PowerRChol: Efficient Power Grid Analysis Based on Fast Randomized Cholesky Factorization

Zhiqiang Liu, Wenjian Yu*
Dept. Computer Science & Tech., BNRist, Tsinghua Univ., Beijing 100084, China
Email: liu-zq20@mails.tsinghua.edu.cn, yuwj@tsinghua.edu.cn

ABSTRACT

Efficient power grid analysis is critical in modern VLSI design. It is computationally challenging because it requires solving large linear equations with millions of unknowns. Iterative solvers are more scalable, but their performance relies on preconditioners. Existing preconditioning approaches suffer from either high construction cost or slow convergence rate, both resulting in unsatisfactory total solution time. In this work, we propose an efficient power grid simulator based on fast randomized Cholesky factorization, named PowerRChol. We first propose a randomized Cholesky factorization algorithm with provable linear-time complexity. Then we propose a randomized factorization oriented matrix reordering approach. Experimental results on large-scale power grids demonstrate the superior efficiency of PowerRChol over existing iterative solvers, showing 1.51X, 1.93X and 3.64X speedups on average over the original RChol [3], feGRASS [11] and AMG [14] based PCG solvers, respectively. For instance, a power grid matrix with 60 million nodes and 260 million nonzeros can be solved (at a 1E-6 accuracy level) in 148 seconds on a single CPU core.

ACM Reference Format:
Zhiqiang Liu, Wenjian Yu. 2024. PowerRChol: Efficient Power Grid Analysis Based on Fast Randomized Cholesky Factorization. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3657308

1 INTRODUCTION

An efficient solver for symmetric diagonally dominant M-matrices (SDDM) plays an important role in many applications, such as on-chip power grid analysis [6, 9–11, 14], thermal simulation [15], computer graphics [7] and semi-supervised learning [16], etc.

Accurate analysis of on-chip power grids is indispensable for designing modern VLSI chips since it can help to reveal critical design issues related to IR drop, electromigration, etc. With the increased design complexity, power grid analysis imposes a severe computational challenge, where linear equations with millions of unknowns (whose coefficient matrix is an SDDM) need to be solved. A majority of power grid analysis algorithms fall into two categories: direct solvers such as Cholesky factorization [4] and iterative solvers such as preconditioned conjugate gradient (PCG) method [6, 9–11, 14]. Iterative solvers are more preferred for large problems due to more favourable memory requirements but may suffer from slow convergence issues. The total solution time of iterative solvers consists of the time for constructing the preconditioner and the time for PCG iteration. Therefore, to reduce the total solution time, it is highly demanded that the preconditioner can not only be constructed efficiently but also lead to fast convergence.

Recently, spectral graph sparsification based preconditioning approaches have shown promising performance in accelerating power grid analysis [6, 9–11]. These methods aim to find a spectrally similar subgraph (sparsifier) from original graph and leverage the Laplacian matrix of subgraph as the preconditioner. GRASS [6] and the approximate trace reduction based approach [10] can construct effective sparsifiers but take a huge amount of time in the sparsification phase. The feGRASS algorithm proposed in [11] achieves a better tradeoff between runtime and effectiveness of sparsifiers. However, graph sparsification based iterative solvers still face a dilemma: too few edges in the sparsifier result in slow convergence while too many edges in the sparsifier lead to large factorization cost. The feGRASS based PCG solver was improved in [9] by incorporating the incomplete Cholesky factorization (IChol) for more efficiently solving to a higher accuracy level. However, the inefficient implementation of PCG algorithm in [9] makes the benefit of incorporating IChol largely exaggerated (see Section 4.2).

A randomized Cholesky factorization named RChol was proposed in [3]. It stems from the theoretical work of "Laplacian Paradigm 2.0" [8] and can construct effective preconditioners for large sparse SDDM. The RChol based iterative solver has great potential for accelerating power grid simulation. However, the preconditioner construction cost of RChol can be high for large problems.

In this work, we aim to develop a more efficient SDDM solver for large-scale power grid analysis, via reducing the preconditioner construction cost of RChol. Our main contributions are as follows.

1) We propose an improved RChol algorithm with provable linear-time complexity, named LT-RChol. It can not only reduce the preconditioner construction cost but also accelerate PCG iteration.

2) We propose a fast and effective LT-RChol orientated matrix reordering approach. It can produce matrix reorderings with comparable quality to the approximate minimum degree (AMD) algorithm [2] in much shorter time thus further reduce the total solution time.

3) Combining these two techniques, we obtain a more efficient SDDM solver named PowerRChol. Extensive experimental results on power grids and real-world problems demonstrate the superior efficiency and the robustness of PowerRChol over the state-of-the-art iterative solvers, including the original RChol [3], feGRASS [11] and AMG [14] based PCG solvers, and the PowerRush simulator.

*This work is supported by National Science and Technology Major Project (No. 2021ZD0114703) and NSFC under grant No. 62090025.

2 BACKGROUND

2.1 Graph Laplacian Matrix and SDDM

Consider a weighted undirected graph $\mathcal{G} = (V, E, w)$, where V and E denote the sets of vertices (nodes) and edges, respectively. w is a positive weight function. We use $w_{i,j}$ to denote the weight of edge (i, j). The Laplacian matrix of $\mathcal{G}$ is denoted by $L_\mathcal{G} \in \mathbb{R}^{n\times n}$.

$$L_\mathcal{G}(i,j) = \begin{cases} -w_{i,j}, & (i,j) \in E \\ \sum\limits_{(i,k)\in E} w_{i,k}, & i = j \\ 0, & \text{otherwise} . \end{cases} \tag{1}$$

A symmetric diagonally dominant M-matrix (SDDM) is a symmetric diagonally dominant (SDD) matrix with positive diagonal and nonpositive off-diagonal elements. It can be written as a graph Laplacian matrix plus some positive diagonal elements:

$$A = L_\mathcal{G} + D . \tag{2}$$

In many applications, the linear equation system whose coefficient matrix is an SDDM is solved, and the matrix also corresponds to the Laplacian matrix of a graph. For example, on-chip power grid can be modeled as a weighted undirected graph whose edge weight is the conductance of resistor. The problem of power grid analysis can be formulated as $Ax = b$, where A is an SDDM and x and b denote the unknown vector of node voltages and the vector of current sources, respectively [10, 11].

