

SF-GRASS: Solver-Free Graph Spectral Sparsification

Ying Zhang *
Stevens Institute of Technology
Hoboken, New Jersey
yzhan232@stevens.edu

Zhiqiang Zhao *
Michigan Technological University
Houghton, Michigan
qzzhao@mtu.edu

Zhuo Feng
Stevens Institute of Technology
Hoboken, New Jersey
zhuo.feng@stevens.edu

ABSTRACT

Recent spectral graph sparsification techniques have shown promising performance in accelerating many numerical and graph algorithms, such as iterative methods for solving large sparse matrices, spectral partitioning of undirected graphs, vectorless verification of power/thermal grids, representation learning of large graphs, etc. However, prior spectral graph sparsification methods rely on fast Laplacian matrix solvers that are usually challenging to implement in practice. This work, for the first time, introduces a solver-free approach (SF-GRASS) for spectral graph sparsification by leveraging emerging spectral graph coarsening and graph signal processing (GSP) techniques. We introduce a local spectral embedding scheme for efficiently identifying spectrally-critical edges that are key to preserving graph spectral properties, such as the first few Laplacian eigenvalues and eigenvectors. Since the key kernel functions in SF-GRASS can be efficiently implemented using sparse-matrix-vector-multiplications (SpMV), the proposed spectral approach is simple to implement and inherently parallel friendly. Our extensive experimental results show that the proposed method can produce a hierarchy of high-quality spectral sparsifiers in nearly-linear time for a variety of real-world, large-scale graphs and circuit networks when compared with prior state-of-the-art spectral methods.

KEYWORDS

Spectral graph sparsification, spectral coarsening, graph signal processing

ACM Reference Format:

Ying Zhang, Zhiqiang Zhao, and Zhuo Feng. 2020. SF-GRASS: Solver-Free Graph Spectral Sparsification. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20)*, November 2–5, 2020, Virtual Event, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3400302.3415629>

1 INTRODUCTION

Spectral methods are playing increasingly important roles in a wide variety of graph and numerical applications [28]. Examples include scientific computing and numerical optimization [8, 13, 26], graph partitioning and data clustering [15, 22], machine learning and data mining [6, 14], as well as integrated circuit modeling, simulation

and verifications [11, 29, 30]. In particular, latest theoretical breakthroughs in spectral graph theory have led to the development of nearly-linear time spectral graph sparsification [8, 9, 16, 25] and coarsening algorithms [18, 19, 31, 33]. These techniques can efficiently produce much smaller graphs that well preserve the key spectral properties of the original graph (e.g., the first few eigenvalues and eigenvectors of the graph Laplacian), which in turn has led to much faster algorithms for solving partial differential equations (PDEs) and linear systems of equations [21, 25, 32], spectral clustering and graph partitioning [9, 15, 22, 31], and dimensionality reduction and data visualization [33].

However, prior spectral graph sparsification methods strongly rely on fast Laplacian matrix solvers that are usually challenging to implement in practice and inherently-difficult accelerate on parallel processors. For example, effective-resistance sampling-based spectral sparsification method [24] requires multiple Laplacian matrix solutions for computing each edge's leverage score, while the latest spectral-perturbation based algorithm [10] leverages a graph-theoretic algebraic multigrid (AMG) solver for computing dominant generalized eigenvectors key to estimating each edge's spectral importance. As a result, the performance (scalability) of Laplacian matrix solver can become a dominating factor in existing spectral sparsification methods. However, after decades of extensive research studies by theoretical computer scientists, it is still not clear if there exist any practically-efficient (nearly-linear time) and robust Laplacian solvers for general large-scale real-world graphs.

This paper for the first time introduces a solver-free spectral graph sparsification framework (SF-GRASS) by leveraging emerging spectral graph coarsening [31] and graph signal processing techniques [23]. Our approach first coarsens the original graph into increasingly smaller graphs while preserving the key graph spectral properties. Since spectral graph coarsening [31] can be considered as a cascade of low-pass graph filters with decreasing bandwidths, spectrally-critical edges for different ranges of eigenvalues can be effectively identified on coarse-level graphs in a stratified manner. For example, considering a coarsest graph that has only a few (e.g. two) nodes, any graph signals smoothed (low-pass filtered) from random vectors can become good approximations of the Fiedler vector corresponding to the few smallest nontrivial Laplacian eigenvalues; when such vectors are leveraged for recovering spectrally-critical edges using spectral-perturbation based approach similar to the one introduced in [8], ultra-sparse spectral graph sparsifiers preserving the smallest few eigenvalues can be efficiently extracted; by iteratively mapping sparsifiers back to each finer level, a hierarchy of spectral sparsifiers with increasing sizes can be incrementally computed for preserving increasing eigenvalues. Our results show SF-GRASS outperforms prior state-of-the-art methods for spectral sparsification considering both efficiency and solution quality. The technical contribution of this work has been summarized as follows:

*Equal contribution

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICCAD '20, November 2–5, 2020, Virtual Event, USA

© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8026-3/20/11...\$15.00
<https://doi.org/10.1145/3400302.3415629>

- (1) For the first time, we present a solver-free spectral graph sparsification framework (SF-GRASS) by leveraging emerging spectral graph coarsening [31] and graph signal processing techniques [23]. It can be implemented using simple sparse-matrix-vector multiplications and thus completely addresses the computational challenges in prior methods that strongly rely on efficient graph Laplacian solvers.
- (2) We introduce a multilevel spectral sparsification framework, which is motivated by the prior graph spectral perturbation analysis approach [8]. Such a scalable framework allows constructing a hierarchy of spectrally-reduced and sparsified graphs in nearly-linear time, which can become key to accelerating many graph-based numerical computing tasks.
- (3) By comprehensively comparing with the state-of-the-art method through extensive experiments, we show that in various numerical and graph-related applications, such as solving sparse SDD matrices, and vectorless verification of power grids, SF-GRASS can always obtain high-quality solution while achieving dramatically improved runtime scalability.

The rest of this paper is organized as follows. Section 2 provides a brief introduction to spectral graph sparsification and coarsening problems. In Section 3, a solver-free, multilevel spectral graph sparsification framework is described in detail. Section 4 demonstrates extensive experiment results for a variety of real-world, large-scale matrix and graph problems, which is followed by the conclusion of this work in Section 5.

2 BACKGROUND

2.1 Graph Laplacians and Quadratic Forms

Consider a weighted, undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega)$ with $|\mathcal{V}| = N$ and $|\mathcal{E}| = M$, where $\mathcal{V}$ denotes a set of vertices, N denotes the number of vertices, $\mathcal{E}$ denotes a set of edges, M denotes the number of edges, and ω denotes a weight function that assigns a positive weight to each edge. The adjacency matrix of graph $\mathcal{G}$ can be defined as follows:

$$\mathcal{A}_{\mathcal{G}}(p, q) = \begin{cases} \omega(p, q) & \text{if } (p, q) \in \mathcal{E} \\ 0 & \text{if otherwise} \end{cases} \quad (1)$$

The Laplacian matrix can be computed by $\mathcal{L}_{\mathcal{G}} = \mathcal{D}_{\mathcal{G}} - \mathcal{A}_{\mathcal{G}}$, where $\mathcal{D}_{\mathcal{G}}$ is an diagonal matrix with elements $\mathcal{D}_{\mathcal{G}}(p, p) = \sum_{t \neq p} \omega(p, t)$.

For any real vector $x \in \mathbb{R}^N$, the Laplacian quadratic form of graph $\mathcal{G}$ is defined as: $x^T \mathcal{L}_{\mathcal{G}} x = \sum_{(p, q) \in \mathcal{E}} \omega(p, q)(x(p) - x(q))^2$.

