address these issues we offer a parallel divide-and-conquer approach in order to construct the compact representation for K^{-1} , i.e. the framework allows for the parallel inversion of the coefficient matrix for simulation. Specifically, we introduce an efficient method for computing the block tridiagonal portion of K^{-1} in order to exploit the process demonstrated above.

3 Parallel Inversion of Block Tridiagonal Matrices

The compact representation of K^{-1} can be computed in a distributed fashion by first creating several smaller sub-matrices ϕ_i . That is, the total number of blocks for the matrix K are divided as evenly as possible amongst the sub-matrices. After each individual sub-matrix inverse has been computed they can be combined in a Radix-2 fashion using the matrix inversion lemma from linear algebra. Figure 4 shows both the decomposition and the two combining levels needed to form the block tridiagonal portion of K^{-1} , assuming K has been divided into four sub-matrices. In general, if K is separated into p sub-matrices there will be $\log p$ combining levels with a total of p-1combining operations or "steps". The notation $\phi_{i \sim j}^{-1}$ is introduced to represent the result of any combining step, through the use of the matrix inversion lemma. For example, $\phi_{1,2}^{-1}$ is the inverse of a matrix comprised of the blocks assigned to both ϕ_1 and ϕ_2 . It is important to note that using the matrix inversion lemma repeatedly to join sub-matrix inverses will result in a prohibitive amount of memory and computation for large simulation problems. This is due to the fact that at each combining step all entries would be computed and stored. Thus, the question remains on the most efficient way to produce the block tridiagonal portion of K^{-1} , given this general decomposition scheme for the matrix K.

In this work, we introduce a mapping scheme to transform compact representations of smaller matrix inverses into the compact representation of K^{-1} . The algorithm is organized as follows:

- Decompose the block tridiagonal matrix *K* into *p* smaller block tridiagonal matrices.
- Assign each sub-matrix to an individual CPU.
- Independently determine the compact representations associated with each submatrix.
- Gather all information that is needed to map the sub-matrix compact representations into the compact representation for K^{-1} .
- Independently apply the mappings to produce a portion of the compact representation for K^{-1} on each CPU.

The procedure described above results in a "distributed compact representation" allowing for reduced memory and computational requirements. Specifically, each CPU will eventually be responsible for the elements from both the ratio sequences and diagonal blocks that correspond to the initial decomposition (e.g. if ϕ_1 is responsible for blocks 1,2, and 3 from the matrix *K*, the mappings will allow for the computation of $S_{1...3}$, $R_{1...3}$, and $D_{1...3}$).

In order to derive the mapping relationships needed to produce a distributed compact representation, it is first necessary to analyze how sub-matrix inverses can be combined to form the complete inverse. Consider the decomposition of the block tridiagonal matrix K into two block tridiagonal sub-matrices and a correction term, demonstrated below:



Figure 4: Decomposition of block tridiagonal matrix *K* into four sub-matrices, where the shaded blocks correspond to the bridge matrices. The two combining levels follow the individual sub-matrix inversions, where $\phi_{i\sim j}^{-1}$ represents the inverse of divisions ϕ_i through ϕ_j from the matrix *K*. Matrix mappings will be used to capture the combining effects and allow for the direct computation of the block tridiagonal portion of K^{-1} .

$$\begin{split} K &= \underbrace{\begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix}}_{\tilde{K}} + XY, \\ \phi_1 &= \operatorname{tri}(A_{1:k}, B_{1:k-1}), \quad \phi_2 &= \operatorname{tri}(A_{k+1:N_y}, B_{k+1:N_y-1}), \quad \text{and} \\ X &= \begin{pmatrix} 0 & \cdots & -B_k^T & 0 & \cdots & 0 \\ 0 & \cdots & 0 & -B_k & \cdots & 0 \\ 0 & \cdots & 0 & I & \cdots & 0 \\ 0 & \cdots & I & 0 & \cdots & 0 \end{pmatrix}^T, \\ Y &= \begin{pmatrix} 0 & \cdots & 0 & I & \cdots & 0 \\ 0 & \cdots & I & 0 & \cdots & 0 \end{pmatrix}. \end{split}$$

Thus, the original block tridiagonal matrix can be decomposed into the sum of a block diagonal matrix (with its two diagonal blocks themselves being block tridiagonal) and a correction term parameterized by the $N_x \times N_x$ matrix B_k , which we will refer to as the "bridge matrix". Using the matrix inversion lemma, we have

$$K^{-1} = (\tilde{K} + XY)^{-1} = \tilde{K}^{-1} - (\tilde{K}^{-1}X) \left(I + Y\tilde{K}^{-1}X\right)^{-1} (Y\tilde{K}^{-1}),$$

where

$$\begin{aligned}
\tilde{K}^{-1}X &= \begin{pmatrix} -\phi_1^{-1}(:,k)B_i & 0\\ 0 & -\phi_2^{-1}(:,1)B_i^T \end{pmatrix}, & (10) \\
(I+Y\tilde{K}^{-1}X)^{-1} &= \begin{pmatrix} I & -\phi_2^{-1}(1,1)B_i^T \\ -\phi_1^{-1}(k,k)B_i & I \end{pmatrix}^{-1}, \\
Y\tilde{K}^{-1} &= \begin{pmatrix} 0 & \phi_2^{-1}(:,1)^T \\ \phi_1^{-1}(:,k)^T & 0 \end{pmatrix},
\end{aligned}$$

and $\phi_1^{-1}(:,k)$ and $\phi_2^{-1}(:,1)$ denote respectively the last and first block columns of ϕ_1^{-1} and ϕ_2^{-1} .

This shows that the entries of \tilde{K}^{-1} are modified through the entries from the first rows and last columns of ϕ_1^{-1} and ϕ_2^{-1} , as well as the bridge matrix B_3 . Specifically, since ϕ_1 is before or "above" the bridge point we only need the last column of its inverse to reconstruct K^{-1} . Similarly, since ϕ_2 is after or "below" the bridge point we only need the first column of its inverse. These observations were noted in [19], where the authors demonstrated a parallel divide-and-conquer approach to determine the diagonal entries for the inverse of block tridiagonal matrices. In this work we build upon these ideas to create a scalable distributed framework for the transient simulation of mesh structures. We begin by generalizing the method from [19] in order to compute all information necessary to determine the distributed compact representation of K^{-1} (8). That is, we would like to create a combining methodology for sub-matrix inverses with two major goals in mind. First, it must allow for the calculation of all information that would be required to repeatedly join sub-matrix inverses, in order to mimic the combining process shown in Figure 4. Second, at the final stage of the combining process it must facilitate the computation of the block tridiagonal portion for the combined inverses.

It is important to note that the method in [19] has been developed specifically to determine the diagonal entries for a matrix with structure similar to that of K^{-1} . We can observe from Figure 4 that the diagonal entries of K^{-1} are restricted to the larger diagonal blocks (corresponding the size of each division). The method from [19] does not illustrate the process for reconstructing any information outside of these ranges. In addition, as the final goal for the application in [19] is the calculation of a small subset of the entries from K^{-1} , there is no discussion of the process for efficiently reconstructing a distributed compact representation and its use in the solution of block tridiagonal systems of linear equations. In this work we offer the following contributions:

- Considering a block tridiagonal coefficient matrix for simulation K, we offer a parallel matrix inversion algorithm. Specifically, we first show that a distributed mapping scheme can be used to determine the block tridiagonal entries from K^{-1} . Subsequently, these entries can be used to form a distributed compact representation of K^{-1} using the procedure demonstrated in (9).
- A computationally efficient parallel matrix-vector multiplication approach using the distributed compact representation of K^{-1} .

