Efficient Transient Analysis of Power Delivery Network With Clock/Power Gating by Sparse Approximation

Hengliang Zhu, *Member, IEEE*, Yuanzhe Wang, Frank Liu, *Senior Member, IEEE*, Xin Li, *Senior Member, IEEE*, Xuan Zeng, *Member, IEEE*, and Peter Feldmann, *Fellow, IEEE*

***Abstract*—Transient analysis of large-scale power delivery network (PDN) is a critical task to ensure the functional correctness and desired performance of today's integrated circuits (ICs), especially if significant transient noises are induced by clock and/or power gating due to the utilization of extensive power management. In this paper, we propose an efficient algorithm for PDN transient analysis based on sparse approximation. The key idea is to exploit the fact that the transient response caused by clock/power gating is often localized and the voltages at many other "inactive" nodes are almost unchanged, thereby rendering a unique sparse structure. By taking advantage of the underlying sparsity of the solution structure, a modified conjugate gradient algorithm is developed and tuned to efficiently solve the PDN analysis problem with low computational cost. Our numerical experiments based on standard benchmarks demonstrate that the proposed transient analysis with sparse approximation offers up to 2.2× runtime speedup over other traditional methods, while simultaneously achieving similar accuracy.**

***Index Terms*—Conjugate gradient (CG), orthogonal matching pursuit (OMP), power delivery network (PDN), sparse approximation, transient analysis.**

I. Introduction

ON-CHIP power delivery network (PDN) is a complex 3-D interconnect circuit that connects off-chip power sources with on-chip circuit blocks [1]–[3]. When a large amount of currents flow through the PDN, large voltage fluctuations due to IR drops and/or *di/dt* noises are often observed. These noises posed by PDN significantly impact the functionality and performance of integrated circuits (ICs). For this reason, appropriate design and verification of on-chip PDNs is an extremely important task for today's system-on-chip.

During the past decade, a large number of computer-aided design (CAD) algorithms have been developed for efficient PDN analysis [4]–[20]. These existing techniques can be classified into several broad categories: 1) Krylov subspace method [9], [10]; 2) hierarchical analysis [11]; 3) multigrid solver [12]–[16]; 4) randomized algorithm [17]; and 5) vectorless analysis [18]–[20]. Most of them focus on DC analysis in order to predict the IR drops which play an important role in both signal integrity (e.g., increased gate delay) and circuit reliability (e.g., electromigration for metal wires). For transient analysis of PDNs, the existing approaches can be further categorized as: 1) implicit numerical integration (e.g., back Euler method, trapezoidal method, two-step backward differentiation method, etc.) and 2) explicit numerical integration (e.g., matrix exponential method [10]). These algorithms have been successfully applied to a large number of practical PDN problems.

While PDN analysis has been extensively studied in the literature, the aggressive scaling of IC technologies brings several recent trends that suggest a need to revisit this topic.

1) *Clock and/or Power Gating:* In order to achieve low-power operation for today's digital ICs, clock and/or power gating techniques are commonly used to dynamically turn on/off one or more functional blocks on the same chip [21]–[23]. When these blocks are simultaneously switched on/off, large transient noises (e.g., overshooting) are often observed in the PDN due to its underdamped nature. In this case, transient analysis over a long time period, instead of DC analysis, is required to accurately capture these transient noises that die out slowly.
2) *Increased PDN Size:* With technology scaling, the complexity of PDN continuously increases. The PDN of a state-of-the-art high-performance microprocessor consists of hundreds of millions of internal nodes. Hence, PDN analysis must be scalable to extremely large problem size to meet today's design reality.

The combination of the aforementioned two trends makes most traditional techniques ill-equipped to perform large-scale transient analysis for PDNs. On one hand, direct linear solvers (e.g., LU decomposition, Cholesky decomposition, etc.) can be extremely efficient for transient analysis of small- and

Manuscript received June 27, 2014; revised October 5, 2014; accepted December 18, 2014. Date of publication January 23, 2015; date of current version February 17, 2015. This work was supported in part by the National Natural Science Foundation of China Research Project under Grant 91330201, Grant 61106032, Grant 61125401, Grant 61376040, and Grant 61228401, in part by the National Basic Research Program of China under Grant 2011CB309701, in part by the National Major Science and Technology Special Project of China under Grant 2014ZX02301002-002, and in part by the Shanghai Science and Technology Committee Project under 13XD1401100. This paper was recommended by Associate Editor A. Demir.

H. Zhu and X. Zeng are with the State Key Laboratory of ASIC & System, Microelectronics Department, Fudan University, Shanghai 201203, China.

Y. Wang and X. Li are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15123 USA (e-mail: xinli@cmu.edu).

F. Liu is with the IBM Austin Research Laboratory, Austin, TX 78758 USA.

P. Feldmann is with the D. E. Shaw Research, New York, NY 10036 USA.

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

Digital Object Identifier 10.1109/TCAD.2015.2391256

medium-scale PDNs, especially if a constant time step is used and, hence, only a single matrix factorization is required [10]. However, these direct linear solvers are not applicable to large-scale PDN analysis problems due to their high computational complexity and memory usage. On the other hand, iterative linear solvers (e.g., multigrid method) are not efficient either, if they must be repeatedly applied to solve large-scale linear equations at a large number of time points during transient analysis. The fundamental question here is how to make transient analysis computationally feasible for large-scale PDNs with consideration of clock/power gating.

In this paper, we propose a novel algorithm for PDN transient analysis based on sparse approximation. We particularly focus on the challenging problem of PDN analysis where direct linear solvers are not computationally feasible due to large circuit size. Our key idea is to exploit the fact that when clock/power gating is applied, only a portion of the entire chip is switched on/off. Hence, the time-domain response of the PDN is localized around the functional blocks that are activated. In other words, the voltages at other nodes of the PDN are almost unchanged. This fact, in turn, renders a unique sparse structure where the voltage difference between two successive time points is almost zero for a large number of nodes inside the PDN. Instead of solving a general solution for transient analysis, the proposed sparse approximation is particularly developed to solve the "sparse" response with low computational cost. Similar ideas based on PDN locality have also been explored in [6] and [7].

An important contribution of this paper is to develop a new modified conjugate gradient (MCG) algorithm to efficiently find the sparse solutions of the large-scale linear equations incurred by PDN transient analysis. Similar to the traditional conjugate gradient (CG) algorithm [24], MCG iteratively selects a set of search directions and then finds the approximated solution within the linear subspace spanned by these search directions. Different from the traditional CG method that explicitly forms the conjugate search directions, MCG directly calculates the approximated solution at each iteration step by solving a small-size linear equation, without explicitly making the search directions mutually conjugate. As such, the computational cost of MCG can be substantially reduced. As will be demonstrated by the numerical examples in Section VI, our proposed sparse approximation based on MCG offers up to 2.2× runtime speedup over other traditional methods, while simultaneously achieving similar accuracy.

The remainder of this paper is organized as follows. In Section II, we briefly review the important background on PDN analysis, and then derive our sparse approximation method in Section III. A novel MCG algorithm is developed in Section IV to efficiently solve the large-scale linear equations for PDN transient analysis. Several implementation details are further discussed in Section V. The efficacy of our proposed technique is demonstrated by a number of numerical examples in Section VI. Finally, we conclude in Section VII.

II. Background

A. Transient Analysis of PDN

A PDN is typically modeled as an RLC network with ideal voltage and current sources. Without loss of generality, we mathematically represent a PDN by the following modified nodal analysis (MNA) equation [25]:

$$\begin{bmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{0} & \mathbf{L} \end{bmatrix} \cdot \begin{bmatrix} \dot{\mathbf{v}}(t) \\ \dot{\mathbf{i}}(t) \end{bmatrix} + \begin{bmatrix} \mathbf{G} & \mathbf{A}_L^T \\ -\mathbf{A}_L & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}(t) \\ \mathbf{i}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{i}_S(t) \\ \mathbf{v}_S(t) \end{bmatrix} \quad (1)$$

where $\mathbf{v}(t) \in \Re^M$ is a vector containing all node voltages, $\mathbf{i}(t) \in \Re^N$ is a vector containing all branch currents associated with inductors, $\mathbf{i}_S(t) \in \Re^M$ is a vector containing all input current sources, $\mathbf{v}_S(t) \in \Re^N$ is a vector containing all input voltage sources, and $\mathbf{C} \in \Re^{M\times M}$, $\mathbf{L} \in \Re^{N\times N}$, $\mathbf{G} \in \Re^{M\times M}$ and $\mathbf{A}_L \in \Re^{N\times M}$ are the system matrices. In our MNA formulation, we assume that all input voltage sources are grounded. Namely, one node of the voltage source is connected to ground. In this case, the voltage of the other node is known and, hence, it is no longer considered as a problem unknown. Instead, its value is directly added to the right-hand-side (RHS) vector in (1) [12].

To numerically solve the differential algebraic equation (DAE) in (1), a numerical integration method must be applied to approximate the differential operator d/dt. In this paper, we use the two-step backward differentiation formula (BDF2) [26] to solve the DAEs related to the PDN problem. However, it should be noted that the proposed idea of sparse approximation is also applicable to other multistep numerical integration methods.

When a multistep numerical integration method is applied, the derivative at the current time step is approximated by a polynomial of the current and past solutions. By doing so, the DAE can be approximated as a system of algebraic equations. In what follows, we use the BDF2 formula with constant time step to derive our proposed method. We will discuss how to handle variable time steps later in the paper.

When BDF2 is applied, the voltages and currents at the (n+1)th time point are expressed as

$$\mathbf{v}(t_{n+1}) = \frac{4}{3}\mathbf{v}(t_n) - \frac{1}{3}\mathbf{v}(t_{n-1}) + \frac{2h_n}{3}\dot{\mathbf{v}}(t_{n+1}) \quad (2)$$

$$\mathbf{i}(t_{n+1}) = \frac{4}{3}\mathbf{i}(t_n) - \frac{1}{3}\mathbf{i}(t_{n-1}) + \frac{2h_n}{3}\dot{\mathbf{i}}(t_{n+1}) \quad (3)$$

where h_n, $\mathbf{v}(t_n)$, and $\mathbf{i}(t_n)$ represent the time step, the node voltages, and the branch currents at the nth time point respectively. Substituting (2) and (3) into (1) yields

$$\left(\frac{3}{2h_n}\mathbf{C} + \mathbf{G} + \frac{2h_n}{3}\mathbf{A}_L^T\mathbf{L}^{-1}\mathbf{A}_L\right) \cdot \mathbf{v}(t_{n+1}) = \frac{2}{h_n}\mathbf{C}\mathbf{v}(t_n) - \frac{1}{2h_n}\mathbf{C}\mathbf{v}(t_{n-1}) - \frac{4}{3}\mathbf{A}_L^T\mathbf{i}(t_n) + \frac{1}{3}\mathbf{A}_L^T\mathbf{i}(t_{n-1}) + \mathbf{i}_S(t_{n+1}) - \frac{2}{3}h_n\mathbf{A}_L^T\mathbf{L}^{-1}\mathbf{v}_S(t_{n+1}) \quad (4)$$

$$\mathbf{i}(t_{n+1}) = \frac{2h_n}{3}\mathbf{L}^{-1}\mathbf{A}_L\mathbf{v}(t_{n+1}) + \frac{4}{3}\mathbf{i}(t_n) - \frac{1}{3}\mathbf{i}(t_{n-1}) + \frac{2h_n}{3}\mathbf{L}^{-1}\mathbf{v}_S(t_{n+1}). \quad (5)$$

It can be shown that the matrix $3/2h_n \cdot \mathbf{C} + \mathbf{G} + 2h_n/3 \cdot \mathbf{A}_L^T \mathbf{L}^{-1} \mathbf{A}_L$ in (4) is symmetric and positive definite [9]. In most practical applications, in order to control local truncation error (LTE), the time step h_n is adaptively adjusted and the polynomial coefficients in (2) and (3) must be adjusted accordingly [25].