2.2 Spectral Graph Sparsification

Spectral sparsifier was first introduced by Spielman and Teng [25], which is a strictly stronger notation than the cut sparsifier [3, 4]. The spectral sparsifier is a weighted subgraph such that the difference of quadratic forms calculated by original graph and the sparsifier is bounded by $(1 \pm \epsilon)$, where ϵ is a constant factor. Given an undirected graph with N vertices and M edges, a nearly-linear time algorithm was introduced for building $(1 \pm \epsilon)$ spectral sparsifiers with $O(N \log N / \epsilon^2)$ edges in [24]. Later, Batson, Spielman, and Srivastava [2] proposed the algorithm for constructing the sparsifier

within $O(N / \epsilon^2)$ edges. Recently, the state-of-the-art work is given by Lee and Sun [16] that computes a $(1 \pm \epsilon)$ sparsifier with $O(qN / \epsilon)$ edges in nearly linear time $O\left(\frac{qMN^{5/4}}{\epsilon^{4+4/q}}\right)$, where q is an integer greater than 10.

Another metric for quantifying spectral similarity of two graphs has been proposed by Spielman and Teng [24]: the subgraph $\mathcal{P} = (\mathcal{V}, E)$ is a σ -spectral sparsifier of the original graph $\mathcal{G}$ if the following inequality holds for any $x \in \mathbb{R}^N$

$$\frac{1}{\sigma} x^T \mathcal{L}_{\mathcal{G}} x \leq x^T \mathcal{L}_{\mathcal{P}} x \leq \sigma x^T \mathcal{L}_{\mathcal{G}} x, \quad (2)$$

where the relative condition number is defined as $\kappa(\mathcal{L}_{\mathcal{G}}, \mathcal{L}_{\mathcal{P}}) \leq \sigma^2$. It indicates that a smaller relative condition number corresponds to a higher spectral similarity.

2.3 Spectral Graph Coarsening

Graph coarsening (reduction) not only reduces the number of edges but also aggregates nodes to form smaller number of nodes for graph approximation. It was at heuristics level until Loukas, and Vanderghenst [18, 19] developed a theoretical framework which guarantees that the spectral properties of coarsened graphs can approximate the original ones under some restricted circumstances.

3 SF-GRASS: SOLVER-FREE GRAPH SPECTRAL SPARSIFICATION

The proposed Solver-Free Graph Spectral Sparsification (SF-GRASS) framework is built upon a multilevel spectral graph coarsening scheme, which allows constructing multilevel spectral sparsifiers in nearly-linear time. Given an undirected graph $\mathcal{G} = \mathcal{G}_0$, a series of reduced graphs $\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_{l_f}$ will be generated through a spectral coarsening procedure with the corresponding node sizes denoted by $N_0, N_1, \dots, N_{l_f}$, where $N_0 > N_1 > \dots > N_{l_f}$. Once the coarsened graphs are constructed, the spectral sparsifier $\mathcal{P}_l$ of the coarsened graph $\mathcal{G}_l$ at level l will be extracted by the spectral perturbation approach introduced in [8], where $l = l_f, l_f - 1, \dots, 0$, and $\mathcal{P}_0 = \mathcal{P}$ is defined as the spectral sparsifier for the original graph $\mathcal{G}$. For the sake of simplicity, all the symbols used in this paper are summarized in Table 1.

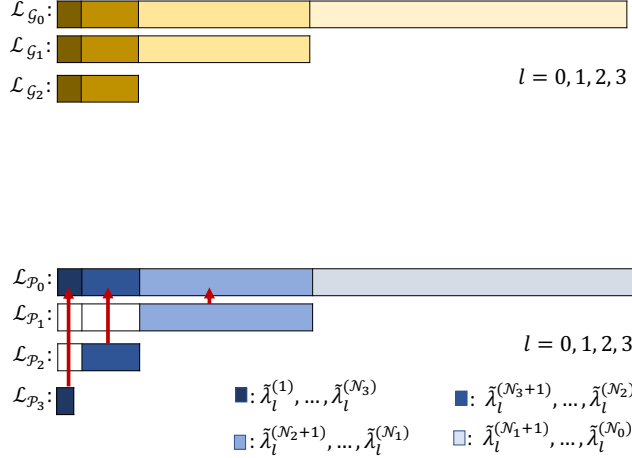
3.1 Overview of Our Approach

Recent research in graph signal processing (GSP) [23] shows that for undirected graphs the smaller eigenvalues and corresponding eigenvectors of its Laplacian are associated to the global structure (long-range distances) of the underlying graph, while the higher eigenvalues and corresponding eigenvectors encode the local structure of the graph. Since spectral sparsification aims to approximate the first few eigenvalues and eigenvectors of the original Laplacian with the minimum number of edges, it can be regarded as a low-pass filter on graphs for removing redundant edges. Spectral sparsification usually involves two steps: the first step is to generate a low-stretch spanning tree (LSST) from the original graph using star or petal decompositions [1, 7]; the next step is to identify and recover spectrally-critical off-tree edges into the LSST to drastically

Table 1: Summary of symbols used in the paper ($l = 0, 1, \dots, l_f, i = 1, \dots, N_l$).

symbols	description	symbols	description
$\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l)$	an undirected graph at level l	$\mathcal{P}_l = (\mathcal{V}_l, \mathcal{E}_l)$	the sparsifier of $\mathcal{G}_l$
$\mathcal{V}_l$	node set at level l	$\mathcal{V}_l$	node set at level l
$\mathcal{E}_l$	edge set of $\mathcal{G}_l$	$\mathcal{E}_l$	edge set of $\mathcal{P}_l$
$\omega_l(p, q)$	edge weight of node (p, q) for $\mathcal{G}_l$	$\omega_l(p, q)$	edge weight of node (p, q) for $\mathcal{P}_l$
$N_l = \mathcal{V}_l $	number of nodes	N_l	number of nodes
$M_l = \mathcal{E}_l $	number of edges in $\mathcal{G}_l$	$M_l = \mathcal{E}_l $	number of edges in $\mathcal{P}_l$
$\mathcal{L}_{\mathcal{G}_l}$	Laplacian of graph $\mathcal{G}_l$	$\mathcal{L}_{\mathcal{P}_l}$	Laplacian of graph $\mathcal{P}_l$
$\mathcal{A}_{\mathcal{G}_l}$	adjacency matrix of graph $\mathcal{G}_l$	$\mathcal{A}_{\mathcal{P}_l}$	adjacency matrix of graph $\mathcal{P}_l$
$\lambda_l^{(i)}$	eigenvalues of $\mathcal{L}_{\mathcal{G}_l}$	$\tilde{\lambda}_l^{(i)}$	eigenvalues of $\mathcal{L}_{\mathcal{P}_l}$
$u_l^{(i)}$	eigenvectors of $\mathcal{L}_{\mathcal{G}_l}$	$\tilde{u}_l^{(i)}$	eigenvectors of $\mathcal{L}_{\mathcal{P}_l}$
$S_{l-1}^{(i)}$	node aggregation set at level $l-1$ with respect to the single node i at level l		

reduce the condition number, and thereby minimizing the spec-

**Figure 2: Eigenvalue distributions of $\mathcal{L}_{\mathcal{P}_l}$**

In this work, we propose a solver-free, multilevel spectral sparsification scheme to generate a hierarchy of increasingly smaller spectral sparsifiers. As aforementioned, given an undirected graph $\mathcal{G}_0 = \mathcal{G}$, a series of coarsened graphs $\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_{l_f}$ will be generated via the multilevel spectral graph coarsening scheme introduced in [31], where $\mathcal{G}_{l_f}$ denotes the coarsest graph. It can be shown that the Laplacian of $\mathcal{G}_l$ can well preserve the low eigenvalues and eigenvectors of the finer graphs $\mathcal{G}_{l-1}, \dots, \mathcal{G}_1, \mathcal{G}_0$ [18, 19]. For example, Figure 1 shows the eigenvalue distributions of the Laplacian matrices corresponding to four consecutive coarse-level graphs, implying that the eigenvalues $(\lambda_1^{(1)}, \dots, \lambda_{N_3}^{(1)})$ of $\mathcal{L}_{\mathcal{G}_3}$ will approximately match the smallest eigenvalues of $\mathcal{L}_{\mathcal{G}_2}$, $\mathcal{L}_{\mathcal{G}_1}$ and $\mathcal{L}_{\mathcal{G}_0}$. In other words, $\mathcal{L}_{\mathcal{G}_3}$ will always approximately preserve the key spectral