Our algorithm is a scalable alternative for the repeated solution of block tridiagonal systems of linear equations, with respect to the presence of additional reluctance coupling terms.

3.1 Matrix Maps

Matrix mappings are constructed in order to eventually produce the block tridiagonal portion of K^{-1} while avoiding any unnecessary computation during the combining process. Specifically, we will show that both the boundary block entries (first block row and last block column) and the block tridiagonal entries from any combined inverse $\phi_{i\sim i}^{-1}$ must be attainable (not necessarily computed) for all combining steps. We begin

by illustrating the initial stage of the combining process given four divisions, where for simplicity each will be assumed to have N_y blocks of size N_x . First, the two submatrices ϕ_1 and ϕ_2 are connected through the bridge matrix B_{N_y} and together they form the larger block tridiagonal matrix $\phi_{1\sim 2}$. By examining Figure 4 it can be seen that eventually $\phi_{1\sim 2}^{-1}$ and $\phi_{3\sim 4}^{-1}$ will be combined and we must therefore produce the boundaries for each combined inverse. From (10) the first block row and last block column of $\phi_{1\sim 2}^{-1}$ can be calculated through the use of an "adjustment" matrix:

$$J = \begin{pmatrix} I & -\phi_2^{-1}(1,1)B_{N_y}^T \\ -\phi_1^{-1}(N_y,N_y)B_{N_y} & I \end{pmatrix}^{-1},$$

as follows:

In addition, the *r*th diagonal block of $\phi_{1\sim 2}^{-1}$ can be calculated using the following relationships for $r \leq N_y$:

$$\begin{split} \phi_{1\sim2}^{-1}(r,r) &= \phi_{1}^{-1}(r,r) - \left(-\phi_{1}^{-1}(r,N_{y})B_{N_{y}}J_{12}\phi_{1}^{-1}(r,N_{y})^{T}\right), \\ \phi_{1\sim2}^{-1}(r+N_{y},r+N_{y}) &= \phi_{2}^{-1}(r,r) - \\ \left(-\phi_{2}^{-1}(r,1)B_{N_{y}}^{T}J_{21}\phi_{2}^{-1}(1,r)\right)^{T}, \end{split}$$
(12)

where the *r*th off-diagonal block of $\phi_{1\sim 2}^{-1}$ can be calculated using the following relationships:

$$\begin{split} \phi_{1\sim2}^{-1}(r,r+1) &= \phi_{1}^{-1}(r,r+1) - \\ \left(-\phi_{1}^{-1}(r,N_{y})B_{N_{y}}J_{12}\phi_{1}^{-1}(r+1,N_{y})^{T}\right), \end{split}$$
(13)
$$\phi_{1\sim2}^{-1}(r+N_{y},r+1+N_{y}) &= \phi_{2}^{-1}(r,r+1) - \\ \left(-\phi_{2}^{-1}(r+1,1)B_{N_{y}}^{T}J_{21}\phi_{2}^{-1}(1,r))\right)^{T}, \\ r &< N_{y}, \end{aligned}$$
(13)
$$\phi_{1\sim2}^{-1}(r,r+1) &= 0 - \left(-\phi_{1}^{-1}(r,N_{y})B_{N_{y}}J_{11}\phi_{2}^{-1}(1,1)\right), \\ r &= N_{y}. \end{split}$$

The combination of ϕ_3 and ϕ_4 through the bridge matrix B_{3N_y} results in similar relationships to those seen above. Thus, in order be able to produce both the boundary and block tridiagonal portions of each combined inverse we assign a total of twelve $N_x \times N_x$ matrix maps for each sub-matrix k. $M_{k;1-4}$ describe effects for the k^{th} portion of the boundary, $M_{k;5-8}$ describe the effects for a majority of the tridiagonal blocks, while $C_{k;1-4}$, which we will refer to as "cross" maps, can be used to produce the remainder of the tridiagonal blocks.

Initially, for each sub-matrix *i* the mappings $M_{k;i} = I$, k = 1, 4, with all remaining mapping terms set to zero. This ensures that initially the boundary of $\phi_{i\sim i}^{-1}$ matches the actual entries from the sub-matrix inverse, and the modifications to the tridiagonal portion due to combining are all set to zero. By examining the first block row, last block column, and the tridiagonal portion of the combined inverse $\phi_{1\sim 2}^{-1}$ we can see how the maps can be used to explicitly represent all of the needed information. The



Figure 5: Mapping dependencies when combining φ_1^{-1} and φ_2^{-1} to form $\varphi_{1\sim 2}^{-1}.$

governing responsibilities of the individual matrix maps are detailed below:

$$\Phi_{1\sim2}^{-1}(1,:) = \begin{pmatrix} \left[M_{1;1} \Phi_1^{-1}(1,:) + M_{2;1} \Phi_1^{-1}(:,N_y)^T \right]^T \\ \left[M_{1;2} \Phi_2^{-1}(1,:) + M_{2;2} \Phi_2^{-1}(:,N_y)^T \right]^T \end{pmatrix}^T, \\ \Phi_{1\sim2}^{-1}(:,2N_y) = \begin{pmatrix} \left[M_{3;1} \Phi_1^{-1}(1,:) + M_{4;1} \Phi_1^{-1}(:,N_y)^T \right]^T \\ \left[M_{3;2} \Phi_2^{-1}(1,:) + M_{4;2} \Phi_2^{-1}(:,N_y)^T \right]^T \end{pmatrix}, \end{cases}$$

$$\begin{split} \phi_{1\sim2}^{-1}(r,s) &= \phi_1^{-1}(r,s) - [\phi_1^{-1}(r,1)M_{5;1}\phi_1^{-1}(1,s) + \\ \phi_1^{-1}(r,1)M_{6;1}\phi_1^{-1}(s,N_y)^T + \phi_1^{-1}(r,N_y)M_{7;1}\phi_1^{-1}(1,s) + \\ \phi_1^{-1}(r,N_y)M_{8;1}\phi_1^{-1}(s,N_y)^T], \end{split}$$
(14)

$$\begin{split} \phi_{1\sim2}^{-1}(r,s+N_y) &= -[\phi_1^{-1}(r,1)C_{1;1}\phi_2^{-1}(1,s) + \\ \phi_1^{-1}(r,1)C_{2;1}\phi_2^{-1}(s,N_y)^T + \phi_1^{-1}(r,N_y)C_{3;1}\phi_2^{-1}(1,s) + \\ \phi_1^{-1}(r,N_y)C_{4;1}\phi_2^{-1}(s,N_y)^T], \end{split}$$

$$\begin{split} & \phi_{1\sim2}^{-1}(r+N_y,s+N_y) = \phi_2^{-1}(r,s) - [\phi_2^{-1}(r,1)M_{5;2}\phi_2^{-1}(1,s) + \\ & \phi_2^{-1}(r,1)M_{6;2}\phi_2^{-1}(s,N_y)^T + \phi_2^{-1}(r,N_y)M_{7;2}\phi_2^{-1}(1,s) + \\ & \phi_2^{-1}(r,N_y)M_{8;2}\phi_2^{-1}(s,N_y)^T], \end{split}$$

$$r, s \leq N_{y},$$

It is important to note that all of the expressions (11)-(13) can be written into the matrix