To calculate the transient response at the $(n+1)$th time point, we need to first solve the linear equation (4) and find the voltage value $\mathbf{v}(t_{n+1})$. Next, the current value $\mathbf{i}(t_{n+1})$ can be calculated by substituting $\mathbf{v}(t_{n+1})$ into (5). Note that the matrix $\mathbf{L}$ in (5) is diagonal, if there is no mutual inductance. Hence, the inverse of $\mathbf{L}$ can be easily calculated. In this case, computing (5) only involves several matrix-vector multiplications and is computationally inexpensive.

Even if mutual inductance does exist, the inverse matrix $\mathbf{L}^{-1}$, instead of the inductance matrix $\mathbf{L}$, is often directly extracted by an extraction tool in order to achieve good locality and sparsity [27]. In this case, since the inverse matrix $\mathbf{L}^{-1}$ is directly provided by the extraction tool, we can again calculate (5) with low computational cost and the overall runtime is dominated by the linear equation solver for (4).

There are many techniques to solve the linear equation (4). In what follows, we will briefly review two important linear solvers: 1) the CG solver [24] and 2) the sparse solver [28]. These two linear solvers are closely related to our proposed technique of sparse approximation. The mathematical concepts behind these two solvers will be further used to derive our proposed MCG algorithm in Section IV.

B. CG Solver

CG is an iterative method to efficiently solve the following linear equation:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{6}$$

where the matrix $\mathbf{A} \in \Re^{M \times M}$ is a symmetric, positive definite matrix, $\mathbf{b} \in \Re^M$ is the RHS vector, and $\mathbf{x} \in \Re^M$ is the unknown vector that needs to be solved. The linear equation (4) posed by PDN transient analysis can be easily mapped to the general form in (6). The key idea of CG is to convert the linear equation (6) to an equivalent optimization problem

$$\min_x \quad \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} - \mathbf{b}^T \mathbf{x}. \tag{7}$$

It is easy to verify that the solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ of (6) satisfies the first-order optimality condition [29]

$$\frac{\partial}{\partial \mathbf{x}} \left(\frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} - \mathbf{b}^T \mathbf{x} \right) = 0. \tag{8}$$

Hence, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ is also the optimal solution of (7).

To solve the optimization problem in (7), CG starts from an initial solution $\mathbf{x}_{(0)}$ and iteratively generates a set of search directions $\{\mathbf{d}_{(k)} \in \Re^M;\ k = 0, 1, 2, \ldots\}$ that are mutually conjugate

$$\mathbf{d}_{(i)}^T \mathbf{A}\mathbf{d}_{(j)} = 0 (i \neq j). \tag{9}$$

At the kth iteration step, CG first calculates the residue corresponding to the current solution $\mathbf{x}_{(k)}$

$$\mathbf{r}_{(k)} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x}_{(k)}. \tag{10}$$

Algorithm 1 CG Solver

1: Start from a given linear equation (6) where the matrix $\mathbf{A}$ is symmetric and positive definite.
2: Choose an initial solution $\mathbf{x}_{(0)}$ and set $k = 0$.
3: Calculate the initial search direction $\mathbf{d}_{(0)} = \mathbf{r}_{(0)} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x}_{(0)}$.
4: Update the solution:

$$\mathbf{x}_{(k+1)} = \mathbf{x}_{(k)} + \mu_{(k)} \mathbf{d}_{(k)}. \tag{13}$$

where

$$\mu_{(k)} = \frac{\mathbf{d}_{(k)}^T \mathbf{r}_{(k)}}{\mathbf{d}_{(k)}^T \mathbf{A}\mathbf{d}_{(k)}}. \tag{14}$$

5: Update the residue:

$$\mathbf{r}_{(k+1)} = \mathbf{r}_{(k)} - \mu_{(k)} \mathbf{A}\mathbf{d}_{(k)}. \tag{15}$$

6: Update the search direction:

$$\mathbf{d}_{(k+1)} = \mathbf{r}_{(k+1)} + \beta_k \mathbf{d}_{(k)}, \tag{16}$$

where

$$\beta_k = \frac{\mathbf{r}_{(k+1)}^T \mathbf{r}_{(k+1)}}{\mathbf{d}_{(k)}^T \mathbf{r}_{(k)}}. \tag{17}$$

7: Set $k = k + 1$ and repeat Step 4–6 until the residue is sufficiently small.

It then determines the search direction $\mathbf{d}_{(k)}$ by implicitly forming the following linear subspace based on an iterative algorithm:

$$\Omega_{(k)} = \text{span}\left\{\mathbf{d}_{(0)}, \mathbf{d}_{(1)}, \ldots, \mathbf{d}_{(k)}\right\} = \text{span}\left\{\mathbf{r}_{(0)}, \mathbf{r}_{(1)}, \ldots, \mathbf{r}_{(k)}\right\}. \tag{11}$$

Next, CG searches the new solution $\mathbf{x}_{(k+1)}$ such that $\mathbf{x}_{(k+1)} - \mathbf{x}_{(0)}$ is within the linear subspace $\Omega_{(k)}$

$$\begin{aligned} \min_{\mathbf{x}_{(k+1)}} \quad & \frac{1}{2}\mathbf{x}_{(k+1)}^T \mathbf{A}\mathbf{x}_{(k+1)} - \mathbf{b}^T \mathbf{x}_{(k+1)} \\ \text{S.T.} \quad & \mathbf{x}_{(k+1)} - \mathbf{x}_{(0)} \in \Omega_{(k)}. \end{aligned} \tag{12}$$

Algorithm 1 summarizes the simplified flow of the CG algorithm.

The convergence rate of CG depends on the condition number of the matrix $\mathbf{A}$ [24]. If the matrix $\mathbf{A}$ is well-conditioned, CG converges quickly. For an ill-conditioned matrix, CG may take a large number of iteration steps to converge, thereby resulting in expensive computational time. To address this issue, various preconditioning techniques have been proposed to further improve the convergence rate of CG [12].

C. Sparse Solver

A sparse solver has been proposed in [28] for incremental DC analysis of PDNs. It exploits the fact that when a PDN is locally updated, its response also changes locally. More specifically, the incremental "changes" are almost zero for most

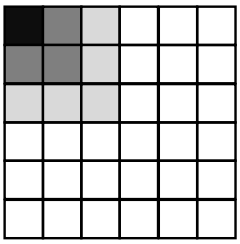


Fig. 1. Simple example illustrates the concept of localized transient response associated with clock/power gating. Only a small portion (black) of the chip is activated and the time-domain response of the PDN is localized within the black and gray regions.

node voltages and branch currents. In this case, the incremental analysis problem for PDN can be formulated in the general form of linear equation (6) where the unknown vector $\mathbf{x}$ is sparse. Hence, instead of solving a general solution $\mathbf{x}$ from (6), we only need to decide the locations and values of a few nonzeros.

Toward this goal, orthogonal matching pursuit (OMP) is applied to solve the sparse solution $\mathbf{x}$ from the linear equation $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$ [28]. OMP rewrites the RHS vector $\mathbf{b}$ as the linear combination of all column vectors of the matrix $\mathbf{A}$

$$\mathbf{b} = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \cdots + x_M\mathbf{a}_M \tag{18}$$

where x_m denotes the mth element of the vector $\mathbf{x}$ and $\mathbf{a}_m$ stands for the mth column vector of the matrix $\mathbf{A}$. Next, the "importance" of each column vector $\mathbf{a}_m$ is quantitatively assessed by the normalized inner product

$$\rho(\mathbf{a}_m, \mathbf{b}) = \left| \frac{\mathbf{a}_m^T\mathbf{b}}{\mathbf{a}_m^T\mathbf{a}_m} \right|. \tag{19}$$

Intuitively, if $\rho(\mathbf{a}_m, \mathbf{b})$ is large, the column vector $\mathbf{a}_m$ plays an important role to represent the RHS vector $\mathbf{b}$. Hence, the corresponding element x_m should be nonzero. OMP iteratively calculates the normalized inner products to select the nonzero elements from the vector $\mathbf{x}$ and then decide their values by finding the least-squares solution of an over-determined linear equation. Algorithm 2 summarizes the major steps of OMP.

Algorithm 2 OMP

1: Start from a given linear equation (6) and rewrite it as (18).
2: Initialize the residue $\mathbf{r}_{(0)} = \mathbf{b}$ and set $k = 0$.
3: Calculate the normalized inner product $\rho(\mathbf{a}_m, \mathbf{r}_{(k)})$ between each column vector $\mathbf{a}_m$ and the residue $\mathbf{r}_{(k)}$:

$$\rho\big(\mathbf{a}_m, \mathbf{r}_{(k)}\big) = \left| \frac{\mathbf{a}_m^T\mathbf{r}_{(k)}}{\mathbf{a}_m^T\mathbf{a}_m} \right|. \tag{20}$$

4: Select the column vectors $\{\mathbf{a}_{m1}, \mathbf{a}_{m2}, \dots\}$ such that:

$$\rho\big(\mathbf{a}_{m1}, \mathbf{r}_{(k)}\big) \geq \varepsilon \ \rho\big(\mathbf{a}_{m2}, \mathbf{r}_{(k)}\big) \geq \varepsilon \cdots, \tag{21}$$

where ϵ is a user-defined threshold.
5: Determine the index set:

$$\mathbf{\Theta}_{(k)} = \begin{cases} \{m_1, m_2, \cdots\} & (k = 0) \\ \mathbf{\Theta}_{(k-1)} \cup \{m_1, m_2, \cdots\} & (k \geq 1) \end{cases}. \tag{22}$$

6: Solve the unknowns $\{x_{(k+1),m};\ m \in \mathbf{\Theta}_{(k)}\}$ by:

$$\min_{x_{(k+1),m}; m\in\mathbf{\Theta}_{(k)}} \left\| \sum_{m\in\mathbf{\Theta}_{(k)}} x_{(k+1),m} \cdot a_m - b \right\|_2^2. \tag{23}$$

7: For any $m \notin \mathbf{\Theta}_{(k)}$, the value of $x_{(k+1),m}$ is set to 0.
8: Update the residue:

$$\mathbf{r}_{(k+1)} = \mathbf{b} - \mathbf{A}\cdot\mathbf{x}_{(k+1)}. \tag{24}$$

9: Set $k = k + 1$ and repeat Step 3–8 until the residue is sufficiently small.