(structural) properties of $\mathcal{L}_{\mathcal{G}_0}$ after coarsening. Similarly, eigenvalues $(\lambda_1^{(N_3+1)}, \dots, \lambda_{N_2}^{(N_2)})$ of $\mathcal{L}_{\mathcal{G}_2}$ will approximately match the first few eigenvalues of $\mathcal{L}_{\mathcal{G}_1}$ and $\mathcal{L}_{\mathcal{G}_0}$. Compared to $\mathcal{G}_3$ and $\mathcal{G}_2$, $\mathcal{G}_1$ will retain more local information of $\mathcal{G}_0$ by approximately preserving the moderate to large eigenvalues of $\mathcal{L}_{\mathcal{G}_0}$. Consequently, spectral coarsening is creating a hierarchy of smaller graphs that can be considered as a **cascade of low-pass graph filters** with gradually decreasing bandwidths: the finest graph always retains the highest bandwidth, whereas the coarsest graph only retains the lowest bandwidth. The theoretical proofs for the multilevel spectral preservation via graph coarsening are provided in Section 3.3.

Once the series of reduced graphs have been obtained via spectral coarsening, we will be able to effectively exploit them for extracting a hierarchy of ultra-sparse spectral sparsifiers. In the following, we show detailed steps for constructing spectral sparsifiers $\mathcal{P}_l$ at each level $l = l_f, \dots, 1, 0$, such that each $\mathcal{P}_l$ will be spectrally-similar to $\mathcal{G}_l$. Unlike the spectral coarsening step that starts at the finest-level (original) graph, SF-GRASS will start from the coarsest-level graph $\mathcal{G}_{l_f}$ and aims to approximate eigenvalues and eigenvectors (in an ascending order) through a stratified scheme: when $\mathcal{G}_{l_f}$ is sufficiently small, we can always efficiently extract the spectral sparsifier $\mathcal{P}_{l_f}$ for level l_f , leading to good approximation of the first few eigenvalues (eigenvectors); then we will map $\mathcal{P}_{l_f}$ to the finer level to facilitate the construction of the next-level sparsifier $\mathcal{P}_{l_f-1}$ so that higher eigenvalues (eigenvectors) can be approximated. The proposed approach SF-GRASS strives to incrementally construct a series of increasingly finer spectral sparsifiers, as shown in Figure 2. After iteratively applying the above procedure for all levels, the spectral sparsifier $\mathcal{P}_0$ for the original graph can be efficiently constructed to well preserve the key spectral properties of $\mathcal{G}_0$.

3.2 Spectral Coarsening via Local Embedding

As shown in Figure 3, an induced subgraph $\mathcal{F}_{l-1}^{(i)}$ can be constructed with the node aggregation set $S_{l-1}^{(i)}$ and the edge set $\mathcal{E}_{l-1}(S_{l-1}^{(i)})$ that includes edges (p, q) in $\mathcal{E}_{l-1}$ with both of the nodes p and q included in the set $S_{l-1}^{(i)}$. The induced subgraphs are strongly-connected components in $\mathcal{G}_{l-1}$, which will be aggregated into a

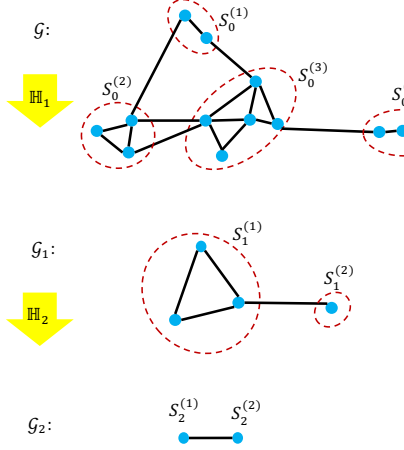


Figure 3: Graph spectral coarsening via hierarchical embedding

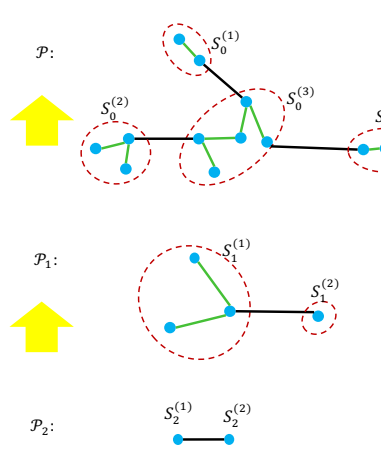


Figure 4: Sparsifier backward mapping

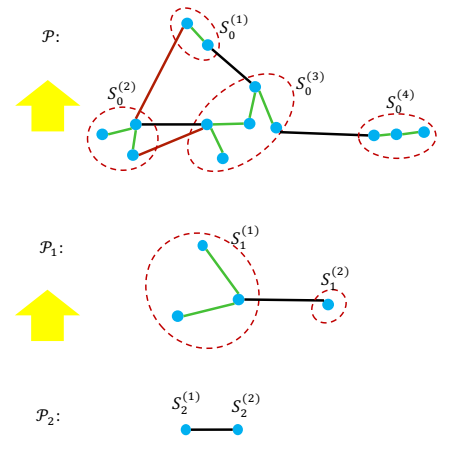


Figure 5: Spectrally-critical edge identification

single node of the coarser graph. Next, we create a node-mapping matrix $\mathbb{H}_l$ that allows constructing $\mathcal{G}_l$ given the finer graph $\mathcal{G}_{l-1}$ with the following equation:

$$\mathcal{L}_{\mathcal{G}_l} := \mathbb{H}_l^\top \mathcal{L}_{\mathcal{G}_{l-1}} \mathbb{H}_l^+, \text{ and } x_l := \mathbb{H}_l x_{l-1}, \text{ for } l = 1, 2, \dots, l_f, \quad (3)$$

where $\mathbb{H}_l \in \mathbb{R}^{N_l \times N_{l-1}}$, $x_l \in \mathbb{R}^{N_l \times 1}$, and $\mathbb{H}_l^\top, \mathbb{H}_l^+, \mathbb{H}_l^\mp$ denote the transpose, pseudoinverse, and transposed pseudoinverse of $\mathbb{H}_l$, respectively. $\mathbb{H}_l, \mathbb{H}_l^+$ can be created as follows [18]:

$$\mathbb{H}_l(i, p) = \begin{cases} \frac{1}{|S_{l-1}^{(i)}|} & \text{if node } p \in S_{l-1}^{(i)} \\ 0 & \text{if otherwise} \end{cases} \quad (4)$$

$$\mathbb{H}_l^+(p, i) = \begin{cases} 1 & \text{if node } p \in S_{l-1}^{(i)} \\ 0 & \text{if otherwise} \end{cases} \quad (5)$$

When creating a coarsening framework, the core task is to cluster the graph into aggregation sets so that we can define matrix $\mathbb{H}_l$. To preserve important spectral properties (e.g., the first few eigenvalues and eigenvectors of the graph Laplacian) on the coarsened graphs, one naive approach is to embed the original graph into a K -dimensional space using the first K nontrivial Laplacian eigenvectors. Then, the nodes that are close to each other in the embedding space can be aggregated for forming a coarser graph. However, such a scheme requires calculating the Laplacian eigenvectors, which will be extremely expensive for large graphs.