map framework of (14). Figure 5 shows the mapping dependencies for the first block row and last block row (or column since *K* is symmetric). From (11) we see that both of the block rows are distributed based upon the location of each sub-matrix with respect to the bridge point, i.e. the mapping terms associated with ϕ_1^{-1} can be used to produce the first portion of the rows while those associated with ϕ_2^{-1} can be used for the remainder. In fact, this implicit division for the mapping dependencies holds for the block tridiagonal portion of the combined inverses as well, enabling an efficient parallel implementation. Thus, from this point we can deduce that the matrix maps for the first block row (14) must be updated in the following manner:

$$M_{1;1} \leftarrow M_{1;1} + (\phi_1^{-1}(1, N_y) B_{N_y} J_{12}) M_{3;1};$$

$$M_{2;1} \leftarrow M_{2;1} + (\phi_1^{-1}(1, N_y) B_{N_y} J_{12}) M_{4;1};$$

$$M_{3;1} \leftarrow (\phi_1^{-1}(1, N_y) B_{N_y} J_{11}) M_{1;2};$$

$$M_{4;1} \leftarrow (\phi_1^{-1}(1, N_y) B_{N_y} J_{11}) M_{2;2};$$

In order to understand these relationships it is important to first recall that the updates to the maps associated with sub-matrix ϕ_1 are dependent on the last block column $\phi_1^{-1}(:,N_y)$. Thus, we see a dependence on the previous state for the last block column $\phi_1^{-1}(:, N_y)$, i.e. the new state of the mapping terms $M_{1;1}$ and $M_{2;1}$ are dependent on the previous state of the mapping terms $M_{3;1}$ and $M_{4;1}$ respectively. Similarly, a dependence on $\phi_2^{-1}(1,:)$ results in the new state of the mapping terms $M_{1;2}$ and $M_{2;2}$ being functions of the previous state of the mapping terms $M_{1,2}$ and $M_{1,2}$ respectively. Finally, although some of the mapping terms remain zero after this initial combining step $(M_{2:2}$ for example), the expressions described in (14) need to be general enough for the methodology. That is, the mapping expressions must be able to capture combining effects for multiple combing stages, regardless of the position of the sub-matrix with respect to a bridge point. For example, if we consider sub-matrix ϕ_2 for the case seen in Figure 4, during the initial combining step it would be considered a lower problem and for the final combining step it would be considered a upper problem. Alternatively, sub-matrix ϕ_3 would be associated with exactly the opposite modifications. It is important to note that every possible modification process, for the individual mapping terms, is encompassed within this general matrix map framework.

3.2 Recursive Combining Process

In order to formalize the notion of a recursive update scheme we will continue the example from Section 3.1. By examining the final combing stage for the case of four divisions, we notice that the approach described in (11)-(13) can again be used to combine sub-matrix inverses $\phi_{1\sim2}^{-1}$ and $\phi_{3\sim4}^{-1}$, through the bridge matrix B_{2N_y} . The first block

row and last block column of $\varphi_{1\sim 4}^{-1}$ can be calculated as follows:

$$\phi_{1\sim4}^{-1}(1,:) = \left(\phi_{1\sim2}^{-1}(1,:) \quad 0\right) - \left(\begin{bmatrix} -\phi_{1\sim2}^{-1}(1,2N_{y})B_{2N_{y}}J_{12}\phi_{1\sim2}^{-1}(:,2N_{y})^{T} \end{bmatrix}^{T} \\ \begin{bmatrix} -\phi_{1\sim2}^{-1}(1,2N_{y})B_{2N_{y}}J_{11}\phi_{3\sim4}^{-1}(1,:) \end{bmatrix}^{T} \end{bmatrix}^{T} ,$$

$$\phi_{1\sim4}^{-1}(:,4N_{y}) = \left(0 \quad \phi_{3\sim4}^{-1}(2N_{y},:) \right)^{T} - \left(\begin{bmatrix} -\phi_{3\sim4}^{-1}(2N_{y},1)B_{2N_{y}}^{T}J_{22}\phi_{1\sim2}^{-1}(:,2N_{y})^{T} \end{bmatrix}^{T} \\ \begin{bmatrix} -\phi_{3\sim4}^{-1}(2N_{y},1)B_{2N_{y}}^{T}J_{21}\phi_{3\sim4}^{-1}(1,:) \end{bmatrix}^{T} \end{bmatrix} ,$$

$$(15)$$

given the adjustment matrix:

$$J = \begin{pmatrix} I & -\phi_{3\sim4}^{-1}(1,1)B_{2N_y}^T \\ -\phi_{1\sim2}^{-1}(2N_y,2N_y)B_{2N_y} & I \end{pmatrix}^{-1}$$

In addition, the r^{th} diagonal block of $\phi_{1\sim4}^{-1}$ can be calculated using the following relationships:

$$\begin{split} \phi_{1\sim4}^{-1}(r,r) &= \phi_{1\sim2}^{-1}(r,r) - \\ \left(-\phi_{1\sim2}^{-1}(r,2N_y) B_{2N_y} J_{12} \phi_{1\sim2}^{-1}(r,2N_y)^T \right), \end{split}$$
(16)
$$\phi_{1\sim4}^{-1}(r+2N_y,r+2N_y) &= \phi_{3\sim4}^{-1}(r,r) - \\ \left(-\phi_{3\sim4}^{-1}(r,1) B_{2N_y}^T J_{21} \phi_{3\sim4}^{-1}(1,r) \right) \Big)^T, \\ r &\leq 2N_y, \end{split}$$

where the *r*th off-diagonal block of $\phi_{1\sim4}^{-1}$ can be calculated using the following relationships:

$$\begin{split} \phi_{1\sim4}^{-1}(r,r+1) &= \phi_{1\sim2}^{-1}(r,r+1) - \\ \left(-\phi_{1\sim2}^{-1}(r,2N_{y})B_{2N_{y}}J_{12}\phi_{1\sim2}^{-1}(r+1,2N_{y})^{T} \right), \\ \phi_{1\sim4}^{-1}(r+2N_{y},r+1+2N_{y}) &= \phi_{3\sim4}^{-1}(r,r+1) - \\ \left(-\phi_{3\sim4}^{-1}(r+1,1)B_{2N_{y}}^{T}J_{21}\phi_{3\sim4}^{-1}(1,r) \right) \right)^{T}, \\ r < 2N_{y}, \end{split}$$

$$\end{split}$$
(17)

$$\begin{split} \phi_{1\sim4}^{-1}(r,r+1) &= 0 - \\ \left(-\phi_{1\sim2}^{-1}(r,2N_y)B_{2N_y}J_{11}\phi_{3\sim4}^{-1}(1,1) \right), \\ r &= 2N_y. \end{split}$$

Again, it is important to note that each of the expressions (15)-(17) are implicitly divided based upon topology. For example, the first $2N_y$ diagonal blocks of $\phi_{1\sim4}^{-1} = K^{-1}$ can be separated into two groups based upon the size of the sub-matrices ϕ_1 and ϕ_2 . That is,

$$\begin{split} & \phi_{1\sim4}^{-1}(r,r) = \phi_{1\sim2}^{-1}(r,r) - \\ & \left(-\phi_{1\sim2}^{-1}(r,2N_y) B_{2N_y} J_{12} \phi_{1\sim2}^{-1}(r,2N_y)^T \right), \\ & r \leq 2N_y, \end{split}$$

can be separated for $r \leq N_y$ as:

$$\begin{split} \phi_{1 \sim 4}^{-1}(r,r) &= \phi_{1}^{-1}(r,r) - \\ \left(\left[\phi_{2}^{-1}(N_{y},1) B_{N_{y}}^{T} J_{22} \phi_{1}^{-1}(r,N_{y})^{T} \right]^{T} \right) \cdot B_{2N_{y}} J_{12} \cdot \\ \left(\left[\phi_{2}^{-1}(N_{y},1) B_{N_{y}}^{T} J_{22} \phi_{1}^{-1}(r,N_{y})^{T} \right] \right), \end{split}$$

$$\begin{split} \phi_{1\sim4}^{-1}(r+N_y,r+N_y) &= \phi_2^{-1}(r,r) - \\ \left(\left[\phi_2^{-1}(N_y,r) + \phi_2^{-1}(N_y,1) B_{N_y}^T J_{21} \phi_2^{-1}(1,r) \right]^T \right) \cdot B_{2N_y} J_{12} \cdot \\ \left(\left[\phi_2^{-1}(N_y,r) + \phi_2^{-1}(N_y,1) B_{N_y}^T J_{21} \phi_2^{-1}(1,r) \right] \right). \end{split}$$

