CSP: Comprehensively-Sparsified Preconditioner for Efficient Nonlinear Circuit Simulation

Yuxuan Zhao
SSSLab, Dept. of CST, China
University of Petroleum-Beijing

Xiaoyu Yang
SSSLab, Dept. of CST, China
University of Petroleum-Beijing

Yinuo Bai
SSSLab, Dept. of CST, China
University of Petroleum-Beijing

Lijie Zeng
SSSLab, Dept. of CST, China
University of Petroleum-Beijing

Dan Niu
School of Automation, Southeast University

Weifeng Liu
SSSLab, Dept. of CST, China
University of Petroleum-Beijing

Zhou Jin
SSSLab, Dept. of CST, China
University of Petroleum-Beijing

ABSTRACT

Solving sparse linear systems dominates the simulation time for nonlinear integrated circuits. Developing an effective preconditioner is crucial for accelerating the iterative solver when dealing with large-scale circuit matrices, yet this remains a challenging task. In this paper, we introduce an efficient sparsification-based preconditioner method that significantly reduces the number of iterations needed in iterative solvers. Our method transforms nonlinear components into symmetric Laplacian matrices, enabling the inclusion of both nonlinear and linear elements in the sparsification process. We then intersect the generated sparsifier with the original Modified Nodal Analysis (MNA) matrix to further reduce the sparsity, thereby decreasing preconditioner factorization time. Furthermore, we enhance the parallelization of the spectral sparsification strategy by integrating block RMQ and point exclusivity algorithms, which substantially speeds up preprocessing. Experiment results demonstrate acceleration of 2.50x, 13.46x, 2.18x on average in serial, 3.72x, 24.23x, 3.86x on average in parallel, and memory reduction of 21.3%, 21.7%, 88.0% on average when solving nonlinear circuit matrices compared to the state-of-the-art solver GPSCP, feGRASS, and direct solver KLU, respectively.

KEYWORDS

Nonlinear circuit simulation, Sparse linear solver, Spectral sparsification, Preconditioned iterative solver

1 INTRODUCTION

High-performance sparse linear solvers emerge as pivotal tools to facilitate rapid and accurate simulation and verification of transistor-level circuits. Along with the fast development of semiconductors, modern integrated circuits have been extremely complex, consisting of hundreds of millions of components, causing linear solvers to consume more time and memory resources for simulation [2].

Compared with direct solvers, e.g., sparse LU factorization [1, 33–36], iterative solvers [9, 27, 37] demonstrate superior performance in handling large-scale linear systems due to their reduced memory requirements and facilitation of parallel acceleration [4]. Many studies [5–8, 10] have thoroughly investigated iterative methods for solving circuit matrices. However, the effectiveness of iterative solvers is heavily dependent on the choice and design of preconditioners, as they directly impact the convergence rate and efficiency of the iterative process [11]. Identifying an effective preconditioning technique becomes particularly paramount.

Despite numerous efforts to explore preconditioning approaches for circuit matrices (e.g., [12, 15, 32]), efficiently handling circuit matrices remains a significant challenge. The reason is that the circuit matrix often exhibits highly heterogeneous properties, such as variable resistances, capacitances, and inductances, leading to complex sparsity patterns and spectral characteristics. Additionally, the presence of nonlinear elements, such as diodes and transistors, further complicates the preconditioning process by introducing nonlinearity and potentially causing numerical instability.

Recently, the spectral graph sparsification methods have demonstrated promising results for accelerating power grid simulation. It produces an ultra-sparse sub-graph with spectral similarity characteristics, which can be served as an effective preconditioner for the iterative sparse linear solver, e.g., preconditioned Conjugate Gradient [14, 16–19]. However, the spectral graph sparsification algorithm can only be performed for symmetric positive definite matrices, which hinders its further application to large-scale nonlinear circuit simulation matrices.

To harness the potential of spectral graph sparsification, several researchers have endeavored to explore sparsification strategies tailored for general asymmetric matrices in nonlinear circuit simulation. A straightforward idea is to segregate the nonlinear and linear elements and apply the sparsification of the spectral graph solely to the linear portion [20]. Subsequently, the resulting sparsifier is combined with the matrix of nonlinear components to construct the preconditioner. However, when nonlinear devices dominate in a design, factorizing the support-circuit preconditioner matrix can

ICCAD '24, October 27–31, 2024, New Jersey, NY, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1077-3/24/10…$15.00
https://doi.org/10.1145/3676536.3676794

be expensive. Zhao et al. proposed a GPSCP method in [21], which distinguishes between the directed and undirected components of the linearized circuit, focusing its graph sparsification efforts solely on the latter. It effectively tackles more general circuit simulation tasks. However, it did not introduce all edges into the computation, which resulted in destroying the original sparsity.

Therefore, in this paper, we propose a comprehensively-sparsified operation to producing high-quality preconditioner, thereby expediting the solver of general asymmetric circuits. To enable all elements in the circuit involved in the sparsification, that is, both linear and nonlinear elements, we need to tackle the following challenges. (1) How to handle nonlinear elements to ensure that all elements are accessible for spectral sparsification operations. (2) How can consider the effects of symmetric and asymmetric edges on relative condition numbers and spectral similarity. (3) How to achieve significant speed-up in the spectral graph sparsification process. Our method efficiently addresses these challenges. Specifically, (1) we achieve a sparser preconditioner matrix through a comprehensive graph sparsification strategy, thereby diminishing preconditioner decomposition time. (2) We enhance the spectral similarity of preconditioners by intersecting the sparsification outcomes with the original Modified Nodal Analysis (MNA) matrix, thereby effectively reducing solver iteration count. (3) We significantly accelerate pre-processing performance by parallelizing the spectral sparsification operation using the block RMQ algorithm and the point exclusion algorithm.

This paper primarily presents the following contributions.

- We effectively extend the spectral graph sparsification to the asymmetric circuit matrix. To the best of our knowledge, this is the first paper to enable comprehensive sparsification including both linear and nonlinear elements for circuit simulation.
- We define effective weights for both directed and undirected edges in our graph sparsification process, leading to a notable reduction in the relative condition number of our preconditioners.
- We substantiate the relationship between our produced sparsifier preconditioner and the original asymmetric matrices through mathematical derivation, highlighting their impact on the number of iterations required by the GMRES algorithm.
- We leverage parallelization techniques such as block RMQ and point exclusion algorithms to markedly enhance the efficiency of spectral sparsification.
- CSP outperforms state-of-the-art spectral methods GPSCP and feGRASS by an average speedup of 2.50x, 13.46x and memory reduction of 21.3%, 21.7% for serving as the preconditioner of GMRES.

