Power Grid Reduction by Sparse Convex Optimization

Wei Ye
ECE Department, UT Austin
weiye@utexas.edu

Meng Li
ECE Department, UT Austin
meng_li@utexas.edu

Kai Zhong
ICES, UT Austin
zhongkai@ices.utexas.edu

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

David Z. Pan
ECE Department, UT Austin
dpan@ece.utexas.edu

ABSTRACT

With the dramatic increase in the complexity of modern integrated circuits (ICs), direct analysis and verification of IC power distribution networks (PDNs) have become extremely computationally expensive. Various power grid reduction methods are proposed to reduce the grid size for fast verification and simulation but usually suffer from poor scalability. In this paper, we present a convex optimization-based framework for power grid reduction. Edge sparsification is formulated as a weighted convex optimization problem with sparsity-inducing penalties, which provides an accurate control over the final error. A greedy coordinate descent (GCD) method with optimality guarantee is proposed along with a novel coordinate selection strategy to improve the efficiency and accuracy of edge sparsification. Experimental results demonstrate that the proposed approach achieves better performance compared with traditional gradient descent methods, and 98% accuracy and good sparsity for industrial benchmarks.

KEYWORDS

Power grid reduction; sparse convex optimization; greedy coordinate descent

1 INTRODUCTION

Power distribution network (PDN) provides power and ground voltage supply to on-chip components. Robust PDNs are essential to ensure reliable operations and high performance of chips. However, with the dramatic increase in the complexity of modern integrated circuits (ICs), direct analysis and verification of PDNs have become extremely computationally expensive. Therefore, fast and accurate modeling and verification techniques are necessary to assist power grid design.

The goal of power grid reduction is to reduce an original large power grid to a significantly smaller one for fast power grid analysis while preserving its electrical behavior and accuracy. However, power grid reduction faces the dilemma in which a reduced grid with a small number of nodes and edges is usually not accurate enough, and an accurate reduced grid may still be too large for power grid verification. Therefore, the primary difficulty with power grid reduction is how to explicitly control the trade-off between sparsity and accuracy. Moreover, the large size of modern power grids harms the efficiency of power grid reduction methods.

In recent years, power grid reduction has been widely studied [1–4]. The moment matching method PRIMA [1] projects the explicit moment space to Krylov subspace. This method is numerically stable but is not applicable to power grids with a large number of ports. TICER [2] based on Gaussian elimination removes the nodes with low degrees efficiently but introduces too many edges for mesh-like power grids. The multigrid method [3] reduces the original power grid to a coarser grid, solves the reduced grid directly, and then maps the solution back to the original grid. Although it can fast produce significantly small grids, it is not applicable to general irregular grids and the error is difficult to control.

To help achieve fast design closure and incremental analysis for PDNs, it is useful to preserve the physical information about the port nodes that are connected to C4 bumps or load devices in the power grid. Schur complement is a widely used technique for linear systems to keep these port nodes and eliminate the non-port nodes that only have internal connections without compromise on accuracy. However, as the number of non-port nodes is becoming much larger than that of port nodes, eliminating them by Schur complement tends to generate smaller but much denser reduced grids which are still intractable for power grid analysis [5]. To this end, edge sparsification is introduced to further sparsify the reduced models. Zhao *et al.* [5] propose a resistance-based port merging scheme to eliminate nodes and leverage a sampling-based spectral graph sparsification [6] to decrease the edge density of reduced grid blocks. However, the method eliminates most of the port nodes, which renders the reduced models less practical for further analysis. Besides, how to explicitly obtain a good trade-off between sparsity and accuracy is not well explored. Recently, Yassine *et al.* [7] propose an iterative power grid reduction framework, which eliminates an entire metal layer at one time and then remove edges topologically. However, the proposed heuristic approach to sparsify the reduced model lacks error guarantee. Wang *et al.* [8] formulate edge sparsification as a convex optimization problem which explores the current range information on the port nodes to achieve better sparsity and solve it by the stochastic gradient descent (SGD) algorithm. This approach has poor runtime and therefore cannot be directly applied to large graphs. Another major drawback is that the SGD algorithm usually spends most effort optimizing the edges

ISPD '18, March 25–28, 2018, Monterey, CA,

ACM ISBN 978-1-4503-5626-8/18/03…$15.00
https://doi.org/10.1145/3177540.3178247

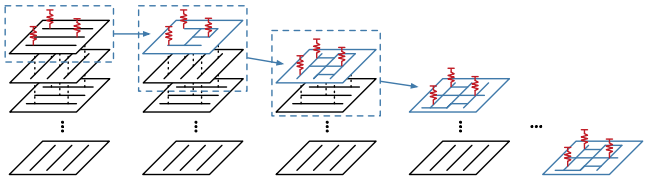

Figure 1: The proposed power grid reduction flow.

directly connected to the voltage sources under the mathematical formulation in [8] and ignores the errors at other port nodes, resulting in a significant overall error.

In this paper, we propose an efficient edge sparsification approach along with a holistic power grid reduction framework. By formulating the edge sparsification problem as a weighted sparse convex optimization problem, a greedy coordinate descent (GCD) algorithm with optimality guarantee is proposed to generate sparse and accurate solutions. To enable scalable power grid reduction, we propose an iterative reduction framework as illustrated in Figure 1, which reduces one layer at a time and eliminates all the internal layers incrementally from top to bottom. This reduction framework explicitly keeps all the port nodes in the topmost and bottommost layers. During the process of node elimination and edge sparsification for a middle layer, the nodes connected to the vias right below this layer are considered as pseudo external nodes, and their electrical properties are preserved with high accuracy. In this way, the reduced graph can be easily connected back to the next layer to process without loss of information, and the entire framework achieves a relatively small error for the final power grid model.

Our main contributions are summarized as follows:

- We propose a weighted convex optimization formulation with sparsity-inducing penalties for edge sparsification, which enables us to explicitly find a good trade-off between sparsity and accuracy. With the port nodes assigned different weights, this formulation provides an accurate control over the final error.
- We propose a novel GCD algorithm with optimality guarantee and runtime efficiency for edge sparsification. Besides, an efficient coordinate selection strategy is proposed for good convergence of GCD. A heap-based optimal coordinate search approach is proposed to improve the time complexity of iteration from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$.
- Experimental results demonstrate that the proposed approach achieves significantly better accuracy and runtime compared with previous work [8] and the traditional coordinate descent method. The proposed reduction framework achieves 98% accuracy for industrial benchmarks and preserves their sparsity.