Thus, the modifications to the diagonal entries can be written as just a function of the first block row and last block column from the individual sub-matrices, using the matrix map framework introduced in (14) for $r \le N_y$:

$$\begin{split} & \phi_{1\sim4}^{-1}(r,r) = \phi_{1}^{-1}(r,r) - \\ & \left(\left[M_{1;1}\phi_{1}^{-1}(1,r) + M_{2;1}\phi_{1}^{-1}(r,N_{y})^{T} \right]^{T} \right) \cdot B_{2N_{y}}J_{12} \cdot \\ & \left(\left[M_{1;1}\phi_{1}^{-1}(1,r) + M_{2;1}\phi_{1}^{-1}(r,N_{y})^{T} \right] \right) , \\ & \phi_{1\sim4}^{-1}(r+N_{y},r+N_{y}) = \phi_{2}^{-1}(r,r) - \\ & \left(\left[M_{1;2}\phi_{2}^{-1}(1,r) + M_{2;2}\phi_{2}^{-1}(r,N_{y}) \right]^{T} \right) \cdot B_{2N_{y}}J_{12} \cdot \\ & \left(\left[M_{1;2}\phi_{2}^{-1}(1,r) + M_{2;2}\phi_{2}^{-1}(r,N_{y}) \right] \right) . \end{split}$$

Here, the matrix maps are assumed to have been updated based upon the formation of the combined inverses $\phi_{1\sim 2}^{-1}$ and $\phi_{3\sim 4}^{-1}$. Therefore, we can begin to formulate the recursive framework for updating the matrix maps to represent the effect of each combining step.

3.3 Update Scheme for Parallel Inversion and the Distributed Compact Representation

The procedure begins with each division of the problem being assigned to one of p available CPUs. In addition, all of the p-1 bridge matrices are made available to

each of the CPUs. After the compact representation for each inverse has been found independently, the combining process begins. Three reference positions are defined for the formation of a combined inverse $\phi_{i\sim j}^{-1}$: the "start" position [st] = i, the "stop" position [sp] = j, and the bridge position $[bp] = \lceil \frac{j-i}{2} \rceil$. Due to the fact that a CPU t will only be involved in the formulation of a combined inverse when $[st] \le t \le [sp]$ all combining stages on the same level (see Figure 4) can be performed concurrently. When forming a combined inverse $\phi_{i\sim j}^{-1}$, each CPU $[st] \le t \le [sp]$ will first need to form the adjustment matrix for the combining step. Assuming a bridge matrix B_k , we begin by constructing four "corner blocks". If the upper combined inverse is assumed to have N_u blocks and the lower to have N_l , the two matrices need from the upper combined inverse are: $[\text{UR}] = \phi_{[st]\sim[bp]}^{-1}(1,N_u)$ and $[\text{LR}] = \phi_{[st]\sim[bp]}^{-1}(N_u,N_u)$, with the two matrices from the lower being: $[\text{UL}] = \phi_{[bp+1]\sim[sp]}^{-1}(1,1)$ and $[\text{LL}] = \phi_{[bp+1]\sim[sp]}^{-1}(N_l,1)$. These matrices can be generated by the appropriate CPU through their respective matrix maps (recall the example shown in Figure 5). Specifically, the CPUs corresponding to the [st], [bp], [bp+1] and [sp] divisions govern the required information. The adjustment matrix for the combining step can then be formed:

$$J = \begin{pmatrix} I & -[\mathrm{UL}]B_k^T \\ -[\mathrm{LR}]B_k & I \end{pmatrix}^{-1}.$$

After the adjustment matrix has been calculated the process of updating the matrix maps can begin. For any combining step, the cross maps each CPU t must be updated first:

if
$$(t < [bp])$$
 then
 $C_1 \leftarrow C_1 - M_{3;t}^T (B_k J_{12}) M_{3;t+1};$
 $C_2 \leftarrow C_2 - M_{3;t}^T (B_k J_{12}) M_{4;t+1};$
 $C_3 \leftarrow C_3 - M_{4;t}^T (B_k J_{12}) M_{3;t+1};$
 $C_4 \leftarrow C_4 - M_{4;t}^T (B_k J_{12}) M_{4;t+1};$
elseif $(t == [bp])$ then (18)
 $C_1 \leftarrow C_1 - M_{3;t}^T (B_k J_{11}) M_{1;t+1};$
 $C_2 \leftarrow C_2 - M_{3;t}^T (B_k J_{11}) M_{2;t+1};$
 $C_3 \leftarrow C_3 - M_{4;t}^T (B_k J_{11}) M_{2;t+1};$
elseif $(t < [sp])$ then
 $C_1 \leftarrow C_1 - M_{1;t}^T (B_k^T J_{21}) M_{1;t+1};$
 $C_2 \leftarrow C_2 - M_{1;t}^T (B_k^T J_{21}) M_{2;t+1};$
 $C_3 \leftarrow C_3 - M_{2;t}^T (B_k^T J_{21}) M_{2;t+1};$
 $C_4 \leftarrow C_4 - M_{2;t}^T (B_k^T J_{21}) M_{1;t+1};$
 $C_4 \leftarrow C_4 - M_{2;t}^T (B_k^T J_{21}) M_{2;t+1};$

Notice that the cross maps for CPU *t* are dependent on information from its neighboring CPU t + 1. This information must be transmitted and made available before the cross updates can be performed. Next, updates to the remaining eight matrix maps can be separated into two categories. The updates to the matrix maps for the upper sub-matrices ($t \leq [bp]$), are summarized below:

$$M_{5;t} \leftarrow M_{5;t} - M_{3;t}^{T} (B_{k}J_{12})M_{3;t};$$

$$M_{6;t} \leftarrow M_{6;t} - M_{3;t}^{T} (B_{k}J_{12})M_{4;t};$$

$$M_{7;t} \leftarrow M_{7;t} - M_{4;t}^{T} (B_{k}J_{12})M_{3;t};$$

$$M_{8;t} \leftarrow M_{8;t} - M_{4;t}^{T} (B_{k}J_{12})M_{4;t};$$

$$M_{1;t} \leftarrow M_{1;t} + ([\text{UR}]B_{k}J_{12})M_{3;t};$$

$$M_{2;t} \leftarrow M_{2;t} + ([\text{UR}]B_{k}J_{12})M_{4;t};$$

$$M_{3;t} \leftarrow ([\text{LL}]^{T}B_{k}^{T}J_{22})M_{3;t};$$

$$M_{4;t} \leftarrow ([\text{LL}]^{T}B_{k}^{T}J_{22})M_{4;t};$$
(19)