III. Transient Analysis by Sparse Approximation

In this paper, we further extend the idea of sparse approximation from DC analysis to transient analysis. Our proposed approach is motivated by the fact that when clock/power gating is applied, only a small portion of the entire chip is activated. In this case, the transient response of the PDN is localized in a small region. Namely, most node voltages and branch currents are almost constant between two consecutive time points. Fig. 1 shows a simple PDN example that conceptually illustrates the aforementioned concept of localized transient response.

Based on this observation, if we formulate the PDN equation with respect to the incremental changes between two successive time points, the unknown solution vector containing the incremental voltage changes is sparse. To derive such an incremental formulation, we subtract $(3/2h_n\cdot\mathbf{C} + \mathbf{G} + 2h_n/3\cdot\mathbf{A}_L^T\mathbf{L}^{-1}\mathbf{A}_L)\cdot\mathbf{v}(t_n)$ from both sides of (4), yielding

$$\begin{aligned}\left(\frac{3}{2h_n}\mathbf{C} + \mathbf{G} + \frac{2h_n}{3}\mathbf{A}_L^T\mathbf{L}^{-1}\mathbf{A}_L\right)\cdot\boldsymbol{\delta}\,(t_{n+1}) &= \frac{2}{h_n}\mathbf{C}\mathbf{v}(t_n) \\ -\frac{1}{2h_n}\mathbf{C}\mathbf{v}(t_{n-1}) - \frac{4}{3}\mathbf{A}_L^T\mathbf{i}(t_n) + \frac{1}{3}\mathbf{A}_L^T\mathbf{i}(t_{n-1}) + \mathbf{i}_S(t_{n+1}) \\ -\frac{2}{3}h_n\mathbf{A}_L^T\mathbf{L}^{-1}\mathbf{v}_S(t_{n+1}) \\ -\left(\frac{3}{2h_n}\mathbf{C} + \mathbf{G} + \frac{2h_n}{3}\mathbf{A}_L^T\mathbf{L}^{-1}\mathbf{A}_L\right)\cdot\mathbf{v}(t_n)\end{aligned} \tag{25}$$

where

$$\boldsymbol{\delta}(t_{n+1}) = \mathbf{v}(t_{n+1}) - \mathbf{v}(t_n). \tag{26}$$

Note that the size of the unknown vector $\boldsymbol{\delta}(t_{n+1}) \in \Re^M$ (i.e., the total number of nodes of the PDN) is extremely large, since today's PDNs often contain tens to hundreds of millions of nodes. However, since the vector $\boldsymbol{\delta}(t_{n+1})$ is sparse with a few nonzeros, we only need to determine the locations and values of these nonzeros, instead of solving the large-scale linear equation by a general-purpose solver.

One possible approach to solve the sparse solution $\boldsymbol{\delta}(t_{n+1})$ from (25) is to directly apply the OMP algorithm (i.e., Algorithm 2) where an over-determined linear equation (23) must be solved at each iteration step. There are two well-known methods to find the least-squares solution of (23) [28]: 1) QR decomposition and 2) pseudo-inverse. Each

method has its own advantages and limitations. QR decomposition is numerically stable, but computationally expensive. On the other hand, pseudo-inverse is computationally inexpensive, but often suffers from numerical issues. This lack of clear choice motivates us to develop a highly efficient-yet-stable algorithm, referred to as the MCG solver, to find the sparse solution $\boldsymbol{\delta}(t_{n+1})$ of (25) without solving any over-determined linear equation. The details of MCG will be discussed in the next section.

IV. MCG Solver

Our proposed MCG solver is derived from the traditional CG algorithm with several important modifications. First, MCG uses a different criterion to form the search directions. Second, it applies a different scheme to update the solution by searching the linear subspace spanned by the search directions. In this section, we derive the mathematical formulation of MCG in detail.

A. Mathematical Formulation

Without loss of generality, we consider the general representation of a linear equation $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$ in (6). Our objective is to efficiently compute the sparse solution $\mathbf{x}$. The incremental formulation (25) can be easily mapped to this general form $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$ by redefining the symbols

$$\mathbf{A} = \frac{3}{2h_n}\mathbf{C} + \mathbf{G} + \frac{2h_n}{3}\mathbf{A}_L^T\mathbf{L}^{-1}\mathbf{A}_L \tag{27}$$

$$\mathbf{x} = \boldsymbol{\delta}\,(t_{n+1}) \tag{28}$$

$$\begin{aligned}\mathbf{b} = {} & \frac{2}{h_n}\mathbf{C}\mathbf{v}(t_n) - \frac{1}{2h_n}\mathbf{C}\mathbf{v}(t_{n-1}) - \frac{4}{3}\mathbf{A}_L^T\mathbf{i}(t_n) \\ & + \frac{1}{3}\mathbf{A}_L^T\mathbf{i}(t_{n-1}) + \mathbf{i}_S(t_{n+1}) - \frac{2}{3}h_n\mathbf{A}_L^T\mathbf{L}^{-1}\mathbf{v}_S(t_{n+1}) \\ & - \left(\frac{3}{2h_n}\mathbf{C} + \mathbf{G} + \frac{2h_n}{3}\mathbf{A}_L^T\mathbf{L}^{-1}\mathbf{A}_L\right)\cdot\mathbf{v}(t_n).\end{aligned} \tag{29}$$

Note that the matrix $\mathbf{A}$ in (27) is symmetric and positive definite.

Since the unknown vector $\mathbf{x}$ contains a few nonzeros only, we propose to form the search directions based on the locations of these nonzeros, instead of the residues used by the traditional CG method (i.e., Algorithm 1). We use several heuristic criteria to quantitatively assess the importance of each column vector $\mathbf{a}_m$ of the matrix $\mathbf{A}$, when calculating the incremental response $\boldsymbol{\delta}(t_{n+1})$ at the $(n+1)$th time point. As such, the corresponding locations of the nonzeros in the unknown vector $\boldsymbol{\delta}(t_{n+1})$ can be quickly identified.

In particular, we develop two heuristic techniques. First, the mth node is considered to be "active" and the corresponding incremental response $\delta_m(t_{n+1})$ [i.e., the mth element of the unknown vector $\boldsymbol{\delta}(t_{n+1})$] is expected to be nonzero, if at least one of the following two conditions is satisfied.

1) The voltage of the mth node significantly varies at the *previous* time point [i.e., the value of $|\delta_m(t_n)|$ is large].
2) The mth node is connected to an input current or voltage source (say, the ith current source i_{Si} or voltage source v_{Si}) that significantly varies at the *current* time point [i.e., the value of $|i_{Si}(t_{n+1}) - i_{Si}(t_n)|$ or $|v_{Si}(t_{n+1}) - v_{Si}(t_n)|$ is large].

Note that the values of $\delta_m(t_n)$, $i_{Si}(t_{n+1})$, $i_{Si}(t_n)$, $v_{Si}(t_{n+1})$, and $v_{Si}(t_n)$ are all known at the $(n+1)$th time point, when calculating the incremental response $\boldsymbol{\delta}(t_{n+1})$. Hence, these two conditions can be easily checked. The indices of these identified nodes are used to form an initial index set $\boldsymbol{\Theta}_{(0)}$.

Second, in addition to the nodes that belong to the initial set $\boldsymbol{\Theta}_{(0)}$, a number of other nodes may also become active at the $(n+1)$th time point, but are not captured by the aforementioned heuristics. For this reason, we further borrow the idea of normalized inner product in (20) to iteratively identify these active nodes, similar to the OMP algorithm (i.e., Algorithm 2). At the kth iteration step of the proposed MCG solver, the indices of a set of activenodes (say, $\{m_1, m_2, \ldots\}$) are selected according to the normalized inner products and these indices are cumulatively added to the index set $\boldsymbol{\Theta}_{(k)}$

$$\boldsymbol{\Theta}_{(k)} = \boldsymbol{\Theta}_{(k-1)} \cup \{m_1, m_2, \ldots\}. \tag{30}$$

Once the set $\boldsymbol{\Theta}_{(k)}$ is formed, we define the following search directions:

$$\left\{\mathbf{e}_m;\, m \in \boldsymbol{\Theta}_{(k)}\right\} \tag{31}$$

where $\mathbf{e}_m \in \Re^M$ is a vector for which the mth element is one and all other elements are zero. When the vector $\mathbf{e}_m$ is selected as a search direction, it implies that the mth element of the solution $\mathbf{x}$ of $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$, where $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{b}$ are defined in (27)–(29), should be nonzero.

After the search directions are determined at the kth iteration step, we form the following linear subspace:

$$\Omega_{(k)} = \text{span}\left\{\mathbf{e}_m;\, m \in \boldsymbol{\Theta}_{(k)}\right\}. \tag{32}$$

If the solution $\mathbf{x}$ is sparse and contains a few nonzeros only, it belongs to a low-dimensional linear subspace. In this case, MCG is able to accurately approximate the solution $\mathbf{x}$ within very few iteration steps. In other words, by exploiting the sparsity of the solution $\mathbf{x}$, MCG can converge more quickly than the traditional CG method.

Given the linear subspace $\Omega_{(k)}$ at the kth iteration step of the MCG solver, we need to further determine the new solution $\mathbf{x}_{(k+1)}$ by solving the following optimization problem:

$$\min_{x_{(k+1)} \in \Omega_{(k)}} \quad \frac{1}{2}\mathbf{x}_{(k+1)}^T\mathbf{A}\mathbf{x}_{(k+1)} - \mathbf{b}^T\mathbf{x}_{(k+1)}. \tag{33}$$

The optimization problem in (33) is similar to that of the traditional CG method shown in (12).

Note that the search directions in (31) are orthogonal but *not* conjugate. Unlike the traditional CG method in which the conjugate search directions can be easily calculated from the residues by (16), explicitly forming a set of conjugate search directions for the linear subspace in (32) is not trivial. While it is possible to make the vectors $\{\mathbf{e}_m;\, m \in \boldsymbol{\Theta}_{(k)}\}$ conjugate by applying Gram–Schmidt conjugation [24], such an approach can be computationally prohibitive, especially for large-scale problems. Without explicitly knowing the conjugate search directions, the new solution $\mathbf{x}_{(k+1)}$ cannot be calculated by directly following the simple equation expressed in (13).