Algorithm 1 RChol: Randomized Cholesky factorization for SDDM

```
Input: An SDDM $A \in \mathbb{R}^{N\times N}$ satisfying $A = L_\mathcal{G} + D$ as Eq. (2).
Output: The lower triangular factor matrix $L \in \mathbb{R}^{N\times N}$.
1:  Set $L \in \mathbb{R}^{N\times N}$ to a zero matrix.
2:  for $k = 1$ to $N$ do
3:      Set $d_k = L_\mathcal{G}(k,k) + D(k,k)$.
4:      Set $L(k,k) = \sqrt{d_k}$ and $L(k+1:n,k) = \frac{L_\mathcal{G}(k+1:n,k)}{\sqrt{d_k}}$.
5:      Sort $N_k = \{i | i > k \;\&\; L_\mathcal{G}(i,k) \neq 0\}$ to $\{n_1, n_2, \dots, n_{|N_k|}\}$,
        such that $|L_\mathcal{G}(n_1,k)| \leq |L_\mathcal{G}(n_2,k)| \leq \dots \leq |L_\mathcal{G}(n_{|N_k|},k)|$.
6:      for $j = 1$ to $|N_k|$ do
7:          Set $D(n_j,n_j) = D(n_j,n_j) - \frac{D(n_j,n_j)L_\mathcal{G}(n_j,k)}{d_k}$.
8:          Set $s_{k,j} = \sum_{j<i\leq|N_k|} |L_\mathcal{G}(n_i,k)|$.
9:          Randomly sample $\{n_i : j < i \leq |N_k|\}$ with probability
            $\frac{|L_\mathcal{G}(n_i,k)|}{s_{k,j}}$ to get $n_l$.
10:         Set $L_\mathcal{G} = L_\mathcal{G} + \frac{s_{k,j}|L_\mathcal{G}(n_j,k)|}{d_k} f_{n_j,n_l} f_{n_j,n_l}^T$.
11:     end for
12: end for
```

2.2 RChol: Randomized Cholesky Factorization

In [3], a randomized Cholesky factorization method named RChol was proposed to construct effective preconditioner for SDDM. Instead of introducing a clique at each elimination step as in classical Cholesky factorization, RChol randomly keeps a spanning tree among the neighbors of the eliminated node to reduce fill-ins. In [3], it is proven that the sampled spanning tree is an unbiased estimator of the clique and the whole procedure is breakdown-free. The algorithm flow of RChol is described in Alg. 1 [3]. Here, $f_{i,j} = f_i - f_j$ and f_i denotes the i-th column of the identity matrix.

In practice, the matrix reordering on A is applied before executing the RChol algorithm, to reduce the computational cost of the RChol factorization. So, if RChol is employed, the time of constructing preconditioner consists of the time of matrix reordering and the time of factorization. In [3], different matrix reordering approaches are compared and the approximate minimum degree (AMD) [2] algorithm achieves the best performance. However, the AMD algorithm itself takes a considerable amount of time. Another drawback of RChol is that the time complexity of RChol is $O(|L|log\frac{|L|}{N})$, which is super-linear with respect to the number of nonzeros in the output factor (i.e. $|L|$). This is because that the clique sampling operation, i.e. line 5-10 in Alg. 1, takes $O(|N_k|log|N_k|)$ time. Thus the preconditioner construction cost of RChol can be high for large-scale problems. For example, to solve the power grid matrix named "thupg10" in [13], which is a large power grid with about 6E7 nodes and 1E8 resistors, RChol takes 98.5 seconds (64.9 seconds for reordering and 33.6 seconds for factorization) to set up the preconditioner and the total solution time is 236 seconds. In this work, we aim to reduce the preconditioner construction cost and develop a more efficient iterative solver for large SDDM.

3 POWERRCHOL: EFFICIENT POWER GRID ANALYSIS BASED ON FAST RANDOMIZED CHOLESKY FACTORIZATION

In this section, we first propose a linear-time clique sampling technique, which can not only reduce the factorization cost but also improve the preconditioning quality, i.e. reduce the PCG iteration time. Then, we propose a novel fast and effective matrix reordering strategy which is customized for randomized factorization. Combining these two techniques, we finally present the efficient PowerRChol solver for power grid analysis.

3.1 Linear-Time Randomized Cholesky Factorization

The core operation in RChol is the clique sampling, i.e. line 5-10 in Alg. 1. It samples a random spanning tree from the clique introduced by node elimination to reduce the fill-ins. Suppose node k is being eliminated and the set of node k's neighbors are denoted as N_k, as in Alg. 1. Now we analyze the time complexity of the clique sampling operation. It first sorts the neighbors of node k by edge weights and the sorting takes $O(|N_k|log|N_k|)$ time. Then it traverses all the neighbors and for each neighbor n_j, it samples another neighbor n_l from those neighbors after neighbor n_j, with probability proportional to edge weights. Each sampling operation can be implemented by a binary search thus the time complexity is $O(log|N_k|)$. So the time complexity of the sampling operations for all neighbors is $O(|N_k|log|N_k|)$. Therefore, the total time complexity of the clique sampling operation is $O(|N_k|log|N_k|)$, which is super-linear with respect to the number of neighbors of node k.