The rest of this paper is organized as follows. Section 2 reviews the background on power grid reduction. Section 3 and Section 4 illustrate the overall reduction flow and the node elimination method. Section 5 provides a detailed explanation of the proposed edge sparsification approach. Section 7 demonstrates the effectiveness of our approaches with comprehensive results, followed by conclusion in Section 8.

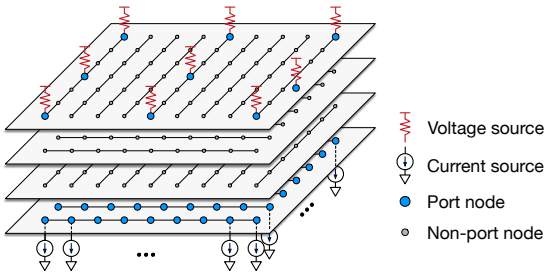

Figure 2: The multiple-layer power grid structure.

2 PRELIMINARIES

2.1 Notations

We first introduce some notations used in the paper. For any positive integers n, m, we use $[n]$ to denote the set $\{1, 2, \cdots, n\}$, and $[n, m]$ to denote the set $\{n, n+1, \cdots, m\}$. Let $\mathbf{A}^\top$ denote the transpose of a given matrix $\mathbf{A}$, $\mathbf{A}_i$ denote the i-th column vector of matrix $\mathbf{A}$ and $A_{i,j}$ denote the entry in the i-th row and j-th column of matrix $\mathbf{A}$. Let $v_{k,i}$ denote the i-th element in vector $\mathbf{v}_k$. Let $\mathbf{u} \circ \mathbf{v}$ denote the Hadamard product between two vectors $\mathbf{u}$ and $\mathbf{v}$, i.e., $\mathbf{y} = \mathbf{u} \circ \mathbf{v}$ if $y_i = u_i v_i, \forall i$.

2.2 Power Grid Model and Reduction

A resistive power grid can be modeled as a weighted undirected graph $G(V, E, w)$, where edges represent the resistors connecting nodes. Weight $w(i, j)$ denotes the physical conductance between node i and node j, and $w(i, j) = 0$ if the two nodes are not directly connected through a metal segment. For any graph G, its connectivity can be encoded as a Laplacian matrix $\mathbf{L}$, with element (i, j) given by:

$$L_{i,j} = \begin{cases} \sum_{k,k\neq i} w(i,k), & \text{if } i = j \\ -w(i,j), & \text{if } i \neq j \text{ and } e(i,j) \in E \\ 0, & \text{otherwise.} \end{cases}$$

Let $\mathbf{v}$ be the vector of node voltages and $\mathbf{i}$ be the vector of currents entering the power grid through all the nodes. The Laplacian matrix $\mathbf{L}$ can be used to characterize the linear behavior between the input current vector $\mathbf{i}$ and the voltage vector $\mathbf{v}$ by

$$\mathbf{i} = \mathbf{L}\mathbf{v}. \tag{1}$$

Figure 2 illustrates the power grid structure that uses multiple layers of metal interconnects. Some nodes in the topmost layer are connected to C4 bumps for external voltage supplies, and the nodes in the bottommost layer are connected to load transistors which are modeled as ideal current sources. Here a port node is defined as a grid node that is electrically connected to a voltage or current source. All other nodes in the power grid are non-port nodes. Power grid analysis verifies the voltages arriving at load transistors and therefore requires calculating the voltages at all the port nodes by solving the linear system in Equation (1). The Laplacian matrices of power grids are typically large and computationally prohibitive to be solved directly. Therefore, in this work, we consider the power grid reduction problem defined as follows:

Problem 1 (Power Grid Reduction)**.** Given an initial large power grid, we reduce it to a small and sparse power grid that contains

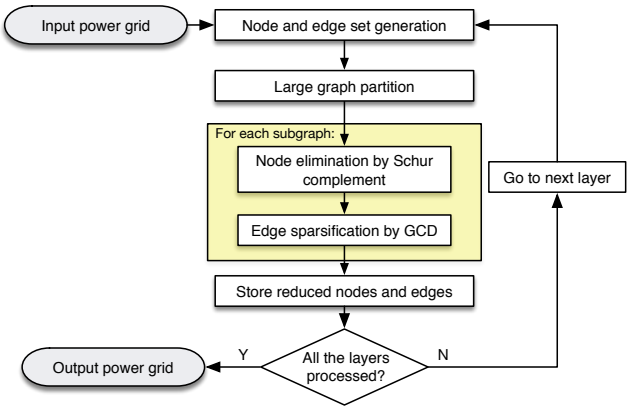

Figure 3: Algorithm flow of power grid reduction.

all the port nodes in the original power grid and preserves the accuracy in terms of the voltage drop error.

3 OVERALL FLOW

In this section, we introduce the overall flow for power grid reduction. The port nodes are required to be kept, while the non-port nodes in the middle layers will be eliminated for reduction of the grid size to the greatest extent. However, as the size of the power grid and the number of metal layers increase, the number of nodes in the middle layers increases dramatically [7]. The runtime and memory costs of removing all these non-port nodes at one time are not affordable anymore. In order to enable scalable power grid reduction, we propose an iterative layer-based node elimination and edge sparsification framework as shown in Figure 3.

The main idea is to perform node elimination and edge sparsification for one layer at a time, and remove all the middle layers incrementally from top to bottom. In each run, a new graph is created by combining the nodes and edges in the layer to be eliminated this time with the reduced graph obtained from the last run. If the combined graph is relatively large, we partition it into several small subgraphs. We explicitly keep the boundary nodes (the nodes connected to the vias below the layer) as well as the port nodes, and apply Schur complement to eliminate the rest of nodes (Section 4). Subsequently, we perform GCD to sparsify the smaller but much denser model (Section 5.2). Note that because the electrical behaviors of these boundary nodes are preserved as external nodes during Schur complement and GCD, the reduced graph can be easily combined with the lower layer through these boundary nodes for next run. Therefore, the layer-by-layer approach can achieve a small error for the reduced grid model.