The updates to the matrix maps for the lower sub-matrices (t > [bp]), will be:

$$M_{5;t} \leftarrow M_{5;t} - M_{1;t}^{T} (B_{k}^{T} J_{21}) M_{1;t};$$

$$M_{6;t} \leftarrow M_{6;t} - M_{1;t}^{T} (B_{k}^{T} J_{21}) M_{2;t};$$

$$M_{7;t} \leftarrow M_{7;t} - M_{2;t}^{T} (B_{k}^{T} J_{21}) M_{1;t};$$

$$M_{8;t} \leftarrow M_{8;t} - M_{2;t}^{T} (B_{k}^{T} J_{21}) M_{2;t};$$

$$M_{3;t} \leftarrow M_{3;t} + ([LL]^{T} B_{k}^{T} J_{21}) M_{1;t};$$

$$M_{4;t} \leftarrow M_{4;t} + ([LL]^{T} B_{k}^{T} J_{21}) M_{2;t};$$

$$M_{1;t} \leftarrow ([UR] B_{k} J_{11}) M_{1;t};$$

$$M_{2;t} \leftarrow ([UR] B_{k} J_{11}) M_{2;t};$$
(20)

The above procedure, shown in (18)-(20), for modifying the matrix maps can be recursively repeated for each of the combining stages beginning with the lowest level of combining the individual sub-matrix inverses. On completion the maps can then be used to generate the block tridiagonal entries of K^{-1} .

Under this framework, matrix maps can be used to determine both the diagonal and off-diagonal block entries for K^{-1} . This subsequently allows for the computation of the ratio sequences for K^{-1} , via the relationships shown in (9), in a purely distributed fashion. This distribution of the compact representation is at the foundation of an efficient parallel method for solving systems of linear equations Kx = c.

The time complexity of the algorithm presented is $O(\frac{N_x^3 N_y}{p} + N_x^3 \log p)$, with memory consumption $O\left(\frac{N_x^2 N_y}{p} + N_x^2\right)$. The first term $\left(\frac{N_x^3 N_y}{p}\right)$ in the computational complexity arises from the embarrassingly parallel nature of both determining the ratio sequences and applying the matrix maps to update the block tridiagonal portion of the inverse. The second term $(N_x^3 \log p)$ is dependent on the number of levels needed to gather combining information for p sub-matrix inverses. Similarly, the first term in the

memory complexity is due to the ratio sequences and diagonal blocks, and the second represents the memory required for the matrix maps of each sub-problem governed.

4 Parallel Solution of Block Tridiagonal Systems

The parallel inversion algorithm described above not only has advantages in computational and memory efficiency but also facilitates the formulation of a fast, and highly scalable, parallel multiplication algorithm. This plays an essential role during the simulation process due to the fact that transient simulation of power mesh structures involves the solution of a linear system $Kx_i = c_i$ at each time step *i*, which in this case translates into performing the operation $x_i = K^{-1}c_i$. Due to the fact that for transient simulation this multiplication operation is repeated many times we would like to precompute as much information as possible, i.e. reduce the amount of computation and parallel communication required to perform each multiplication during simulation.

4.1 Parallel Matrix-Vector Multiply

Recall that our initial state for this procedure would assume that portions of the ratio matrices (corresponding to the size and location of the division from within the complete problem that was assigned to the CPU) have been calculated and stored. Subsequently, we can formulate the matrix-vector product of K^{-1} and c. In order to simplify notation for the presentation of the parallel matrix-vector multiply, i.e. $K^{-1}c$, we will use two subscripts for each of the sequences involved in the calculation. The first subscript will be used to denote the CPU and the second will refer to a block within the sequence, present on that particular CPU. We are interested in solving Kx = c, where two sequences $\{W_{k,l}\}$ and $\{T_{k,l}\}$ must be defined in order to compute the solution for the system of equations. Specifically, *k* refers to a CPU with p_k total blocks indexed by *l*:

$$W_{k} = \begin{bmatrix} W_{k,1}^{T} \\ W_{k,2}^{T} \\ W_{k,3}^{T} \\ \vdots \\ W_{k,p_{k}}^{T} \end{bmatrix}, T_{k} = \begin{bmatrix} T_{k,1}^{T} \\ T_{k,2}^{T} \\ T_{k,3}^{T} \\ \vdots \\ T_{k,p_{k}}^{T} \end{bmatrix}, W_{k,l}, T_{k,l} \in \mathbb{R}^{N_{x} \times 1}.$$

The elements from each sequence can be computed through the following recursions:

$$W_{k,p_k} = D_{k,p_k} c_{k,p_k}, W_{k,l} = D_{k,l} c_{k,l} + R_{k,l} W_{k,l+1}, \quad l = p_k - 1, \dots, 1, T_{k,1} = S_{k,1}^T D_{k,1} c_{k,1}, T_{k,l} = S_{k,l}^T (T_{k,l-1} + D_{k,l} c_{k,l}), \quad l = 2, \dots, p_k - 1,$$
(21)

where $D_{k,l}$ is diagonal block *l* of the inverse for CPU *k*, and the terms $R_{k,l}$ and $S_{k,l}$ are the corresponding elements of the ratio sequences (8). The resulting vector *x* can then be determined using *T* and *W* and the product matrices:

$$\tilde{R}_{q,k} = \begin{cases} \prod_{m=q}^{(k+1)} \prod_{n=p_m}^{1} R_{m,n} & \text{if } q > k, \\ I & \text{if } q = k, \\ 0 & \text{otherwise} \end{cases}$$
$$\tilde{S}_{q,k} = \begin{cases} \prod_{m=q}^{(k-1)} p_m \\ \prod_{m=q}^{n} \prod_{n=1}^{1} S_{m,n} & \text{if } q < k, \\ I & \text{if } q = k, \\ 0 & \text{otherwise} \end{cases}$$
$$R_{k,l} = \prod_{n=p_k}^{l} R_{k,n}, \ S_{k,l} = \prod_{n=1}^{l} S_{k,n}.$$

Here, the ratio terms that are present on a given CPU are denoted as local, and the accumulations of the ratios from the remaining CPUs are denoted as skip. Thus, the following expression for the vector x can be constructed:

$$x_{k,l} = \sum_{q=k}^{p} R_{k,l} \tilde{R}_{q-1,k} W_{q,1}$$

+ $\sum_{q=1}^{k} S_{k,l} \tilde{S}_{q+1,k} T_{q,pq} + W_{k,l} + T_{k,l-1},$ (22)
 $\forall k = 1, \dots, p \text{ and } l = 1, \dots, p_k,$

where:

$$T_{k,-1} = \begin{cases} 0 & \text{if } k = 1, \\ T_{k-1,p_{k-1}} & \text{if } k > 1, \end{cases}$$

Although this multiplication procedure seems to be of a strictly recursive nature (and hence not readily parallel) it will be shown that the general formulation (22) can in fact be computed efficiently in a distributed fashion.

Initially, each skip product term for the CPUs, i.e. $\prod_{n=p_m}^{1} R_{m,n}$ and $\prod_{n=1}^{p_m} S_{m,n}$ within the expressions for $\tilde{R}_{q,k}$ and $\tilde{S}_{q,k}$ respectively, are pre-computed after the inversion process. In addition, during the setup time between the inversion algorithm and time step calculations, the necessary skip product terms for the above procedure are distributed amongst the appropriate CPUs. The goal of this multiplication procedure is to evenly distribute the computation across the CPUs while minimizing both the parallel communication and the amount of redundant operations.

