A Hierarchical Matrix Inversion Algorithm for Vectorless Power Grid Verification

Xuanxing Xiong and Jia Wang
Electrical and Computer Engineering Department
Illinois Institute of Technology, Chicago, IL 60616, USA

***Abstract*—Vectorless power grid verification is a powerful technique to validate the robustness of the on-chip power distribution network for all possible current waveforms. Formulated and solved as linear programming problems, vectorless power grid verification demands intensive computational power due to the large number of nodes in modern power grids. Previous work showed that the performance bottleneck of this powerful technique is within the sub-problem of power grid analysis, which essentially computes the inverse of the sparse but large power grid matrix. In this paper, we propose a hierarchical matrix inversion algorithm to compute the rows of the inverse efficiently by exploiting the structure of the power grid. The proposed algorithm is integrated with a previous dual algorithm addressing an orthogonal sub-problem for vectorless power grid verification. Results show that the proposed hierarchical algorithm accelerates the matrix inversion significantly, and thus makes the overall vectorless power grid verification efficient.**

I. Introduction

The increasing complexity of modern integrated circuits (ICs) has made the power grid verification a challenging task. A robust power grid is essential to guarantee the correct functionality at desired performance of the circuit. When designing ICs, it is indispensable to verify that the power grid design is "safe", which means that the power supply noise at each node is acceptable for all possible runtime situations. There are mainly two sources of power supply noises: IR drops and Ldi/dt noise. In modern designs, the increasing number of transistors and the shrinking interconnect sizes lead to large IR drops, and the high clock frequency results in substantial amount of Ldi/dt noise. Moreover, as supply voltages are lowered to reduce power consumption while subthreshold voltages are decreased for better performance, the circuit become more vulnerable to power supply noises than ever before. Hence, full-chip power grid verification with high accuracy has become critical.

Today, the power grid is typically verified by simulation. Using the current waveforms of the circuit blocks attached to each node, one can simulate the power grid to evaluate the power supply noises. However, this simulation-based technique has two major drawbacks. First, it is either intractable or computationally prohibitive to enumerate all the possible waveform combinations. Second, one cannot perform power grid verification until the current waveforms of the circuit blocks are available, which makes accurate early verification impossible. Therefore, a verification approach, which is not dependent on simulation, is highly desirable. To satisfy this need, vectorless power grid verification under linear current constraints has been proposed in [1], and studied in [2], [3], [4]. All possible current waveforms of the circuit blocks are captured by a group of linear current constraints defining a feasible excitation set of currents. Then, the worst-case power supply noise at each node of an RC power grid is evaluated by solving a linear programming (LP) problem because the noises, in terms of voltage drops, are affine functions of the currents. However, due to large number of nodes in modern power grid, solving these LP problems directly is very time-consuming.

Recent works [3], [4] have achieved significant speed-ups by decomposing each LP problem into two steps. In the first step, the affine function that relates the voltage drop and the currents is obtained by solving a power grid analysis problem. In the second step, the maximum voltage drop is computed by maximizing the affine function subject to the current constraints, which is also an LP problem but with reduced number of decision variables and constraints. In [4], after an efficient dual algorithm was proposed that greatly reduces the running time for the second step, it was observed that the first step consumes more than 80% of the total running time for large power grids, and thus is the performance bottleneck of the overall approach.

For each node, the power grid analysis in the first step computes the corresponding row in the inverse of the power grid matrix. Therefore, for all the nodes, essentially the whole inverse of the power grid matrix is computed in a row-by-row manner. In comparison to [4] where the rows are computed independently by the preconditioned conjugate-gradient (PCG) method [5], [6], we propose a hierarchical algorithm to speed up the matrix inversion by exploiting the fact that there are dependencies among the rows in the inverse of the matrix.

Different from the conventional hierarchical matrix approaches [7], where the inverse matrix is computed by using the inversions of sub-matrices, our hierarchical matrix inversion algorithm partitions the power grid into several clusters and a set of external neighbor nodes, performs partial inversions on each cluster directly, computes the rows corresponding to these external neighbor nodes with the PCG method at first, then combines partial inversions and computed rows to generate the rows corresponding to the nodes within each cluster. Our algorithm also differs from the hierarchical power grid analysis [8], which uses macro-modeling to reduce the problem size for solving a single voltage noise vector (e.g. a row in the inverse) and does not exploit the row dependencies. The proposed hierarchical algorithm can be viewed as a combination of the direct method for solving linear system, e.g. LU factorization, and the iterative method, e.g. the PCG method. The advantages of both are seen in our algorithm – while small clusters are solved efficiently by direct methods, extremely large power grids that cannot be usually handled by direct methods and require iterative solvers can be

verified with our algorithm.

We integrate the proposed hierarchical matrix inversion algorithm with the dual algorithm in [4] to solve the vectorless power grid verification problem for RC power grids. Experimental results show that our hierarchical matrix inversion algorithm is more efficient than the approach in [4] that applies the PCG method to each node independently.

The rest of the paper is organized as follows. The problem formulation and previous approaches are summarized in Section II. The motivation of our algorithm is shown in Section III. The technical details of the proposed algorithm is presented in Section IV. After experimental results are shown in Section V, we conclude the paper in Section VI.

II. Preliminaries

A. Problem Formulation

We follow recent works [3], [4] to formulate the vectorless RC power grid verification problem. Consider an RC power grid of VDD pads, non-VDD nodes, and wire branches. Each branch is represented by a resistor, and each non-VDD node is connected to ground through a capacitor and a current source. Let $\mathbf{v}(t)$ be the time-varying voltage drops of those non-VDD nodes, $\mathbf{i}(t)$ be the time-varying current sources connected to them, G be the conductance matrix, and C be the capacitance matrix. The following system equation must be satisfied,

$$G\mathbf{v}(t) + C\dot{\mathbf{v}}(t) = \mathbf{i}(t). \tag{1}$$

For n non-VDD nodes in the power grid, both $\mathbf{v}(t)$ and $\mathbf{i}(t)$ are $n \times 1$ vectors, G is an $n \times n$ symmetric M-matrix, and C is an $n \times n$ diagonal matrix.

Given a current waveform, one can compute the voltage drops at each node by solving Eq.(1) through transient analysis. However, to validate the robustness of the power grid, such analysis either assumes peak currents for all current sources, which is intrinsically pessimistic, or requires to enumerate all possible current waveforms, which is computationally prohibitive. Current constraints [1] are introduced to address those issues since then the power grid verification can be performed on an optimization framework.