To achieve good efficiency, SF-GRASS leverages a linear-time local spectral embedding scheme based on low-pass filtering of random graph signals [6, 31]. Let $X_l = [x_l^{(1)}, x_l^{(2)}, \dots, x_l^{(K)}]$, where $x_l^{(i)} \in \mathbb{R}^{N_l \times 1}$ denote the test vectors computed by applying a few steps of Gaussian-Seidel relaxations for solving the linear system of equations $\mathcal{L}_{\mathcal{G}_l} x_l^{(i)} = 0$ for $i = 1, \dots, K$ with K initial random vectors that are orthogonal to the all-one vector [17]. The above smoothing procedure can be regarded as a low-pass filtering process applied to K random graph signals. The resultant K smoothed test vectors will consist of linear combinations of the first few Laplacian

eigenvectors, and thus can be subsequently leveraged for spectral graph embedding.

Since modern graph signal processing (GSP) based filtering functions are dominated by SpMV operations that are massively-parallel-friendly, the local spectral embedding scheme can be effectively accelerated on modern parallel computing platforms, such as CPUs, GPUs, and FPGAs [12, 27].

3.3 Spectral Similarity Between Coarse Graphs

After finding the $\mathbb{H}_l$, will Eq (2) still hold between $\mathcal{G}_l$ and $\mathcal{G}_{l-1}$? If yes, how does the smaller graph $\mathcal{G}_l$ spectrally preserve the finer graph $\mathcal{G}_{l-1}$? How will $\mathbb{H}_l$ affect the spectral properties of $\mathcal{G}_l$? For the above questions, we provide detailed explanation through the following comprehensive theoretical analysis. Let $\lambda_l^{(1)}, \lambda_l^{(2)}, \dots, \lambda_l^{(N_l)}$, and $u_l^{(1)}, u_l^{(2)}, \dots, u_l^{(N_l)}$ denote the non-decreasing eigenvalues and their corresponding eigenvectors for $\mathcal{L}_{\mathcal{G}_l}$. The **restricted spectral similarity** [18] is defined as follows

$$\frac{1}{\sigma_{l-1}} \|x_{l-1}\|_{\mathcal{L}_{\mathcal{G}_{l-1}}} \leq \|x_l\|_{\mathcal{L}_{\mathcal{G}_l}} \leq \sigma_{l-1} \|x_{l-1}\|_{\mathcal{L}_{\mathcal{G}_{l-1}}}, \forall x_{l-1} \in U_{l-1}^k, \quad (6)$$

where $U_{l-1}^k = [u_{l-1}^{(1)}, u_{l-1}^{(2)}, \dots, u_{l-1}^{(k)}]$ includes the first k eigenvectors of $\mathcal{L}_{\mathcal{G}_{l-1}}$. The restricted spectral similarity can also be denoted as the $(U_{l-1}^k, \sigma_{l-1})$ -spectral similarity. If $\mathcal{L}_{\mathcal{G}_{l-1}}$ and $\mathcal{L}_{\mathcal{G}_l}$ are $(U_{l-1}^k, \sigma_{l-1})$ -similar, we have

$$\gamma_1 \lambda_{l-1}^{(i)} \leq \lambda_l^{(i)} \leq \gamma_2 \frac{(1+\epsilon)^2}{1-\tau\epsilon^2} \lambda_{l-1}^{(i)}, \quad i = 1, \dots, N_l \quad (7)$$

where $\tau = \lambda_{l-1}^{(k)} / \lambda_{l-1}^{(2)}$, $\epsilon = (\sigma_{l-1}^2 - 1) / (\sigma_{l-1}^2 + 1)$ and $\sigma_{l-1} \leq (\frac{1+\sqrt{\tau}}{1-\sqrt{\tau}})^{\frac{1}{2}}$. γ_1, γ_2 will be the smallest and largest eigenvalues of $(\mathbb{H}_l \mathbb{H}_l^\top)^{-1}$. Therefore, the spectral similarity between $\lambda_{l-1}^{(i)}$ and $\lambda_l^{(i)}$ can be controlled by σ_{l-1} . The canonical angles between the principal eigenspace of $\mathcal{L}_{\mathcal{G}_{l-1}}$ and $\mathcal{L}_{\mathcal{G}_l}$ are defined as follows:

$$\Theta(U_{l-1}^k, \mathbb{H}_l^\top U_l^k) = \arccos(U_{l-1}^{\top} \mathbb{H}_l^\top U_l^k U_l^{\top} \mathbb{H}_l U_{l-1}^k)^{-\frac{1}{2}}. \quad (8)$$

Consequently, a smaller canonical angles implies a higher similarity between two eigenspaces.

3.4 Sparsifier Backward Mapping

We aim to iteratively find spectral sparsifiers for achieving desired spectral similarity or relative condition numbers $\kappa(\mathcal{L}_{\mathcal{G}}, \mathcal{L}_{\mathcal{P}}) = \sigma_l^2$, $l = l_f, \dots, 0$. To this end, we will first extract an LSST, and subsequently, add extra off-tree edges to form the sparsifier at the coarsest level. Next, sparsifiers at finer levels can be obtained by iteratively mapping the coarser sparsifiers via the procedures illustrated in Figures 3 and 4, where $\mathcal{P}$, $\mathcal{P}_1$, $\mathcal{P}_2$ denote the sparsifiers of $\mathcal{G}$, $\mathcal{G}_1$, $\mathcal{G}_2$, respectively. $\mathcal{P}_l$ and $\mathcal{G}_l$ share the same aggregation sets $S_{l-1}^{(i)}$. $\mathcal{P}_l$ is the series of coarsened graphs for $\mathcal{P}$ where $l = 1, 2$. In the following, we describe how to map two types of edges in the proposed sparsifier backward mapping procedure:

- **Inner-cluster Edges.** We can conveniently locate all the inner-cluster nodes and edges within each aggregation set according to $\mathbb{H}_l$ and $\mathcal{G}_{l-1}$. Since each of the aggregation sets is a strongly-connected component, we can extract an LSST for each aggregation set (highlighted by the red dash line in Figure 4). Since each aggregation set size is pretty small, LSSTs can be well approximated using all-pairs shortest-path trees or maximum spanning trees (MSTs).
- **Inter-cluster Edges.** We could get all the inter-cluster edges between these aggregation sets in the graph $\mathcal{G}_{l-1}$ and only keep the edges with the largest weights in $\mathcal{P}_{l-1}$. As a result, all the aggregation sets will be connected through inter-cluster edges in $\mathcal{P}_{l-1}$, forming a good spectral sparsifier for $\mathcal{G}_{l-1}$ (as shown in Figure 4).