For example, if we examine a subset of the operations for the case of p = 4 divisions, we can see how this general parallel computational scheme can be constructed. The portion of the solution from (22) present on CPU k = 1 has the form:

$$\begin{split} x_{1,l} &= \sum_{q=1}^{4} R_{1,l} \tilde{R}_{q-1,1} W_{q,1} \\ &+ \sum_{q=1}^{1} S_{1,l} \tilde{S}_{q+1,1} T_{q,p_q} + W_{1,l} + T_{1,l-1}, \\ &= \sum_{q=1}^{4} R_{1,l} \tilde{R}_{q-1,1} W_{q,1} + W_{1,l} + T_{1,l-1}, \\ &= R_{1,l} \tilde{R}_{1,1} W_{2,1} + R_{1,l} \tilde{R}_{2,1} W_{3,1} \\ &+ R_{1,l} \tilde{R}_{3,1} W_{4,1} + W_{1,l} + T_{1,l-1} \\ &= (W_{1,l} + T_{1,l-1}) + R_{1,l} W_{2,1} \\ &+ R_{1,l} \left(\tilde{R}_{2,1} \left(W_{2,1} + \tilde{R}_{3,2} W_{4,1} \right) \right). \end{split}$$

We can see the sum being separated into two types of operations: those that can be computed independently for CPU k, namely $(W_{k,l} + T_{k,l-1})$, and the remainder of which require information from other CPUs. In addition, it is important to note that the information coming from the other CPUs is essentially a single vector such as $W_{q,1}$ which is cascaded through a sequence of ratio terms. The vectors must finally arrive at the given CPU in order to be multiplied by its governed ratios $R_{k,l}$. Finally, we can clearly see from the example above the advantages of having skip product terms such as $\tilde{R}_{2,1}$ stored on each CPU. This allows for the effects of multiplying through all of the ratios R from CPU 2 to be replaced by a single skip term. Also, this single matrix-vector multiply can be performed by CPU 1, thus reducing the number of stages required for the entire process to be completed.

The computation associated with calculating the sequences T_k and W_k has complexity $O(p_k N_x^2)$, with that of the $\log_2 p$ product stages each being of order $O(p_k N_x^2)$. Therefore, if the problem is distributed evenly across the *p* CPUs ($p_k = \frac{N_y}{p}$, $\forall k$), the total complexity of the process is $O(\frac{N_x^2 N_y}{p} + \frac{\log p}{p} N_x^2 N_y)$. Finally, in the case of a single CPU it should be noted that *x* is generated in its entirety from the recursions of *W* and *T* (21), where p = 1 and $p_1 = N_y$.

5 Power Mesh Simulation

There are two main categories of algorithms for solving sparse linear systems of equations: direct and iterative. In this section we will first demonstrate the advantages in scalability of direct methods for the *accurate* transient simulation of mesh structures. Here, the transient simulation time using both direct and iterative methods are compared for varying levels of reluctive coupling. Finally, the ability to trade-off computing resources for both increased mesh sizes and transient simulation time using the parallel inversion process will be highlighted. Given the need for objective comparisons against current and future algorithms, we investigate the performance of standard direct and iterative methods. Specifically, our parallel direct approach is compared with two commonly available sparse linear solvers and the conjugate gradient (CG) method using the universally accepted incomplete LU (ILU) preconditioner. These algorithms all have well documented computational complexities, memory requirements, and speed of converge in the case of CG.

In order to perform meaningful simulations of the transient behavior seen on a power mesh there are several factors that need to be considered. First, the locations of the power pads on the mesh will vary with the packaging that is used for the design. Next, switching activity on realistic devices can be observed across the entire chip area. Finally, based upon the circuit equation formulation employed and the actual numerical method of solution, the accuracy for the simulation may be affected. The simulations considered in this work are for square power meshes of dimension $m \times$ m. The sizes of the variables N_x and N_y for the block tridiagonal representation of the coefficient matrix will depend on the amount of inductive coupling considered, as discussed in Section 2.1. For example, a windowing based technique would result in a coefficient matrix with $N_y = m$ blocks of size $N_x = 3m - 2$ (based upon the fact that a distributed RLC model is used for the interconnects, refer to Figure 1). For all SPAI based analyses we constrain the additional inductive coupling considered to be within a block size $N_x = 2(3m-2)$ with $N_y = \frac{m}{2}$. All inductance values used as inputs for the windowing and SPAI approximation procedures were generated with the FastHenry extraction tool [20]. We consider V_{dd} pins to be placed at $(2 \cdot (\frac{N_v}{2} - 1))^2$ equally spaced positions throughout the grid, where the minimum mesh size analyzed is m = 16. A random subset from the remaining nodes are considered current sinks with a square waveform triggered by a rising clock edge. All simulations consisted of 1500 time steps, given a step size of 0.1ps and clock signal of 50ps. All interconnects were assumed to be of uniform size: $1\mu m \times 2\mu m \times 100\mu m$.

5.1 Simulation Methods

For the transient simulation of power mesh structures we are considering the solution of the system $Kx_i = c_i$ at each time step *i*, where the coefficient matrix *K* does not change with time. Regardless of the method of solution it can be of great benefit to consider some amount of pre-processing on the matrix *K* in order to accelerate the solution at each time step. In this work we limit the comparison of our algorithm to several of the best known and most widely available software packages in order to provide standard comparisons for future analysis of algorithms in this area. The conjugate gradient method is an iterative search method which is known to perform extremely well, especially for matrices that are symmetric, positive-definite, and sparse [10]. In addition, the ILU factorization or ILU preconditioner is almost universally accepted as a pre-processing technique that can be used in conjunction with CG to improve solve times for matrices of this type. A good choice of a preconditioner *M* for a matrix *A* would be determined by two factors:

- How closely does the matrix M approximate A^{-1} ? We would like $AM \approx I$.
- How easy is it to solve with the matrix *M*? We would like to solve My = z fast.

Mesh Size	128×128	192×192	256×256	384×384	512×512
Matrix Size	4.89E+04	1.10E+05	1.96E+05	4.42E+05	7.85E+05
NNZ					
Window	4.35E+05	9.81E+05	1.75E+06	3.93E+06	6.99E+06
$\tau = 0.95$	1.43E+06	3.22E+06	5.73E+06	1.29E+07	2.29E+07
$\tau = 0.97$	3.54E+06	7.99E+06	1.42E+07	3.21E+07	5.70E+07

Table 2: Scaling trend for reluctance approximations with respect to the number of unknowns or "Matrix Size" and the number of non-zeros or "NNZ", given a mesh of size $m \times m$.

By specifying a drop tolerance the ILU method computes a sparse LU approximation that meets both of these criteria and can be used to efficiently guide the search process. Note that all simulations performed using the CG method in this work have a fixed stopping criterion tolerance of 10^{-6} . It should be clear that as the drop tolerance decreases more terms will be included in the preconditioner and the number of iterations needed for the method to converge will potentially decrease. However, in this case the ILU preconditioner will be less sparse and across all drop tolerances with the same number of iterations needed for convergence, the longest solve time will correspond to the smallest drop tolerance.

In addition to the CG method, we also provide analysis of the performance of the UMFPACK and the MATLAB Sparse LU direct linear solvers when used for power mesh simulation. UMFPACK [11] is a multi-frontal method that uses minimum fill-in orderings to efficiently perform an elimination procedure. The factorization involves four matrices: a permutation matrix for stability, a reduced fill-in ordering matrix for efficiency, and the LU factors. These terms can be stored and subsequently only back solves are needed to compute the solution at each time step. A similar pre-processing factorization and evaluation procedure can be employed for the MATLAB Sparse LU solver.