There are two kinds of current constraints: *local constraints* and *global constraints*. The local constraints are the upper-bounds on current sources,

$$0 \le \mathbf{i}(t) \le \mathbf{I}_L, \forall t,$$

where $\mathbf{I}_L \ge 0$ is an $n \times 1$ vector. The global constraints are the upper-bounds for groups of current sources. Assume that there are m current source groups (usually $m \ll n$). The global constraints are that,

$$U\mathbf{i}(t) \le \mathbf{I}_G, \forall t,$$

where U is an $m \times n$ $0/1$ matrix that assigns current sources to groups, and $\mathbf{I}_G \ge 0$ is an $m \times 1$ vector.

Vectorless power grid verification can be performed for either DC analysis model or transient analysis model (with a time-step h). The following maxVD-LCC (maximum voltage drop under linear current constraints) problem is identified in [4] as the key problem that should be solved for vectorless power grid verification.

Problem 1 (maxVD-LCC): Consider a power grid of n non-VDD nodes with a conductance matrix G and capacitance matrix C. Let $A = G$ for DC analysis model or $A = G + \frac{C}{h}$ for transient analysis model with time-step h. Given local and global current constraints with parameters $\mathbf{I}_L$, $\mathbf{I}_G$, and U, solve for every $1 \le l \le n$,

$$\text{Maximize } v_l \quad \text{s.t.} \quad A\mathbf{v} = \mathbf{i}, 0 \le \mathbf{i} \le \mathbf{I}_L, U\mathbf{i} \le \mathbf{I}_G. \tag{2}$$

Here $\mathbf{v}$ and $\mathbf{i}$ are the decision variables of voltage drops and current sources, respectively, and v_l is the l'th component of $\mathbf{v}$.

Note that Eq.(2) is a linear programming (LP) problem, and the problem size is proportional to the power grid size n. As n is usually large for practical power grids, solving all these LP problems directly requires a large amount of runtime and thus is prohibitively expensive. It is proposed in [3], [4] that Eq.(2) should be decomposed by taking advantage of the property of A. As A is an $n \times n$ M-matrix and thus invertible, we have $\mathbf{v} = A^{-1}\mathbf{i}$. Let $\mathbf{c}_l$ be the l'th row of A^{-1}. Then $v_l = \mathbf{c}_l\mathbf{i}$. Therefore, instead of letting an LP solver to handle the constraints $A\mathbf{v} = \mathbf{i}$, one can simplify the LP problem in Eq.(2) by computing $\mathbf{c}_l$ first. Let $\mathbf{e}_l$ be the $n \times 1$ vector of 0's except for its l'th component being 1. Since both A and A^{-1} are symmetric, we have $\mathbf{c}_l = A^{-1}\mathbf{e}_l$. Therefore, $\mathbf{c}_l$ can be computed by solving $A\mathbf{x} = \mathbf{e}_l$, which can be done by a power grid analyzer. In summary, Eq.(2) is decomposed into the following two sub-problems.

$$\text{I: Compute } \mathbf{c}_l \text{ by solving } A\mathbf{x} = \mathbf{e}_l, \tag{3}$$

$$\text{II: Maximize } v_l = \mathbf{c}_l\mathbf{i} \quad \text{s.t.} \quad 0 \le \mathbf{i} \le \mathbf{I}_L, U\mathbf{i} \le \mathbf{I}_G. \tag{4}$$

B. Previous Approaches

The aforementioned problem decomposition has been explored in both [3] and [4]. As there are still a substantial number of decision variables and constraints in Eq.(4), these two works take different approaches to solve both sub-problems efficiently. In [3], it is proposed to compute an approximated $\mathbf{c}_l$ with a small number of non-zero components in the first sub-problem, so that most decision variables and constraints can be dropped from Eq.(4), which can then be solved efficiently by any LP solver. However, as the number of non-zero components depends on the accuracy of the approximation, when a higher level of accuracy is desired, the reduction of the decision variables and constraints diminishes and so does the efficiency to solve Eq.(4). Moreover, to generate the approximated $\mathbf{c}_l$ introduces significant running time overhead in comparison to efficient power grid analysis techniques as demonstrated by [4].

Different from [3], [4] adopts an approach whose running time is not strongly dependent on the accuracy of the solution. The accuracy of the overall solution is established from those of the two sub-problems. Let δ_{inv} and δ_{lp} be two user-specified error-tolerances for the two sub-problems respectively. For the first sub-problem, let the residual vector $\mathbf{r} \stackrel{\Delta}{=} \mathbf{e}_l - A\mathbf{c}_l$. Let r_j be the j'th component of $\mathbf{r}$ and Δ be the maximum component of the solutions of the linear equations $A\mathbf{x} = \mathbf{I}_L$. The computed

$\mathbf{c}_l$ should satisfy that,

$$||\mathbf{r}||_1 \triangleq \sum_{j=1}^{n} |r_j| \leq \frac{\delta_{\text{inv}}}{\Delta}. \tag{5}$$

For the second sub-problem, let $\widehat{v_l}$ be the optimal value of Eq.(4) with the computed $\mathbf{c}_l$, any algorithm that solves Eq.(4) can be terminated with a solution v_l^+ when

$$v_l^+ - \delta_{\text{lp}} \leq \widehat{v_l} \leq v_l^+. \tag{6}$$

Let v_l^* be the optimal value of Eq.(4) for the exact $\mathbf{c}_l$. It is stated in [4] that,

Lemma 1: Suppose both Eq.(5) and (6) hold. Let $\overline{v_l} \triangleq v_l^+ + \Delta \sum_{j=1}^{n} \max(r_j, 0)$ be the conservative bound of the worst-case voltage drop. Then,

$$\overline{v_l} - \delta_{\text{inv}} - \delta_{\text{lp}} \leq v_l^* \leq \overline{v_l}.$$

A dual approach is proposed in [4], to solve Eq.(4) efficiently. It is shown that Eq.(4) is equivalent to the following simplified dual problem,

$$\begin{aligned} &\text{Minimize } D(\boldsymbol{\gamma}) \quad \text{s.t.} \quad \boldsymbol{\gamma} \geq 0, \\ &\text{where } D(\boldsymbol{\gamma}) \triangleq \mathbf{I}_G^T \boldsymbol{\gamma} + \textstyle\sum_{j=1}^{n} I_{L,j} \max(0, c_{l,j} - \mathbf{u}_j^T \boldsymbol{\gamma}). \end{aligned} \tag{7}$$

Here $\boldsymbol{\gamma}$ are the m decision variables corresponding to the Lagrangian multipliers of the global constraints, $I_{L,j}$ is the j'th component of $\mathbf{I}_L$, $c_{l,j}$ is the j'th component of $\mathbf{c}_l$, and $\mathbf{u}_j$ is the j'th column of U. It is further shown that $D(\boldsymbol{\gamma})$ is a convex function of $\boldsymbol{\gamma}$ and thus Eq.(7) is a convex programming problem that is then solved by Kelley's cutting-plane method [9]. Since Eq.(7) only has m decision variables and the constraints are extremely simple, solving Eq.(7) is much more efficient than solving Eq.(4) directly. In addition, it is proposed in [4] that the preconditioned conjugate gradient (PCG) method [5], [6] can be employed to compute $\mathbf{c}_l$ with a proper choice of preconditioner. The stochastic preconditioning technique proposed in [10] is applied in [4] to generate the preconditioner using random walks [11].