3.5 Spectrally-Critical Edges Identification

To further improve the spectral approximation in sparsifiers, additional spectrally-critical off-tree edges need to be identified and added into the latest sparsifiers. Specifically, $O(\frac{M_l \log \log N_l}{\sigma^2})$ spectrally-critical off-tree edges need to be added into LSSTs to obtain a σ -similar spectral sparsifier $\mathcal{P}_l$ for $\mathcal{G}_l$. Let $\tilde{u}_l^{(i)}$ denote the i -th eigenvector of $\mathcal{L}_{\mathcal{P}_l}$ corresponding to the i -th eigenvalue $\tilde{\lambda}_l^{(i)}$ that satisfies:

$$\mathcal{L}_{\mathcal{P}_l} \tilde{u}_l^{(i)} = \tilde{\lambda}_l^{(i)} \tilde{u}_l^{(i)}, \quad (9)$$

then we have the following eigenvalue perturbation analysis:

$$(\mathcal{L}_{\mathcal{P}_l} + \delta \mathcal{L}_{\mathcal{P}_l}) (\tilde{u}_l^{(i)} + \delta \tilde{u}_l^{(i)}) = (\tilde{\lambda}_l^{(i)} + \delta \tilde{\lambda}_l^{(i)}) (\tilde{u}_l^{(i)} + \delta \tilde{u}_l^{(i)}), \quad (10)$$

where a perturbation $\delta \mathcal{L}_{\mathcal{P}_l}$ that includes a new edge connection is applied to $\mathcal{L}_{\mathcal{P}_l}$, resulting in perturbed eigenvalues and eigenvectors $\tilde{\lambda}_l^{(i)} + \delta \tilde{\lambda}_l^{(i)}$ and $\tilde{u}_l^{(i)} + \delta \tilde{u}_l^{(i)}$ for $i = 1, \dots, N_l$, respectively. Keeping only the first-order terms leads to:

$$\mathcal{L}_{\mathcal{P}_l} \delta \tilde{u}_l^{(i)} + \delta \mathcal{L}_{\mathcal{P}_l} \tilde{u}_l^{(i)} = \tilde{\lambda}_l^{(i)} \delta \tilde{u}_l^{(i)} + \delta \tilde{\lambda}_l^{(i)} \tilde{u}_l^{(i)}. \quad (11)$$

Expressing $\delta \tilde{u}_l^{(i)}$ in terms of the original eigenvectors $\tilde{u}_l^{(j)}$ for $j = 1, \dots, N_l$ leads to:

$$\delta \tilde{u}_l^{(i)} = \sum_{j=1}^{N_l} \alpha_j \tilde{u}_l^{(j)}. \quad (12)$$

Substituting (12) into (11) leads to:

$$\mathcal{L}_{\mathcal{P}_l} \sum_{j=1}^{N_l} \alpha_j \tilde{u}_l^{(j)} + \delta \mathcal{L}_{\mathcal{P}_l} \tilde{u}_l^{(i)} = \tilde{\lambda}_l^{(i)} \sum_{j=1}^{N_l} \alpha_j \tilde{u}_l^{(j)} + \delta \tilde{\lambda}_l^{(i)} \tilde{u}_l^{(i)}. \quad (13)$$

Multiplying $\tilde{u}_l^{(i)\top}$ to both sides of (13) results in:

$$\begin{aligned} \tilde{u}_l^{(i)\top} \mathcal{L}_{\mathcal{P}_l} \sum_{j=1}^{N_l} \alpha_j \tilde{u}_l^{(j)} + \tilde{u}_l^{(i)\top} \delta \mathcal{L}_{\mathcal{P}_l} \tilde{u}_l^{(i)} \\ = \tilde{\lambda}_l^{(i)} \tilde{u}_l^{(i)\top} \sum_{j=1}^{N_l} \alpha_j \tilde{u}_l^{(j)} + \delta \tilde{\lambda}_l^{(i)} \tilde{u}_l^{(i)\top} \tilde{u}_l^{(i)}. \end{aligned} \quad (14)$$

Since $\tilde{u}_l^{(i)}$ for $i = 1, \dots, N_l$ are unit-length, mutually-orthogonal eigenvectors, we have:

$$\begin{aligned} \tilde{u}_l^{(i)\top} \mathcal{L}_{\mathcal{P}_l} \sum_{j=1}^{N_l} \alpha_j \tilde{u}_l^{(j)} &= \alpha_i \tilde{u}_l^{(i)\top} \mathcal{L}_{\mathcal{P}_l} \tilde{u}_l^{(i)}, \\ \tilde{\lambda}_l^{(i)} \tilde{u}_l^{(i)\top} \sum_{j=1}^{N_l} \alpha_j \tilde{u}_l^{(j)} &= \alpha_i \tilde{u}_l^{(i)\top} \tilde{\lambda}_l^{(i)} \tilde{u}_l^{(i)}. \end{aligned} \quad (15)$$

Then the eigenvalue perturbation due to $\delta \mathcal{L}_{\mathcal{P}_l}$ is given by:

$$\delta \tilde{\lambda}_l^{(i)} = \omega(p, q) \left(\tilde{u}_l^{(i)\top} e_{p,q} \right)^2. \quad (16)$$

Therefore, if an edge (p, q) has a large $\omega(p, q) \left(\tilde{u}_l^{(i)\top} e_{p,q} \right)^2$ value, it is considered **spectrally critical** to $\tilde{\lambda}_l^{(i)}$. In other words, including this edge into the latest sparsifier will significantly perturb the Laplacian eigenvalue $\tilde{\lambda}_l^{(i)}$ and eigenvector $\tilde{u}_l^{(i)}$. Construct a subspace matrix for K -dimensional spectral graph embedding using the first K Laplacian eigenvectors as follows:

$$U = \left[\tilde{u}_l^{(1)}, \tilde{u}_l^{(2)}, \dots, \tilde{u}_l^{(K)} \right], \quad (17)$$

then the overall K -eigenvalue perturbation Δ_K becomes

$$\Delta_K = \sum_{i=1}^K \delta \tilde{\lambda}_l^{(i)} = \omega(p, q) (U^\top e_{p,q})^2, \quad (18)$$

which is similar to the effective-resistance edge sampling probability [24] computed by $P = \omega(p, q) R_{p,q}^{eff}$ when $K = N_l$, considering the close connection between the spectral embedding distance $(U^\top e_{p,q})^2$ and the effective resistance distance computed by

$$R_{p,q}^{eff} = \left(U_{eff}^\top e_{p,q} \right)^2, \text{ where } U_{eff} = \left[\frac{\tilde{u}_l^{(1)}}{\sqrt{\tilde{\lambda}_l^{(1)}}}, \dots, \frac{\tilde{u}_l^{(N_l)}}{\sqrt{\tilde{\lambda}_l^{(N_l)}}} \right]. \quad (19)$$

Instead of computing exact Laplacian eigenvectors for identifying spectrally-critical edges, we will adopt the local spectral embedding approach described in Section 3.2 for approximately computing the spectral embedding distance $(U^\top e_{p,q})^2$. Consequently, we can compute spectral criticalities for all candidate edges in linear time using only a few times of sparse-matrix-vector multiplications.