To address this issue, we derive a new computing scheme to efficiently find the new solution $\mathbf{x}_{(k+1)}$ from the linear subspace $\Omega_{(k)}$. Our proposed approach is based upon the property of orthogonality described by the following theorem.

Theorem 1: Given the solution $\mathbf{x}_{(k+1)}$ solved by (12) where the initial value $\mathbf{x}_{(0)}$ is $\mathbf{0}$ and the linear subspace $\Omega_{(k)}$ is defined in (32), the residue $\mathbf{r}_{(k+1)} = \mathbf{b} - \mathbf{A}\cdot\mathbf{x}_{(k+1)}$ is orthogonal to all search directions $\{\mathbf{e}_m;\ m \in \mathbf{\Theta}_{(k)}\}$

$$\mathbf{e}_m^T \mathbf{r}_{(k+1)} = 0 \left(m \in \mathbf{\Theta}_{(k)}\right). \tag{34}$$

Proof: Since the solution $\mathbf{x}_{(k+1)}$ is within the linear subspace $\Omega_{(k)}$, it can be represented as the linear combination of all search directions

$$\mathbf{x}_{(k+1)} = \sum_{m \in \mathbf{\Theta}_{(k)}} x_{(k+1),m} \cdot \mathbf{e}_m. \tag{35}$$

Hence, minimizing the cost function in (12) requires us to find the values of $\{x_{(k+1),m};\ m \in \mathbf{\Theta}_{(k)}\}$ to satisfy the following first-order optimality condition [29]:

$$\frac{\partial}{\partial x_{(k+1),m}} \left[\frac{1}{2}\mathbf{x}_{(k+1)}^T \mathbf{A}\mathbf{x}_{(k+1)} - \mathbf{b}^T \mathbf{x}_{(k+1)}\right] = 0 \left(m \in \mathbf{\Theta}_{(k)}\right). \tag{36}$$

Substituting (35) into (36) yields

$$\mathbf{e}_m^T \cdot \left[\mathbf{A}\mathbf{x}_{(k+1)} - \mathbf{b}\right] = 0 \left(m \in \mathbf{\Theta}_{(k)}\right). \tag{37}$$

Since the residue $\mathbf{r}_{(k+1)}$ is equal to $\mathbf{b} - \mathbf{A}\cdot\mathbf{x}_{(k+1)}$, (37) implies the orthogonal property in (34). ■

Assume that there are $M_{(k)}$ search directions $\{\mathbf{e}_m;\ m \in \mathbf{\Theta}_{(k)}\}$, therefore $M_{(k)}$ nonzeros $\{x_{(k+1),m};\ m \in \mathbf{\Theta}_{(k)}\}$ are selected at the kth iteration step of the MCG algorithm. Namely, the cardinality (i.e., the size) of the set $\mathbf{\Theta}_{(k)}$ is $M_{(k)}$

$$\left|\mathbf{\Theta}_{(k)}\right| = M_{(k)}. \tag{38}$$

In this case, there are $M_{(k)}$ linear equations in (37) that we can use to solve the $M_{(k)}$ unknowns $\{x_{(k+1),m};\ m \in \mathbf{\Theta}_{(k)}\}$. In other words, the property of orthogonality described by Theorem 1 reveals a crucial fact that solving the optimization problem in (12) is equivalent to solving the linear equations in (37). This fact provides an alternative way to efficiently update the solution $\mathbf{x}_{(k+1)}$ at the kth iteration step.

To further derive a compact representation of the reduced linear equations in (37), we substitute (35) into (37)

$$\mathbf{e}_m^T \cdot \mathbf{A} \cdot \sum_{m \in \mathbf{\Theta}_{(k)}} x_{(k+1),m} \cdot \mathbf{e}_m = \mathbf{e}_m^T \mathbf{b} \left(m \in \mathbf{\Theta}_{(k)}\right). \tag{39}$$

Equation (39) can be further rewritten as

$$\mathbf{E}_{\mathbf{\Theta}(k)}^T \mathbf{A}\mathbf{E}_{\mathbf{\Theta}(k)} \cdot \mathbf{x}_{\mathbf{\Theta}(k+1)} = \mathbf{E}_{\mathbf{\Theta}(k)}^T \mathbf{b} \tag{40}$$

where $\mathbf{E}_{\mathbf{\Theta}(k)}$ is an M-by-$M_{(k)}$ matrix containing the column vectors $\{\mathbf{e}_m;\ m \in \mathbf{\Theta}_{(k)}\}$ and $\mathbf{x}_{\mathbf{\Theta}(k+1)}$ is an $M_{(k)}$-by-one column vector containing the unknowns $\{x_{(k+1),m};\ m \in \mathbf{\Theta}_{(k)}\}$. Remember that $\mathbf{e}_m$ is a vector for which the mth element is one and all other elements are zero. Hence, the reduced matrix ${\mathbf{E}_{\mathbf{\Theta}(k)}}^T \mathbf{A}\mathbf{E}_{\mathbf{\Theta}(k)}$ in (40) is simply generated by selecting the corresponding $M_{(k)}$ rows and columns from the matrix $\mathbf{A}$. Namely, the matrix $\mathbf{E}_{\mathbf{\Theta}(k)}^T \mathbf{A}\mathbf{E}_{\mathbf{\Theta}(k)}$ is a principal minor [30] of the matrix $\mathbf{A}$. Similarly, the reduced RHS vector $\mathbf{E}_{\mathbf{\Theta}(k)}^T \mathbf{b}$ in (40) is generated by selecting the corresponding $M_{(k)}$ elements from the vector $\mathbf{b}$. Fig. 2 shows a simple example to illustrate how the reduced matrix $\mathbf{E}_{\mathbf{\Theta}(k)}^T \mathbf{A}\mathbf{E}_{\mathbf{\Theta}(k)}$ and the reduced vector $\mathbf{E}_{\mathbf{\Theta}(k)}^T \mathbf{b}$ are constructed.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \quad \mathbf{E}_{\Theta(k)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$\mathbf{E}_{\Theta(k)}^T \cdot \mathbf{A} \cdot \mathbf{E}_{\Theta(k)} = \begin{bmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{bmatrix} \quad \mathbf{x}_{\Theta(k+1)} = \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} \quad \mathbf{E}_{\Theta(k)}^T \cdot \mathbf{b} = \begin{bmatrix} b_1 \\ b_3 \end{bmatrix}$$

Fig. 2. Simple example illustrates the "reduced" linear equation in (40), where the first and third rows/columns of the matrix $\mathbf{A}$ are selected to form the reduced matrix $\mathbf{E}_{\mathbf{\Theta}(k)}^T \mathbf{A}\mathbf{E}_{\mathbf{\Theta}(k)}$, and the first and third elements of the vector $\mathbf{b}$ are selected to form the reduced RHS vector $\mathbf{E}_{\mathbf{\Theta}(k)}^T \mathbf{b}$.

Algorithm 3 MCG Solver

1: Start from the linear equation $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$, where the symbols $\mathbf{A}$, $\mathbf{x}$ and $\mathbf{b}$ are defined by (27)–(29) in order to solve the incremental response $\delta(t_{n+1})$ at the $(n+1)$-th time point.
2: Set $k = 0$. Form the initial index set $\mathbf{\Theta}_{(0)}$ by identifying the active nodes based on two criteria: (i) the voltage of a node significantly varies at the previous time point, or (ii) a node is connected to an input current or voltage source that significantly varies at the current time point.
3: Solve the unknowns $\{x_{(k+1),m};\ m \in \mathbf{\Theta}_{(k)}\}$ from the reduced linear equation (40).
4: For any $m \notin \mathbf{\Theta}_{(k)}$, the value of $x_{(k+1),m}$ is set to 0.
5: Calculate the residue $\mathbf{r}_{(k+1)}$ by using (24).
6: Set $k = k + 1$, update the index set $\mathbf{\Theta}_{(k)}$ based on the normalized inner products by using (20)–(21) and (30), and repeat Step 3–6 until the residue is sufficiently small.

Algorithm 3 summarizes the major steps of the proposed MCG algorithm. There are several important clarifications that should be made. First, since the matrix $\mathbf{A}$ in (27) is positive definite, all its principal minors are also positive definite [30]. As a principal minor, the reduced matrix $\mathbf{E}_{\mathbf{\Theta}(k)}^T \mathbf{A}\mathbf{E}_{\mathbf{\Theta}(k)}$ is positive definite and nonsingular. Hence, the linear equation (40) is guaranteed to have a unique solution.

Second, if the solution of the linear equation $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$ is sparse, the dimension of the linear subspace (32) should be small. Hence, the linear equation (40) is substantially smaller than the original equation $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$. Its solution $\mathbf{x}_{\mathbf{\Theta}(k+1)}$ can be efficiently solved by either a direct linear solver (e.g., Cholesky decomposition) or an iterative linear solver (e.g., the CG method, the multigrid method, etc.) with low computational cost. In addition, an appropriately designed algorithm can incrementally update the solution $\mathbf{x}_{\mathbf{\Theta}(k+1)}$ during each iteration step of the proposed MCG solver, thereby further reducing the computational cost. For instance, if Cholesky decomposition is applied to solve (40), the factorization process can be incrementally updated. On the other hand, if the CG method or the multigrid method is applied to solve (40),

the solution $\mathbf{x}_{\mathbf{\Theta}(k)}$ from the previous iteration step can be used to define an initial guess for the solution $\mathbf{x}_{\mathbf{\Theta}(k+1)}$ of the current iteration step so that the iterative linear solver converges quickly. Implementing such an incremental update scheme is straightforward and, therefore, the details are not further discussed in this paper.

B. Comparison With Traditional Sparse Solver

Our proposed MCG algorithm (i.e., Algorithm 3) is similar to the traditional OMP algorithm (i.e., Algorithm 2), because both algorithms apply the same heuristics to iteratively identify the nonzeros for the unknown solution vector based on normalized inner product. However, once the nonzero locations are determined and stored in the index set $\mathbf{\Theta}_{(k)}$ at the kth iteration step, MCG and OMP rely on different algorithms to calculate the values of these nonzeros $\{x_{(k+1),m};\, m \in \mathbf{\Theta}_{(k)}\}$. As shown in (23), OMP aims to find the least-squares solution by minimizing the total squared error. The optimization problem in (23) can be rewritten as

$$\min_{x_{(k+1)} \in \Omega_{(k)}} \quad \frac{1}{2}\mathbf{x}_{(k+1)}^T \mathbf{A}^T \mathbf{A}\mathbf{x}_{(k+1)} - \mathbf{b}^T \mathbf{A}\mathbf{x}_{(k+1)} \tag{41}$$

where $\Omega_{(k)}$ denotes the linear subspace defined in (32). On the other hand, MCG formulates a different optimization (33) to find the solution $\mathbf{x}_{(k+1)}$.