4 NODE ELIMINATION

In this section, we introduce the elimination process of non-port grid nodes using the Schur complement method [9]. It should be noted that different from previous work [5], our elimination process explicitly keeps all the port nodes to be analyzed for power grid verification. Although the number of the remaining nodes is larger, the reduction result contains all the necessary physical information, which also benefits the design closure flow.

For each run of Schur complement, according to whether a node is going to be kept or removed, two different subsets of nodes are distinguished: external nodes V_{ext} and internal nodes V_{int}. Specifically, for the layer-by-layer reduction framework in Figure 3, the external nodes are the port nodes on the topmost and bottommost layers and the boundary nodes between layers. Assume $|V_{\text{ext}}| = n$ and $|V_{\text{int}}| = p$. Let $\mathbf{i}_{\text{ext}}$ ($\mathbf{i}_{\text{int}}$) denote the vector for the current injected into external (internal) nodes, and $\mathbf{v}_{\text{ext}}$ ($\mathbf{v}_{\text{int}}$) denote the column vector for voltage at external (internal) nodes. Then, we know that $i_{\text{int}}(a) = 0$ for $a \in V_{\text{int}}$, because no external current is injected into the internal node. Let $\mathbf{v} = (\mathbf{v}_{\text{ext}}, \mathbf{v}_{\text{int}})$, and $\mathbf{i} = (\mathbf{i}_{\text{ext}}, \mathbf{i}_{\text{int}}) = (\mathbf{i}_{\text{ext}}, 0 \cdots 0)$. For a resistive network with its Laplacian given by $\widetilde{\mathbf{L}}$, we can rewrite $\widetilde{\mathbf{L}}\mathbf{v} = \mathbf{i}$ as follows:

$$\begin{bmatrix} \widetilde{\mathbf{L}}_{11} & \widetilde{\mathbf{L}}_{12} \\ \widetilde{\mathbf{L}}_{12}^{\top} & \widetilde{\mathbf{L}}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{v}_{\text{ext}} \\ \mathbf{v}_{\text{int}} \end{bmatrix} = \begin{bmatrix} \mathbf{i}_{\text{ext}} \\ \mathbf{0} \end{bmatrix}, \tag{2}$$

where $\widetilde{\mathbf{L}}_{11} \in \mathbb{R}^{n \times n}$ represents the connections between the external nodes, $\widetilde{\mathbf{L}}_{12} \in \mathbb{R}^{n \times p}$ represents the connections between the internal nodes and the external nodes, and $\widetilde{\mathbf{L}}_{22} \in \mathbb{R}^{p \times p}$ represents the connections between the internal nodes.

We can derive $\mathbf{v}_{\text{int}} = -\widetilde{\mathbf{L}}_{22}^{-1}\widetilde{\mathbf{L}}_{12}^{\top}\mathbf{v}_{\text{ext}}$ from Equation (2), then for the external nodes,

$$(\widetilde{\mathbf{L}}_{11} - \widetilde{\mathbf{L}}_{12}\widetilde{\mathbf{L}}_{22}^{-1}\widetilde{\mathbf{L}}_{12}^{\top})\mathbf{v}_{\text{ext}} = \mathbf{i}_{\text{ext}}.$$

Therefore, the Schur complement of $\widetilde{\mathbf{L}}_{22}$ in $\widetilde{\mathbf{L}}$ is given by

$$\mathbf{L} = \widetilde{\mathbf{L}}_{11} - \widetilde{\mathbf{L}}_{12}\widetilde{\mathbf{L}}_{22}^{-1}\widetilde{\mathbf{L}}_{12}^{\top}.$$

Although $\mathbf{L}$ has a smaller dimension compared with $\widetilde{\mathbf{L}}$, it is much denser. Hence, it is necessary to perform edge sparsification to further reduce the number of connections in the reduced model.

5 EDGE SPARSIFICATION

In this section, we give the details of the proposed edge sparsification techniques.

5.1 Mathematical Formulation

Graph sparsification refers to the approximation of a given graph with fewer nodes or edges. For a Laplacian matrix, the number of nonzero diagonal elements (i.e., ℓ_0-norm of diagonal elements) equals the number of nodes in the graph, and the number of nonzero elements off the diagonal (i.e., ℓ_0-norm of off-diagonal elements) indicates the number of edges. Therefore, the sparsity of the Laplacian matrix can act as a measure of the graph sparsity, and the goal of edge sparsification can be regarded as sparsifying the corresponding Laplacian matrix to reduce the nonzero elements off the diagonal.

Since ℓ_0-norm is non-convex and cannot be used directly as the sparsity penalty for edge sparsification, [8] relaxes ℓ_0-norm to ℓ_1-norm, the closest convex norm to it, and proposes a convex optimization-based edge sparsification formulation with ℓ_1-norm regularization. However, the formulation in [8] may not produce solutions that preserve the current behaviors of real-life circuits. The objective in [8] implies that all the port nodes are assigned the same weight. Typically, the currents flowing into the port nodes connected to voltage sources are much larger than the currents flowing out of other port nodes connected to current sources. Thus,

the first term in the objective given by [8], the total error over all the port nodes, will be dominated by the errors associated with the port nodes connected to voltage sources. Gradient descent algorithms including SGD tend to spend considerable effort reducing these errors by repeatedly updating the edges connected to the voltage sources, and therefore cannot guarantee the accuracy of the currents flowing to current sources, which are undoubtedly more important for accurate power grid verification.