The key idea of our algorithm is to modify the clique sampling operation and reduce its complexity from $O(|N_k|log|N_k|)$ to $O(|N_k|)$, thus the complexity of the randomized factorization can be reduced to $O(\Sigma_k|N_k|) = O(|L|)$. To achieve this goal, we need to modify the two steps in Alg. 1: sorting (line 5) and sampling (line 9).

It is known that the counting sorting [5] takes only linear time but it only applies to integers. Here, we use an approximate counting sorting which first quantizes real numbers to limited bits and then sorts them approximately using counting sorting. The maximum edge weight between node k and its neighbors, denoted as:

$$m_k = \max_{n_j \in |N_k|} |L_{\mathcal{G}}(n_j, k)| \,, \tag{3}$$

can be found in $O(|N_k|)$ time. Then we can normalize those edge weights to $(0, 1]$ via dividing them by m_k, which takes also linear time. Suppose we use b buckets in approximate counting sorting. Then the neighbor n_j should be put into the $\lceil \frac{|L_{\mathcal{G}}(n_j,k)|}{m_k} b \rceil$-th bucket. The neighbors in the same bucket are thought to be with nearly the same edge weight so the sorting is approximate. The sorted neighbors can be obtained by simply outputting the neighbors bucket by bucket. It can be easily verified that the approximate couting sorting takes only $O(|N_k|)$ time.

The clique sampling operation traverses all the neighbors of node k. For each neighbor $n_j \in N_k$, it samples another neighbor n_{l_j} from those neighbors after neighbor n_j, with probability proportional to edge weights. Now we take a deeper look at the implementation details of the clique sampling operation. Suppose the prefix sum array of the sorted edge weights between node k and its neighbors is denoted as pfs_j:

$$pfs_j = \sum_{1 \le i \le j} |L_{\mathcal{G}}(n_i, k)| \,. \tag{4}$$

When the traversal reaches the neighbor n_j, the sampling operation firstly generates a random number $r_j \in (0, 1)$ and then searches for the target $t_j = pfs_j + r_j \times (pfs_{|N_k|} - pfs_j)$ in the prefix sum array to find:

$$l_j = \min_{j < i \le |N_k|} \{i \,| pfs_i \ge t_j\}. \tag{5}$$

Then the n_{l_j} is the sampled neighbor at this step. This can be done via a binary search ($O(log|N_k|)$ complexity) and it can be easily shown that it is equal to sampling n_l with probability proportional to edge weights (line 8 in Alg. 1).

To remove the log in the complexity, instead of generating a random number r_j for each neighbor n_j, we can fisrt generate a random number $r \in (0, 1)$ and compute $r_j = \frac{j-1+r}{|N_k|}$ for each neighbor $n_j \in N_k$. It ensures that r_j is an ascending array. The resultant target array, which satisfies that:

$$t_j = pfs_j + \frac{j-1+r}{|N_k|} \times (pfs_{|N_k|} - pfs_j) \,, \tag{6}$$

is also ascending. It is because that suppose $i > j$, we have:

$$t_i - t_j > (1 - r_j)(pfs_i - pfs_j) > 0 \,. \tag{7}$$

Instead of calling binary search at each step, searching for an ascending array (t_j with length $|N_k|$) in another ascending array (pfs_j with length $|N_k|$) can be implemented in a more compact manner, which is summarized as Alg. 2. It can be easily verified that the time complexity of Alg. 2 is $O(m + n)$, which is linear with respect to the total length of the two arrays.

Therefore we obtain a more efficient randomized Cholesky factorization algorithm (named LT-RChol), which is summarized as Alg. 3. Since the clique sampling operation in Alg. 3 takes only $O(|N_k|)$ time, the total time complexity of Alg. 3 is $O(\Sigma_k |N_k|) = O(|L|)$, which is linear with respect to the number of nonzeros in the output factor. Experimental results in Section 4.1 demonstrates that the proposed LT-RChol can not only reduce the factorization cost but also improve the preconditioning quality, leading to shorter PCG iteration time.

Algorithm 2 Locate an ascending array in another ascending array

```
Input: Two ascending arrays $\{a_i\}_{i=1}^n$ and $\{t_j\}_{j=1}^m$: $a_1 \le a_2 \le ... \le a_n$ and $t_1 \le t_2 \le ... \le t_m$.
Output: $\{l_j\}_{j=1}^m$: the locations of $\{t_j\}$ in $\{a_i\}$, i.e. $l_j = min_{1 \le i \le n}\{i \mid a_i \ge t_j\}$, $j = 1, 2, ..., m$.
1: Set $c = 1$.
2: for $j = 1$ to $m$ do
3:   while $a_c < t_j$ do
4:     Set $c = c + 1$.
5:   end while
6:   Set $l_j = c$.
7: end for
```

Algorithm 3 Linear-time randomized Cholesky factorization

```
Input: An SDDM $A \in \mathbf{R}^{N \times N}$ satisfying $A = L_{\mathcal{G}} + D$ as Eq. (2).
Output: The lower triangular factor matrix $L \in \mathbf{R}^{N \times N}$.
1: Set $L$ to a zero matrix.
2: for $k = 1$ to $N$ do
3:   Set $d_k = L_{\mathcal{G}}(k, k) + D(k, k)$.
4:   Set $L(k, k) = \sqrt{d_k}$ and $L(k + 1 : n, k) = \frac{L_{\mathcal{G}}(k+1:n,k)}{\sqrt{d_k}}$.
5:   Sort $N_k = \{i | i > k \;\&\; L_{\mathcal{G}}(i, k) \ne 0\}$ with approximate counting sorting such that $|L_{\mathcal{G}}(n_1, k)| \le |L_{\mathcal{G}}(n_2, k)| \le ... \le |L_{\mathcal{G}}(n_{|N_k|}, k)|$ holds approximately.
6:   Generate a random number $r$. Compute the prefix sum array $\{pfs_i\}$ with Eq. (4) and the target array $\{t_j\}$ with Eq. (6).
7:   Call Alg. 2 to find the locations of $\{t_j\}$ in $\{pfs_i\}$: $\{l_j\}$.
8:   for $j = 1$ to $|N_k|$ do
9:     Set $D(n_j, n_j) = D(n_j, n_j) - \frac{D(n_j,n_j) L_{\mathcal{G}}(n_j,k)}{d_k}$.
10:    Set $s_{k,j} = \sum_{j<i\le|N_k|} |L_{\mathcal{G}}(n_i, k)|$.
11:    Set $L_{\mathcal{G}} = L_{\mathcal{G}} + \frac{s_{k,j} |L_{\mathcal{G}}(n_j,k)|}{d_k} f_{n_j,n_{l_j}} f_{n_j,n_{l_j}}^T$.
12:  end for
13: end for
```