There are two consequences from the differences between these two optimization formulations. First, the solutions from (33) and (41) are different if the solution of $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$ does not exactly lie in the subspace $\Omega_{(k)}$. As proven by Theorem 1, the MCG solution of (33) leads to a residue $\mathbf{r}_{(k+1)}$ that is orthogonal to all search directions $\{\mathbf{e}_m;\, m \in \mathbf{\Theta}_{(k)}\}$. On the other hand, based on the theory of least-squares fitting [30], the OMP residue $\mathbf{r}_{(k+1)}$ from (41) should be orthogonal to the column vectors $\{\mathbf{a}_m;\, m \in \mathbf{\Theta}_{(k)}\}$

$$\mathbf{a}_m^T \mathbf{r}_{(k+1)} = 0 \left(m \in \mathbf{\Theta}_{(k)}\right). \tag{42}$$

Recall that $\mathbf{e}_m$ is a vector for which the mth element is one and all other elements are zero. Hence, the following equality holds:

$$\mathbf{A}\mathbf{e}_m = \mathbf{a}_m \,(m = 1, 2, \ldots, M). \tag{43}$$

Substituting (43) into (42) yields

$$\mathbf{e}_m^T \mathbf{A}\mathbf{r}_{(k+1)} = 0 \left(m \in \mathbf{\Theta}_{(k)}\right). \tag{44}$$

It implies that the OMP residue $\mathbf{r}_{(k+1)}$ and all search directions $\{\mathbf{e}_m;\, m \in \mathbf{\Theta}_{(k)}\}$ are conjugate but not orthogonal.

Second and more importantly, the computational costs of solving (33) and (41) are different. As discussed in Section IV-A, solving (33) at each MCG iteration step is equivalent to solving a reduced linear equation (40). In this case, either Cholesky decomposition or the CG method can be applied, because the linear equation (40) is positive definite. On the other hand, the least-squares solution of (41) can be computed by either a direct linear solver (e.g., QR decomposition) or an iterative linear solver (e.g., LSQR [31]). However, the least-squares solver is more expensive than a positive definite linear equation solver, especially for large-scale problems. Due to this reason, the proposed MCG algorithm is expected to offer superior runtime efficiency over the traditional OMP method, as will be demonstrated by our numerical examples in Section VI.

V. Implementation Details

Our proposed PDN transient analysis based on sparse approximation is made efficient by carefully addressing a number of implementation issues. In this section, we discuss these implementation details and then summarize the overall transient analysis flow.

A. Linear Solver Selection

As previously mentioned, the proposed MCG algorithm needs to solve the linear equation (40) at each iteration step, as shown by Step 3 of Algorithm 3. Ideally, if the incremental response $\boldsymbol{\delta}(t_{n+1})$ in (25) is sparse, (40) is small and it can be efficiently solved by a direct linear solver based on Cholesky decomposition. However, if the incremental response $\boldsymbol{\delta}(t_{n+1})$ is not sparse at a particular time point t_n, a large linear equation must be solved. In this case, directly applying Cholesky decomposition to solve (40) is not computationally efficient, since the computational complexity of Cholesky decomposition grows quickly with the problem size. On the other hand, an iterative algebraic multigrid (AMG) solver [14] can be extremely efficient for large-scale equations; however, AMG is not as efficient as Cholesky decomposition for small-size problems due to the overhead of constructing the interpolation and restriction operators [14].

To address this issue, we propose to adaptively select the appropriate linear solver based on the size of (40). If the equation size is sufficiently small (say, less than 0.5×10^6), Cholesky decomposition is used to solve (40). Otherwise, if the equation size is large (say, greater than 0.5×10^6), an AMG solver is applied to find the solution of (40). Such an adaptive strategy for solver selection allows us to fully take advantage of the trade-offs between computational efficiency and problem size for different linear solvers in order to reduce the overall runtime for a broad range of PDN analysis problems.

The aforementioned strategy of linear solver selection is mainly driven by computational cost. On the other hand, the accuracy of these linear solvers is equally important and must be carefully considered. In particular, AMG is an iterative algorithm and its error tolerance must be set to be sufficiently small due to the following two reasons. First, the AMG error directly impacts the convergence of Algorithm 3. If the AMG solver is not sufficiently accurate, the residue calculated by Step 5 of Algorithm 3 may be large, even if a lot of active nodes are selected. Second, the AMG error also plays an important role in estimating the LTE. Without an accurate AMG solver, LTE cannot be accurately calculated and, consequently, the time step for transient analysis cannot be appropriately determined, as will be discussed in detail in the next subsection.

B. Adaptive Time Step Control

The LTE of a numerical integration method is strongly affected by the time step size. In many practical applications,

the time step has to be adjusted to ensure that the LTE is less than a predefined threshold. On the other hand, the time step should also be increased to speed up the transient simulation when the circuit behavior is relatively quiet. Although we can continuously adjust the time step based on the BDF2 formula, such an implementation can be computationally expensive and unnecessary. Instead, we implement a simple control scheme with discrete time step in this paper.

In particular, if the estimated LTE is too large, we reduce the time step by a factor of the power of two (i.e., 2, 4, 8, etc.). Otherwise, if the LTE is too small, we double the time step. Because the BDF2 formula in (2) and (3) requires the solutions at two past time points with the same time step, we use an interpolation scheme to compute the solutions at the missing time points whenever the time step changes.

More specifically, assume that transient simulation is progressing with the time step h from the time point t_{n-2} to t_{n-1} and then from t_{n-1} to t_n. At the time point t_n, if the LTE estimation indicates that the LTE is greater than a predefined threshold, we need to reduce the time step and recalculate the solution at t_n. Instead of adopting a continuous time step that may be only slightly smaller than h, we reduce the time step to $h/2$ (or even $h/4$ if necessary). After that, we interpolate the node voltages $\mathbf{v}(t_{n-3/2})$ and the branch currents $\mathbf{i}(t_{n-3/2})$ at the middle of t_{n-2} and t_{n-1} (denoted as $t_{n-3/2}$). Since the time steps $t_{n-1} - t_{n-3/2}$ and $t_n - t_{n-1}$ are now both equal to $h/2$, we can proceed to use the BDF2 formula with constant time step in (2) and (3) to continue our transient simulation. On the other hand, when the time step increases to $2h$, we only need to select a few past solutions to make the equivalent time step equal to $2h$ so that the BDF2 formula in (2) and (3) can again be applied.

Given the aforementioned setup, the LTE of a capacitor voltage or inductor current x at the nth time point can be estimated by [32]

$$LTE_x(t_n) = \frac{h_n^2 (h_n + h_{n-1})^2}{2h_n + h_{n-1}} \times \langle x(t_n), x(t_{n-1}), x(t_{n-2}), x(t_{n-3}) \rangle \quad (45)$$

where the operator $<x(t_n), x(t_{n-1}), x(t_{n-2}), x(t_{n-3})>$ is recursively defined as

$$\langle x(t_n), x(t_{n-1}), \cdots, x(t_{n-i}) \rangle = \begin{cases} x(t_n) & (i = 0) \\ \frac{\langle x(t_n), \cdots, x(t_{n-i+1}) \rangle - \langle x(t_{n-1}), \cdots, x(t_{n-i}) \rangle}{h_n + h_{n-1} + \cdots + h_{n-i+1}} & (i \geq 1) \end{cases}. \quad (46)$$

For a given PDN containing RLC elements, the LTE values are calculated by (45) for all capacitor voltages and inductor currents and compared against the predefined threshold for adaptive time step control.

C. Summary

Algorithm 4 summarizes the major steps of the proposed transient analysis algorithm for PDN. Before the transient analysis starts, a DC analysis is first performed to determine the initial condition of all node voltages and branch currents. Since, we focus on large-scale PDN analysis problems that cannot be efficiently solved by a direct linear solver, the DC analysis in Step 4 of Algorithm 4 is implemented with an AMG solver [14]. Next, our proposed MCG solver (i.e., Algorithm 3) is used to solve the linear equation (25) at each time point. As mentioned in Section V-A, Cholesky decomposition and AMG solver are adaptively selected to solve the linear equation (40) at Step 3 of Algorithm 3. Finally, once the PDN response is known at the current time point, LTE values are calculated to adaptively adjust the time step, as discussed in Section V-B.

Algorithm 4 Transient Analysis for PDN

1: Start from a given PDN that is described by (1) and the time interval [0, t_{STOP}] for transient analysis.
2: Derive the incremental formulation (25).
3: Set an initial index $n = 0$, an initial time point $t_0 = 0$, and an initial time step h_0 specified by the user.
4: Apply a DC analysis to solve the initial node voltages $\mathbf{v}(t_0)$ and the initial branch currents $\mathbf{i}(t_0)$.
5: Set $t_{n+1} = t_n + h_n$, and solve the incremental response $\boldsymbol{\delta}(t_{n+1})$ by the MCG solver (i.e., Algorithm 3).
6: Calculate the node voltages $\mathbf{v}(t_{n+1})$ by (26), and then the branch currents $\mathbf{i}(t_{n+1})$ by (5).
7: Calculate the LTE values for all capacitor voltages and inductor currents by (45).
8: If all LTE values are less than a user-defined threshold (say, LTE_{LOW}), set $h_{n+1} = 2 \cdot h_n$.
9: If at least one of these LTE values is greater than a user-defined threshold (say, LTE_{UP}), set $h_{n+1} = h_n/2$ and go to Step 5.
10: Set $n = n + 1$. Repeat Step 5–10 until t_n reaches t_{STOP}.

The runtime of the MCG solver at each time point dominates the overall computational time for transient analysis. Since MCG solves the reduced linear equation in (40) at each iteration step (i.e., Step 3 of Algorithm 3), its computational complexity can be expressed as $O(M_{(k)}^{\alpha})$, where $M_{(k)}$ denotes the number of unknowns of the reduced linear equation at the kth iteration step and the constant α is usually between 1 and 2. For many practical problems, the incremental response defined by (26) is sparse and contains few nonzeros. Hence, $M_{(k)}$ is much less than the total number of nodes of the PDN (i.e., M). For this reason, our proposed MCG solver is computationally more efficient than the traditional linear solver for which the computational complexity is $O(M^{\alpha})$.

VI. Numerical Examples

A. Experimental Setup

In this section, we demonstrate the efficacy of our proposed PDN transient analysis for six large-scale circuit examples, as shown in Table I. These six test cases, labeled as PND1–PDN6 in Table I, are from the IBM benchmarks ibmpg3-ibmpg6, ibmpgnew1, and ibmpgnew2 given in [33]. In this paper, the two benchmarks ibmpg1 and ibmpg2 are excluded, since their circuit sizes are small and, hence, they are not considered as good examples to validate our proposed algorithm that is particularly developed for large-scale circuits.