The PCG method, combined with the dual approach to solve Eq.(4), allows [4] to achieve significant speed-ups over previous works including [3] without sacrificing the solution accuracy. As observed in [4], the PCG method consumes more than 80% of the total running time for all large power grids. Therefore, any further reduction in running time must address the problem of computing $\mathbf{c}_l$'s more efficiently than the PCG method, which is clearly non-trivial given the large number of nodes in power grids.

III. Motivation

As solving $\mathbf{c}_l$ is now the performance bottleneck of power grid verification when the dual approach is applied, it is of great interest to speed up computing $\mathbf{c}_l$, so that large power girds can be verified in a timely manner. Clearly, to achieve such a goal, one must be able to explore the structure of the power grid, as the voltage drops of a node and its neighboring nodes are closely related.

Let's fist look at a single node l in the power grid for DC analysis model, as illustrated in Fig. 1. Applying Kirchhoff's

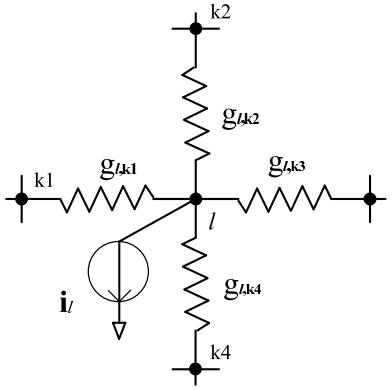

Fig. 1. A representative node l in the power grid.

Current Law (KCL), we have

$$\sum_{\forall k \in \mathcal{N}(l)} g_{l,k}(v_l - v_k) = i_l, \tag{8}$$

where $\mathcal{N}(l)$ is the set of node l's neighboring nodes, $g_{l,k}$ is the conductance between node l and k, v_l and v_k are the voltage drop at node l and k respectively, and i_l is the current source attached to node l. Define $g_l \triangleq \sum_{\forall k \in \mathcal{N}(l)} g_{l,k}$, then Eq.(8) can be rearranged into

$$v_l = \frac{1}{g_l}\big(i_l + \sum_{\forall k \in \mathcal{N}(l)} g_{l,k} v_k\big). \tag{9}$$

Recall that $v_l = \mathbf{c}_l^T \mathbf{i}$, $v_k = \mathbf{c}_k^T \mathbf{i}$ and $i_l = \mathbf{e}_l^T \mathbf{i}$. Eq.(9) is equivalent to

$$\begin{aligned} \mathbf{c}_l^T \mathbf{i} &= \frac{1}{g_l}\big(\mathbf{e}_l^T \mathbf{i} + \sum_{\forall k \in \mathcal{N}(l)} g_{l,k} \mathbf{c}_k^T \mathbf{i}\big) \\ &= \Big(\frac{1}{g_l}\big(\mathbf{e}_l + \sum_{\forall k \in \mathcal{N}(l)} g_{l,k} \mathbf{c}_k\big)\Big)^T \mathbf{i}. \end{aligned} \tag{10}$$

Eq.(10) holds for any current vector $\mathbf{i}$, therefore,

$$\mathbf{c}_l = \frac{1}{g_l}\big(\mathbf{e}_l + \sum_{\forall k \in \mathcal{N}(l)} g_{l,k} \mathbf{c}_k\big). \tag{11}$$

According to Eq.(11), one can directly compute the $\mathbf{c}_l$ of a node by using the $\mathbf{c}_l$ of its neighboring nodes. Due to the structure of the power grid, a node only has a few number of neighboring nodes, so computing $\mathbf{c}_l$ using Eq.(11) consumes much less runtime than solving $\mathbf{c}_l$ with the PCG method. In order to apply Eq.(11) to speed up computing $\mathbf{c}_l$, one shall first use the PCG method to solve the $\mathbf{c}_l$ of the nodes, which can cover all the edges in the power grid, then employ Eq.(11) to compute the $\mathbf{c}_l$ of the other nodes. Note that finding the nodes that need to be solved by the PCG method is a vertex cover problem. Our preliminary experimental results show that only about 35%-38% of the nodes' $\mathbf{c}_l$ can be computed by using Eq.(11). Intuitively, replacing node l by a subset of nodes in Fig. 1, if one can formulate the $\mathbf{c}_l$ of these nodes similarly, then the $\mathbf{c}_l$ of more nodes can be computed using this approach. Motivated by the desire to speed up computing $\mathbf{c}_l$ to the most limit, we explore this method to compute the $\mathbf{c}_l$ of a subset of nodes.

Our major contribution in this paper is an efficient hierarchi-

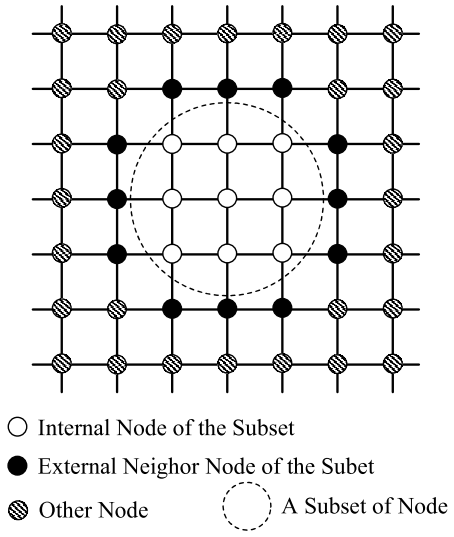

Fig. 2. A subset of nodes in the power grid.

cal approach to compute all the $\mathbf{c}_l$. We exploit the structure of the power grid to formulate the $\mathbf{c}_l$ of a subset of nodes in terms of their external neighbor nodes. We propose to partition the power grid into a set of external neighbor nodes and a number of partitions using the hypergraph partitioning technique, and compute the $\mathbf{c}_l$ of the nodes within each partition using the $\mathbf{c}_l$ of these external neighbor nodes computed by the PCG method [4]. We design the HierarchicalVD algorithm to solve the *maxVD-LCC* problem efficiently using our hierarchical approach to compute $\mathbf{c}_l$, and the dual approach of [4] to solve Eq.(4). The details are presented in the next section.