2 BACKGROUND

2.1 Nonlinear Circuit Simulation

In electronic circuit simulations involving nonlinear characteristics, effective modeling can be achieved by employing the following nonlinear differential algebraic equation (DAE): $F(x) = f(x(t)) + \frac{dq(x(t))}{dt} + u(t)$ [21, 22]. To solve this DAE, the typical approach involves first discretizing it with a numerical integration algorithm, such as backward Euler, etc. Then we need to solve a series of nonlinear algebraic equations at each discrete time-point. The Newton-Raphson iterative method is usually employed at each time point. For nonlinear circuits, the Jacobian matrix derived from the conductance matrix G and the capacitance matrix C usually exhibits asymmetry, primarily influenced by the MOS transistors, bipolar transistors, and control sources, e.g., voltage-controlled current sources, etc. As the complexity of the circuit increases rapidly, the size of asymmetric Jacobian matrices grows to millions or even billions[3]. In tackling such challenges, preconditioner-based iterative solvers prove to be more advantageous than direct methods.

2.2 Preconditioned Iterative Solver for Asymmetric Circuit Matrix

Numerous established iterative methods have been employed for solving large-scale asymmetric circuit matrices, such as CG [23], BICGSTAB [24], and GMRES [25]. With ongoing semiconductor scaling and increasing device integration, the size of the matrix has expanded to billions of entries, presenting significant computational challenges for efficient and accurate circuit analysis. To enhance the efficiency of iterative solvers used for solving large linear systems, preconditioners are often employed. The iteration complexity of preconditioned iterative solvers is typically proportional to the relative condition number λ between the original circuit matrix and the preconditioner, which is defined as follow:

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}}, \tag{1}$$

$$Au = \lambda Bu, \tag{2}$$

where A and B are denoted as the original matrix and the preconditioner respectively, λ is the generalized eigenvalue and u is the generalized eigenvector.

2.3 Graph Sparsification Preconditioner

The main idea of graph sparsification is to make G_P sparser while ensuring that G_P is similar to G_A. Graph sparsification usually involves two key points: 1) Extract a maximum spanning tree from G_T, 2) Recover a few spectrally critical off-tree edges from G_T and add them into the spanning tree to form preconditioned sparse graph G_P. For the first part, the simple maximum-spanning tree can be used in practical problems. For the second part, [29] employs a sampling technique reliant on effective resistance. [18, 21] use approximate dominant generalized eigenvectors to identify and recover the edge critical to the tree, which can enhance spectral similarity while recovering the same number of edges, thereby substantially reducing the required iteration steps.

3 OUR PROPOSED CSP FRAMEWORK

3.1 Overview

In this paper, we introduce CSP, a comprehensive graph sparsification preconditioning method to better enhance the solver performance. It is different from the conventional approach shown in Fig. 1(B), (C) and (D), which involves segregating the undirected and directed segments of the graph and conducting graph sparsification solely on the undirected portion and then the sparsified subgraphs are merged with the original graph to produce preconditioners that have a large number of extra edges.

CSP begins by converting the entire original graph to an undirected graph, as shown in Fig. 1(E), so that all edges can be involved in the sprasification. Then, to ensure comprehensive integration of

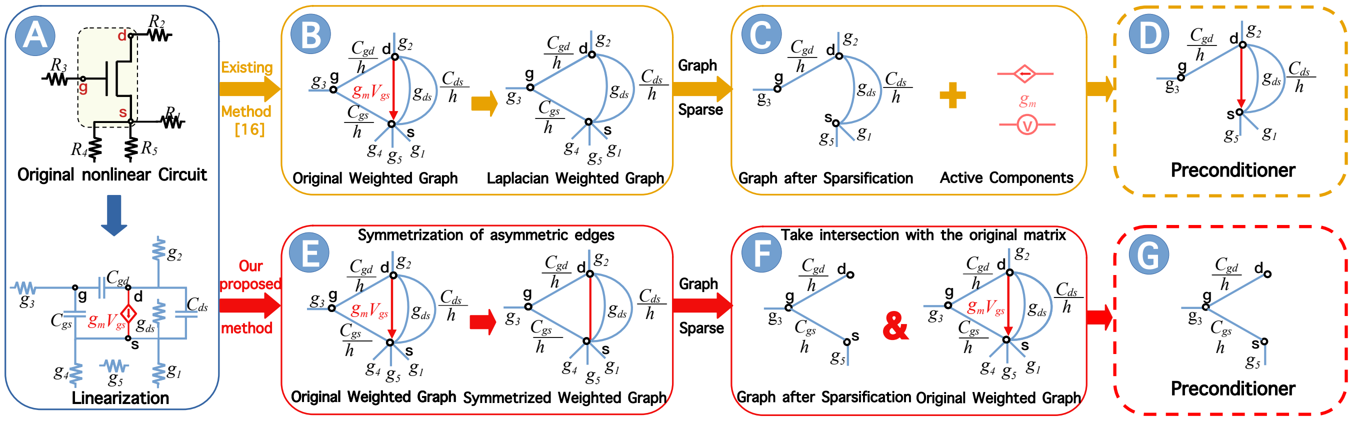

Figure 1: Comparison of preconditioner construction algorithms: conventional methods range from A to D, while our proposed approach spans from E to G. After linearizing the circuit (step A), the circuit can be represented as a graph comprising undirected and directed edges. Conventional method, GPSCP, separates the undircted edges (step B) and perform spectral sparsification. After that, the directed edges are added to obtain the final preconditioner (step C). Conversely, in our proposed CSP approach, directed edges are considered as undirected edges, and the entire graph is subjected to spectral sparsification (step E). Then preconditioner is produced by executing an "AND" operation (step F) between the sparsifier and the original graph.

Algorithm 1 The CSP Preconditioned Iterative Solver.

```
Input: Original asymmetric matrix $A$
Output: Iterative solution results
1: Set up the collection of $A$ edges: $e_A$
2: Initialize array $Issym$
3: for $i = 0$ to $nnzR_A$ do
4:     Assign value to $Issym[i]$ based on the features of the edge $e_A[i]$.
5: end for
6: for $i = 0$ to $nnzR_A$ do
7:     if $e_A[i]! = 1$ then Make symmetric $e_A[i]$ by the large value symmetry rule
8:     end if
9: end for
10: Perform spectral sparsification and obtain sparse submatrix $A_s$
11: Set up the collection of $A_s$ edges: $e_s$
12: for $i = 0$ to $nnzR_{A_s}$ do
13:     if $e_s[i]$ is not in $e_A$ then Delete $e_s[i]$
14:     end if
15:     if $e_s[i]$ is in $e_A$ then Reuse the elements of the original matrix $A$
16:     end if
17: end for
18: Obtain preconditioner $A_p$
19: Run GMRES operation
```

edge information from the original graph, CSP executes a sparsification process on the corresponding undirected graph, employing a directed graph-oriented sparsification strategy to further enhance the spectral similarity. Subsequently, the sparsified subgraph is combined with the original graph using the "AND" operation (that is, perform intersection between the produced sparsifier and the original matrix need to be solved) to produce preconditioners, as shown in Fig. 1(F). Furthermore, we propose an efficient parallel technique to reduce the pre-processing overhead of performing sparsification, thereby enhancing its practicality.