3.2 Fast and Effective Matrix Reordering for Randomized Cholesky Factorization

To achieve better performance, the input SDDM should be reordered by some matrix reordering strategy prior to factorization. In [3], different matrix reordering strategies are compared and it is concluded that the approximate minimum degree (AMD) algorithm achieves the best total solution time. However, the AMD itself takes a considerable amount of time. Take the power grid named "thupg10" in [13] as an example, AMD takes 64.9 seconds, which accounts for about 30% of the total solution time (236 seconds).

The AMD algorithm is based on the heuristics that the node with the minimum degree should be eliminated first [2]. It computes upper bounds of actual node degrees during factorization. The computation is based on the fact that after eliminating a node with d neighbors, node degrees of all its neighbors increase by $d - 2$. However, things become different for randomized factorization. To derive a fast and effective reordering strategy for randomized factorization, now we analyze the change of node degrees after eliminating a node in randomized factorization.

Suppose node k is eliminated and its neighbors $N_k = \{i | i > k \;\&\; L_{\mathcal{G}}(i, k) \ne 0\}$ are sorted such that $|L_{\mathcal{G}}(n_1, k)| \le |L_{\mathcal{G}}(n_2, k)| \le$

$... \leq |L_{\mathcal{G}}(n_{|N_k|}, k)|$. Let $\Delta deg(n_j)$ denote the change of degree of neighbor n_j and $\Delta deg(n_j)$ is a random variable which relies on $l_1, l_2, ... l_{j-1}$. Here, $\{l_i\}$ denotes the sampled neighbor at previous steps. Let $I_{i,j}$ denote the indicator variable such that:

$$I_{i,j} = \begin{cases} 1, & \text{if } l_i = j \\ 0, & \text{otherwise} . \end{cases} \tag{8}$$

The probability of the indicator variable being 1 is:

$$\mathbb{P}(I_{i,j} = 1) = \frac{|L_{\mathcal{G}}(k, n_j)|}{\sum_{i<l\leq|N_k|} |L_{\mathcal{G}}(k, n_l)|} \leq \frac{1}{|N_k| - i} . \tag{9}$$

It can be shown that for $j < |N_k|$:

$$\Delta deg(n_j) = \sum_{i<j} I_{i,j} . \tag{10}$$

Therefore, the expectation of $\Delta deg(n_j)$ satisfies that:

$$\mathbb{E}(\Delta deg(n_j)) = \sum_{i<j} \mathbb{E}(I_{i,j}) \leq \sum_{i<j} \frac{1}{|N_k| - i} < \ln \frac{|N_k| - 1}{|N_k| - j} . \tag{11}$$

So, for $j \leq |N_k|(1 - \frac{1}{e}) + \frac{1}{e} \approx 63\%|N_k|$, we have $\mathbb{E}(\Delta deg(n_j)) < 1$. It means that for most neighbors of node k, the increase of node degrees after eliminating node k is smaller than 1 in expectation. It is different from complete Cholesky factorization, where the node degree of every neighbor increases by $|N_k| - 2$ after eliminating node k. So for most nodes in randomized factorization, the initial node degree can serve as a good lower bound of the actual degree.

Another difference from complete Cholesky factorization is that the pattern of fill-ins depends on not only the pattern of nonzeros but also the values. Consider the last neighbor of node k, i.e. $n_{|N_k|}$, which is the node with the maximum increase of node degree. It can be shown that:

$$\begin{aligned} \mathbb{E}(\Delta deg(n_{|N_k|})) &= \sum_{i<|N_k|} \frac{|L_{\mathcal{G}}(k, n_{|N_k|})|}{\sum_{i<l\leq|N_k|} |L_{\mathcal{G}}(k, n_l)|} - 1 \\ &\geq \frac{(|N_k| - 1)|L_{\mathcal{G}}(k, n_{|N_k|})|}{\sum_{1\leq l\leq|N_k|} |L_{\mathcal{G}}(k, n_l)|} - 1 . \end{aligned} \tag{12}$$

If $|L_{\mathcal{G}}(k, n_{|N_k|})|$ is large, i.e. it accounts for a large portion of $\sum_{1\leq l\leq|N_k|} |L_{\mathcal{G}}(k, n_l)|$, the degree of $n_{|N_k|}$ will increase by about $|N_k|-1$ after eliminating node k. Those nodes adjacent to edges with large weight should be eliminated firstly when the degrees ($|N_k|$) are relatively small. Otherwise, the degree of its largest neighbor will increase a lot after its elimination. In our implementations, the edges with weight larger than 10 times average weight are regarded as edges with large weight.

Therefore, we obtain a randomized factorization oriented matrix reordering algorithm, which is summarized as Alg. 4. It is very simple and can be implemented efficiently. Experimental results in Section 4.1 show that the proposed algorithm can produce permutations of comparable quality with AMD in much shorter time.