TABLE I
PDN BENCHMARK INFORMATION

Name	Number of Current Sources	Number of Nodes
PDN1	0.20M	1.04M
PDN2	0.27M	1.21M
PDN3	0.54M	1.01M
PDN4	0.76M	1.53M
PDN5	0.36M	0.99M
PDN6	0.36M	1.80M

To emulate the effect of clock/power gating, we simultaneously turn on/off all current sources within a local region that is defined by a bounding box. The percentage of the current sources being activated or deactivated is referred to the activation rate in this paper. In our experiment, the size of the bounding box is appropriately set so that the activation rate is around 10% for each benchmark circuit. The waveform of each current source is set to a step function in time domain where the rising time equals 0.1 ns.

For testing and comparison purposes, three different techniques are implemented for PDN transient analysis. All these three implementations use the BDF2 formula [26] for numerical integration with adaptive time step control. The only difference between them lies in the linear solver that is used to solve the linear equation (25) at each time point: 1) the AMG solver [14]; 2) the OMP solver [28]; and 3) the proposed MCG solver.

To measure and compare the accuracy for different techniques, a "golden" linear solver is further implemented to generate the golden solution for each test case. The golden solver applies the BDF2 formula [26] for numerical integration with a fixed time step that is sufficiently small and, hence, the LTE of the golden solver is negligible. In addition, the golden solver uses Cholesky decomposition to solve the linear equation (25) at each time point. Hence, the resulting residue of (25) is also negligible for the golden solver.

It is important to emphasize that the computational complexity of Cholesky decomposition grows quickly with problem size. Even though, Cholesky decomposition remains feasible for all test cases shown in Table I, it would quickly become computationally infeasible as circuit size further increases. In this paper, the proposed MCG algorithm is particularly developed to handle large-scale PDN analysis problems where direct linear solvers are not computationally feasible. Here, we only use a number of medium-scale PDN examples to validate the MCG algorithm, because we need to apply Cholesky decomposition to calculate the golden solution for accuracy comparison.

For all test cases, the transient analysis is performed over the time interval [0, 100 ns]. The time interval is set to be sufficiently long for the transient response to die out. All experiments are executed on a Linux server with Intel Xeon 2.67 GHz CPU and 512 GB memory.

B. Comparison on Accuracy and Speed

Table II shows the computational time, memory consumption, and simulation error for three different implementations.

TABLE II
COMPARISON ON ACCURACY AND RUNTIME FOR TRANSIENT ANALYSIS OF PDNS (ACTIVATION RATE $\approx 10\%$)

CKT	Solver	Runtime (Sec.)	Memory Usage (GB)	$Error_{Mean}$	$Error_{Max}$	#Time Steps
PDN1	AMG	1.43×10^{3}	1.12	1.77×10^{-4}	1.01×10^{-2}	56
	OMP	3.24×10^{3}	1.07	1.95×10^{-4}	1.01×10^{-2}	56
	MCG	1.04×10^{3}	0.91	3.99×10^{-4}	1.01×10^{-2}	56
PDN2	AMG	1.58×10^{3}	1.37	7.06×10^{-6}	7.08×10^{-3}	53
	OMP	1.55×10^{3}	0.84	1.97×10^{-4}	8.56×10^{-2}	122
	MCG	9.70×10^{2}	1.04	4.87×10^{-5}	7.13×10^{-3}	53
PDN3	AMG	3.22×10^{3}	1.44	6.72×10^{-5}	7.31×10^{-3}	54
	OMP	2.00×10^{3}	0.84	2.70×10^{-4}	4.59×10^{-2}	118
	MCG	1.70×10^{3}	1.13	7.27×10^{-5}	7.89×10^{-3}	54
PDN4	AMG	4.20×10^{3}	2.21	1.56×10^{-4}	9.29×10^{-3}	55
	OMP	3.55×10^{3}	1.64	2.17×10^{-4}	6.32×10^{-2}	93
	MCG	3.24×10^{3}	2.00	3.66×10^{-4}	1.05×10^{-2}	55
PDN5	AMG	1.49×10^{3}	1.17	7.76×10^{-5}	1.23×10^{-2}	56
	OMP	7.85×10^{2}	0.80	2.09×10^{-4}	1.48×10^{-2}	75
	MCG	6.72×10^{2}	0.86	3.38×10^{-4}	1.15×10^{-2}	57
PDN6	AMG	3.12×10^{3}	2.02	1.27×10^{-4}	1.37×10^{-2}	59
	OMP	6.13×10^{3}	1.87	1.63×10^{-4}	1.37×10^{-2}	59
	MCG	1.95×10^{3}	1.49	3.88×10^{-4}	1.37×10^{-2}	59

Here, the mean and maximum errors are respectively defined as

$$\begin{aligned}\text{Error}_{\text{Mean}} &= \frac{1}{M}\cdot\frac{1}{t_{\text{STOP}}}\cdot\sum_{m=1}^{M}\int_{0}^{t_{\text{STOP}}}|v_m(t)-\tilde{v}_m(t)|\cdot dt \\ &\approx \frac{1}{M}\cdot\frac{1}{t_{\text{STOP}}}\cdot\sum_{m=1}^{M}\sum_{n=0}^{N}|v_m(t_{n+1})-\tilde{v}_m(t_{n+1})|\cdot h_n\end{aligned} \tag{47}$$

$$\text{Error}_{\text{Max}} = \max_{m,n}|v_m(t_n)-\tilde{v}_m(t_n)| \tag{48}$$

where $v_m(t)$ and $\tilde{v}_m(t)$ denote the exact and estimated node voltages at the time t respectively, t_{STOP} represents the upper bound of the time interval for transient analysis, and h_n stands for the time step at the nth time point.

Studying Table II, we notice that the OMP solver can be computationally expensive for a number of benchmarks. Taking PDN1 as an example, we observe from our experiment that the least-squares problem in (23) is ill-conditioned. In our implementation of OMP, the least-squares solution of (23) is computed by pseudo-inverse, instead of QR decomposition, in order to minimize the computational cost. Therefore, OMP cannot accurately find the least-squares solution for PDN1 and the OMP solver eventually has to use AMG to solve the full-size linear equation at almost all time points. It, in turn, results in expensive computational cost.

In addition, the numerical issue of OMP can further impact the LTE estimation during transient analysis. Considering PDN2 as an example, we observe from Table II that OMP takes significantly more time steps than AMG and MCG to finish the transient analysis, thereby resulting in expensive computational cost. Such an observation is made, because the OMP solver is not numerically robust and its solution is not highly accurate. Therefore, the LTE is not accurately estimated and the transient analysis fails to use the appropriate time step to perform numerical integration in this example.

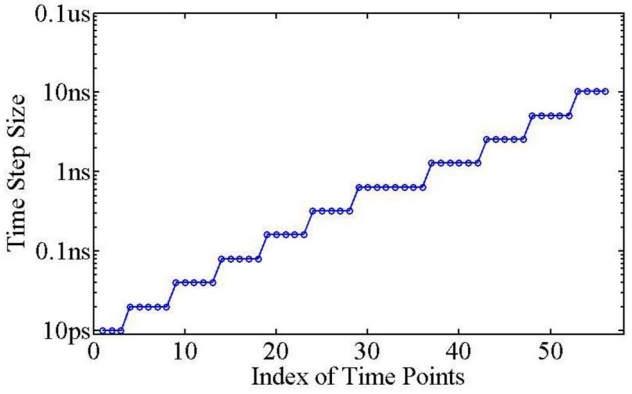


Fig. 3. Time step sizes of MCG are plotted at different time points for PDN6.

AMG and MCG, on the other hand, work perfectly with the adaptive time step control based on LTE. The numbers of time steps associated with AMG and MCG are almost identical, as shown in Table II. To further validate the time step control scheme for MCG, we closely examine the time step sizes during transient analysis. Since the rising time of the input current sources is set to 0.1 ns, the initial time step size is chosen as 10 ps for all benchmarks. As the time evolves and the transient response dies out, the time step sizes are adaptively increased to larger values (e.g., more than 10 ns at the end of the transient analysis). The average time step size over all six benchmarks is equal to 1.7 ns. Taking PDN6 as an example, Fig. 3 shows its time step sizes at different time points where the time step size gradually increases over time.

Compared to AMG, MCG reduces the runtime by up to 2.2×, while simultaneously achieving similar accuracy, as shown in Table II. In particular, the maximum error is almost identical for AMG and MCG over all benchmarks. Note that the maximum error is often of greater importance than the mean error for many practical applications, since we want to predict the worst-case voltage droop, instead of the average voltage droop, by running transient simulation. For these test cases, MCG demonstrates superior performance, because it efficiently exploits the sparse property of the incremental response $\boldsymbol{\delta}(t_{n+1})$, while AMG is a general-purpose linear solver and is not particularly tuned to solve sparse solutions.

The memory usages of AMG, OMP, and MCG are measured at each time step by counting the size of the major data structures, as shown in Table II. Note that OMP and MCG consume less memory than AMG. Such an observation is made, because OMP and MCG only need to solve a reduced linear equation with reduced memory consumption, when the circuit response is sparse.

C. Visualization of Results

In this subsection, we take the benchmark PDN6 as an example to show several important findings related to the proposed transient analysis based on sparse approximation. These studies provide a number of intuitive insights about our proposed method, thereby facilitating us to further understand its advantages and limitations.

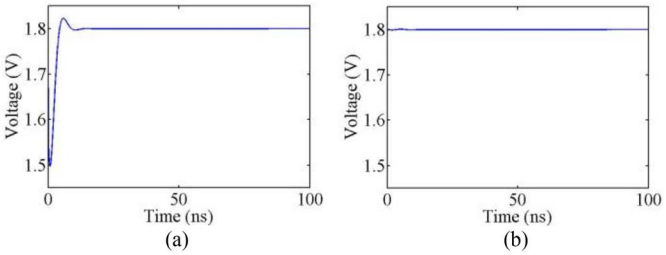


Fig. 4. Golden PDN response is calculated by transient analysis implemented with a direct linear solver and is plotted for two selected nodes of PDN6. (a) One node within the activated region. (b) One node outside the activated region.

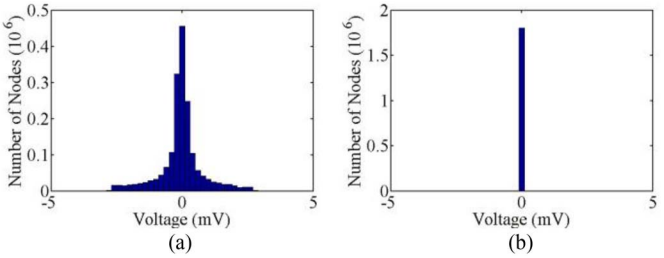


Fig. 5. Golden incremental voltage response $\boldsymbol{\delta}(t)$ is solved from (25) and (26) by a direct linear solver and its histogram is plotted at two selected time points for PDN6. (a) t = 3 ns at the beginning of the transient analysis. (b) t = 100 ns at the end of the transient analysis.