3.2 Analysis about Preconditioner for Asymmetric Circuit Matrix

To better design the sparsification algorithm, we first analyze how the preconditioner influences the solver's performance. Assume the original matrix is L_G, the laplacian matrix L_{G_u} after symmetrization, the preconditioned matrix L_{S_u} after sparsification.

An effective preconditioner should significantly reduce the condition number of a matrix. A lower condition number indicates that the solution is less sensitive to noise and input errors, thereby enhancing the stability of the equation. For the symmetrized matrix L_{G_u} and its sparsified preconditioner matrix L_{S_u}, the theory of spectral graph sparsification suggests that L_{S_u} can effectively reduce $\kappa(L_{S_u}^{-1} \cdot L_{G_u})$. According to the definition of the condition number, for any n-by-n matrix A, we have $\cos(A, I) = \frac{\text{tr}(A)}{\sqrt{n}|A|}$ and $\frac{\sqrt{n}}{\cos^2(A,I)} \leq \kappa(A)$. Based on these equations, we can find that

$$\kappa(L_{S_u}^{-1} \cdot L_{G_u}) \geq \frac{\sqrt{n}}{\cos^2(L_{S_u}^{-1} \cdot L_G + L_{S_u}^{-1} \cdot L_G^T, I)}. \tag{3}$$

Further, it can be concluded that,

$$n^{-3/4}(\kappa(L_{S_u}^{-1} \cdot L_G + L_{S_u}^{-1} \cdot L_G^T))^{1/2} \geq \frac{|L_{S_u}^{-1} \cdot L_G|}{2\max(\text{tr}(L_{S_u}^{-1} \cdot L_G), \text{tr}(L_{S_u}^{-1} \cdot L_G^T))}. \tag{4}$$

this leads to $\kappa(L_{S_u}^{-1} \cdot L_G) \leq \delta \cdot \kappa(L_{S_u}^{-1} \cdot L_{G_u})$, where

$$\delta \leq 2 \cdot n^{-3/4} \cdot \frac{\max(\lambda_{\max}(L_{S_u}^{-1} \cdot L_G), \lambda_{\max}(L_{S_u}^{-1} \cdot L_G^T))}{\lambda_{\min}(L_{S_u}^{-1} \cdot L_G)}. \tag{5}$$

As n increases, δ can be considered less than 1. Consequently, $\kappa(L_{S_u}^{-1} \cdot L_G) < \kappa(L_{S_u}^{-1} \cdot L_{G_u})$. This indicates that the preconditioner not only reduces the condition number of L_{G_u} but also that of L_G. Therefore, for iterative methods like GMRES, L_{S_u} can serve as an effective preconditioner for L_G.

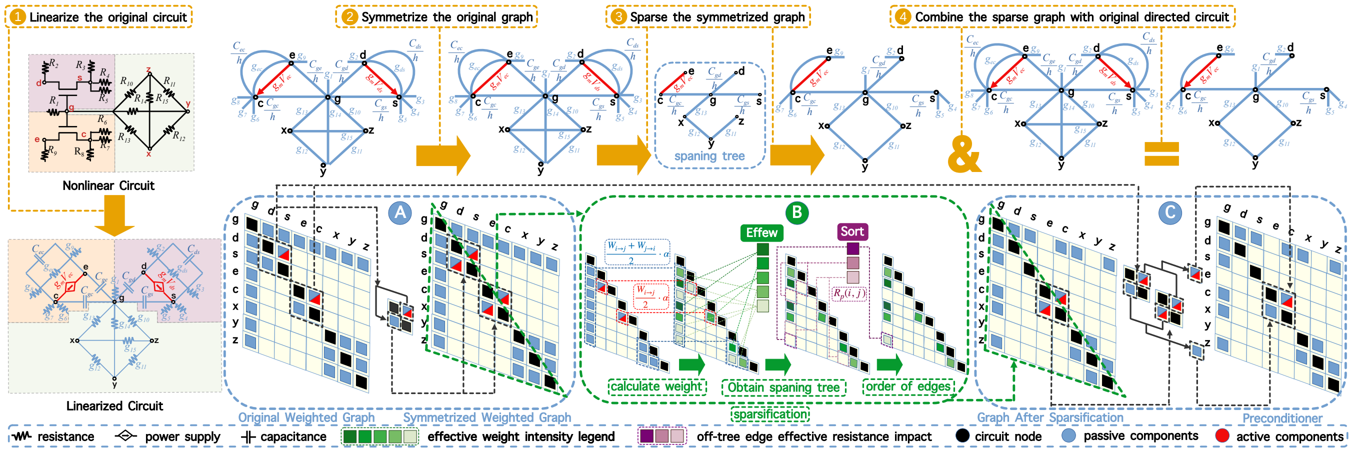
1 Linearize the original circuit
2 Symmetrize the original graph
3 Sparse the symmetrized graph
4 Combine the sparse graph with original directed circuit
Nonlinear Circuit
Linearized Circuit
spaning tree
Original Weighted Graph
Symmetrized Weighted Graph
Effew
Sort
calculate weight
Obtain spaning tree
order of edges
sparsification
Graph After Sparsification
Preconditioner
resistance
power supply
capacitance
effective weight intensity legend
off-tree edge effective resistance impact
circuit node
passive components
active components

Figure 2: Nonlinear components are firstly transformed into an equivalent linear circuits (Part A). The linear circuit simplifies to a graph, with undirected edges for resistive components and directed edges for active components. Directed edges become undirected for graph sparsification. Sparsification involves building a maximum-weight spanning tree and sorting off-tree edges by their impact on effective resistances (Part B). Asymmetric and symmetric elements affect the matrix differently, resulting in specific sparsification for directed edges. The representative support graph results from an "AND" operation between the sparsified and original graphs (Part C). It acts as an effective precondition to improve iterative methods.

3.3 Operations about Asymmetric Part

Motivated by the previous derivation in Section 3.2, we employ a "symmetric with large edges" strategy. This approach retain that edges with higher weights in the spanning tree and decrease their average stretch, consequently minimizing the number of iterations required for GMRES to reach a solution. In our work, CSP employs a "symmetry with large edges" strategy. As shown in Fig. 2(A), upon acquiring the circuit matrix corresponding to the original diagram, we identify and mark its asymmetric elements (lines 3-5 in Algorithms 1). For such asymmetric elements, we symmetrize them with the value of the larger element and update the values of the diagonal elements accordingly (lines 6-10 in Algorithms 1). Through the aforementioned method, we establish a symmetric matrix as equivalent to one possessing Laplacian properties, thereby enabling its integration into the spectral sparsification operation.