To overcome the drawback mentioned above, we propose a weighted sparse convex optimization formulation for edge sparsification. Given a Laplacian matrix $\mathbf{L} \in \mathbb{R}^{n\times n}$, a set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \in \mathbb{R}^n$, a weight vector $\mathbf{w} \in \mathbb{R}^n$ and a parameter $\lambda > 0$, we output a Laplacian matrix $\mathbf{X}$ with fewer edges by solving the following constrained problem:

$$\min_{\mathbf{X}\in\mathbb{R}^{n\times n}} \quad \frac{1}{2m}\sum_{k=1}^{m} \|((\mathbf{X}-\mathbf{L})\mathbf{v}_k)\circ\mathbf{w}\|_2^2 + \lambda\sum_{i=1}^{n} X_{i,i} \tag{3}$$

$$\text{s.t. } X_{i,j} \le 0, \qquad \forall i \ne j \tag{3a}$$

$$X_{i,j} = X_{j,i}, \qquad \forall i \ne j \tag{3b}$$

$$X_{i,i} = -\sum_{j\in[n]\setminus i} X_{i,j}, \qquad \forall i. \tag{3c}$$

Since Laplacian matrices are symmetric (Constraint (3b)) and have zero sum over rows or columns (Constraint (3c)), we can use the elements below (or above) the diagonal to represent the whole Laplacian matrix and drop these two constraints. The number of variables is also reduced from n^2 to $n(n-1)/2$. We define function $f : \mathbb{R}^{n(n-1)/2} \to \mathbb{R}$ such that

$$f(\mathbf{y}) = \frac{1}{2m}\sum_{k=1}^{m} \|((\mathbf{X}-\mathbf{L})\mathbf{v}_k)\circ\mathbf{w}\|_2^2 + \lambda\sum_{i=1}^{n} X_{i,i}, \tag{4}$$

where $X_{i,j} = X_{j,i} = y_{i,j}$, $X_{i,i} = -\sum_{j\ne i} y_{i,j}$, and $y_{i,j} \le 0$, $\forall i \in [2,n]$, $j \in [i-1]$.

Gradient methods can converge to the global solution of a convex function. The formulation proposed by [8] is convex. We next demonstrate the above function is also convex.

Lemma 1. *The function f defined in Equation (4) is strongly convex and coordinate-wise Lipschitz smooth.*

This lemma can be proved by calculating the Hessian matrix of f. The proof details are omitted here for lack of space.

The gradient of f at $y_{i,j}$ is

$$\frac{\partial f}{\partial y_{i,j}} = -\frac{1}{m}(w_i^2(\mathbf{X}-\mathbf{L})_i^\top - w_j^2(\mathbf{X}-\mathbf{L})_j^\top)\sum_{k=1}^{m}(v_{k,i}-v_{k,j})\mathbf{v}_k - 2\lambda.$$

For simplicity of notation, we define a gradient matrix $\mathbf{G} \in \mathbb{R}^{n\times n}$ as in Equation (5).

5.2 GCD-based Algorithm

In this part, we introduce the GCD-based algorithm to solve the optimization problem in Formulation (4). The GCD optimization process starts from the initial solution $\mathbf{X}^1 = \mathbf{0}$ and generates a sequence of matrices $\{\mathbf{X}^t\}_{t=1}^{T+1}$ in T iterations [10]. At each iteration, the GCD method selects the coordinate along which maximum progress can be made, and updates the coordinate by minimizing

$$\mathbf{G} = \begin{bmatrix} 0 & \frac{\partial f}{\partial y_{2,1}} & \cdots & \frac{\partial f}{\partial y_{n-1,1}} & \frac{\partial f}{\partial y_{n,1}} \\ \frac{\partial f}{\partial y_{2,1}} & 0 & \cdots & \frac{\partial f}{\partial y_{n-1,2}} & \frac{\partial f}{\partial y_{n,2}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial f}{\partial y_{n-1,1}} & \frac{\partial f}{\partial y_{n-1,2}} & \cdots & 0 & \frac{\partial f}{\partial y_{n,n-1}} \\ \frac{\partial f}{\partial y_{n,1}} & \frac{\partial f}{\partial y_{n,2}} & \cdots & \frac{\partial f}{\partial y_{n,n-1}} & 0 \end{bmatrix}. \tag{5}$$

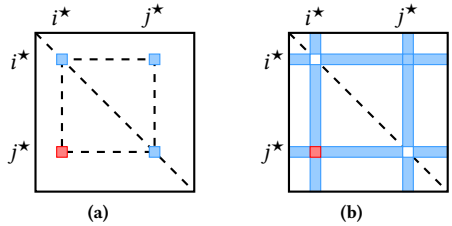

Figure 4: If select coordinate $(i^\star, j^\star)$ to update, we need to update (a) four elements in the Laplacian matrix X, and (b) two rows and columns in the gradient matrix G.

a single-variable subproblem. More specifically, for the edge sparsification problem, the initial solution is constructed as a graph that only consists of the nodes from the input graph as illustrated in Figure 5. Then the GCD method chooses the most important edge $e(i,j)$ at a time and adds the new edge in the graph (i.e., increases $w(i,j)$ from 0) or updates the weight on the existing edge (i.e., increases or decreases $w(i,j)$).

We first demonstrate how to determine the optimal coordinate to update, denoted (i^*, j^*). At each iteration GCD is supposed to select the coordinate direction with largest directional derivative; this is the same as choosing the largest (in absolute value) component of the gradient matrix $\mathbf{G}$:

$$(i^*, j^*) = \underset{(i,j)\in[n]\times[n]}{\arg\max} |G_{i,j}|. \tag{6}$$

Next, we decide how far to move along the coordinate (i^*, j^*) for updating $\mathbf{X}^t$ to $\mathbf{X}^{t+1}$. Since the objective function in Equation (4) is quadratic, we can apply exact coordinate optimization for better performance, i.e., updating the coordinate to the exact minimum point of the quadratic function as follows:

Claim 1. *Given the coordinate to update, (i,j), the next iterate for $y_{i,j}^t$ can be written as*

$$y_{i,j}^{t+1} = \underset{y_{i,j}\in\mathbb{R}_{\le 0}}{\operatorname{argmin}} f(y_{i,j}) = \min(0, y_{i,j}^t - \alpha_{i,j}), \tag{7}$$

where $\alpha_{i,j}$ is defined as

$$\left.\frac{\partial f}{\partial y_{i,j}}\right|_{y_{i,j}=y_{i,j}^t} \cdot \left(\frac{1}{m}(w_i^2 + w_j^2)\sum_{k=1}^{m}(v_{k,i}-v_{k,j})^2\right)^{-1}.$$

As illustrated in Figure 4(a), once we decide to update y_{i^*,j^*} by $\Delta y_{i^*,j^*}$, we will update four coordinates in $\mathbf{X}$, i.e., $(i^\star, i^\star)$, $(i^\star, j^\star)$,