Fig. 4 shows the transient response of two selected nodes from PDN6. One is inside the activated region, and the other is outside the activated region. A large-scale oscillation is observed for the voltage response associated with the node inside the activated region, due to the underdamped nature of the PDN. On the other hand, the voltage response is almost constant over time for the node outside the activated region, as is expected. In this case, the incremental voltage response associated with the inactive node is almost zero. It, in turn, facilitates our proposed sparse approximation to achieve extremely high efficiency for transient analysis.

Fig. 5 plots the incremental voltage response $\boldsymbol{\delta}(t)$ at two different time points: 1) $t = 3$ ns at the beginning of the transient analysis and 2) $t = 100$ ns at the end of the transient analysis. Two important observations can be made from Fig. 5. First, since a small portion of current sources are turned on, only a small number of nodes are activated in this example. Hence, the incremental voltage response $\boldsymbol{\delta}(t)$ is close to zero for most nodes at the beginning of the transient analysis, as shown in Fig. 5(a). Second, but more importantly, as the transient response dies out at the end of the transient analysis, the incremental voltage response $\boldsymbol{\delta}(t)$ becomes almost zero for all nodes at t = 100 ns. In this case, since the vector $\boldsymbol{\delta}(t)$ contains few nonzeros, it can be efficiently determined by the proposed MCG solver with low computational cost.

Figs. 6 and 7 further plot the incremental voltage response $\boldsymbol{\delta}(t)$ as a function of the spatial location for different layers (one bottom layer and one top layer) at $t = 3$ ns and $t = 100$ ns, respectively. Studying Figs. 6 and 7 reveals an important fact that the incremental response $\boldsymbol{\delta}(t)$ is close to zero for a large number of nodes at $t = 3$ ns and it becomes almost zero for

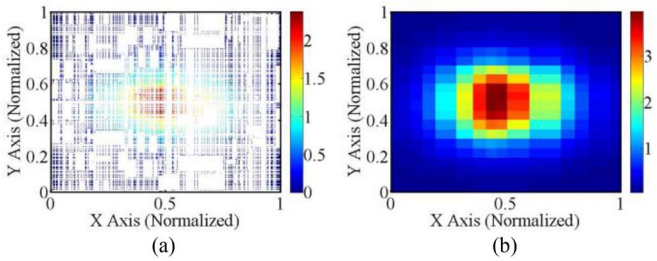


Fig. 6. Golden incremental voltage response $\delta(t_n)$ is solved from (25) and (26) by a direct linear solver and $|\delta(t_n)|$ is plotted for two selected metal layers of PDN6 at t = 3 ns. (a) Bottom metal layer (b) Top metal layer. The color indicates the magnitude of $|\delta(t_n)|$ with the unit of mV.

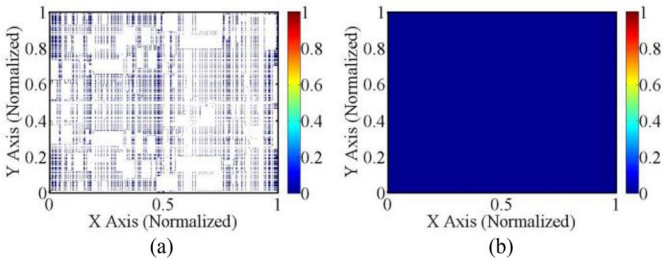


Fig. 7. Golden incremental voltage response $\delta(t_n)$ is solved from (25) and (26) by a direct linear solver and $|\delta(t_n)|$ is plotted for two selected metal layers of PDN6 at t = 100 ns. (a) Bottom metal layer (b) Top metal layer. The color indicates the magnitude of $|\delta(t_n)|$ with the unit of mV.

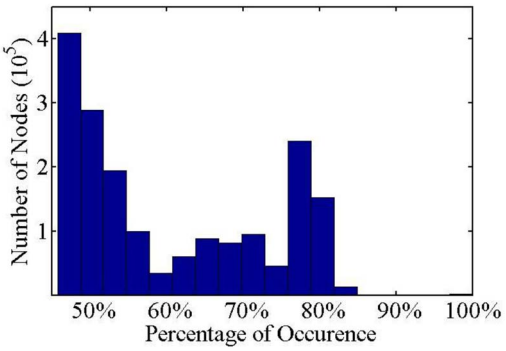


Fig. 8. Histogram of the percentage of occurrence is plotted for transient simulation of PDN6 by using MCG.

all nodes at $t = 100$ ns. This observation is consistent with that shown in Fig. 5.

Fig. 8 shows the histogram of the percentage of occurrence for each node being selected by the MCG solver during transient simulation. Here, the occurrence of a node is defined as the number of time points for the node being selected, and the percentage of occurrence of a node is defined as the ratio of its occurrence over the total number of time points. In other words, if the percentage of occurrence is high for a particular node, this node is considered to be active at most time points. As shown in Fig. 8, for more than half of the nodes, the percentage of occurrence is less than 60%, implying that a large number of nodes are "inactive" and the corresponding incremental response is zero at many time points. This is additional evidence to explain the reason why the proposed transient analysis based on sparse approximation is computationally efficient.

TABLE III
COMPARISON ON ACCURACY AND RUNTIME FOR TRANSIENT ANALYSIS OF PDN6 BY VARYING ACTIVATION RATE

Activation Rate	Solver	Runtime (Sec.)	$Error_{Mean}$	$Error_{Max}$	#Time Steps
2%	AMG	2.84×10^{3}	2.81×10^{-5}	9.09×10^{-3}	54
	MCG	1.20×10^{3}	3.51×10^{-4}	9.06×10^{-3}	54
5%	AMG	2.96×10^{3}	6.53×10^{-5}	1.10×10^{-2}	56
	MCG	1.39×10^{3}	2.84×10^{-4}	1.10×10^{-2}	57
10%	AMG	3.12×10^{3}	1.27×10^{-4}	1.37×10^{-2}	59
	MCG	1.95×10^{3}	3.88×10^{-4}	1.37×10^{-2}	59
20%	AMG	3.40×10^{3}	2.47×10^{-4}	1.63×10^{-2}	61
	MCG	2.56×10^{3}	8.19×10^{-4}	1.63×10^{-2}	62
30%	AMG	3.53×10^{3}	3.30×10^{-4}	1.59×10^{-2}	62
	MCG	2.63×10^{3}	5.46×10^{-4}	1.59×10^{-2}	62
40%	AMG	3.60×10^{3}	4.38×10^{-4}	1.62×10^{-2}	62
	MCG	2.76×10^{3}	5.97×10^{-4}	1.62×10^{-2}	62
50%	AMG	3.82×10^{3}	5.56×10^{-4}	1.64×10^{-2}	62
	MCG	3.16×10^{3}	7.58×10^{-4}	1.64×10^{-2}	62

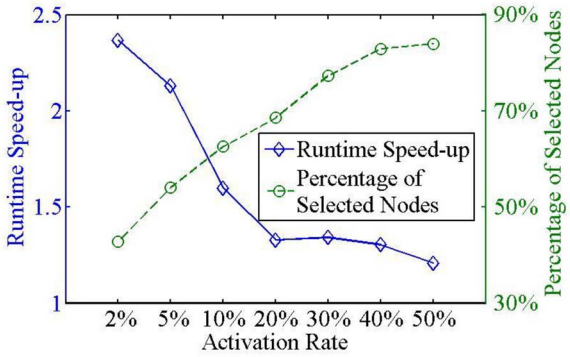


Fig. 9. Runtime speed-up of MCG over AMG and the percentage of selected nodes by MCG are plotted as functions of the activation rate for PDN6.

D. Impact of Activation Rate

In this subsection, we further study the efficiency of MCG by varying the activation rate. Similar to the previous subsection, we again take the benchmark PDN6 as an example. Table III compares the runtime and error for two different solvers: 1) AMG and 2) MCG. Here, we do not include the OMP results, because the OMP solver suffers from numerical issues, as is demonstrated in Section VI-B.

To clearly explain the impact of activation rate, Fig. 9 plots the runtime speed-up of MCG over AMG and the percentage of selected nodes by MCG as functions of the activation rate. Here, the percentage of selected nodes is equal to the average number of selected nodes over all time points divided by the total number of circuit nodes.

Several important observations can be made from the data in Table III and Fig. 9. First, as the activation rate increases, the number of time steps increases for both AMG and MCG. Intuitively, as more circuit components become active, a smaller time step must be used to perform transient analysis. Second, the runtime of both AMG and MCG increases with the

activation rate. The runtime of AMG increases, mostly because the number of time steps increases. On the other hand, the runtime of MCG increases due to two reasons: 1) the increase of the number of time steps as shown in Table III and 2) the increase of the number of active nodes selected by MCG as shown in Fig. 9.

Finally, the runtime speed-up of MCG over AMG decreases with the activation rate, as the sparsity of the solution is reduced. However, MCG is still more computationally efficient than AMG (i.e., less computational time but similar simulation accuracy), even if the activation rate is as large as 50%. Note that the maximum error is almost identical for AMG and MCG over all cases in Table III.

VII. Conclusion

In this paper, an efficient method based on sparse approximation is proposed for transient analysis of large-scale PDNs. By exploiting the unique sparse structure of the transient response of PDNs with clock/power gating, an MCG algorithm is developed to efficiently find the sparse solution (i.e., the incremental voltage response of the PDN) of a linear, positive definite equation. The MCG algorithm facilitates the proposed transient analysis to reduce runtime over other general-purpose linear solvers. As being demonstrated by the numerical examples in this paper, the proposed transient analysis based on sparse approximation offers up to 2.2× speedup over other traditional methods, while achieving similar accuracy.

The proposed MCG solver is computationally efficient if the solution vector is sparse. If the solution is not sparse, a traditional linear solver (i.e., either a direct linear solver or an iterative linear solver) is preferred. In practice, a heuristic scheme for sparsity detection can be possibly incorporated into the MCG algorithm. More specifically, if a large number of active nodes are detected within the iteration loop of Algorithm 3, we should abort the MCG solver and switch to the traditional linear solver. The detailed implementation of the aforementioned heuristics for sparsity detection is beyond the scope of this paper and will be further studied in our future research.

It is worth mentioning that the large overhead of clock/power gating often prevents today's IC designers from using a large number of clock/power domains. However, the advance of several emerging technologies (e.g., on-chip DC-DC converter [34] and 3-D integration [35]) has brought up several new opportunities to explore the feasibility of using fine-grain clock/power domains for future ICs [36]. Due to this reason, the efficacy of the proposed MCG algorithm is expected to be more pronounced for simulating future PDNs.