3.4 Spectral Sparsification for Asymmetric Matrices

Definition of Effective Weights of Edges. In order to reduce the number of relative conditions in the spanning tree, we consider more information about the asymmetric edges in the construction of the spanning tree T. Consider the largest mismatch of the volume of the vertex between the undirected graph G_u and the spanning tree T:

$$\mathrm{VolMis}_{\max} = \max_{a\in V} \frac{\mathrm{Vol}_{G_u}(a)}{\mathrm{Vol}_T(a)}, \tag{6}$$

where $Vol_{G_u}(a)$ is the volume of vectex a in graph, same as $\mathrm{Vol}_T(a)$. When the largest mismatch of the vertex volume is smaller, it typically indicates a reduced relative condition number. When the largest mismatch of the vertex volume is smaller, it typically indicates a reduced relative condition number. So we have $\lambda_{max}(L_{G_u}, L_T) \geq textVolMis_{\max}$. Thus, the largest mismatch can be constrained from below by $\mathrm{VolMis}_{\max} \geq \frac{\sum_{i\in V} Vol_{G_u}(a)}{\sum_{a\in V} Vol_{MWEST}(a)}$, where $Vol_{MWEST}(a)$ represents the volume of vertex a.

Algorithm 2 The Proposed Sparsification Algorithm.

```
Input: Symmetric positive definite matrix $G_{A_l} = (V, E, w)$.
Output: Completely sparse matrix $G_{A_s}$.
 1: Initialize the arrays and variables.
 2: Set the sparsity $\alpha$.
 3: for $e(a,b) \in E$ do
 4:     if $e(a,b)$ is a undirected side then
 5:         $effwt(a,b) = 2 * w(a,b)\frac{Vol_e}{Layer_a + Layer_b}$.
 6:     end if
 7:     if $e(a,b)$ is a directed side then
 8:         $effwt(a,b) = (w_{a\to b} + w_{b\to a})\frac{Vol_e}{Layer_a + Layer_b}$.
 9:     end if
10: end for
11: Obtain the graph $G' = (V, E, w_{effwt})$.
12: Execute the Kruskal algorithm on $G'$ to get $T$.
13: Compute effective resistances and sort off-tree edges to get an edge list
    $e_{Offtree}$.
14: for $e(a,b) \in e_{offtree}$ do
15:     if then$\alpha$ edges have been added into $G_{A_s}$
16:         Break
17:     end if
18:     if $e(a,b)$ is not marked then
19:         Add $e(a,b)$ into $G_{A_s}$
20:         Identify the outer edges of the tree resembling $e(a,b)$
21:         based on $a$, $b$, and the vertices reached in the BFS.
22:     end if
23: end for
```

Based on the analyses provided above, it is evident that information regarding both node degree and distance to a root node should be taken into account when constructing the tree. The edge weights generated anew are referred to as effective weights. It is positively correlated with the degree of the node because we prioritize edges with higher endpoints. It should be negatively correlated with the distance to the root node because we prefer edges closer to the root. The effective weights also consider the directed edge information of the original graph $G(V, E, w)$. In this context, undirected edges are treated as a composite of two directed edges. Thus, the CSP-based effective weight is defined as

$$EffeW(e) = \frac{(w_G(a \to b) + w_G(b \to a)) log(Vol_T(a) + Vol_T(b))}{2(Dis(r \to a) + Dis(r \to b))}, \quad (7)$$

where $w_G(a \to b) + w_G(b \to a)$ represents the sum of the absolute values of the weights of edges $e(a \to b)$ and $e(b \to a)$ in the original weighted graph G. $Vol_T(a) + Vol_T(b)$ represents the sum of the volumes of nodes a and b in the generated tree. $Dis(r \to a) + Dis(r \to b)$ is the sum of the shortest paths from the root node to nodes a and b, as shown in Fig. 2(B). In a symmetric graph, the edges are endowed with distinct effective weight values based on their properties in the original graph, such as whether they are directed or undirected (lines 3-10 in Algorithms 2). Then, $|E| - 1$ edges are retained to construct the maximum spanning tree and the edges off the tree.

Accession of Off-tree Edges. After creating the spanning tree, critical off-tree edges are added to produce the sparsified subgraph S_u, further reducing the number of conditions $\kappa(G, S_u)$. So, it is crucial to select a small number of spectrally critical off-tree edges. By deriving the formula for the trace of the extended matrix, assuming that λ_k is the eigenvalue of G and u_k is the eigenvalue of subgraph S_u, we can see that the variation of the trace satisfies the equation:

$$\sum_k \lambda_k - \sum_k u_k \geq \frac{(\frac{1}{2}(w_G(a \to b) + w_G(b \to a))R_p(a, b))^2}{1 + \frac{1}{2}(w_G(a \to b) + w_G(a \to b))R_p(a, b)}. \quad (8)$$

In this, the sum of the edge weights (a, b) in the original weighted graph $(w_G(a \to b) + w_G(b \to a)$ contributes to the maximum node mismatch, as shown in Fig.2(B). The effective resistance value of edge (a, b) is denoted as $R_p(a, b)$. Adding external edges reduces the condition number between the matrices G and S_u. To maximize this reduction, effective resistance values $R_p(a, b)$ is computed for all external edges, sorted, and prioritized based on larger reduction. Directed and undirected edges possess distinct effective weights, leading to discrepancies in $R_p(a, b)$ values. Undirected edges exhibit larger the effective resistance value, thereby exerting a greater influence on the reduction of conditions compared to directed edges. Previous research [30] has shown that non-symmetric elements significantly impact the preconditioner, leading to a higher condition number. Therefore, priority is given to adding undirected edges, resulting in a subgraph with a smaller relative condition number (lines 13-23 in Algorithms 2).

3.5 The "AND" Operation

Finally, we reintegrate the asymmetric information from the original graph into the preconditioned matrix through the "AND" operation. The core idea of "AND" is that edges are retained when they exist in both the sparsified subgraph and the original weight graph. Otherwise, they are ignored. As shown in Fig. 2(C), from the matrix viewpoint, elements existing in the sparsified matrix are preserved only if they also exist in the original matrix. The values of the elements are updated accordingly to match the values within the original matrix. Otherwise, elements not found in the original matrix are removed (lines 13-20 in Algorithms 1). Then, the sparse matrix resulting from the "AND" operation is employed as a preconditioner in the GMRES algorithm to solve the original matrix (lines 21-22 in Algorithms 1).

3.6 Parallelization

Though our proposed CSP could produce a better preconditioner, the sparsification itself usually consumes additional overhead. To further improve efficiency and enhance its practicality, we use OpenMP to parallelize the sparsification process.

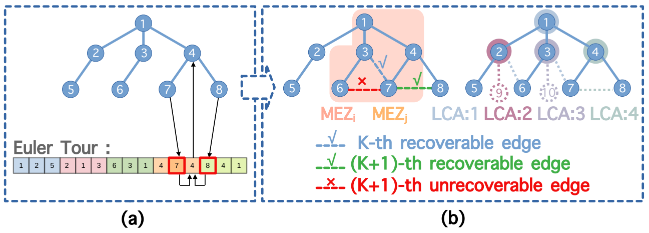

Figure 3: The figure (a) is an example of getting the LCA. The figure (b) is an example of edge grouping and edge exclusion.