3.3 The Overall Algorithm

The overall algorithm flow of the proposed PowerRChol is:

1) Form the linear equation system (3) where A is an SDDM for the resistance circuit representing the power grid.

2) Call Alg. 4 to compute permutation matrix P.

3) Call Alg. 3 to factorize the reordered SDDM $PAP^T \approx LL^T$.

4) Run PCG algorithm to solve (3) with a preset convergence criterion, where at each iteration the preconditioning operation is multiplying $P^T L^{-T} L^{-1} P$ and the vector x.

5) Output the solution x, which includes the node voltages.

Algorithm 4 LT-RChol oriented matrix reordering

```
Input: An SDDM $A \in \mathbf{R}^{N\times N}$ satisfying $A = L_{\mathcal{G}} + D$ as Eq. (2).
Output: A permutation matrix $P$.
1: Compute the average edge weight $w_{avg}$ in graph $\mathcal{G}$.
2: For each node $i$, compute the maximum edge weight adjacent
   to $i$ in graph $\mathcal{G}$, denoted as $w_{max}(i)$.
3: Sort all nodes according to their degrees in ascending order.
   Set $d_{max}$ as the maximum node degree.
4: for $d = 1$ to $d_{max}$ do
5:     Move those nodes $\{i \mid deg(i) = d \;\&\&\; w_{max}(i) > 10w_{avg}\}$ to
       the front of $\{i \mid deg(i) = d\}$.
6: end for
7: Return the corresponding permutation matrix $P$.
```

4 EXPERIMENTAL RESULTS

We first validate the ideas proposed in Section 3.1 and Section 3.2. Then the proposed PowerRChol is compared with other recently developed iterative solvers for power grid analysis on the power grid benchmarks [12, 13], including feGRASS [11], feGRASS-IChol [9] and algebraic multigrid (AMG) [14] based PCG solvers. Finally, we test the performance of PowerRChol on other real-world SDDM from [1]. All solvers are implemented in C++. The relative tolerance of PCG is always set to 10^{-6} if not stated explicitly. All experiments are conducted using a single CPU core of a computer with Intel Xeon E5-2630 CPU @2.40 GHz and 256 GB RAM.

4.1 Validation of the Proposed Techniques

We first compare the proposed LT-RChol (Alg. 3) with the original RChol (Alg. 1). Both algorithms utilizes the same AMD reordering prior to factorization [2]. The results are listed in Table 1. For all the power grid cases, the proposed LT-RChol can reduce both factorization time (T_f) and PCG iteration time (T_i), leading to 1.15X speedups in total solution time (T_{tot}) on average over the RChol based solver. This demonstrates the efficiency and effectiveness of the proposed LT-RChol algorithm.

Then, we compare different matrix reordering strategies prior to LT-RChol, including the proposed LT-RChol orientated matrix reordering (Alg. 4), the AMD reordering and also the natural order (i.e. without reordering). The results are listed in Table 2. From the results we see that, although utilizing natural order takes no reordering time, the number of nonzeros is 45% larger than that of the AMD reordering on average and it takes more total solution time than the AMD based solver. On the other hand, the proposed LT-RChol orientated matrix reordering is much faster than the AMD reordering and the number of nonzeros in the resultant factor increases by only 12% on average. Consequently, the Alg. 4 based solver achieves an average 1.32X speedups in total solution time over the AMD based solver. Compared with the original RChol based solver (AMD+Alg. 1), PowerRChol (Alg. 4+Alg. 3) achieves 1.51X speedups on average for these power grid cases.

4.2 Comparison with feGRASS Based Solvers

In this subsection, we compare the proposed PowerRChol with the feGRASS based solver [11][1] and feGRASS-IChol based solver [9]. For feGRASS, $2\%|V|$ off-tree edges are recovered as [11]. For

[1]http://numbda.cs.tsinghua.edu.cn/packages/feGRASSEXE.zip

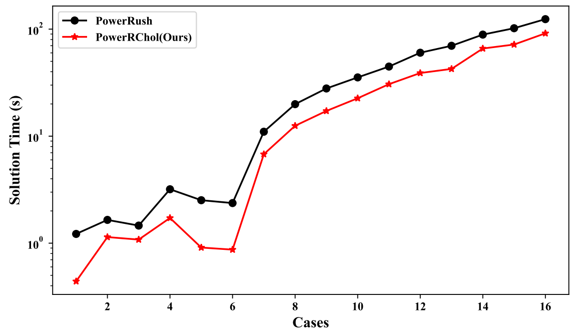

Figure 1: Comparison of total solution time between PowerRChol and PowerRush.

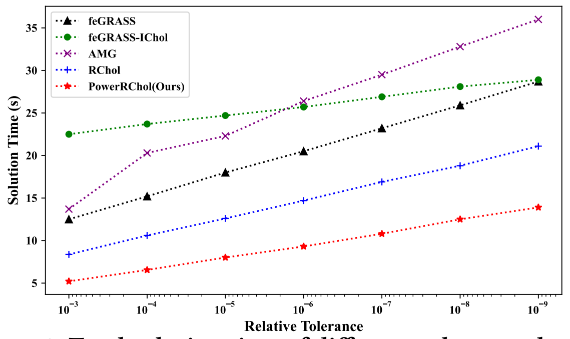

Figure 2: Total solution time of different solvers under different relative tolerance, for the case "thupg1".

feGRASS-IChol, 50%$|V|$ off-tree edges are recovered by the feGRASS algorithm and then the resulted Laplacian matrix is factorized with the incomplete Cholesky factorization (IChol) with drop tolerance 8.5×10^{-6} as [9]. The results are listed in Table 3.