Finally, it is important to note that the proposed idea of sparse approximation is not limited to linear circuits (e.g., PDNs) only. It can be further extended to transient analysis of nonlinear circuits, such as logic circuits and SRAM circuits, where the transient response is localized in a small region and a large portion of the circuit is inactive. In these cases, a number of linear equations must be repeatedly solved during the Newton–Raphson iteration [25]. Since the Jacobian matrix of a general nonlinear circuit may not be symmetric and positive definite, a different iterative algorithm such as the generalized minimal residual (GMRES) method [37], instead of the CG method, must be used to construct the sparse linear solver. More details along this direction are not included in this paper but will be studied in our future research.

References

[1] S. Nassif and J. Kozhaya, "Fast power grid simulation," in *Proc. Design Autom. Conf.*, Los Angeles, CA, USA, 2000, pp. 156–161.

[2] S. Nassif, "Power grid analysis benchmarks," in *Proc. Asia South Pac. Design Autom. Conf.*, Seoul, Korea, 2008, pp. 376–381.

[3] Q. Zhu, *Power Distribution Network Design for VLSI*. Hoboken, NJ, USA: Wiley, 2004.

[4] P. Ghanta, S. Vrudhula, R. Panda, and J. Wang, "Stochastic power grid analysis considering process variations," in *Proc. Design Autom. Test Eur.*, vol. 2. Munich, Germany, 2005, pp. 964–969.

[5] S. Pant, D. Blaauw, V. Zolotov, S. Sundareswaran, and R. Panda, "A stochastic approach to power grid analysis," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, 2004, pp. 171–176.

[6] E. Chiprout, "Fast flip-chip power grid analysis via locality and grid shells," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2004, pp. 485–488.

[7] S. Pant and E. Chiprout, "Power grid physics and implications for CAD," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, 2006, pp. 199–204.

[8] R. Mandrekar, K. Srinivasan, E. Engin, and M. Swminathan, "Causality enforcement in transient co-simulation of signal and power delivery networks," *IEEE Trans. Adv. Packag.*, vol. 30, no. 2, pp. 270–278, May 2007.

[9] T. Chen and C. Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods," in *Proc. Design Autom. Conf.*, Las Vegas, NV, USA, 2001, pp. 559–562.

[10] S. Weng, Q. Chen, and C. Cheng, "Time-domain analysis of large-scale circuits by matrix exponential method with adaptive control," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 8, pp. 1180–1193, Aug. 2012.

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

[12] J. Yang, Z. Li, Y. Cai, and Q. Zhou, "PowerRush: An efficient simulator for static power grid analysis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,* vol. 22, no. 10, pp. 2103–2116, Oct. 2014.

[13] J. Kozhaya, S. Nassif, and F. Najm, "A multigrid-like technique for power grid analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 10, pp. 1148–1160, Oct. 2002.

[14] H. Su, E. Acar, and S. Nassif, "Power grid reduction based on algebraic multigrid principles," in *Proc. Design Autom. Conf.*, Anaheim, CA, USA, 2003, pp. 109–112.

[15] Z. Feng and P. Li, "Multigrid on GPU: Tackling power grid analysis on parallel SIMT platforms," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2008, pp. 647–654.

[16] Z. Feng and Z. Zeng, "Parallel multigrid preconditioning on graphics processing units (GPUs) for robust power grid analysis," in *Proc. Design Autom. Conf.*, Anaheim, CA, USA, 2010, pp. 661–666.

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

[18] D. Kouroussis and F. Najm, "A static pattern-independent technique for power grid voltage integrity verification," in *Proc. Design Autom. Conf.*, Anaheim, CA, USA, 2003, pp. 99–104.

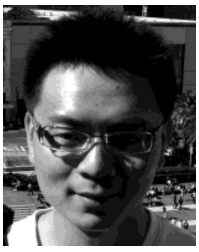
[19] H. Qian, S. Nassif, and S. Sapatnekar, "Early-stage power grid analysis for uncertain working modes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 5, pp. 676–682, May 2005.

[20] X. Xiong and J. Wang, "Vectorless verification of RLC power grids with transient current constraints," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2011, pp. 548–554.

[21] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 47, no. 3, pp. 415–420, Mar. 2000.

[22] K. Agarwal, K. Nowka, H. Deogun, and D. Sylvester, "Power gating with multiple sleep modes," in *Proc. Int. Symp. Qual. Electron. Design*, San Jose, CA, USA, 2006, pp. 633–637.

[23] W. Zhang *et al.*, "Efficient power network analysis considering multidomain clock gating," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 9, pp. 1348–1358, Jan. 2009.
[24] J. Shewchuk. (Aug. 1994). *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. [Online]. Available: http://www.cs.cmu.edu/~quake-papers/ painless-conjugate-gradient.pdf
[25] L. Pillage, R. A. Rohrer, and C. Visweswariah, *Electronic Circuit and System Simulation Methods*. New York, NY, USA: McGraw-Hill, 1998.
[26] C. Gear, "Simultaneous numerical solution of differential-algebraic equations," *IEEE Trans. Circuit Theory*, vol. 18, no. 1, pp. 89–95, Jan. 1971.
[27] A. Devgan, H. Ji, and W. Dai, "How to efficiently capture on-chip inductance effects: Introducing a new circuit element *K*," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2000, pp. 150–155.
[28] P. Sun, X. Li, and M. Ting, "Efficient incremental analysis of on-chip power grid via sparse approximation," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, 2011, pp. 676–681.
[29] D. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 1999.
[30] R. Horn and C. Johnson, *Matrix Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 1990.
[31] C. Paige and M. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Trans. Math. Softw.*, vol. 8, no. 1, pp. 43–71, Mar. 1982.
[32] J. Verner, "Explicit Runge-Kutta methods with estimates of the local truncation error," *SIAM J. Numer. Anal.*, vol. 15, no. 4, pp. 772–790, Aug. 1978.
[33] Z. Li, P. Li, and S. Nassif. (Aug. 2011). *IBM Power Grid Benchmarks*. [Online]. Available: http://dropzone.tamu.edu/~pli/PGBench/
[34] H. Le, S. Randers, and E. Alon, "Design techniques for fully integrated switched-capacitor DC-DC converters," *IEEE J. Solid-State Circuits*, vol. 48, no. 9, pp. 2120–2131, Sep. 2011.
[35] Y. Xie, J. Cong, and S. Sapatnekar, *Three-Dimensional IC: Design, CAD, and Architecture*. New York, NY, USA: Springer, 2009.
[36] H. Esmaeilzadeh, E. Blern, R. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. Int. Symp. Comput. Archit.*, Portland, OR, USA, 2011, pp. 365–376.
[37] Y. Saad and M. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, pp. 856–869, Jul. 1986.



Hengliang Zhu (S'07–M'09) received the B.E. degree in electronic engineering from the University of Science and Technology of China, Hefei, China, and the Ph.D. degree in microelectronics from Fudan University, Shanghai, China, in 2004 and 2009, respectively.

He joined the State Key Laboratory of ASIC & System, Microelectronics Department, Fudan University, as an Assistant Professor, in 2009. His current research interests include circuit analysis, interconnect parameter extraction, and model order reduction.

Yuanzhe Wang received the B.Eng. and M.Phil. degrees in electrical engineering from Tianjin University, Tianjin, China, and the University of Hong Kong, Hong Kong, in 2009 and 2011, respectively.

He is currently with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA. His current research interests include self-healing of analog circuits and power grid analysis.

Frank Liu (S'95–M'99–SM'09) received the M.S. degree in applied mathematics from the University of Minnesota, Minneapolis, MN, USA, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA.

He is currently a Research Staff Member with IBM Austin Research Laboratory, Austin, TX, USA. He has authored and co-authored over 60 conference and journal papers.

Dr. Liu was the recipient of the Best Paper Award at Asian-Pacific Design Automation Conference, the IEEE Donald O. Pederson Best Paper Award, and the Multiple IBM Research Division Accomplishment Awards. He was an ACM SIGDA Executive Committee Member from 2012 to 2015.

Xin Li (S'01–M'06–SM'10) received the B.S. and M.S. degrees in electronics engineering from Fudan University, Shanghai, China, in 1998 and 2001, respectively, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2005.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Carnegie Mellon University. His current research interests include computer-aided design, neural signal processing, and power system analysis and design.

Dr. Li was the recipient of the National Science Foundation Faculty Early Career Development Award in 2012, the Best Paper Award from Design Automation Conference in 2010, and the IEEE/ACM William J. McCalla ICCAD Best Paper Awards twice in 2004 and 2011.

Xuan Zeng (M'97) received the B.S. and Ph.D. degrees in electrical engineering from Fudan University, Shanghai, China, in 1991 and 1997, respectively.

She is currently a Full Professor with the Department of Microelectronics, Fudan University, where she was the Director of the State Key Laboratory of ASIC & System, from 2008 to 2012. She was a Visiting Professor at the Department of Electrical Engineering, Texas A&M University, College Station, TX, USA, and the Department of Microelectronics, Technische Universiteit Delft, Delft, The Netherlands, in 2002 and 2003, respectively. Her current research interests include design for manufacturability, high-speed interconnect analysis and optimization, analog behavioral modeling, circuit simulation, and ASIC design.

Dr. Zeng was the recipient of the Chinese National Science Funds for Distinguished Young Scientists in 2011 and the First-Class of Natural Science Prize of Shanghai in 2012. She is the Changjiang Distinguished Professor with the Ministry of Education Department of China in 2014.

Peter Feldmann (F'00) was born in Timisoara, Romania. He received the B.Sc. degree (*summa cum laude*) in computer engineering and the M.Sc. degree in electrical engineering, both from the Technion–Israel Institute of Technology, Haifa, Israel, in 1983 and 1987, respectively, and the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA, USA, in 1991.

He was a Designer in digital signal processors at Zoran Microelectronics, Haifa. He was a Distinguished Technical Staff Member at Design Principles Research Department, Bell Laboratories, Murray Hill, NJ, USA, and also a Research Staff Member at the IBM T.J. Watson Research Center, Yorktown Heights, NY, USA. He is currently with D. E. Shaw Research, New York City, NY, USA, researching on hardware accelerated molecular dynamics simulation. He was the Vice President of the VLSI and Integrated Electro-optics at CeLight, Silver Spring, MD, USA, a fiber-optic communications start-up. He was an Adjunct Professor at the Columbia University Electrical Engineering Department, New York. His current research interests include analysis, modeling, design, and optimization methods for integrated electronic circuits and communication systems, analysis and modeling for timing, power, and noise in digital VLSI circuits, and also on the large-scale dynamic analysis and modeling of the electricity transmission grid. He has authored over hundred papers and patents.