Block RMQ algorithm. To calculate the effective resistances, we need to obtain the lowest common ancestor (LCA). However, traditional LCA algorithms are difficult to parallelize. As shown in Fig. 3(a), in this paper, we propose an effective strategy to transform the LCA problem into the range maximum query (RMQ) problem on the Euler tour. By dividing the block, we can conduct parallel operations on each block, thereby maintaining the maximum interval between them.

Point exclusive algorithm. When considering adding the off-tree edge $e = (i, j)$, we need to mark the other off-tree edges. However, establishing connections between newly added edges and other non-tree edges can present challenges. In this paper, to improve the efficiency, we create a mutually exclusive zone with i and j as starting points for edge e, so that the outside of an off-tree with mutually exclusive endpoints marked by the same edge can be skipped, as shown in Fig. 3(b). Finally, to reduce memory and time overhead, we group the edges with the LCA as a set.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

We compare the proposed CSP method with SOTA spectral sparsification methods GPSCP and feGRASS as the preconditioner for GMRES, as well as the most widely used sparse direct solver KLU (version 2.3.3). GPSCP is designed for nonlinear asyemetric circuits while feGRASS is not specifically designed for nonlinear asysmetric circuit matrix and can only be performed to syemetric matrix. Therefore, we leverage half of the asymmetric matrix (that is, the upper or lower triangular matrix) for symmetrization and serve it as the input of feGRASS for sparsification. ALL algorithms were developed using C++. The sparsity α is uniformly configured to 2% for all three sparsifiers. The residual bound is set to 10^{-5} and the maximum number of iterations was capped at 500 (exceeded ones are considered as non-convergent). Our experiments are on a 32-core CPU operating at a frequency of 3.7 GHz, as well as 16 GB of DDR4 memory.

Table 1: Experimental setup of test cases

MatrixName	n(A)	nnz(A)	asym_ratio
ww_36_pmec_36	66	1194	95.31%
fpga_dcop_43	1220	5892	53.94%
fpga_trans_01	1220	7382	66.30%
adder_dcop_35	1813	11246	83.36%
add20	2395	17319	29.40%
adder_trans_02	1814	14579	84.89%
mult_dcop_02	25187	193276	86.86%
rajat30	643994	6175377	88.90%
circuit_4	80209	307604	47.05%
ASIC_320ks	321671	1827807	40.66%
circuit5M_dc	3523317	19194193	24.41%
Freescale1	3428755	18920347	51.55%

We use 178 circuit matrics from SuiteSparse Matrix Collection [31] for experimental verification and also present a detailed performance comparison for 12 cases in which all methods have converged. The details of these cases are shown in Table 1. "$\#n(A)$" denote the dimensions of the matrix. "$\#nnz(A)$" denotes the number of non-zero elements in the original matrix of the system. "$\#asym\_ratio$" denotes the percentage of asymmetric elements in the matrix.

4.2 Solver Performance Comparison

We first compare the total solution time of the iterative solver, which uses CSP, feGRASS and GPSCP as the preconditioner, respectively. Here, for a fair comparison, all algorithms are performed in serial. The total solver time includes both the pre-processing time and the iterative solver time in GMRES. Results are shown in Fig. 4. Of the 178 circuit matrices tested, both CSP and GPSCP successfully converged in 175 matrices, while feGRASS only achieved successful convergence in 60 matrices. In the set of matrices where successful convergence is achieved, CSP outperforms feGRASS and GPSCP with a 13.46x and 2.50x speedup on geometric mean, respectively.

We further demonstrate the detailed solver time of our proposed CSP over fe-GRASS, GPSCP and KLU using 12 circuit matrices of varying sizes and classes. As depicted in Fig. 5 (where KLU is considered as the baseline), CSP exhibits the highest acceleration among the three graph sparsification-based preconditioning solvers. Furthermore, CSP exhibits an average speedup of 2.18x compared to KLU. It demonstrates significant effectiveness of our proposed methods compared with not only SOTA iterative solvers but also the most widely used direct solver.

4.3 Quality of Preconditioners

We further validate the preprocessing quality of our proposed CSP in the following three aspects and compare them with the feGRASS and GPSCP methods.

The number of *nnz* in preconditioners. As shown in Table 2 "$\#nnz(P)$", our proposed CSP demonstrates better sparsity compared with the other two methods. The primary reason for this is

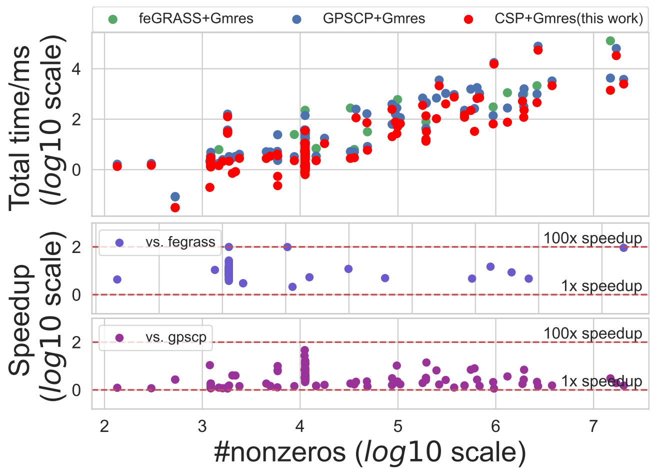

Figure 4: The total time (log10 scale) of three preconditioned iteration, with Speedup(log10 scale) between CSP+GMRES and other two methods.

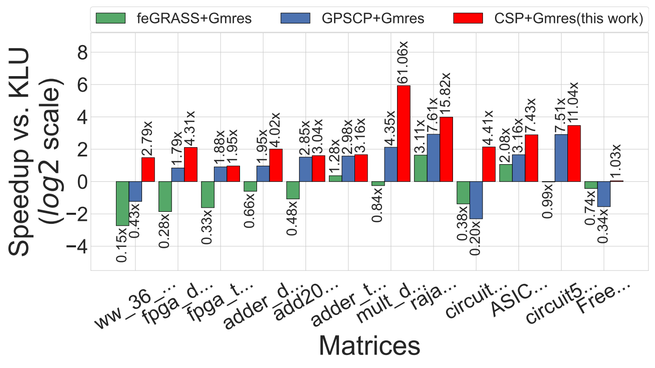

Figure 5: Peformance comparisons of feGRASS, GPSCP, CSP and KLU, where KLU is used as the baseline. The number on each bar represents the speedup ratio.

that our sparsification strategies are applied to the entire graph, thereby minimizing the addition of extraneous nonlinear elements and further reducing the preconditioner's factorization time and the complexity. Note that a higher sparsity in the preconditioner brings a significant advantage of reducing the preconditioner's factorization time and the complexity.