$(j^\star, i^\star)$ and $(j^\star, j^\star)$. Therefore, for each iteration $t \in [T]$, $|\operatorname{supp}(\Delta \mathbf{X}^t)| = 4$. Besides, we update the entires in $\mathbf{G}$ in the following sense:

$$\Delta G_{r,c} = \begin{cases} \Delta y_{i^*,j^*} w_{i^*}^2 h(c, i^*, i^*, j^*), & \text{if } r = i^*, c \in [n] \\ \Delta y_{i^*,j^*} w_{j^*}^2 h(j^*, r, i^*, j^*), & \text{if } r \in [n], c = j^* \\ \Delta y_{i^*,j^*} w_{j^*}^2 h(j^*, c, i^*, j^*), & \text{if } r = j^*, c \in [n] \\ \Delta y_{i^*,j^*} w_{i^*}^2 h(r, i^*, i^*, j^*), & \text{if } r \in [n], c = i^* \\ -\Delta y_{i^*,j^*} (w_{i^*}^2 + w_{j^*}^2) h(i^*, j^*, i^*, j^*), & \text{if } r = i^*, c = j^* \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

where

$$h(i_1, j_1, i_2, j_2) = -\frac{1}{m} \sum_{k=1}^{m} (v_{k,i_1} - v_{k,j_1})(v_{k,i_2} - v_{k,j_2}). \tag{9}$$

Accordingly, the update in $\mathbf{X}$ only affects two rows (columns) in the gradient matrix $\mathbf{G}$ (Figure 4(b)). Therefore, the number of elements to update in $\mathbf{G}$ is $\mathcal{O}(n)$.

However, the GCD algorithm described above suffers from a high computational cost per iteration and sometimes even convergence failures. To achieve edge sparsification with better convergence and runtime, we propose an improved GCD algorithm. Our three key contributions are listed as follows.

Firstly, **efficient coordinate selection**. It is worth noting that Formulation (4) is constrained by $\mathbf{y} \leq \mathbf{0}$, which could make the selection rule in Equation (6) fail in some cases. For any coordinate (i, j), when other coordinates are fixed, $f(y_{i,j})$ is essentially a single-variable quadratic function. Sometimes the maximum absolute component of $\mathbf{G}$ given by Equation (6) has $y_{i^\star,j^\star} = 0$ and $G_{i^\star,j^\star} < 0$. For the quadratic function $f(y_{i^\star,j^\star})$ subject to $y_{i^\star,j^\star} \leq 0$, $y_{i^\star,j^\star} = 0$ is already the optimal solution when $G_{i^\star,j^\star} \leq 0$. Consequently, $y_{i^\star,j^\star}$ is updated to 0, and $\mathbf{X}$ and $\mathbf{G}$ are not changed. This selection strategy continues picking this coordinate in the following iterations because the gradient matrix is not changed, and therefore get stuck here. To this end, we propose an efficient coordinate selection strategy to avoid failure of GCD convergence as follows:

$$(i^*, j^*) = \underset{(i,j) \in [n] \times [n]}{\arg\max} |G_{i,j}| \text{ s.t. } G_{i,j} > 0 \text{ or } y_{i,j} \neq 0. \tag{10}$$

Secondly, **heap-based optimal coordinate search**. In the straightforward implementation, at each iteration GCD needs to traverse $\mathcal{O}(n^2)$ elements to find the coordinate with the largest directional derivative, and it is as expensive as a full gradient evaluation. We observe that the structure of the gradient in Equation (4) enables an efficient implementation of the coordinate selection rule in Equation (10). More specifically, because each node has at most n neighbors, we can track the gradient of all the variables and use a max-heap data structure to fast search the optimal coordinate. Each node in the max-heap stores the coordinate index (i, j), the variable value $y_{i,j}$, and the absolute value of the gradient component $G_{i,j}$. In this way, we can directly get the largest element in the gradient matrix from the heap in $\mathcal{O}(1)$ time. Besides, we need to update at most $\mathcal{O}(n)$ nodes in the heap at the end of each iteration. In this way, the time complexity of each iteration in GCD is reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$ as proven in Section 6.1.

Thirdly, **lookup table-based gradient update**. It is clear that the calculation for updating any element in the gradient matrix $\mathbf{G}$ according to Equation (9) has a linear dependence on the sample size m, and it is not affordable as the size of voltage samples increases. To speed up the calculation of gradient update, we define function $r : \mathbb{N}^2 \to \mathbb{R}$ such that

$$r(i, j) = -\frac{1}{m} \sum_{k=1}^{m} v_{k,i} v_{k,j}, \quad \forall i \in [2, n], j \in [i-1].$$

Thus, we have

$$h(i_1, j_1, i_2, j_2) = r(i_1, i_2) - r(i_1, j_2) - r(j_1, i_2) + r(j_1, j_2). \tag{11}$$

It inspires that we can store the set of values for r in a lookup table (LUT), and simply add or subtract them to get the value of the element to update in $\mathbf{G}$ by Equation (11). In this way, the calculation cost of gradient update for each element decreases from $\mathcal{O}(m)$ to $\mathcal{O}(1)$.

Algorithm 1 summarizes the proposed GCD method. The algorithm takes as input the sparsity control parameter λ, the Laplacian matrix $\mathbf{L}$ of the input graph, the maximum number of iterations T, and a batch of sampled voltage vectors $\mathcal{V} = \{\mathbf{v}_k\}_{k=1}^m$ [8].

6 ALGORITHM ANALYSIS

6.1 Convergence and Runtime Analysis

The goal of this section is to prove our main theoretical results, Theorem 1 and Theorem 2.

Theorem 1. *GCD (Algorithm 1) converges to the global optimum.*

Theorem 1 demonstrates the correctness of our algorithm, and the proof is similar to [11] considering that function f is strongly convex (Lemma 1).

Before we prove Theorem 2, we first show a useful claim.