Compared with the feGRASS based solver [11], PowerRChol consumes less time in both preconditioner construction phase and PCG iteration phase, thus achieves an average 1.93X speedup in total solution time. The preconditioner constructed by the feGRASS-IChol method [9] can lead to fast convergence for small cases, but the PCG iteration number increases as the matrix dimension increases. On the contrary, the convergence rate of PowerRChol is more stable. Overall, PowerRChol achieves an average 2.37X speedup over the feGRASS-IChol based solver.

PowerRush is a highly efficient tool for static power grid analysis [14][2]. It is based on algebraic multigrid preconditioned conjugate gradient method (AMG-PCG) and utilizes a trick of merging small

Table 1: Comparison between LT-RChol and original RChol [3] (Time in unit of second)

Case	|V|	nnz	RChol [3]					LT-RChol (Alg. 3)					Sp
			T_r	T_f	T_i	N_i	T_{tot}	T_r	T_f	T_i	N_i	T_{tot}	
ibmpg3	8.5E5	3.7E6	0.84	0.41	0.76	22	2.08	0.84	0.39	0.58	17	1.89	1.10
ibmpg4	9.5E5	4.1E6	0.84	0.50	0.74	19	2.17	0.84	0.44	0.69	17	2.07	1.05
ibmpg5	1.1E6	4.3E6	0.62	0.48	1.05	25	2.25	0.62	0.43	0.97	23	2.13	1.06
ibmpg6	1.7E6	6.6E6	1.44	0.73	1.73	25	4.01	1.44	0.66	1.59	23	3.85	1.04
ibmpg7	1.5E6	6.2E6	1.54	0.72	1.26	20	3.65	1.54	0.62	1.04	17	3.34	1.09
ibmpg8	1.5E6	6.2E6	1.53	0.71	1.29	21	3.65	1.53	0.61	1.05	17	3.36	1.09
thupg1	5.0E6	2.2E7	4.14	2.73	7.33	26	14.7	4.13	2.43	5.18	18	12.3	1.20
thupg2	9.0E6	3.9E7	7.15	5.40	15.2	25	29.3	7.17	4.74	11.9	20	25.9	1.13
thupg3	1.2E7	5.1E7	9.19	6.53	20.9	29	37.8	9.24	5.90	14.5	20	30.8	1.23
thupg4	1.5E7	6.6E7	13.5	8.32	30.1	32	53.4	13.8	7.48	18.2	19	41.1	1.30
thupg5	1.9E7	8.5E7	19.6	10.5	33.8	28	65.6	19.7	9.42	25.7	21	56.8	1.15
thupg6	2.4E7	1.1E8	22.4	13.1	44.2	29	81.8	22.4	11.6	33.6	22	69.8	1.17
thupg7	2.8E7	1.2E8	24.4	15.5	52.1	29	94.5	24.4	14.0	37.4	21	78.5	1.20
thupg8	4.0E7	1.8E8	42.4	22.2	79.5	30	148	42.4	19.1	57.6	22	124	1.19
thupg9	5.2E7	2.2E8	59.5	31.0	108	30	203	59.4	25.6	80.2	24	168	1.21
thupg10	6.0E7	2.6E8	64.9	33.6	130	32	236	65.0	30.1	103	25	205	1.15
Average	-	-	-	-	-	-	-	-	-	-	-	-	1.15

T_r, T_f and T_i are the time of matrix reordering, randomized factorization and PCG iteration, respectively. T_{tot} is the total solution time and N_i is the PCG iteration number. Sp is the speedup in total solution time.

Table 2: Comparison among different matrix reordering strategies. (Time in unit of second)

Case	AMD order (Alg.3)					Natural order				Alg. 4 (PowerRChol)					Sp_a	Sp_b
	T_r	NNZ	T_i	N_i	T_{tot}	NNZ	T_i	N_i	T_{tot}	T_r	NNZ	T_i	N_i	T_{tot}		
1	0.84	4.6E6	0.58	17	1.89	5.3E6	0.89	24	1.52	0.03	4.4E6	0.77	23	1.33	1.42	1.56
2	0.84	5.5E6	0.69	17	2.07	6.4E6	0.90	21	1.72	0.03	5.6E6	0.63	16	1.32	1.57	1.64
3	0.62	5.8E6	0.97	23	2.13	7.2E6	1.29	26	2.05	0.03	6.6E6	1.18	25	1.88	1.13	1.20
4	1.44	8.7E6	1.59	23	3.85	1.2E7	1.70	20	2.97	0.05	1.0E7	2.55	31	3.68	1.05	1.09
5	1.54	8.0E6	1.04	17	3.34	9.2E6	1.42	22	2.52	0.05	8.2E6	1.29	20	2.33	1.43	1.57
6	1.53	8.0E6	1.05	17	3.36	9.1E6	1.49	23	2.57	0.05	8.2E6	1.22	19	2.26	1.49	1.62
7	4.13	3.0E7	5.18	18	12.3	4.7E7	8.90	22	14.8	0.21	3.4E7	4.87	14	9.31	1.32	1.58
8	7.17	5.4E7	11.9	20	25.9	8.5E7	16.5	21	27.3	0.37	6.1E7	10.8	16	19.1	1.36	1.53
9	9.24	7.0E7	14.5	20	30.8	1.1E8	24.2	23	38.6	0.50	8.1E7	13.7	15	24.4	1.26	1.55
10	13.8	9.1E7	18.2	19	41.1	1.4E8	30.3	22	48.6	0.63	1.0E8	19.2	16	33.2	1.24	1.61
11	19.7	1.1E8	25.7	21	56.8	1.8E8	42.3	24	65.9	0.81	1.3E8	25.0	16	43.2	1.31	1.52
12	22.4	1.4E8	33.6	22	69.8	2.2E8	48.3	22	77.0	1.02	1.6E8	33.5	17	55.8	1.25	1.47
13	24.4	1.7E8	37.4	21	78.5	2.8E8	59.0	23	94.7	1.21	1.9E8	36.9	16	63.8	1.23	1.48
14	42.4	2.4E8	57.6	22	124	3.8E8	90.7	24	141	1.76	2.8E8	61.0	18	98.9	1.25	1.50
15	59.4	3.1E8	80.2	24	168	5.1E8	118	25	183	2.21	3.5E8	70.0	16	119	1.41	1.71
16	65.0	3.6E8	103	25	205	5.9E8	158	27	235	2.58	4.1E8	89.3	17	148	1.39	1.59
Avg	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1.32	1.51