Relative condition numbers. As shown in Table 2 "$\#\kappa(A, P)$", the smaller the value of $\#\kappa(A, P)$, the more similar the spectral characteristics of matrices A and P are. Among the three graph sparsification methods, CSP yields a smaller relative condition number, indicating superior spectral similarity. This is primarily attributed to our approach of considering all elements globally during the sparsification process and reintroducing information from the original matrix into the preconditioner at the end.

Number of iterations and convergence performance. As shown in Table 2 "$\#iter(A, P)$", improved spectral similarity between matrices A and P leads to reduced iteration numbers required by the iterative method. As a result, the CSP-based preconditioned

Table 2: Sparsity and validity of three different sparsification-based preconditioner (The maximum number of iterations is 500).

MatrixName	feGRASS+GMRES[18]			GPSCP+GMRES[21]			CSP+GMRES			IterDown	
	nnz(P)	iter	$\kappa(A,P)$	nnz(P)	iter	$\kappa(A,P)$	nnz(P)	iter	$\kappa(A,P)$	vs. [18]	vs. [21]
ww_36_pmec_36	186	500	309.50	888	30	68.67	**140**	**30**	**66.72**	–	1.00x
fpga_dcop_43	3562	41	632.14	4380	4	12.99	**3421**	**2**	**9.13**	20.50x	2.00x
fpga_trans_01	3616	500	328.01	3616	10	33.70	**3616**	**9**	**23.75**	–	1.11x
adder_dcop_35	5457	338	1248.11	8774	64	64.03	**5039**	**56**	**43.21**	6.04x	1.14x
add20	7229	500	2268.92	7229	500	2266.72	**5953**	**7**	**12.44**	–	–
adder_trans_02	5463	11	221.00	5463	9	216.87	**5463**	**9**	**216.06**	1.22x	1.00x
mult_dcop_02	75567	500	2271.40	140424	500	2266.40	**71321**	**10**	**11.91**	–	–
rajat30	1932663	500	3875.64	1947406	30	171.29	**1919717**	**28**	**161.27**	–	1.07x
circuit_4	200872	500	1956.00	225206	500	1952.12	**186364**	**11**	**119.71**	–	–
ASIC_320ks	970947	30	99.40	970947	4	1.03	**465838**	**3**	**1.03**	10.00x	1.33x
circuit5M_dc	10638293	500	2556.00	10638293	500	2592.12	**6062913**	**11**	**119.71**	–	–
Freescale1	10352717	500	7823.20	10352717	20	504.20	**7804691**	**16**	**419.06**	–	1.25x
Average	–	–	–	–	–	–	–	–	–	9.44x	1.23x

iterative method solver can proceed through fewer steps. As shown in Fig. 6, we further illustrate the convergence details of some tests. It can be seen that CSP+GMRES reaches the convergence bound earlier in most cases.

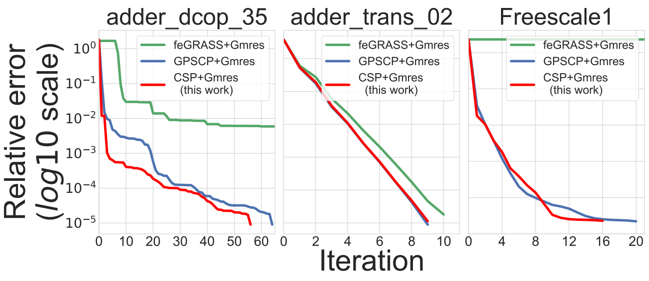

Figure 6: The convergence of residuals during iteration is observed for three representitive matrices.

4.4 Pre-processing Time

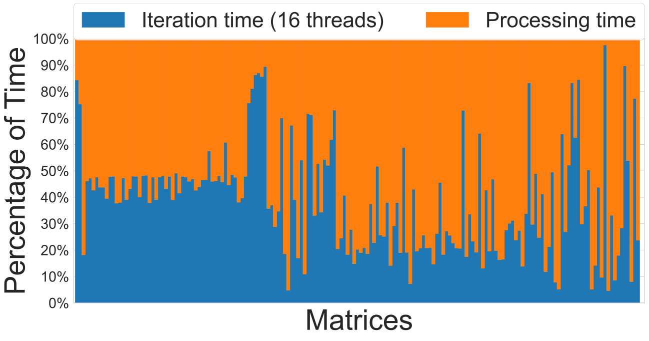

Figure 7: The percentage of time spent on pre-processing and iterative solver with 16-threads for 178 matrices.

We further show the breakdown time to veify the pre-processing overhead. As depicted in Table 3, where "$\#t_{set}(ms)$" and "$\#t_{iter}(ms)$" denote the time to generate the preconditioner (with symmetric step, sparsification step, "AND" intersection step) and the time of the GMRES iterative solver, respectively. It can be found that CSP is slightly more time-consuming compared to the other two methods. While generation process of preconditioners may be time-consuming, the exceptional quality of CSP enables iterative methods to significantly decrease solution times.

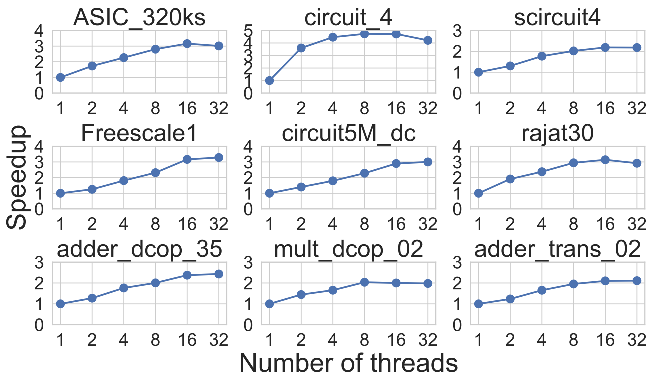

Figure 8: Parallel performance for sparsification.

Another observation is that the pre-processing time usually takes a protion of 7.30%-51.39% compared with the iterative solver. This indicates an obvious pre-processing overhead, which is mainly introduced by the sparsification process. Fig. 7 further demonstrate the breakdown time comparison when we perform a parallel iterative solver (with 16 threads). The pre-processing phase may even account for more than 50% of the total time. This emphasis a siginificant importance and necessity of accelerating the pre-precoessing step. Since the symmetric and the "AND" intersection step is quite easy to parallize, in this paper, we mainly focus on the parallalism on the sparsification step.

4.5 Parallelization

As shown in Fig. 8, our parallel sparsification algorithm demonstrates notable parallel efficiency and scalability. In 178 tested matrices, our parallelization strategy achieves an average speedup 2.99x compared to traditional serial graph sparsification algorithms. This

Table 3: The breakdown time for setup stage and the iterative solver stage for three spaesification-based iterative solvers.