IV. Hierarchical Power Grid Verification

A. Computing $\mathbf{c}_l$ for a Subset of Nodes

Consider a subset of nodes in the power grid as illustrated in Fig. 2, let's call the nodes within this subset *internal nodes*, and these internal nodes' neighbors which are not included in this subset *external neighbors*. In other words, for any internal node of a subset, any of its neighbors is either an internal node or an external neighbor of this subset.

Let's first consider the DC analysis model, where $A = G$ is the conductance matrix. Let n' and m' be the number of internal nodes and external neighbors of a subset respectively. Applying KCL for all the internal nodes of the subset, we have

$$[G_{in} \quad G_{ex}] \begin{bmatrix} \mathbf{v}_{in} \\ \mathbf{v}_{ex} \end{bmatrix} = \mathbf{i}_{in}, \tag{12}$$

where $G_{in} \in \mathcal{R}^{n' \times n'}$ is the conductance matrix of the internal nodes, $G_{ex} \in \mathcal{R}^{n' \times m'}$ is the conductance links between internal nodes and external neighbors, $\mathbf{v}_{in}$ and $\mathbf{v}_{ex}$ are the voltage drop vector of the internal nodes and external neighbors respectively, and $\mathbf{i}_{in}$ is the current source vector of the internal nodes. Rearrange Eq.(12), we get

$$\mathbf{v}_{in} = G_{in}^{-1}(\mathbf{i}_{in} - G_{ex}\mathbf{v}_{ex}). \tag{13}$$

Introduce $C_{in} \in \mathcal{R}^{n' \times n}$, $C_{ex} \in \mathcal{R}^{m' \times n}$ and $E_{in} \in \mathcal{R}^{n' \times n}$. Let each row of C_{in} be the corresponding row of each internal node in A^{-1}, each row of C_{ex} be the corresponding row of each external neighbor node in A^{-1}, and each row of E_{in} be the transposed $\mathbf{e}_l$ of the corresponding internal node. We have $\mathbf{v}_{in} = C_{in}\mathbf{i}, \mathbf{v}_{ex} = C_{ex}\mathbf{i}$, and $\mathbf{i}_{in} = E_{in}\mathbf{i}$, then Eq.(13) can be rewritten as

$$\begin{aligned} C_{in}\mathbf{i} &= G_{in}^{-1}(E_{in}\mathbf{i} - G_{ex}C_{ex}\mathbf{i}) \\ &= \big(G_{in}^{-1}(E_{in} - G_{ex}C_{ex})\big)\mathbf{i}. \end{aligned} \tag{14}$$

Eq.(14) holds for any current vector $\mathbf{i}$, so

$$C_{in} = G_{in}^{-1}(E_{in} - G_{ex}C_{ex}). \tag{15}$$

For the transient analysis model, $A = G + \frac{C}{h}$. One can create an *adjusted power grid* by attaching a VDD pad to each non-VDD node and setting the conductances of these newly added edges according to the diagonal components of $\frac{C}{h}$. Then A is exactly the conductance matrix of the adjusted power grid. If one constructs G_{in} and G_{ex} for this adjusted power grid, then Eq.(15) still holds. In summary, for either DC or transient analysis, one can always treat A as a conductance matrix, create the corresponding G_{in} and G_{ex} according to A, and derive Eq.(15). Let A_{in} and A_{ex} denote the G_{in} and G_{ex} constructed according to the power grid conductance matrix A, then we have the following lemma.

Lemma 2: $C_{in} = A_{in}^{-1}(E_{in} - A_{ex}C_{ex})$.

Obviously, the $\mathbf{c}_l$ of a subset of nodes (C_{in}) can be computed by using the $\mathbf{c}_l$ of their external neighbors (C_{ex}). Consider the nodes within a sub-block of the power grid as such a subset. If all the $\mathbf{c}_l$ of its external neighbors have been solved, one can directly compute the $\mathbf{c}_l$ of its internal nodes. As we will show in the next subsection, the power grid can be divided into a set of external neighbors and a number of partitions. Each partition is a subset of nodes, and it is almost always a sub-block of the power grid. Using such a power grid partitioning technique, only a small amount of nodes are external neighbors, and most of the nodes are internal nodes of partitions. Then, one only need to compute a small amount of $\mathbf{c}_l$'s corresponding to the external neighbors independently with the PCG method, and compute most $\mathbf{c}_l$'s corresponding to the internal nodes using Lemma 2.

The computation in Lemma 2 can be made more efficient by exploiting the properties of these matrices. Let's first consider their density, $A_{in}^{-1} \in \mathcal{R}^{n' \times n'}$ and $C_{ex} \in \mathcal{R}^{m' \times n}$ are dense, $E_{in} \in \mathcal{R}^{n' \times n}$ is a sparse 0/1 matrix with only n' 1s, and $A_{ex} \in \mathcal{R}^{n' \times m'}$ is also sparse since only a few internal nodes are connected with external neighbors. Because of the special characteristic of E_{in}, the computation associated with E_{in} is negligible. Most time complexity is attributable to computing $A_{in}^{-1}A_{ex}C_{ex}$. Recall that the power grid is partitioned into a number of partitions. We observe that the number of internal nodes in a partition is usually larger than the number of its external neighbors in our preliminary experiments. In other words, we usually have $n \gg n' > m'$. Hence, computing $A_{in}^{-1}A_{ex}$ first consumes less runtime and memory than computing $A_{ex}C_{ex}$ first in order to solve $A_{in}^{-1}A_{ex}C_{ex}$, then Lemma 2 can be rearranged into

$$C_{in} = \big(A_{in}^{-1}(-A_{ex})\big)C_{ex} + A_{in}^{-1}E_{in}. \tag{16}$$

By keeping the number of internal nodes in the subset within some bound, A_{in}^{-1} can be solved by LU factorization together with a forward solve and a backward solve.