Algorithm 1 $\mathcal{P} = \text{SF-GRASS}(\mathcal{G}, \sigma)$

```

1:  $\mathcal{P}_l = \emptyset$  for  $l = 0, \dots, l_f$ ;
2:  $[\mathcal{G}_1, \dots, \mathcal{G}_{l_f}; \mathbb{H}_1, \dots, \mathbb{H}_{l_f}] = \text{Multilevel\_spectral\_graph\_reduction}(\mathcal{G})$ ;
3:  $\mathcal{P}_{l_f} = \mathcal{G}_{l_f}$ ;  $l = l_f$ 
4: while  $l \geq 1$  do
5:   for each node  $i \in \mathcal{V}_l$  do
6:     Find the induced subgraph  $\mathcal{F}_{l-1}^{(i)}$  formed by the nodes set  $\mathcal{S}_{l-1}^{(i)}$  in  $\mathcal{G}_{l-1}$ ;
7:     Extract the LSST  $\mathcal{T}_{l-1}^{(i)}$  of  $\mathcal{F}_{l-1}^{(i)}$ ;
8:      $\mathcal{P}_{l-1} = \mathcal{P}_{l-1} \cup \mathcal{T}_{l-1}^{(i)}$ ;
9:   end for
10:  for each edge  $(p, q) \in E_l$  do
11:    Node set  $s_p = \mathcal{S}_{l-1}^{(p)}$ , node set  $s_q = \mathcal{S}_{l-1}^{(q)}$ ;
12:    Find the edge with maximum weight between  $s_p$  and  $s_q$ , and add the edge into  $\mathcal{P}_{l-1}$ ;
13:  end for
14:  Embed  $\mathcal{G}_{l-1}$  to  $K$ -dimensional space  $X_{l-1} = [x_{l-1}^{(1)}, x_{l-1}^{(2)}, \dots, x_{l-1}^{(K)}]$ ;
15:  For each subgraph edge  $(p, q) \in (E_{l-1} - E_{l-1})$ , calculate the edge distortion  $d(p, q) \propto \omega_{l-1}(p, q) (X_{l-1}^\top e_{p,q})^2$ ;
16:  Include top few edges with large distortion into  $\mathcal{P}_{l-1}$ ;
17:   $l = l - 1$ 
18: end while
19: let  $\mathcal{P} = \mathcal{P}_0$  and return graph  $\mathcal{P}$ ;

```

3.6 Algorithm Flow and Complexity

Algorithm 1 shows the algorithm flow for the proposed SF-GRASS framework. The complexity of spectral graph coarsening is $O(|\mathcal{E}_l|)$ for each level, the complexity of backward graph mapping procedure is $O(|\mathcal{E}_l|)$ for each level l , and the complexity of off-subgraph identification is $O(|\mathcal{E}_l|)$ for each level l . If the spectral coarsening step will produce $O(\log |\mathcal{V}|)$ graphs with a fixed coarsening ratio for two consecutive levels, the overall runtime complexity of SF-GRASS is nearly linear for an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega)$.

4 EXPERIMENTAL RESULTS

The proposed spectral sparsification algorithm has been implemented in Matlab and C++. The test cases used in this paper have been selected from a great variety of matrices that have been used in circuit simulation, finite element analysis, machine learning, and data mining applications. If the original matrix is not a graph Laplacian, it will be converted into a graph Laplacian by setting each edge weight using the absolute value of each nonzero entry in the lower triangular matrix; if edge weights are not available in the original matrix file, a unit edge weight will be assigned to all edges. All of our experiments have been conducted using a single CPU core of a computing platform running 64-bit RHEW 7.2 with a 2.67GHz 12-core CPU and 50 GB memory. Several test cases have been tested in the experiments.

4.1 SF-GRASS for Spectral Graph Sparsification

Table 2 shows the spectral graph sparsification results on various graphs when comparing to the state-of-the-art sparsification tool GRASS¹ [8–10], where $\mathcal{N}$ (M) represents the number of nodes

(edges) in the original graph; T_{grass} denotes the the sparsifier construction time using GRASS; T_r denotes the multilevel graph coarsening time; T_{spar} denotes the multilevel sparsifier construction time by SF-GRASS; $|\mathcal{E}_{off}|$ denotes the number of off-tree edges added for forming the final sparsifier from the initial spanning-tree sparsifier. $\kappa(\mathcal{L}_{\mathcal{G}}, \mathcal{L}_{\mathcal{P}})$ denotes the final relative condition number between the Laplacians of the original graph $\mathcal{G}$ and the sparsifier $\mathcal{P}$. $\kappa(\mathcal{L}_{\mathcal{G}}, \mathcal{L}_{\mathcal{S}})$ denotes the relative condition number between the Laplacians of the original graph $\mathcal{G}$ and the initial spanning-tree sparsifier $\mathcal{S}$ generated by SF-GRASS. When the original graph is relatively small, the runtime of GRASS and SF-GRASS are comparable. However, SF-GRASS can become substantially faster when confronting greater graph sizes and densities since GRASS requires a Laplacian solver to compute dominant generalized eigenvectors while SF-GRASS does not.

Figure 6 and Figure 7 show the changes of the relative condition numbers with increasing number of off-tree edges added to the initial spanning-tree sparsifier for PPI and fe_4elt graphs. It can be observed that smaller condition number can be achieved with greater number of off-tree edges included, which indicates that very desired (flexible) tradeoffs between graph complexity and approximation quality can be obtained.

4.2 SF-GRASS for PCG Iterations

The spectral sparsifier generated by the proposed algorithm is leveraged as a preconditioner in a PCG solver for solving linear system equations $Ax = b$. The preconditioner matrix is factorized with Cholmod solver [5]. The right-hand-side (RHS) vector b is generated randomly, while the solver is set to converge to an accuracy level $\|Ax - b\|/\|b\| < 1E - 3$ for all test cases. We compare the PCG solver using SF-GRASS with the direct method and the PCG solver using GRASS, as shown in Table 3. T represents the total runtime for each solver, $iter$ represents the number of iterations, and $relres$ is the relative residue. It shows that SF-GRASS is the fastest among all three solvers, which can achieve up to 167X and 98X speedups when comparing to direct solver and PCG solver using GRASS, respectively. Also, SF-GRASS has achieved a faster convergence rate than GRASS.

Figure 8 shows the runtime scalability of GRASS and SF-GRASS on different sizes of 3D mesh graphs. It indicates that SF-GRASS scales linearly with the graph size, which is more scalable than GRASS, especially on larger and denser graphs, such as 3D mesh graphs.

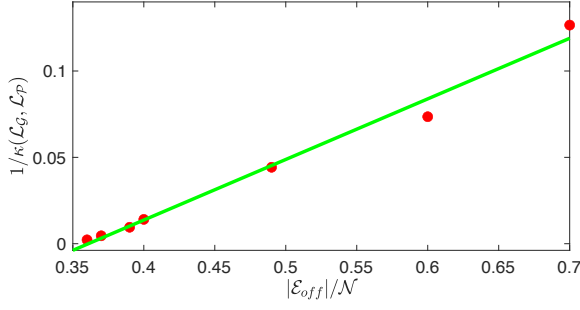
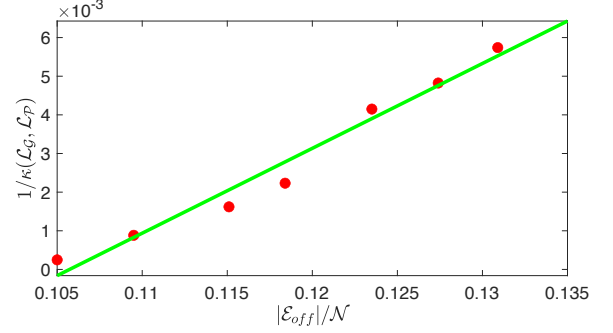
Figure 9 shows the convergence rate of PCG solver when using the sparsifiers generated by GRASS and SF-GRASS on a 3D thermal grid with $1.0E5$ nodes and $3.0E5$ edges. To generate the sparsifiers, we add $0.018\mathcal{N}$ off-tree edges to the sparsifiers for both SF-GRASS and GRASS settings. It shows that SF-GRASS has achieved a better convergence rate than GRASS.