MatrixName	feGRASS+GMRES[18]		GPSCP+GMRES[21]		CSP+GMRES	
	$t_{set}\,(ms)$	$t_{iter}\,(ms)$	$t_{set}\,(ms)$	$t_{iter}\,(ms)$	$t_{set}\,(ms)$	$t_{iter}\,(ms)$
ww_36_pmec_36	0.13	5.95	0.03	2.12	0.17	0.15
fpga_dcop_43	0.43	13.73	0.30	1.89	0.49	1.83
fpga_trans_01	0.52	14.41	0.55	2.06	0.56	1.94
adder_dcop_35	0.69	11.32	0.57	5.18	1.05	1.11
add20	1.17	23.15	1.37	2.69	1.14	2.68
adder_trans_02	1.14	6.80	1.28	2.14	1.14	2.08
mult_dcop_02	15.71	207.24	10.73	32.35	25.15	5.50
rajat30	624.21	5402.98	711.45	820.35	969.5	471.54
circuit_4	75.60	538.14	62.85	947.55	102.68	420.09
ASIC_320ks	142.91	1121.58	148.35	117.03	160.61	28.87
circuit5M_dc	1553.48	30806.12	2146.52	1990.06	2274.72	753.55
Freescale1	2496.39	36868.10	2596.70	61167.55	2836.50	35549.32

is primarily attributed to our proposed Block RMQ algorithm and point-exclusive algorithm, facilitating parallelism in both the LCA solution process and the edge addition process. This improvement significantly enables parallel sparsification process and reduces the pre-processing overhead, enhancing the practicality of spectral sparsification method.

4.6 Ablation Experiment

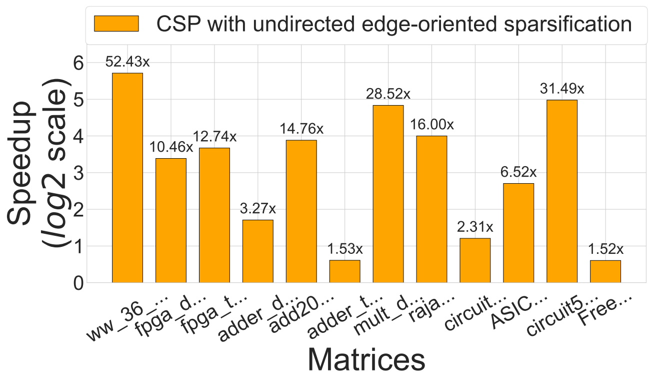

Figure 9: Ablation experiment to demonstrate the effectiveness of our proposed sparsification approach (in Section 3.4).

For ablation experiment to demonstrate the effectiveness of our proposed undirected edge-oriented sparsification strategies (approaches in Section. 3.4), we use converntional feGRASS as the sparsification strategy at the sparsification step in CSP as the comparison baseline. All other steps are the same, e.g., operations about Asymmetric Part and the "AND" operation. As shown in Fig.9, CSP with specific undirected edge-oriented sparsification strategy demonstrates an average 15.13x speedup (with a maximum 52.43x) compared with the one without.

4.7 Memory Cost Comparison

Finally, we compare the memory consumption of the direct solver KLU, and three preconditioning methods CSP, feGRASS and GPSCP with iterative solver. As shown in Fig.10, iterative solvers demonstrate significantly lower memory consumption costs compared with direct solvers. Furthermore, CSP+GMRES exhibits the lowest memory consumption compared to the other two preconditioned iterative methods. The main reason is that our proposed CSP generates a sparser preconditioner compared with other methods.

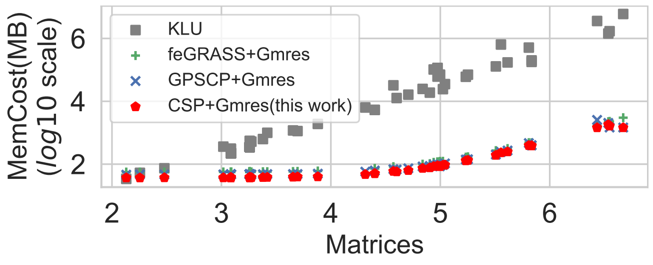

Figure 10: Memory cost comparison of KLU and three sparsification-based iteration operation.

5 CONCLUSION

In this paper, we propose CSP, an efficient preconditioning method for solving nonlinear asymmetric circuit matrices. Our method significantly improves the efficiency and robustness of iterative solvers by using a comprehensively sparse subgraph as a preconditioner. The proposed method enhances the sparisity of the produced preconditioner to lower the factorization time and complexity, while maintaing a better spectral similarity to reduce the iterative number and improve the performance. Moreover, parallelization technique is also proposed to further reduce the overhead of matrix sparsification. CSP boosts the solving efficiency of large circuit matrices, with its superior speed and reduced memory usage.

6 ACKNOWLEDGMENTS

Zhou Jin is the corresponding author of this paper. This work was supported by the National Key R&D Program of China (Grant No. 2022YFB4400400), NSFC (Grant No. 62204265, U23A20301, 62234010, 62374031, 62372467), the State Key Laboratory of Computer Architecture (ICT, CAS) (Grant No. CARCHA202115) and Natural Science Foundation of Jiangsu Province (Grant No. BK20240173).

REFERENCES

[1] Xu Fu, Bingbin Zhang, Tengcheng Wang, Wenhao Li, Yuechen Lu, Enxin Yi, Jianqi Zhao, Xiaohan Geng, Fangying Li, Jingwen Zhang, Zhou Jin, Weifeng Liu. PanguLU: A Scalable Regular Two-Dimensional Block-Cyclic Sparse Direct Solver on Distributed Heterogeneous Systems. *36th International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '23)*, 2023.

[2] Yufei Chen, Haojie Pei, Xiao Dong, Zhou Jin, Cheng Zhuo. Application of Deep Learning in Back-End Simulation: Challenges and Opportunities. *27th ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC '22)*, 2022.

[3] Ganqu Cui, Wenjian Yu, Xin Li, Zhiyu Zeng, and Ben Gu. Machine-Learning-Driven Matrix Ordering for Power Grid Analysis. *Design, Automation & Test in Europe Conference & Exhibition (DATE '19)*, 2019.

[4] Zhiqiang Liu and Wenjian Yu. Accuracy-Preserving Reduction of Rparsified Reduced Power Grids with a Multilevel Node Aggregation Scheme. *IEEE/ACM International Conference on Computer Aided Design (ICCAD '23)*, 2023.

[5] Anirudh Maringanti, Viraj Athavale, and Sachin B Patkar. Acceleration of Conjugate Gradient Method for Circuit Simulation using Cuda. *International Conference on High Performance Computing (HIPC '09)*, 2009.

[6] KF Tsang, RS Chen, Mo Lei, and Edward KN Yung. Application of the Preconditioned Conjugate-Gradient Algorithm to the Integral Equations for Microwave Circuits. *Microwave and Optical Technology Letters*, 2002.

[7] Zhao Li and C-JR Shi. An Efficiently Preconditioned GMRES Method for Fast Parasitic-sensitive Deep-Submicron VlSI Circuit Simulation. *Design, Automation & Test in Europe Conference & Exhibition (DATE '05)*, 2005.