The time complexity of the PCG method to compute a single

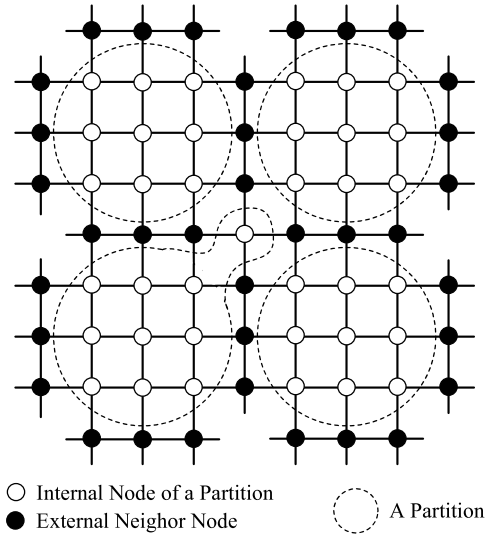

Fig. 3. Part of a partitioned power grid.

$\mathbf{c}_l$ is $O(KM)$ where K is the number of iterations toward convergence and M is the number of non-zero elements in the preconditioner. To apply Eq.(16) to a single partition, if n' is reasonably small, the most time-consuming computation is the dense matrix-matrix multiplication of $A_{in}^{-1}A_{ex}$ and C_{ex}, which takes $O(m'n'n)$ time. Let $\rho \in [0,1]$ be the ratio of internal nodes to n. Assume that each partition has roughly the same size, the overall time complexity will be,

$$\begin{aligned} &O(m'n'n) \times \frac{\rho n}{n'} + O(KM)(1-\rho)n \\ &= \rho O(m'n^2) + (1-\rho)O(KMn). \end{aligned}$$

In other words, for each internal node, we replace the PCG solution that takes $O(KM)$ time by a vector-matrix multiplication that takes $O(m'n)$ time, which will result in a significant reduction of running time if the parameters are chosen properly. Clearly, if n' is too small, then ρ is small. In Section III, we show when $n' = 1$, $\rho \approx 0.35$ to 0.38. As n' increases, ρ increases and the overall complexity may decrease depending on the changes in m'. In addition, n' should be kept small enough such that the computation of A_{in}^{-1} will not dominate that of Eq.(16). Basically, there is a trade-off between the increasing partition size and the increasing time complexity for each internal node, and we will demonstrate it in Section V.

B. Power Grid Partitioning

In order to apply Lemma 2 to compute $\mathbf{c}_l$, one has to find a subset of nodes as external neighbors, which split the power grid into a number of non-connected partitions. Moreover, it is also desired that the size of each partition is in some user-controllable bound, so that the $\mathbf{c}_l$ of the nodes in these partitions can be computed efficiently according to Eq.(16). Fig. 3 shows an example of such a desired partitioned power grid. For simplicity, there are only a total of 36 external neighbors and four partitions, each of which has 9 or 10 internal nodes. Note that the desired partitioning is different from the partitioning in hierarchical power grid analysis [8], where all the nodes are included in partitions. Our power grid partitioning problem can be formulated mathematically as follows.

Problem 2 (PG-Partition): Let $\mathcal{V}$ be the set of all the non-VDD nodes in the power grid, $\mathcal{E}$ be the set of all the edges connecting these nodes, rps be a user-specified rough partition size, $npart$ ($npart \gg 1$) be the desired number of partitions corresponding to rps. Let $\mathcal{P}_k$ and $\mathcal{N}(\mathcal{P}_k)$ be the set of internal nodes and external neighbors of the k'th partition respectively, ϵ be a user-specified imbalance bound of the partition sizes. Find a minimum set $\mathcal{S}$ of external neighbor nodes, which partition the power grid $(\mathcal{V}, \mathcal{E})$ into $npart$ mutually disjoint partitions $\mathcal{P}_k$ $(1 \le k \le npart)$, such that

$$\begin{aligned} &\mathcal{V} = \mathcal{S} \bigcup (\bigcup_{k=1}^{npart} \mathcal{P}_k), \mathcal{S} = \bigcup_{k=1}^{npart} \mathcal{N}(\mathcal{P}_k), \\ &\mathcal{S} \bigcap \mathcal{P}_k = \emptyset, \forall 1 \le k \le npart, \\ &\mathcal{P}_j \bigcap \mathcal{P}_k = \emptyset, \forall 1 \le j \ne k \le npart, \\ &|\mathcal{P}_k| \le \overline{P}, \forall 1 \le k \le npart, \end{aligned}$$

where $\overline{P}$ is a partition size upper bound determined by rps and ϵ.

We propose to solve this problem by introducing a hypergraph model of the power grid, and performing hypergraph partitioning on it. Let $\mathcal{V}_h$ and $\mathcal{E}_h$ be the set of vertices and hyperedges in the hypergraph model respectively. Define

$$\mathcal{V}_h \triangleq \mathcal{E}, \mathcal{E}_h \triangleq \mathcal{V}. \tag{17}$$

Then $(\mathcal{V}_h, \mathcal{E}_h)$ is the hypergraph model. The vertices in $\mathcal{V}_h$ represent the edges in the power gird, and the hyperedges in $\mathcal{E}_h$ represent the nodes. In conventional hypergraphs, each vertex has a specific area, and each hyperedge has a specific weight. Hypergraph partitioning algorithms typically partition the hypergraph by removing some hyperedges (also called hyperedge-cuts) to get isolated partitions of the hypergraph. The objective is to minimize the total weight of hyperedge-cuts while keeping the area of different partitions being balanced. For this hypergraph model, the hyperedge-cuts represent the external neighbors of the power grid, the hyperedges in each partition represent the internal nodes, and the hyperedge-cuts associated with each partition represent the external neighbors of that partition.

In order to apply the hypergraph partitioning algorithms to solve Problem 2, we define the area of each vertex and the weight of each hyperedge in the hypergraph model as follows. Let $a(j)$ be the area of the j'th vertex in $\mathcal{V}_h$, $\mathcal{H}_{j,1}$ and $\mathcal{H}_{j,2}$ be the two hyperedges that include vertex j, $w(k)$ be weight of the k'th hyperedge in $\mathcal{E}_h$. Define

$$a(j) \triangleq \frac{1}{|\mathcal{H}_{j,1}|} + \frac{1}{|\mathcal{H}_{j,2}|}, \forall 1 \le j \le |\mathcal{V}_h|, \tag{18}$$

$$w(k) \triangleq 1, \forall 1 \le k \le |\mathcal{E}_h|. \tag{19}$$

Then it can be proved that there exists some partition size upper bound $\overline{P}$, which is determined by rps and ϵ.