The parallel divide-and-conquer approach detailed in Sections 3 and 4 fits the same mold as the UMFPACK and sparse LU algorithms described above. However, instead of a factorization being stored for future use, the distributed compact representation for the inverse of the coefficient matrix K^{-1} is stored across several computers. In addition, instead of performing a back solve for each time step *i*, as is the case with the elimination style algorithms, our algorithm involves a parallel matrix-vector multiply $K^{-1}c_i$. Our algorithm has been implemented, in C, and compared against pre-compiled UMF-PACK and MATLAB Sparse LU included as part of the MATLAB release (R14SP3). It is important to note that both the ILU factorization and any back solve needed as part of the preconditioning process for CG are pre-compiled subroutines called from within MATLAB as well. These computational issues are necessary to ensure no bias for our algorithm over other methods. All simulations were performed on a cluster of 32-bit 3Ghz Intel Xeon workstations with 2GB of memory for each node.

Data	Our Algorithm	LU	UMF
16×16 $\tau = 0.99$	2.01E-03	2.78E-03	2.61E-03
$32 \times 32 \tau = 0.99$	1.45E-02	2.25E-02	1.90E-02
48×48 $\tau = 0.99$	4.43E-02	6.21E-02	6.03E-02
64×64 $\tau = 0.99$	8.73E-02	1.41E-01	1.48E-01
$128 \times 128 \ \tau = 0.97$	7.29E-01	1.25E+00	7.71E-01
$192 \times 192 \ \tau = 0.90$	1.50E+00	1.67E+00	-
256×256 window	2.25E+00	-	-
384×384 window	6.57E+00	-	-
512×512 window	1.08E+01	-	-

Table 3: Performance of direct algorithms across mesh size. Transient step solve times are shown in seconds; the lack of memory scalability for existing direct approaches is shown through an inability to perform the larger simulations.

5.2 Simulation Time

In order to analyze the improvement in computational efficiency achieved by the divideand-conquer method it is first necessary to understand the breakdown of total simulation time. The total simulation time, given any level of inductance approximation, is dominated by the fixed time cost of inversion or factorization plus the variable time cost to multiply or solve at each time step. When considering transient simulations involving a large number of time steps, any speed-up seen in the variable time cost will dominate the fixed time cost. For example, the LU algorithm used to construct the waveform in Figure 2 has factorization time: 20ms and solve time: 1.54ms. Therefore, after thirteen time steps the transient time would be larger than the factorization time. In the scope of this analysis, we focus on transient simulations involving several time steps and compare solve times for various examples of interest.

5.2.1 Comparison against direct solvers

In order to gain perspective for the computational limitations for each of the direct algorithms considered in this work, several large-scale simulations m = 128, 192, 256, 384, and 512were performed. Table 2 shows the size of coefficient matrix and the number of nonzero entries, considering different amounts of reluctance coupling. Table 3 shows the transient step solve times of the direct algorithms for these mesh sizes. It was determined that the UMFPACK algorithm was only able to handle up to a mesh size of m = 192 with $\tau = 0.85$. This case corresponded to a coefficient matrix with approximately 110K unknowns, but less than 412K non-zero entries. The memory consumption of the Sparse LU algorithm scaled slightly better, being able to perform the simulation for a mesh size of m = 192 with $\tau = 0.90$ which corresponds to over $2 \times$ the number of non-zero entries as compared to $\tau = 0.85$. The divide-and-conquer method was clearly the most memory scalable of the direct algorithms, it was able to perform the largest example in this work: m = 512 with window based approximation (involving 785K unknowns and 7 million non-zero entries). The divide-and-conquer approach was able to perform the largest simulation m = 512 with window based approximation using p = 32 computers and the remaining cases using p = 16 or lower. For the case of

		16×16	32×32	48×48		64×64
Data	р	Total	Total	Total	p	Total
		Time (s)	Time (s)	Time (s)		Time (s)
window	1	1.86E+00	1.56E+01	5.60E+01	2	1.18E+02
window	2	1.80E+00	1.10E+01	3.75E+01	4	7.40E+01
window	4		2.71E+01	5.22E+01	8	5.22E+01
$\tau = 0.95$	1	3.70E+00	3.77E+01	1.95E+02	2	3.47E+02
$\tau = 0.97$	1	3.81E+00	3.89E+01	1.75E+02	2	3.56E+02
$\tau = 0.99$	1	4.00E+00	4.26E+01	1.85E+02	2	3.75E+02
$\tau = 0.95$	2	3.15E+00	3.00E+01	1.72E+02	4	2.13E+02
$\tau = 0.97$	2	3.18E+00	2.95E+01	1.74E+02	4	1.81E+02
$\tau = 0.99$	2	3.41E+00	3.23E+01	1.53E+02	4	2.40E+02
$\tau = 0.95$	4		2.36E+01	8.35E+01	8	1.34E+02
$\tau = 0.97$	4		2.39E+01	1.04E+02	8	1.37E+02
$\tau = 0.99$	4		2.50E+01	7.79E+01	8	1.54E+02

Table 4: Simulation time for divide-and-conquer algorithm mesh sizes m = 16,32,48,64.

m = 512 we are considering a compact representation for the inverse of the coefficient matrix, shown in (8), that would account for nearly 30GB of memory. Next, in order to give a practical measure for the improvement of the divide-and-conquer approach we examine in more detail the effect of increasing the amount of reluctance coupling.

Using several smaller mesh examples m = 16, 32, 48, and 64 we examine the sensitivity of each algorithm to the inclusion of reluctance coupling terms, i.e. larger values of the parameter τ . It is important to note that the same analysis can be performed for larger mesh sizes, however, as we are interested in the overall scaling rate we need only perform a relevant subset of all possible cases. Table 4 shows the results for the divide-and-conquer method and Table 5 for the Sparse LU and UMFPACK algorithms. From Table 5 first notice that although simulations using the UMFPACK algorithms, the simulation time was often substantially larger than that seen using the more dense SPAI based approximation. This can be attributed to the fact that the UMFPACK algorithm was unable to properly decide on an efficient ordering for elimination given the windowing coefficient matrix.

5.2.2 Comparison against iterative solvers

We now turn our attention to the performance of iterative methods for the transient simulation of power mesh structures. Specifically, we analyze the lack of scalability for the CG algorithm with respect to the addition of reluctance coupling. Although the CG method has the smallest memory consumption of any algorithm considered in this work, we observe that the iterative CG method using ILU scaled the worst with respect to the inclusion of reluctance coupling. A comparison of transient step solve times for the example m = 64 are shown in Figure 6. It is important to note that the times for the divide-and-conquer method, shown as horizontal lines in the Figure 6, represent the average time across the different SPAI thresholds. This is due to the fact that all

Data	16 × 16 Total	32×32 Total	48×48 Total	64×64 Total
	Time (s)	Time (s)	Time (s)	Time (s)
LU				
window	2.33E+00	1.74E+01	4.45E+01	9.75E+01
$\tau = 0.95$	3.58E+00	2.57E+01	8.35E+01	1.92E+02
$\tau = 0.97$	4.23E+00	3.32E+01	1.03E+02	2.24E+02
$\tau = 0.99$	4.28E+00	3.54E+01	1.01E+02	2.34E+02
UMF				
window	2.07E+00	1.57E+01	1.39E+02	1.31E+03
$\tau = 0.95$	2.97E+00	1.77E+01	5.05E+01	1.02E+02
$\tau = 0.97$	3.52E+00	2.49E+01	6.98E+01	1.54E+02
$\tau = 0.99$	4.03E+00	2.95E+01	9.40E+01	2.31E+02

Table 5: Simulation time for MATLAB Sparse LU and UMFPACK mesh sizes m = 16,32,48,64.

terms involved in the parallel matrix-vector multiply (21) are dense and the multiply time subsequently will not change as the number of non-zero entries in the coefficient matrix increases. This property does not hold for any other algorithm considered in this work.