Figure 10 shows the edge sampling probabilities on each coarse level graph of a 3D thermal mesh grid, where $0.08\mathcal{N}$ and $0.32\mathcal{N}$ off-tree edges have been added to the initial spanning-tree sparsifier $\mathcal{P}$ across all levels, respectively. $|\mathcal{E}_{l,add}|$ denotes the number of off-subgraph edges added on level l graph, and $|\mathcal{E}_{l,off}|$ denotes the total off-subgraph edges on level l . As shown, the edges on

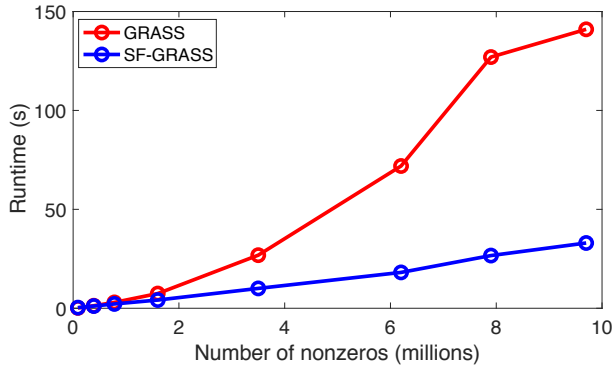
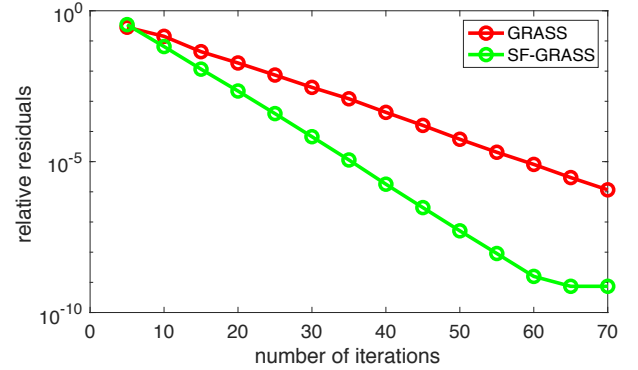
¹ <https://sites.google.com/mtu.edu/zhuofeng-graphspar/home>

Table 2: Comparison of spectral sparsification results between SF-GRASS and GRASS.

Test cases	N	M	GRASS			SF-GRASS					
			T_{grass}	$\frac{ \mathcal{E}_{off} }{N}$	$\kappa(\mathcal{L}_G, \mathcal{L}_P)$	T_r	T_{spar}	$\frac{ \mathcal{E}_{off} }{N}$	$\kappa(\mathcal{L}_G, \mathcal{L}_P)$	$\kappa(\mathcal{L}_G, \mathcal{L}_S)$	$\frac{\kappa(\mathcal{L}_G, \mathcal{L}_S)}{\kappa(\mathcal{L}_G, \mathcal{L}_P)}$
fe_4elt	1.1E4	3.3E4	0.10s	21.6%	50	0.01s	0.16s	19.3%	51	6.36E4	1.25E3X
fe_ocean	1.4E5	4.1E5	2.21s	9.0%	271	1.37s	0.17s	9.7%	276	2.16E6	7.82E3X
Gmat_airfoil	4.3E4	1.2E4	0.03s	7.4%	131	0.08s	0.06s	7.5%	99	1.10E4	1.11E2X
G2_circuit	1.5E5	2.9E5	1.24s	3.0%	306	1.26s	0.12s	3.1%	423	1.43E5	3.39E2X
Gmat_laplacian_0.25M	2.5E5	7.4E5	4.89s	10.0%	238	2.47s	0.26s	10.0%	357	6.31E6	1.77E4X

**Figure 6: Condition number change with number of off-tree edges added for fe_4elt graph****Figure 7: Condition number change with number of off-tree edges added for PPI graph****Table 3: Results of the PCG solver for SF-GRASS.**

Test cases	N	M	directed solver	PCG for SF-GRASS				PCG for GRASS			
				T	$\frac{ \mathcal{E}_{off} }{N}$	iter	relres	T	$\frac{ \mathcal{E}_{off} }{N}$	iter	relres
Thermal1	2.5E4	7.2E4	1.12s	0.13s	2.7%	3	5.6E-4	0.26s	2.6%	7	4.5E-4
Thermal2	1.0E5	2.9E5	5.95s	0.79s	2.9%	3	5.9E-4	3.31s	2.9%	7	5.7E-4
Thermal3	2.0E5	5.9E5	19.42s	3.41s	3.2%	4	1.8E-4	9.44s	2.9%	8	6.3E-4
Thermal4	4.0E5	1.2E6	72.47s	8.01s	3.1%	3	6.0E-4	63.52s	2.9%	6	8.6E-4
Thermal5	9.0E5	2.6E6	974.87s	21.28s	3.1%	3	6.2E-4	919.35s	3.0%	6	8.2E-4
Thermal6	1.6E6	4.6E6	3637.79s	42.02s	3.1%	3	6.1E-4	1695.92s	3.0%	6	5.6E-4
Thermal7	2.0E6	5.9E6	2787.94s	69.62s	3.2%	3	6.2E-4	7932.55s	3.0%	6	6.2E-4
Thermal8	2.5E6	7.2E6	9341.48s	56.36s	3.2%	3	6.1E-4	5476.88s	3.0%	6	6.2E-4

**Figure 8: Runtime scalability comparison: GRASS vs SF-GRASS (3D meshes of different sizes)****Figure 9: Convergence rate comparison for a 3D thermal grid: GRASS vs SF-GRASS**

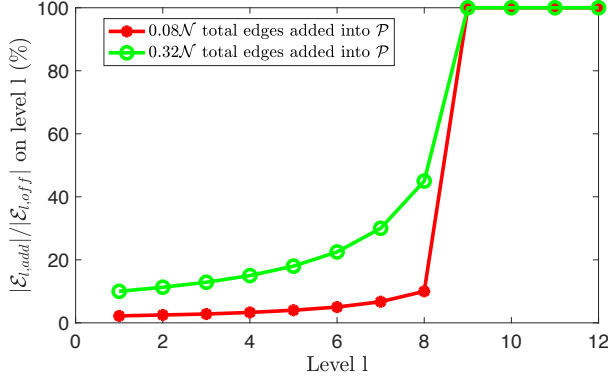
coarser graphs have been assigned with higher sampling probabilities since they will have greater effective-resistances and thus be more important for retaining the original graph structural (spectral) properties.

4.3 SF-GRASS for Vectorless Verification

We also evaluated SF-GRASS for vectorless power grid verifications using industrial power grid designs with different sizes [20], as shown in Table 4. The vectorless verification framework is adopted

Table 4: Results of the proposed vectorless power grid integrity verification method.