Let $\mathcal{P}_k^h (1 \le k \le npart)$ be the set of vertices in the k'th partition of the hypergraph model, $a(\mathcal{P}_k^h)$ be the total area of

the k'th partition. Define

$$a(\mathcal{P}_k^h) \triangleq \sum_{\forall \text{ vertex } j \in \mathcal{P}_k^h} a(j).$$

With this kind of area definition, the area of each partition is larger than the number of hyperedges within it, because the associated hyperedge-cuts of a partition also contribute to its area. For example, let's look at any partition with 9 internal nodes as shown in Fig. 3. It has 24 edges, 9 nodes, and 12 external neighbors. In its corresponding hypergraph model, it has 24 vertices, 9 hyperedges and 12 associated hyperedge-cuts. The area of each vertex in this partition is $\frac{1}{4}+\frac{1}{4}=\frac{1}{2}$. Then the total area of this partition is $24 \times \frac{1}{2} = 12 > 9$. As each hyperedge in a partition represents an internal node within that partition, the number of hyperedges in the k'th partition is equal to $|\mathcal{P}_k|$, so we have the following lemma.

Lemma 3: For every $1 \le k \le npart$, $|\mathcal{P}_k| < a(\mathcal{P}_k^h)$.

Define the average area of all the partitions

$$\begin{aligned} \overline{a_{\mathcal{P}}} &\triangleq \frac{\sum_{k=1}^{npart} a(\mathcal{P}_k^h)}{npart} = \frac{\sum_{\forall \text{ vertex } j \in \mathcal{V}_h} a(j)}{npart} \\ &= \frac{|\mathcal{E}_h|}{npart} = \frac{|\mathcal{V}|}{npart} = \frac{n}{npart}. \end{aligned}$$

In order to balance the area of different partitions, hypergraph partitioning algorithms typically keep the imbalance of $a(\mathcal{P}_k^h)$ among all partitions within the user-specified imbalance bound ϵ. We have

$$\frac{|a(\mathcal{P}_k^h) - \overline{a_{\mathcal{P}}}|}{\overline{a_{\mathcal{P}}}} \le \epsilon, \ \forall 1 \le k \le npart.$$

Therefore,

$$a(\mathcal{P}_k^h) \le (1+\epsilon) \cdot \overline{a_{\mathcal{P}}} = (1+\epsilon) \cdot \frac{n}{npart}. \tag{20}$$

Combine Lemma 3 with Eq.(20), we get

$$|\mathcal{P}_k| < (1+\epsilon) \cdot \frac{n}{npart}.$$

Define

$$npart \triangleq \lceil \frac{n}{rps} \rceil, \tag{21}$$

then

$$|\mathcal{P}_k| < (1+\epsilon) \cdot \frac{n}{npart} \le (1+\epsilon) \cdot rps,$$

and we have the following lemma.

Lemma 4: Define $\overline{P} \triangleq (1+\epsilon) \cdot rps$, then for every $1 \le k \le npart$, $|\mathcal{P}_k| < \overline{P}$.

Clearly, the vertex area definition guarantees that the number of internal nodes in each partition ($|\mathcal{P}_k|$) is bounded by $\overline{P}$. One can adjust the size of partitions by using different rps. As the weight of each hyperedge in the hypergraph model is set to 1, the hypergraph partitioning algorithms will minimize the number of hyperedge-cuts, thus deriving a minimum set of external neighbors. After the hypergraph model is partitioned, the partitioning result can be mapped back to the original power grid to derive $\mathcal{S}, \mathcal{P}_k$, and $\mathcal{N}(\mathcal{P}_k)$, since $\mathcal{S}$ is the set of nodes corresponding to the hyperedge-cuts, $\mathcal{P}_k$ is the set of nodes corresponding to the hyperedges with in the k'th partition, and $\mathcal{N}(\mathcal{P}_k)$ is the set of nodes corresponding to the associated hyperedge-cuts of the k'th partition.

C. The HierarchicalVD Algorithm

Algorithm HierarchicalVD

Inputs

A, $\mathbf{I}_L$, $\mathbf{I}_G$, U: as specified in Problem 1.
rps, ϵ: as specified in Problem 2.
δ_{inv}, δ_{lp}: user-specified error-tolerances.

Outputs

Maximum voltage drops at each node.

```
1  Create the hypergraph model according to Eq.(17), (18) and (19)
2  Compute npart according to Eq.(21)
3  Partition the hypergraph model into npart partitions to derive S, P_k, N(P_k) (1 ≤ k ≤ npart) as defined in Problem 2
4  For k = 1 to npart
5    For all un-solved external neighbors in N(P_k)
6      Apply the PCG method in [4] to compute c_l
7      Apply the dual approach in [4] to solve Eq.(4)
8      Report the maximum voltage drop v_l
9    For all internal nodes in P_k
10     Apply Eq.(16) to compute an approximated c_l
11     Check whether the approximated c_l satisfies Eq.(5), if not, use the PCG Method to refine the approximated c_l until it satisfies Eq.(5)
12     Apply the dual approach in [4] to solve Eq.(4)
13     Report the maximum voltage drop v_l
```

Fig. 4. The HierarchicalVD Algorithm.

Combining our hierarchical approach to compute $\mathbf{c}_l$, the PCG method and the dual approach in [4], we design the HierarchicalVD algorithm to solve the *maxVD-LCC* problem as illustrated in Fig. 4. Two user-specified error-tolerances δ_{inv} and δ_{lp} are employed to control the accuracy of the solution of Eq.(3) and (4) respectively. In this algorithm, the power grid is partitioned at first, then the partitions are verified sequentially. The verification of a partition has two steps. First, verify its un-solved external neighbors and store the computed $\mathbf{c}_l$ of these external neighbors in memory. Second, solve its internal nodes, where the $\mathbf{c}_l$ of these internal nodes are computed by Eq.(16). As all the $\mathbf{c}_l$ of external neighbors are computed by the PCG method with error-tolerance δ_{inv}, thus not being exact, one can only get an approximated $\mathbf{c}_l$ of each internal node using Eq.(16). In order to guarantee the accuracy of the solution, we check whether the computed $\mathbf{c}_l$ is acceptable, and use the PCG method to refine it if necessary.

D. Practical Memory Management

In order to apply the HierarchicalVD algorithm, there must be at least enough memory to store the external neighbors' rows C_{ex} and the partial inversion A_{in}^{-1} of any partition. With limited amount of memory available in the computer, one must keep the number of external neighbors and internal nodes for any partition within some bound, so that C_{ex} and A_{in}^{-1} can be stored. Recall that n' and m' denote the number of internal nodes and the number of external neighbors for a partition respectively. In our experiments, we observe that $rps \ge n' > m'$ almost always. Hence, we adopt rough partition size rps

from a few dozens to one thousand, such that C_{ex} and A_{in}^{-1} can easily be buffered. Since we verify the internal nodes one by one, there is no need to store the internal nodes' rows C_{in}.