Table 6 shows the sensitivity of the CG algorithm to the inclusion of additional reluctance coupling terms, again using several smaller mesh examples. On average the time for CG was more than $11 \times$ slower when comparing the SPAI approximation with $\tau = 0.99$ to the basic windowing approach. If we use this fact we can arrive at speed-up factors when comparing the dominant computational task for transient simulation. Specifically, if we consider the maximum scaling of the divide-and-conquer method for the cases m = 256,384, and 512, we calculate speed-up factors of $8.9 \times, 7.2 \times$, and $9.2 \times$ respectively, when considering transient solve times for a $\tau = 0.99$ quantity of reluctance coupling. We have restricted the analysis of large-scale simulations to the window based approach due to the excessive simulation time requirements for the CG method, given increased values of the parameter τ .

In summary, the use of a distributed sparse approximate inverse technique, for the inclusion of additional reluctance terms, has resulted in improved accuracy for the transient simulation of mesh structures. Given the computational burden associated with accurate RLC modeling of interconnects, the divide-and-conquer approach shows superior computational performance when compared to three of the best known algorithms applicable to this simulation problem. Specifically, the commercially available direct solvers showed advantages in computation time for small problems, but were unable to perform large simulations due to memory restrictions. Alternatively, the CG method was able to perform larger simulations, but the computational performance suffered heavily with the addition of branch coupling terms. The distributed computing approach presented in this paper has been shown to be *scalable* when considering the size of the mesh structure, as we are able to perform simulations with meshes over $2 \times$ as large as other direct methods. In addition, the proposed method is *scalable* with respect to the number of non-zero entries considered, as it does not lead to an increase in either time or memory requirements associated with the primary computational task



Figure 6: Comparison of CG and divide-and-conquer approach. Average time is shown for each transient step of m = 64 power mesh using SPAI: $\tau = 0.94 - 0.99$.

Data	16 × 16	32 × 32	48 × 48	64 × 64	Avg
	Total	Total	Total	Total	#
	Time (s)	Time (s)	Time (s)	Time (s)	Iter
CG window $\tau = 0.95$ $\tau = 0.97$ $\tau = 0.00$	5.94E+00	2.84E+01	7.49E+01	1.41E+02	1.51
	1.43E+01	7.22E+01	2.20E+02	3.76E+02	1.54
	2.86E+01	1.35E+02	4.10E+02	6.87E+02	1.50

Table 6: Simulation time for CG using ILU, 20 drop tolerances from 10^{-1} through 10^{-20} , for mesh sizes m = 16, 32, 48, 64.

in transient simulation.

6 Conclusion

Currently employed techniques for the simulation of mesh structures attempt to address the issue of increased problem sizes by trading off accuracy for simulation time via iterative schemes. Our algorithm is a direct tool that facilitates the simulation of large mesh structures through a divide-and-conquer approach. Due to the inherently parallel nature of the algorithm computing resources can be flexibly allocated toward either speeding up the simulation of a problem of a given size, or solving problems of larger sizes in comparable time. In addition, the flexibility of the algorithm is shown through the use of a more sophisticated approximation technique for the capturing of inductive coupling effects. The new technique offered a reduction of over $12 \times$ to the RMSE error when compared to a standard windowing technique. The scalability of the divide-andconquer method is clearly demonstrated by the absence of increases to time for the primary computational task for transient simulation, when considering the inclusion of these additional coupling terms. This attribute is not shared by any of the other methods analyzed in this work. In addition, the divide-and-conquer method was able to show substantial computational improvement over the most widely used numerical techniques applicable for these large-scale simulations. Specifically, the divide-andconquer approach allows for the the simulation of a 512×512 RLC mesh with a speedup factor over $9 \times$ when compared to the CG method with ILU. Therefore, we conclude that the divide-and-conquer algorithm presented here offers a framework which can be built upon for the large-scale accurate simulation of power mesh structures.

References

- H. Qian, S. R. Nassif, and S. S. Sapatnekar. Power grid analysis using random walks. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(8):1204–1224, 2005.
- [2] K. Sun, Q. Zhou, K. Mohanram, and D. C. Sorensen. Parallel domain decomposition for simulation of large-scale power grids. In *Proc. Int. Conf. on Computer Aided Design*, pages 54–59, 2007.
- [3] G. Weikun, S. X.-D. Tan, Z. Luo, and X. Hong. Partial random walk for large linear network analysis. In *Proc. IEEE Int. Symp. on Circuits and Systems*, pages 173–176, 2004.
- [4] M. Zhao, R. V. Panda, S. S. Sapatnekar, and D. Blaauw. Hierarchical analysis of power distribution networks. *IEEE Trans. on Computer-Aided Design of Inte*grated Circuits and Systems, 21(2):159–168, February 2002.
- [5] Y. Zhong and M. Wong. Fast algorithms for IR drop analysis in large power grid. In Proc. Int. Conf. on Computer Aided Design, pages 351–357, 2005.
- [6] J. Kozhaya, S. R. Nassif, and F. N. Najm. A multigrid-like technique for power grid analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 21(10):1148–1160, 2002.
- [7] H. Qian, S. R. Nassif, and S. S. Sapatnekar. A hybrid linear equation solver and its application in quadratic placement. In *Proc. Int. Conf. on Computer Aided Design*, pages 905–909, 2005.
- [8] U. M. Ascher and L. R. Petzold. Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. SIAM, 1998.
- [9] T. H. Chen, C. Luk, H. Kim, and C. C.-P. Chen. INDUCTWISE: Inductancewise interconnect simulator and extractor. In *Proc. Int. Conf. on Computer Aided Design*, pages 215–220, 2002.
- [10] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, 1996.

- [11] T. Davis and I. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. SIAM Journal on Matrix Analysis and Applications, 18(1):140– 158, 1997.
- [12] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM Journal on Scientific Computing*, 7:856–869, 1986.
- [13] G. Zhong, C.-K. Koh, and K. Roy. On-chip interconnect modeling by wire duplication. In Proc. Int. Conf. on Computer Aided Design, pages 341–346, 2002.
- [14] M. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. SIAM Journal on Scientific Computing, 18(3):838–853, 1997.
- [15] R. Nabben. Decay rates of the inverses of nonsymmetric tridiagonal and band matrices. SIAM Journal on Matrix Analysis and Applications, 20(3):820–837, 1999.
- [16] G. Meurant. A review on the inverse of symmetric block tridiagonal and block tridiagonal matrices. SIAM Journal on Matrix Analysis and Applications, 13(3):707–728, 1992.
- [17] R. Bevilacqua, B. Codenotti, and F. Romani. Parallel solution of block tridiagonal linear systems. *Linear Algebra Appl.*, 104:39–57, 1988.
- [18] J. Jain, S. Cauley, H. Li, C.-K. Koh, and V. Balakrishnan. Numerically stable algorithms for inversion of block tridiagonal and banded matrices. *Purdue ECE Technical Report* (1358), 2007.
- [19] S. Cauley, J. Jain, C.-K. Koh, and V. Balakrishnan. A scalable distributed method for quantum-scale device simulation. *Journal of Applied Physics*, 101(123715), 2007.
- [20] M. Kamon, M. J. Tsuk, and J. White. FastHenry: A multipole-accelerated 3-D inductance extraction program. *IEEE Trans. on Microwave Theory and Techniques*, 42(9):1750–1758, 1994.