Claim 2. *For each iteration $t \in [T]$, for each nonzero element of $\Delta \mathbf{G}^t$, it requires $\mathcal{O}(\log n)$ time to update the position of the corresponding entry in the max-heap.*

Proof. We use a max-heap to store $\mathbf{G}$. Once we update one certain node in this max-heap with $\mathcal{O}(n^2)$ elements, we also need to update the locations of some other nodes to make the heap still

Algorithm 1 GCD-based Edge Sparsification Algorithm

```
1: procedure GCD(λ, L, V, T)
2:     X^1 ← 0, initialize G^1 according to L and V;
3:     heap.init(X^1, G^1);
4:     for t = 1 → T do
5:         i*, j* ← heap.findMax();
6:         Compute Δy_{i*,j*};                    ▷ Equation (7)
7:         Compute ΔG^t;                          ▷ Equation (8)
8:         Compute ΔX^t by Δy_{i*,j*};
9:         X^{t+1} ← X^t + ΔX^t;
10:        heap.update(ΔX^t, ΔG^t);
11:    end for
12:    return X^{T+1};
13: end procedure
```

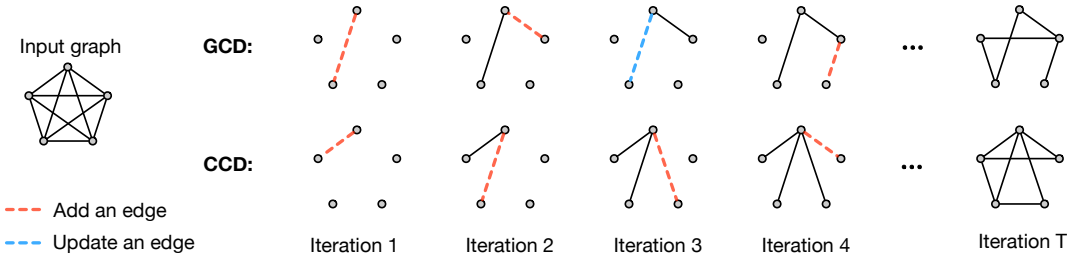

Figure 5: Illustration of the proposed GCD algorithm for edge sparsification.

valid. Because of the property of heap, we can finish the updates in $\mathcal{O}(\log n)$ time. □

Theorem 2. *GCD (Algorithm 1) takes $\mathcal{O}(Tn \log n)$ time and $\mathcal{O}(n^2)$ storage space. In particular, each iteration of GCD takes $\mathcal{O}(n \log n)$ time.*

Proof. It has been shown that at each iteration GCD takes $\mathcal{O}(1)$ time to update an element in $\mathbf{X}$ while $|\operatorname{supp}(\Delta \mathbf{X})| = 4$, and GCD takes $\mathcal{O}(1)$ time to evaluate an element in $\mathbf{G}$ and $\mathcal{O}(\log n)$ time to update it in the heap while $|\operatorname{supp}(\Delta \mathbf{G})| = \mathcal{O}(n)$. Therefore, each iteration of GCD takes $\mathcal{O}(n \log n)$ time. The overall running time is $\mathcal{O}(Tn \log n)$ if GCD runs T iterations. The storage cost of GCD comes from two parts. The first part is that we use $\mathcal{O}(n^2)$ space to store $\mathbf{X}$ in the array and $\mathcal{O}(n^2)$ space to store $\mathbf{G}$ in the heap for speedup. The second part is that after obtaining the training dataset, instead of storing each data individually, we calculate and store all possible values for r. Once r is known, it only requires $\mathcal{O}(1)$ time to compute h. □

6.2 Comparison with Other Algorithms

Various gradient descent methods can be applied to the convex optimization problem in Formulation (4). The stochastic gradient descent (SGD) algorithm chosen by [8] needs to update the entire Laplacian matrix $\mathbf{X}$ and its gradient at each iteration. Therefore, the computational cost for one SGD iteration is $\mathcal{O}(n^2)$, where n is the number of nodes in the graph, and the cost increases dramatically with the size of the input graph. Moreover, the SGD method has a slow convergence rate $\mathcal{O}(1/\epsilon^2)$. However, we observe that updating one edge in the graph only causes $\mathcal{O}(1)$ updates of elements in the Laplacian $\mathbf{X}$ and $\mathcal{O}(n)$ time to evaluate new gradients. This fact benefits the family of coordinate descent (CD) methods that minimize a single coordinate at a time.

There are several kinds of CD algorithms: cyclic gradient descent (CCD) [10] that goes through all coordinates repeatedly, randomized coordinate descent (RCD) [11] that randomly picks a coordinate each time, and GCD that selects the coordinate along which maximum progress can be made. It can be proved that these CD methods can achieve $\mathcal{O}(\log(1/\epsilon))$ convergence rate on the edge sparsification problem due to its strong convexity and coordinate-wise Lipschitz smoothness [12]. Nonetheless, their actual performance varies. Figure 5 illustrates the key difference between CCD and GCD. It is observed that the CCD (or RCD) method touches every coordinate and introduces too many edges. These edges usually have very small weights, and therefore the ℓ_1-norm regularization cost in f is small. On the contrary, GCD selects the most significant coordinates to update and only adds a small number of edges to the output graph. In this way, GCD has the appealing advantage to produce a sparser graph than the other two methods and better preserves the ℓ_0-norm sparsity. In addition, GCD has a provably faster convergence rate for this problem with smoothness and strongly convexity [12, 13]. For the above reasons, the GCD method is chosen to solve Formulation (4). Furthermore, we improve the runtime complexity of GCD at each iteration from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$ with the usage of max-heap and propose the efficient coordinate selection strategy (Equation (10)) as well as the fast LUT-based gradient update method (Equation (11)).

7 EXPERIMENTAL RESULTS

The proposed power grid reduction framework is implemented in C++ with Intel MKL library [14], and all experiments are performed on an 8-core 3.4GHz Linux machine with 32GB memory. METIS [15] is used for graph partition.

7.1 Edge Sparsification Comparison

The edge sparsification algorithm is tested on the dense synthetic benchmarks `rand1`–`rand4`. The node and edge count of each circuit are shown in columns "#Nodes" and "#Edges" in Table 1 and we normalize the maximum voltage drop in the circuit as listed in column "V_{drop}" to 100 mV by scaling the values of input current sources. For each benchmark, there is only one external voltage source and the node directly connected to it is labeled as the first node. To validate the effectiveness of the edge sparsification algorithm, we regard all the nodes as port nodes and do not perform node elimination on them.