As illustrated in Fig. 3, a node can be the external neighbor of two or more partitions, then all these associated partitions will use its $\mathbf{c}_l$ to compute the approximate $\mathbf{c}_l$ of internal nodes. For such an external neighbor node, once the $\mathbf{c}_l$ is computed, one need to keep it in memory until all of its associated partitions have been verified. However, this approach can be problematic. Consider the worst-case scenario, if the first partition and the last partition share a lot of external neighbors, then the $\mathbf{c}_l$ of their common external neighbors need to be stored in memory for the whole duration of the power grid verification. Although such scenario seldom happens, one usually need to keep a number of useful $\mathbf{c}_l$ for the un-verified partitions, but the available memory in the computer may not be sufficient to store all of the useful $\mathbf{c}_l$. Thus it is desired that the partitions of the power grid have good locality, such that the neighboring partitions, which often share some external neighbors, can be verified by the HierarchicalVD algorithm in a small time frame, thus minimizing the number of useful $\mathbf{c}_l$ that one need to buffer at runtime.

In our implementation, we use a fixed-size buffer to store the computed $\mathbf{c}_l$ of external neighbors, and assign a weight for each buffered $\mathbf{c}_l$ to be the number of un-verified partitions that will use it for computation. If we have to store a new $\mathbf{c}_l$ when the buffer is full, then the $\mathbf{c}_l$ with the minimum weight is freed. After finishing the verification of a partition, we decrease the weight of the buffered $\mathbf{c}_l$ for its external neighbors by 1. A zero weight $\mathbf{c}_l$ implies that no partition will use it for computation, so any buffered $\mathbf{c}_l$ is freed as soon as its weight is decreased to 0. Using this scheme, if any useful $\mathbf{c}_l$ is freed when the buffer is full, we need to re-compute it when we need it for computation.

In our experiments, we set the buffer size to be 1000 $\mathbf{c}_l$, and employ the hypergraph partitioning tool hMETIS [12] to partition the hypergraph model. Results with a rough partition size $rps = 100$ for different power grids show that the maximum number of buffered useful $\mathbf{c}_l$ is usually less than 800, and no $\mathbf{c}_l$ is re-computed, except for the largest power grid of 562K non-VDD nodes. When verifying the largest grid, although the buffer is fully utilized, only about 5% of the external neighbors' $\mathbf{c}_l$ are re-computed. Therefore, it is shown that the power grid partitions generated by hMETIS have the desired locality, and there is no need to find a order to verify these partitions with small memory requirement.

V. Experimental Results

We implement our HierarchicalVD algorithm and the DualVD algorithm [4] for comparison in C++. Both algorithms share the same routine to compute the stochastic preconditioner, to perform PCG iterations to compute $\mathbf{c}_l$ when necessary, and to solve Eq.(4) (using MOSEK [13] as the LP solver for the cutting-plane method). In addition, our HierarchicalVD algorithm employs the proposed hierarchical matrix inversion algorithm to compute $\mathbf{c}_l$. The hypergraph model of the power grid is partitioned by hMETIS [12], a hypergraph partitioning tool. The LU factorization tool SuperLU [14] is used to compute A_{in}^{-1} in Eq.(16). Both algorithms are compiled with the same GCC compiler and executed on one core of the same 64-bit Linux workstation with 2.4GHz Intel Q6600 processor and 8GB memory.

We adopt the 7 power grids used in [4] and follow [4] to generate 4 global constraints and to set both error tolerances δ_{inv} and δ_{lp} to 0.1mV. Additional settings of global constraints are not experimented since they introduce marginal running time overhead as shown in [4] for the second sub-problem and do not affect the computation of $\mathbf{c}_l$.

Since hMETIS allows to choose between the recursive bisection partitioning and the k-way partitioning, we experiment with both options. The "unbalance factor" for these two options are set to 2 and 10 respectively. We observe that the k-way partitioning consumes much more memory than the recursive bisection partitioning, and it fails to partition the largest power grid of 562K non-VDD nodes into more than 1000 partitions due to lack of enough memory. For all the test cases that we have evaluated, recursive bisection partitioning consumes less runtime and generates slightly smaller number of hyperedge-cuts. Therefore, we only report the results using recursive bisection partitioning. For both of these two options, the runtime spent on partitioning the hypergraph model of the power grid ranges from a few seconds to about 10 minutes, and it is negligible in comparison with the overall runtime of the power grid verification.

The performance of the proposed hierarchical matrix inversion algorithm depends on the choice of the rough partition size rps. For all rps settings from a few dozens to one thousand, A_{in}^{-1} is computed by SuperLU which usually consumes less than 1 seconds and thus is negligible. On the other hand, as we increase rps, the number of partitions decreases and the number of internal nodes increases. Therefore, more $\mathbf{c}_l$'s can be obtained using Eq.(16) instead of the PCG method. However, because the number of external neighbors for a single partition also increases as rps increases, it takes more runtime to compute the $\mathbf{c}_l$ of each internal node. So, there is a trade-off between the increasing number of internal nodes and the increasing time required to compute the $\mathbf{c}_l$'s for them. To evaluate this trade-off, we perform a set of experiments using the largest power grid of 562K non-VDD nodes. We verify about 10K nodes of this power grid with rough partition size $rps \in \{20, 40, 60, 100, 250, 500, 750, 1000\}$, and estimate the average runtime per internal node and the overall average runtime per node. The results are illustrated in Fig. 5 where the x-axis is the percentage of the internal nodes. When rps increases from 20 to 1000, the percentage of the internal nodes increases from 72% to 95%, and it takes more runtime to verify an internal node. The overall average first decreases as the number of internal nodes increases but then increases as it takes more runtime to compute $\mathbf{c}_l$. We observe the minimum is reached at $rps = 100$ while the other choices would result in a maximum performance degradation close to 30%. We perform additional experiments with the aforementioned rps setting for the other power grids, and notice that the minimum runtime per node is achieved at different rps for different power grids, but in general a non-optimal but reasonable choice of rps would not result in significant performance degradations. Therefore, we always choose $rps = 100$ for the HierarchicalVD algorithm hereafter.