We use number 1-16 in column "Case" to refer to the PG cases in Table 1. The meanings of T_r, T_i, N_i, T_{tot} are the same as those in Table 1. NNZ denotes the number of nonzeros in resultant factor L. Sp_a is the speedup of PowerRChol (Alg. 4+LT-RChol) over AMD+LT-RChol and Sp_b is that of PowerRChol over original RChol. The results of original RChol are in Table 1.

Table 3: Comparing PowerRChol with the feGRASS based solvers and AMG-PCG solver. (Time in unit of second)

Case	feGRASS			feGRASS-IChol			AMG	PowerRChol			Speedups		
	T_i	N_i	T_{tot}	T_i	N_i	T_{tot}	T_{tot}	T_i	N_i	T_{tot}	Sp_1	Sp_2	Sp_3
1	1.22	49	2.19	0.40	11	1.79	-	0.77	23	1.33	1.65	1.35	-
2	0.79	27	1.92	0.47	6	3.37	2.45	0.63	16	1.45	2.55	1.35	1.86
3	1.86	59	3.00	0.52	8	2.93	3.22	1.18	25	1.88	1.60	1.56	1.71
4	5.43	82	6.17	0.75	8	4.34	22.4	2.55	31	3.68	1.68	1.18	6.09
5	2.30	53	4.09	0.50	5	4.51	16.6	1.29	20	2.33	1.76	1.94	7.12
6	2.39	54	4.14	0.73	10	3.52	-	1.22	19	2.26	1.83	1.56	-
7	10.3	59	20.5	4.01	11	25.7	26.4	4.87	14	9.31	2.20	2.76	2.84
8	20.2	62	40.6	9.86	15	51.0	-	10.8	16	19.1	2.13	2.67	-
9	26.0	62	52.6	14.7	17	68.2	85.0	13.7	15	24.4	2.16	2.80	3.48
10	32.5	59	66.9	19.8	18	87.7	-	19.2	16	33.2	2.02	2.64	-
11	41.8	61	89.1	26.2	20	111	-	25.0	16	43.2	2.06	2.57	-
12	51.9	61	112	33.2	21	127	186	33.5	17	55.8	2.01	2.28	3.33
13	62.3	61	135	51.7	24	187	-	36.9	16	63.8	2.12	2.93	-
14	90.0	62	196	77.5	26	262	-	61.0	18	98.9	1.98	2.65	-
15	122	61	257	128	28	406	345	70.0	16	119	2.16	3.41	2.90
16	139	62	306	154	30	468	501	89.3	17	148	2.07	3.16	3.39
Avg	-	-	-	-	-	-	-	-	-	-	1.93	2.37	3.64

We use number 1-16 in column "Case" to refer to the PG cases in Table 1. The meanings of T_i, N_i, T_{tot} are the same as those in Table 1. Sp_1, Sp_2, Sp_3 denote speedups of PowerRChol over feGRASS [11], feGRASS-IChol [9] and AMG-PCG [14] in T_{tot}. "-" means it does not converge in 500 iterations.

[2]http://act.buaa.edu.cn/jianlei/PowerRush/

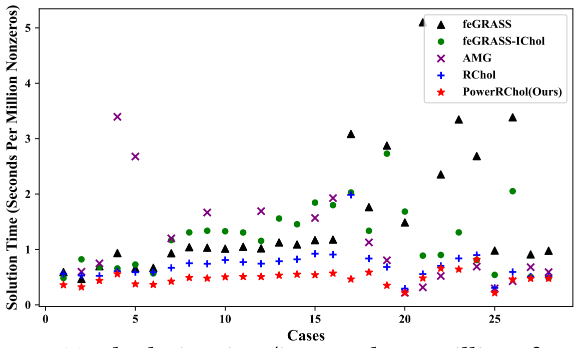

Figure 3: Total solution time (in seconds per million of nonzeros in coefficient matrix) of different solvers.

via resistors to reduce the size and condition number of the matrix to be solved. We first compare the proposed PowerRChol with the AMG-PCG solver inside PowerRush [14]. The results are listed in Table 3. For the cases where the AMG-PCG solver converges in fewer than 500 iterations, PowerRChol is 3.64X faster than it on average. We also compare PowerRChol with PowerRush directly. Here a similar trick of merging small resistors as in PowerRush is employed. The results are plotted in Fig. 1. PowerRChol outperforms PowerRush on all cases and achieves an average 1.76X speedup.

To show the performance of these solvers under different accuracy requirement, we gradually decrease the relative tolerance of PCG from 10^{-3} to 10^{-9} and record the total solution time of these solvers on the case "thupg1" from [13]. The results are plotted in Fig. 2, demonstrating that PowerRChol can achieve the best performance under different accuracy requirement.

4.3 Results on More Test Cases

In this subsection, we test the performance of PowerRChol on some other large sparse SDDM from [1]. We compare the proposed PowerRChol with the feGRASS based solver [11], the feGRASS-IChol based solver [9], the AMG-PCG solver [14] and the RChol based solver [3]. The results are listed in Table 4. PowerRChol achieves the best performance among these solvers for most cases. Although the AMG-PCG solver is faster for some cases, it is not as robust as other solvers and does not converge within 500 iterations sometimes. For those cases where AMG-PCG converges, PowerRChol runs 1.25X faster than it on average. Besides, PowerRChol achieves 1.54X or larger speedups over the other three solvers on these cases.