In the first experiment, we examine the importance of the weight vector $\mathbf{w}$ in Formulation (4) on edge sparsification. We consider three weighting schemes: (1) $w_1 = w_i = 1, \forall i \in [2, n]$; (2) $w_1 = 1/n$, $w_i = 1, \forall i \in [2, n]$; (3) $w_1 = 0$, $w_i = 1, \forall i \in [2, n]$. In Scheme 1, all the port nodes are assigned the same unit weight and the formulation in this scenario is actually the same as the formulation in [8]. The GCD algorithm runs in the above schemes individually on the synthetic benchmarks for the same number of iterations and cross-validation is applied to choose the best λ. The results are listed in Table 1, where "I_{error}" gives the maximum relative current error over all the port nodes when given the voltage vector $\mathbf{v}$, and "V_{error}" gives the maximum error of voltage drop at all the nodes when given the current vector $\mathbf{i}$. As observed from Table 1, the runtime in the three schemes is nearly the same, and Scheme 2 and Scheme 3 have similarly accurate results, both of which are better

Table 1: Comparison of different weighting schemes on the synthetic benchmarks.

CKT	Bench. Stats.			Scheme 1: $w_1 = w_i = 1, \forall i \in [2, n]$				Scheme 2: $w_1 = 1/n, w_i = 1, \forall i \in [2, n]$				Scheme 3: $w_1 = 0, w_i = 1, \forall i \in [2, n]$			
	#Nodes	#Edges	V_{drop} (mV)	#Edges	I_{error} (%)	V_{error} (mV)	Time (s)	#Edges	I_{error} (%)	V_{error} (mV)	Time (s)	#Edges	I_{error} (%)	V_{error} (mV)	Time (s)
rand1	100	4×10^3	100	581	1.68	0.05	0.08	1063	1.17	0.03	0.08	1068	1.10	0.02	0.06
rand2	500	1×10^5	100	955	2.02	1.60	0.82	2690	1.22	0.01	0.88	2743	1.42	0.01	0.74
rand3	1000	4×10^5	100	1216	1.74	1.77	2.84	3821	1.04	0.02	3.26	3920	1.07	0.01	2.75
rand4	5000	1×10^7	100	4999	6.17	6.57	34.04	9436	1.88	1.34	36.00	10003	1.63	1.30	32.60
avg.				1937.8	2.90	2.50	9.45	4252.5	1.33	0.35	10.05	4433.5	1.31	0.34	9.04
ratio				**0.44**	2.21	7.35	1.05	0.96	1.02	1.03	1.11	1.00	**1.00**	**1.00**	**1.00**

Table 2: Comparison of three gradient descent methods for edge sparsification.

CKT	Ref. [8]				CCD				Ours			
	#Edges	I_{error} (%)	V_{error} (mV)	Time (s)	#Edges	I_{error} (%)	V_{error} (mV)	Time (s)	#Edges	I_{error} (%)	V_{error} (mV)	Time (s)
rand1	99	5.05	5.09	0.12	3169	4.22	0.07	0.01	1068	1.10	0.02	0.06
rand2	499	5.07	5.33	2.93	71548	3.24	0.05	1.92	2743	1.42	0.01	0.74
rand3	999	5.27	5.23	29.82	151106	2.42	0.03	19.09	3920	1.07	0.01	2.75
rand4	4999	7.00	7.52	144.43	304675	4.03	2.18	230.52	10003	1.63	1.30	32.60
avg.	1649	5.60	5.79	44.32	132624.5	3.48	0.58	62.89	4433.5	1.31	0.34	9.04
ratio	**0.37**	4.28	17.18	4.91	29.91	2.66	1.72	6.96	1.00	**1.00**	**1.00**	**1.00**

than Scheme 1. It is worth mentioning that the reduced graphs in Scheme 1 have much smaller edge counts than the other two schemes, which is aligned with our expectation: this weighting scheme emphasizes the node where the voltage source is attached due to the large current flowing from the voltage source, and the gradient descent method iteratively optimizes the edges having connections to the voltage source. We select Scheme 3 with slightly faster runtime in the following experiments.

We further study the trade-off between sparsity and accuracy for different sparsity control parameter λ during the GCD process. It is distinctly visible from Figure 6(a) that as the number of GCD iterations goes up, the voltage error decreases and the number of edges in the output graph increases as expected. We also run GCD for the same number of iterations with different λ as shown in Figure 6(b). It is shown that the smaller λ produces more accurate but denser results than the bigger λ, and therefore we can control the trade-off between the sparsity and accuracy of the reduced model by tuning the sparsity control parameter λ. Note that when λ is small enough ($\leq 10^{-4}$ in this case), it makes a negligible difference in the final accuracy and sparsity.

We compare our GCD algorithm with the SGD algorithm with its original formulation [8] and the CCD algorithm. Since the SGD algorithm in [8] was implemented in MATLAB and the voltage error was not reported, we implement it in C++ for a fair comparison. The results are summarized in Table 2, where "#Edges" denotes the edge count in the resultant graph, "V_{error}" is the maximum voltage drop error, and "Time" is the runtime in seconds. Table 2 shows that the SGD algorithm produces very sparse results but suffers from long runtime. Besides, the voltage error is not ideal because it uses only $n - 1$ edges to connect n nodes. We set the same λ for running CCD and GCD, and let them exit when the cost function value on the validation set is below the threshold or the maximum number of iterations is reached. It is clear from the edge value distributions in Figure 7, CCD produces much denser results than our GCD. As we have explained in Section 6.2, this is because CCD cycles through each coordinate and iteratively adds very small edges to the output graph; conversely, GCD only adds the most important edges. Besides, CCD has worse runtime than GCD because, even though CCD has less time complexity per iteration than GCD, it coverages much slower and wastes time on updating insignificant edges.

7.2 Reduction Framework Validation

Table 3 shows the experimental results of the proposed framework on the IBM power grid benchmarks [16]. "#L", "#V" and "#I" give the number of metal layers, voltage sources, and current sources in the circuit, respectively. "#Port nodes" gives the number of nodes that are directly connected to the external voltage/current sources, while "#Non-port nodes" gives that of nodes that only have internal connections. "#Edges" denotes the number of resistors in the power grid. "V_{drop}" denotes the maximum voltage drop in the power grid. It is important to remark that [5] deletes at least 50% of port nodes, whereas we do not allow elimination of any port nodes because all of the port nodes have important physical information for measurement and verification [7]. Therefore, the numbers of nodes and edges in [5] are not listed here. We also cannot perform a comparison with [7] due to unavailability of their benchmarks and binary.