The runtime comparison of the HierarchicalVD and DualVD is presented in Table V. The runtime can be decomposed

TABLE I
RUNTIME COMPARISON OF HIERARCHICALVD AND DUALVD

Benchmarks	DualVD [4]		HierarchicalVD				Speedup	
Nodes	Inverse	Total	Partitions	External Neighbors	Inverse	Total	Inverse	Total
5875	28.40 s	38.45 s	59	766	10.31 s	20.22 s	2.75	1.90
22939	8.11 m	9.95 m	230	3227	2.06 m	3.83 m	3.93	2.60
35668	21.83 m	25.91 m	357	5052	5.07 m	8.96 m	4.31	2.90
51195	45.95 m	54.55 m	512	7440	10.61 m	18.86 m	4.33	2.89
90643	2.57 h	3.01 h	907	13036	34.38 m	1.01 h	4.48	2.99
141283	6.37 h	7.47 h	1413	20455	1.44 h	2.50 h	4.43	2.99
562363	4.58 d	5.37 d	5624	82535	1.20 d	2.00 d	3.83	2.69

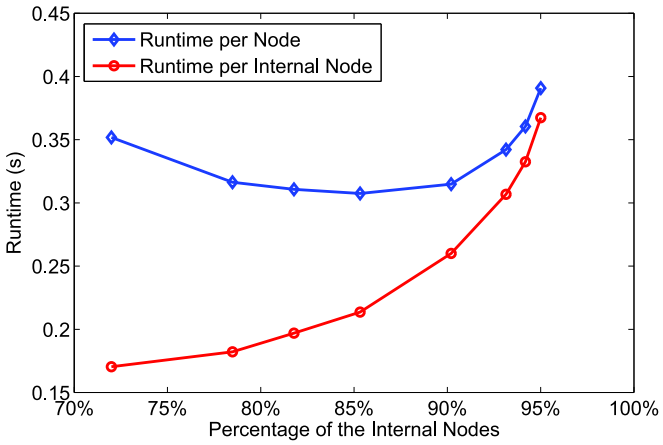

Fig. 5. Average Runtime v.s. Percentage of Internal Nodes for the Largest Power Grid.

into two parts: the runtime to compute $\mathbf{c}_l$ and the runtime to solve the LP problem Eq.(4). For the HierarchicalVD algorithm, the former consists of power grid partitioning runtime, PCG runtime, and the runtime to compute the $\mathbf{c}_l$ of the internal nodes using Eq.(16). For the DualVD algorithm, the runtime to compute $\mathbf{c}_l$ is just the PCG runtime. Both algorithms have approximately the same runtime for solving LP problems as they share the same code.

Since we focus on the efficiency of the hierarchical matrix inversion algorithm, we report the runtime for computing $\mathbf{c}_l$ under columns "Inverse". Moreover, we also report the total runtime under the columns "Total". The time units are abbreviated such that "s", "m", "h" and "d" denote "seconds", "minutes","hours", and "days" respectively. As the DualVD algorithm takes too much time to verify the power grid of 562K non-VDD nodes, the reported runtime of the DualVD algorithm for that power grid is an estimation from the runtime of 1000 nodes chosen randomly. It can be seen that the HierarchicalVD algorithm achieves close to $3\times$ speed-ups for all large power grids in comparison to the DualVD algorithm, which is highly non-trivial since DualVD is already very efficient.

Moreover, for the HierarchicalVD algorithm, we report the number of partitions under the column "Partitions" and the number of external neighbor nodes under the column "External Neighbors". It confirms the effectiveness of the proposed power grid partitioning approach since only about 15% of all the nodes are identified as external neighbors with a small rps setting of 100.

VI. CONCLUSION

In this paper, we proposed a hierarchical matrix inversion algorithm for vectorless power grid verification under linear current constraints. This hierarchical approach partitioned the nodes in the power grid into a set of external neighbors and a number of partitions. For each partition, the rows in the inverse of the power grid matrix corresponding to its external neighbors were solved by the PCG method at first. Then the rows corresponding to its internal nodes were computed directly using the solved rows. Our overall vectorless power grid verification algorithm HierarchicalVD combined the proposed hierarchical matrix inversion algorithm and the dual approach in [4]. It was shown that the proposed hierarchical matrix inversion algorithm achieved substantial speed-ups to compute the inverse of the power grid matrix, and our HierarchicalVD algorithm was highly efficient in comparison to the previous DualVD algorithm [4]. Essentially, our hierarchical algorithm can be applied to compute the inverse of general symmetric M-matrix, and we also expect to achieve further speed-ups by extending our algorithm to multi-level hierarchies.

REFERENCES

[1] D. Kouroussis and F. N. Najm, "A static pattern-independent technique for power grid voltage integrity verification," in *Proc. Design Automation Conf. (DAC)*, 2003, pp. 99–104.

[2] I. A. Ferzli, F. N. Najm, and L. Kruse, "A geometric approach for early power grid verification using current constraints," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 2007, pp. 40–47.

[3] N. H. Abdul Ghani and F. N. Najm, "Fast vectorless power grid verification using an approximate inverse technique," in *Proc. Design Automation Conf. (DAC)*, 2009, pp. 184–189.

[4] X. Xiong and J. Wang, "An efficient dual algorithm for vectorless power grid verification under linear current constraints," in *Proc. Design Automation Conf. (DAC)*, 2010, pp. 837–842.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, 1996.

[6] T.-H. Chen and C. C.-P. Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods," in *Proc. Design Automation Conf. (DAC)*, 2001, pp. 559–562.

[7] L. Grasedyck and W. Hackbusch, "Construction and arithmetics of H-matrices," *Computing*, vol. 70, no. 4, pp. 295–334, Aug. 2003.

[8] M. Zhao, R. V. Panda, S. S. Sapatnekar, and D. Blaauw, "Hierarchical analysis of power distribution networks," *IEEE Trans. Computer-Aided Design*, vol. 21, no. 2, pp. 159–168, Feb. 2002.

[9] J. E. Kelley, "The cutting-plane method for solving convex programs," *J. Soc. Indust. and Appl. Math.*, vol. 8, no. 4, pp. 703–712, Dec. 1960.

[10] H. Qian and S. S. Sapatnekar, "Stochastic preconditioning for diagonally dominant matrices," *SIAM Journal on Scientific Computing*, vol. 30, no. 3, pp. 1178–1204, Mar. 2008.

[11] H. Qian, S. R. Nassif, and S. S. Sapatnekar, "Power grid analysis using random walks," *IEEE Trans. Computer-Aided Design*, vol. 24, no. 8, pp. 1204–1224, Aug. 2005.

[12] The hMETIS Tool, http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview.

[13] The MOSEK Optimization Software, http://www.mosek.com.

[14] The SuperLU Tool, http://crd.lbl.gov/~xiaoye/SuperLU.