[8] Wim Bomhof and Henk A van der Vorst. A Parallelizable Gmres-Type Method for p-cyclic Matrices, with Applications in Circuit Simulation. *Scientific Computing in Electrical Engineering: Proceedings of the 3rd International Workshop*, 2001.

[9] Dechuang Yang, Yuxuan Zhao, Yiduo Niu, Weile Jia, En Shao, Weifeng Liu, Guangming Tan.Zhou Jin. Mille-feuille: A Tile-Grained Mixed Precision Single-Kernel Conjugate Gradient Solver on GPUs. *37th International Conference for High Performance Computing, Networking, Storage, andAnalysis (SC '24)*, 2024.

[10] Richard Barrett, Michael Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*, 1994.

[11] Michele Benzi. Preconditioning Techniques for Large Linear Systems: A Survey. *Journal of computational Physics*, 2002.

[12] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Transactions on Mathematical Software*, 2008.

[13] Zhengyong Zhu, Bo Yao, and Chung-Kuan Cheng. Power Network Analysis Using an Adaptive Algebraic Multigrid Approach. *Proceedings of the 40th annual Design Automation Conference*, 2003.

[14] Yinuo Bai, Xiaoyu Yang, Yicheng Lu, Dan Niu, Cheng Zhuo, Zhou Jin, Weifeng Liu. Efficient Spectral-Aware Power Supply Noise Analysis for Low-Power Design Verification. *21st Design, Automation and Test in Europe Conference (DATE '24)*, 2024.

[15] Jianlei Yang, Zuowei Li, Yici Cai, and Qiang Zhou. Powerrush: An Efficient Simulator for Static Power Grid Analysis. *IEEE Transactions on Very Large Scale Integration Systems (VLSI)*, 2013.

[16] Zhuo Feng. Grass: Graph Spectral Sparsification Leveraging Scalable Spectral Perturbation Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[17] Zhiqiang Liu and Wenjian Yu. Pgrass-Solver: A Parallel Iterative Solver for Scalable Power Grid Analysis Based on Graph Spectral Sparsification. *IEEE/ACM International Conference On Computer Aided Design (ICCAD '21)*, 2021.

[18] Zhiqiang Liu, Wenjian Yu, and Zhuo Feng. Fegrass: Fast and Effective Graph Spectral Sparsification for Scalable Power Grid Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[19] Zhiqiang Liu and Wenjian Yu. Accuracy-preserving Reduction of Sparsified Reduced Power Grids with a Multilevel Node Aggregation Scheme. *IEEE/ACM International Conference on Computer Aided Design (ICCAD '23)*, 2023.

[20] Lengfei Han, Xueqian Zhao, and Zhuo Feng. An Adaptive Graph Sparsification Approach to Scalable Harmonic Balance Analysis of Strongly Nonlinear Post-layout RF Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2014.

[21] Xueqian Zhao and Zhuo Feng. Gpscp: A General-Purpose Support-circuit Preconditioning Approach to Large-scale Spice-accurate Nonlinear Circuit Simulations. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '12)*, 2012.

[22] Alper Demir and Alberto Sangiovanni-Vincentelli. Analysis and simulation of Noise in Nonlinear Electronic Circuits and Systems, 2012.

[23] ChungHan Chou, NienYu Tsai, Hao Yu, CheRung Lee, Yiyu Shi, and ShihChieh Chang. On the Preconditioner of Conjugate Gradient Method—a power Grid Simulation Perspective. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '11)*, 2011.

[24] Roman Iakymchuk, Stef Graillat, and José I Aliaga. General Framework for Reassuring Numerical Reliability in Parallel Krylov Solvers: A Case of Bi-Conjugate Gradient Stabilized Methods. *The international journal of high performance computing applications*, 2024.

[25] Lingyun Ouyang, Chao Jin, and Quan Chen. A Fast Recycling Gmres Method with Smart Frequency Sweeping for Efficient Periodic Small-Signal Analysis. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2024.

[26] Xueqian Zhao, Lengfei Han, and Zhuo Feng. A Performance-Guided Graph Sparsification Approach to Scalable and Robust Spice-accurate Integrated Circuit Simulations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015.

[27] Yuechen Lu, Lijie Zeng, Tengcheng Wang, Xu Fu, Wenxuan Li, Helin Cheng, Dechuang YangZhou Jin, Marc Casas, Weifeng Liu. AmgT: Algebraic Multigrid Solver on Tensor Cores. *37th International Conference for High Performance Computing, Networking, Storage, andAnalysis (SC'24)*, 2024.

[28] Nabil Gmati and Bernard Philippe. Comments on the Gmres Convergence for Preconditioned Systems. *International Conference on Large-Scale Scientific Computing*, 2007.

[29] Daniel A Spielman and Nikhil Srivastava. Graph Sparsification by Effective Resistances. *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 2008.

[30] Yuanzhe Xi and Yousef Saad. A Rational Function Preconditioner for Indefinite Sparse Linear Systems. *SIAM Journal on Scientific Computing*, 2017.

[31] Timothy A Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 2011.

[32] Zhang, Ying and Zhao, Zhiqiang and Feng, Zhuo. DiGRASS: Directed Graph Spectral Sparsification via Spectrum-Preserving Symmetrization. *ACM Transactions on Knowledge Discovery from Data*, 2024.

[33] Zhou Jin, Wenhao Li, Yinuo Bai, Tengcheng Wang, Yicheng Lu and Weifeng Liu. Machine Learning and GPU Accelerated Sparse Linear Solvers for Transistor-Level Circuit Simulation: A Perspective Survey (Invited Paper). *29th ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC '24)*, 2024.

[34] Guofeng Feng, Hongyu Wang, Zhuoqiang Guo, Mingzhen Li, Tong Zhao, Zhou Jin, Weile Jia, Guangming Tan, Ninghui Sun. MAccelerating Large-scale Sparse LU Factorization for RF Circuit Simulation. *30th International European Conference on Parallel and Distributed Computing (Euro-Par '24)*, 2024.

[35] Tengcheng Wang, Wenhao Li, Haojie Pei, Yuying Sun, Zhou Jin, Weifeng Liu. Accelerating Sparse LU Factorization with Density-Aware Adaptive Matrix Multiplication for Circuit Simulation. *60th ACM/IEEE Design Automation Conference (DAC '23)*, 2023.

[36] Jianqi Zhao, Yao Wen, Yuchen Luo, Zhou Jin, Weifeng Liu, Zhenya Zhou. SFLU: Synchronization-Free Sparse LU Factorization for Fast Circuit Simulation on GPUs. *58th ACM/IEEE Design Automation Conference (DAC '21)*, 2021.

[37] Mingjia Fan, Xiaoming Chen, Dechuang Yang, Zhou Jin, Weifeng Liu. ReCG: ReRAM-Accelerated Sparse Conjugate Gradient. *61st ACM/IEEE Design Automation Conference (DAC '24)*, 2024.