The total number of the partitioned blocks for each circuit to facilitate reduction scalability is shown in column "#Blks", and

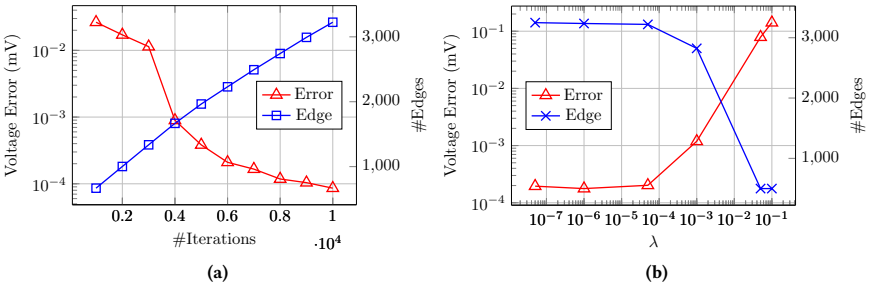

Figure 6: (a) Voltage error and edge count in the reduced circuit of `rand2` versus the number of iterations; (b) Accuracy and sparsity comparisons for different λ.

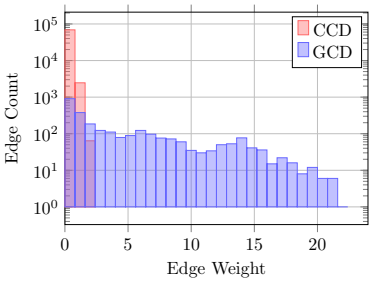

Figure 7: CCD v.s. GCD on circuit `rand2`.

Table 3: Experimental results on IBM power grid benchmarks.

CKT	Bench. Stats.							Ref. [5]		Ours						
	#L	#V	#I	#Port nodes	#Non-port nodes	#Edges	V_{drop} (mV)	V_{error} (mV)	Relative error (%)	#Blks	#Port nodes	#Non-port nodes	#Edges	V_{error} (mV)	Relative error (%)	Time (s)
ibmpg2	4	210	18963	19173	46265	106607	365.4	-	-	9	19173	0	48367	4.41	1.21	37.84
ibmpg3	5	461	100527	100988	340088	724184	181.8	1.4	0.77	68	100988	0	243011	1.32	0.73	105.71
ibmpg4	6	650	132972	133622	345122	779946	3.6	0.19	5.28	76	133622	0	284187	0.17	4.81	131.65
ibmpg5	3	177	270400	270577	311072	871182	42.9	1.2	2.80	40	270577	0	717026	0.96	2.23	122.81
ibmpg6	3	249	380742	380991	481675	1283371	114.1	2.4	2.10	56	380991	0	935322	2.23	1.96	281.25
avg.				181070	304844	753058					181070	0	445583			135.85

we observe from experiments that graph partitioning is usually necessary to process the lower layers because they contain a large number of vias or current sources. It is clear from Table 3 that the port nodes are kept in the reduced grids and all the non-port nodes are removed successfully. Moreover, the average sparsity of the reduced power grids is 0.01%. In terms of reduction accuracy, the voltage error on average is 2.19%, which is smaller than that of [5]. It is observed that as the number of remaining nodes increases, the runtime reported by [5] goes up. Our reduction framework intentionally keeps all the port nodes, and therefore the larger runtime than [5] makes sense.

8 CONCLUSION

In this work, we present a scalable power grid reduction framework. A weighted sparse convex optimization formulation for edge sparsification is proposed to reduce the number of connections in the power grid while preserving the electrical properties of the port nodes. A novel GCD method with optimality guarantee and runtime efficiency is proposed to leverage the sparsity of the reduced grids and offer a trade-off between the final accuracy and sparsity. The experimental results demonstrate that the proposed reduction framework efficiently reduces industrial power grids and preserves the accuracy and sparsity of these grids.

REFERENCES

[1] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: passive reduced-order interconnect macromodeling algorithm," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1997, pp. 58–65.

[2] B. N. Sheehan, "TICER: Realizable reduction of extracted RC circuits," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1999, pp. 200–203.

[3] H. Su, E. Acar, and S. R. Nassif, "Power grid reduction based on algebraic multigrid principles," in *ACM/IEEE Design Automation Conference (DAC)*, 2003, pp. 109–112.

[4] P. Li and W. Shi, "Model order reduction of linear networks with massive ports via frequency-dependent port packing," in *ACM/IEEE Design Automation Conference (DAC)*. ACM, 2006, pp. 267–272.

[5] X. Zhao, Z. Feng, and C. Zhuo, "An efficient spectral graph sparsification approach to scalable reduction of large flip-chip power grids," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 218–223.

[6] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM Journal on Computing (SICOMP)*, vol. 40, no. 6, pp. 1913–1926, 2011.

[7] A.-A. Yassine and F. N. Najm, "A fast layer elimination approach for power grid reduction," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, p. 101.

[8] Y. Wang, M. Li, X. Yi, Z. Song, M. Orshansky, and C. Caramanis, "Novel power grid reduction method based on $l1$ regularization," in *ACM/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[9] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[10] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, 2015.

[11] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM Journal on Optimization (SIOPT)*, vol. 22, no. 2, pp. 341–362, 2012.

[12] J. Nutini, M. Schmidt, I. H. Laradji, M. Friedlander, and H. Koepke, "Coordinate descent converges faster with the gauss-southwell rule than random selection," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1632–1641.

[13] A. Beck and L. Tetruashvili, "On the convergence of block coordinate descent type methods," *SIAM Journal on Optimization (SIOPT)*, vol. 23, no. 4, pp. 2037–2060, 2013.

[14] "Intel Math Kernel Library," http://software.intel.com/en-us/mkl.

[15] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[16] S. R. Nassif, "Power grid analysis benchmarks," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2008, pp. 376–381.