CKT	Power Grid Specs.			Single Level			Multilevel w/o Sparsifier				Multilevel w/ Sparsifier				
	$N_{\#}$	$C_{\#}$	$L_{\#}$	T_{chol}	T_{sol}	T_{lp}	T_{chol}	T_{sol}	T_{lp}	$Err(\%)$	T_{chol}	T_{sol}	T_{lp}	$Err(\%)$	κ
<i>ibmpg3</i>	8.5E5	9.0E4	2	11.91s	0.40s	1.63s	15.55s	0.51s	0.05s	1.76%	1.70s	0.04s	0.03s	1.85%	239
<i>ibmpg4</i>	1.0E6	1.0E5	2	14.97s	0.53s	1.67s	20.99s	0.73s	0.14s	2.71%	1.68s	0.04s	0.10s	3.52%	1136
<i>ibmpg5</i>	1.1E6	1.6E5	2	8.48s	0.27s	2.08s	12.58s	0.43s	0.22s	2.43%	2.10s	0.05s	0.17s	2.52%	218
<i>ibmpg6</i>	1.7E6	1.7E5	2	12.24s	0.36s	3.21s	17.76s	0.51s	0.20s	1.36%	3.05s	0.07s	0.06s	3.83%	248
<i>thupg1</i>	5.0E6	5.0E5	2	72.44s	2.07s	9.40s	290.36s	4.06s	28.31s	1.73%	11.96s	0.25s	4.93s	3.31%	464
<i>thupg2</i>	9.0E6	9.0E5	2	955.00s	4.53s	33.40s	1142.46s	6.59s	14.26s	4.20%	52.75s	0.50s	9.90s	2.64%	465

**Figure 10: Edge sampling probabilities for each coarse-level graph of Thermal3, where total $0.08N$ and $0.32N$ number of edges are added into $\mathcal{P}$, respectively**

from [30]. “Single Level”, “Multilevel w/o Sparsifier”, and “Multilevel w/ Sparsifier” denote the verification methods using single level (direct), multilevel grids w/o sparsification and w/ sparsification using SF-GRASS, respectively. Note that we choose to apply sparsified power grid on each level generated by SF-GRASS during the verification process. $N_{\#}$, $C_{\#}$, $L_{\#}$ are the numbers of grid nodes, current sources, and hierarchical levels, respectively. T_{chol} , T_{sol} and T_{lp} denote the runtime for Cholesky factorizations, adjoint sensitivity calculation using matrix factors and the total LP solution time including all levels, respectively. Err denotes the relative error of maximum voltage drop compared to the single-level method, and κ denotes the relative condition number.

For all test cases, it is observed that matrix factorization, sensitivity calculation, and LP solving can be significantly accelerated using the SF-GRASS while maintaining excellent accuracy. The “Multilevel w/o Sparsifier” method is always the slowest due to the fast-growing matrix densities at coarse levels.

5 CONCLUSIONS

For the first time, we present a solver-free spectral graph sparsification approach (SF-GRASS) by leveraging emerging spectral graph coarsening and graph signal processing (GSP) techniques. Such a scalable framework allows constructing a hierarchy of spectrally-reduced and sparsified graphs in nearly-linear time, which can become key to accelerating many graph-based numerical computing tasks. The proposed spectral approach is simple to implement and inherently parallel friendly. Our extensive experimental results show that the proposed method can produce a hierarchy of high-quality spectral sparsifiers in nearly-linear time for a variety of

real-world, large-scale graphs and circuit networks when compared with prior state-of-the-art spectral methods.

6 ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under Grants CCF-1350206 (CAREER), CCF-2021309 (SHF), and CCF-2011412 (SHF).

REFERENCES

- [1] I. Abraham and O. Neiman. Using petal-decompositions to build a low stretch spanning tree. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*, pages 395–406. ACM, 2012.
- [2] J. Batson, D. Spielman, and N. Srivastava. Twice-Ramanujan Sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.
- [3] A. A. Benczúr and D. R. Karger. Approximating st minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (STOC)*, pages 47–55. ACM, 1996.
- [4] A. A. Benczúr and D. R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.
- [5] T. Davis. CHOLMOD: sparse supernodal Cholesky factorization and update/downdate. [Online]. Available: <http://www.cise.ufl.edu/research/sparse/cholmod/>, 2008.
- [6] C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. *International Conference on Learning Representations (ICLR)*, 2020.
- [7] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008.
- [8] Z. Feng. Spectral graph sparsification in nearly-linear time leveraging efficient spectral perturbation analysis. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, 2016.
- [9] Z. Feng. Similarity-aware spectral sparsification by edge filtering. In *Design Automation Conference (DAC)*, 2018 55nd ACM/EDAC/IEEE, pages 1–6. IEEE, 2018.
- [10] Z. Feng. Grass: Graph spectral sparsification leveraging scalable spectral perturbation analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.
- [11] L. Han, X. Zhao, and Z. Feng. An Adaptive Graph Sparsification Approach to Scalable Harmonic Balance Analysis of Strongly Nonlinear Post-Layout RF Circuits. *Computer-Aided Design of Integrated Circuits and Systems*, *IEEE Transactions on*, 34(2):173–185, 2015.
- [12] C. Hong, A. Sukumaran-Rajam, B. Bandyopadhyay, J. Kim, S. E. Kurt, I. Nisa, S. Sabhlok, Ü. V. Çatalyürek, S. Parthasarathy, and P. Sadayappan. Efficient sparse-matrix multi-vector product on gpus. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pages 66–79. ACM, 2018.
- [13] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford. An Almost-linear-time Algorithm for Approximate Max Flow in Undirected Graphs, and Its Multicommodity Generalizations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 217–226. SIAM, 2014.
- [14] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv e-print, arXiv:1609.02907*, 2016.
- [15] J. R. Lee, S. O. Gharan, and L. Trevisan. Multiway spectral partitioning and higher-order cheeger inequalities. *Journal of the ACM (JACM)*, 61(6):37, 2014.
- [16] Y. T. Lee and H. Sun. An SDP-based Algorithm for Linear-sized Spectral Sparsification. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 678–687, New York, NY, USA, 2017. ACM.
- [17] O. Livne and A. Brandt. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.
- [18] A. Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

- [19] A. Loukas and P. Vndergheynst. Spectrally approximating large graphs with smaller graphs. In *International Conference on Machine Learning*, pages 3243–3252, 2018.
- [20] S. R. Nassif. *IBM power grid benchmarks*. [Online]. Available: <http://dropzone.tamu.edu/pli/PGBench/>, 2008.
- [21] R. Peng. *Algorithm Design Using Spectral Graph Theory*. PhD thesis, Carnegie Mellon University, 2013.
- [22] R. Peng, H. Sun, and L. Zanetti. Partitioning well-clustered graphs: Spectral clustering works. In *Proceedings of The 28th Conference on Learning Theory (COLT)*, pages 1423–1455, 2015.
- [23] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vndergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.
- [24] D. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.
- [25] D. Spielman and S. Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.
- [26] D. Spielman and S. Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014.
- [27] M. Steinberger, R. Zayer, and H.-P. Seidel. Globally homogeneous, locally adaptive sparse matrix-vector multiplication on the gpu. In *Proceedings of the International Conference on Supercomputing*, page 13. ACM, 2017.
- [28] S.-H. Teng. Scalable algorithms for data and network analysis. *Foundations and Trends® in Theoretical Computer Science*, 12(1–2):1–274, 2016.
- [29] X. Zhao, L. Han, and Z. Feng. A Performance-Guided Graph Sparsification Approach to Scalable and Robust SPICE-Accurate Integrated Circuit Simulations. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 34(10):1639–1651, 2015.
- [30] Z. Zhao and Z. Feng. A spectral graph sparsification approach to scalable vectorless power grid integrity verification. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 68. ACM, 2017.
- [31] Z. Zhao and Z. Feng. Effective-resistance preserving spectral reduction of graphs. In *Proceedings of the 56th Annual Design Automation Conference (DAC) 2019*, page 109. ACM, 2019.
- [32] Z. Zhao, Y. Wang, and Z. Feng. SAMG: Sparsified Graph Theoretic Algebraic Multigrid for Solving Large Symmetric Diagonally Dominant (SDD) Matrices. In *Proceedings of the 36th International Conference on Computer-Aided Design (ICCAD)*. ACM, 2017.
- [33] Z. Zhao, Y. Wang, and Z. Feng. Nearly-linear time spectral graph reduction for scalable graph partitioning and data visualization. *arXiv e-print, arXiv:1812.08942*, 2018.