To illustrate the results more clearly, the total solution time of these solvers on all test cases is plotted in Fig. 3. The cases numbered 1 to 16 correspond to the power grid matrices in [12, 13] and the cases numbered 17 to 28 correspond to the sparse SDDM used in Table 4. For all cases, the solution time (in seconds per million nonzeros) of PowerRChol is below 1, which demonstrates the robustness and the scalability of the proposed PowerRChol.

Table 4: Results on more test cases. (Time in unit of second)

Case	|V|	nnz	Total Solution Time					Speedups			
			feGRASS	[9]	AMG	RChol	Ours	Sp_1	Sp_2	Sp_3	Sp_4
com-Youtube	1.1E6	7.1E6	21.9	14.4	-	14.1	**3.29**	6.66	4.38	-	4.29
com-Amazon	3.3E5	2.2E6	3.88	2.94	2.48	1.84	**1.29**	3.01	2.28	1.92	1.43
com-DBLP	3.2E5	2.4E6	6.90	6.55	1.93	1.64	**0.84**	8.21	7.80	2.30	1.95
coPaperDBLP	5.4E5	3.1E7	46.1	52.2	6.76	9.09	**6.69**	6.89	7.80	1.01	1.36
ecology2	1.0E6	5.0E6	25.5	4.44	**1.58**	2.78	2.41	10.6	1.84	0.66	1.15
thermal2	1.2E6	8.2E6	19.3	7.38	**4.27**	5.77	5.39	3.58	1.37	0.79	1.07
G3_circuit	1.6E6	7.8E6	26.1	10.2	-	6.54	**5.00**	5.22	2.04	-	1.31
NACA0015	1.0E6	7.3E6	19.6	5.91	**5.04**	6.56	5.97	3.28	0.99	0.84	1.10
fe_tooth	7.8E4	9.8E5	0.96	0.53	0.29	0.30	**0.21**	4.57	2.52	1.38	1.43
fe_ocean	1.4E5	9.6E5	3.25	1.97	**0.41**	0.57	0.44	7.39	4.48	0.93	1.30
mo2010	3.4E5	2.0E6	1.82	1.01	1.36	1.02	**0.95**	1.92	1.06	1.43	1.07
oh2010	3.7E5	2.1E6	2.05	1.02	1.24	1.07	**1.00**	2.05	1.02	1.24	1.07
Average	-	-	-	-	-	-	-	5.28	3.13	1.25	1.54

Sp_1, Sp_2, Sp_3, Sp_4 denote speedups of PowerRChol over feGRASS based solver [11], feGRASS-IChol based solver [9], AMG [14] and RChol based solver [3], respectively. "-" means it does not converge in 500 iterations.

5 CONCLUSIONS

In this work, we propose an efficient iterative linear equation solver named PowerRChol for large-scale power grid analysis. It is based on the RChol algorithm and incorporates two novel contributions for producing effective preconditioner: an improved LT-RChol factorization algorithm with provable linear-time complexity, and a fast randomized factorization orientated matrix reordering algorithm. PowerRChol not only reduces the precondioner construction cost but also brings faster convergence of PCG iteration. Extensive experimens on power-grid benchmarks and other real-world matrices demonstrate the robustness and the efficiency of PowerRChol. On average, it runs 1.5X faster than the original RChol based solver [3], and 1.93X and 5.28X faster than the feGRASS based PCG solver [11] for power-grid and other cases respectively.

REFERENCES

[1] [n. d.]. SuiteSparse Matrix Collection. https://sparse.tamu.edu/
[2] P. R. Amestoy, T. A. Davis, and I. S. Duff. 1996. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.* 17, 4 (1996), 886–905.
[3] C. Chen, T. Liang, and G. Biros. 2021. RCHOL: Randomized Cholesky factorization for solving SDD linear systems. *SIAM Journal on Scientific Computing* 43, 6 (2021), C411–C438.
[4] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. 2008. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Software* 35, 3 (2008), 22.
[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition.* The MIT Press.
[6] Z. Feng. 2020. GRASS: graph spectral sparsification leveraging scalable spectral perturbation analysis. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 39, 12 (2020), 4944–4957.
[7] D. Krishnan, R. Fattal, and R. Szeliski. 2013. Efficient preconditioning of Laplacian matrices for computer graphics. *ACM Trans. Graph.* 32, 4, Article 142 (2013).
[8] R. Kyng and S. Sachdeva. 2016. Approximate Gaussian elimination for Laplacians - Fast, sparse, and simple. In *Proc. FOCS.* 573–582.
[9] C. Li, C. An, Z. Gao, F. Yang, Y. Su, and X. Zeng. 2023. Unleashing the power of graph spectral sparsification for power grid analysis via incomplete Cholesky factorization. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 42, 9 (2023), 3053–3066.
[10] Z. Liu and W. Yu. 2022. Pursuing more effective graph spectral sparsifiers via approximate trace reduction. In *Proc. DAC.* 613–618.
[11] Z. Liu, W. Yu, and Z. Feng. 2022. feGRASS: Fast and effective graph spectral sparsification for scalable power grid analysis. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 41, 3 (2022), 681–694.
[12] S. R. Nassif. [n. d.]. IBM power grid benchmarks. https://web.ece.ucsb.edu/~lip/PGBenchmarks/ibmpgbench.html
[13] J. Yang and Z. Li. [n. d.]. THU power grid benchmarks. http://tiger.cs.tsinghua.edu.cn/PGBench/
[14] J. Yang, Z. Li, Y. Cai, and Q. Zhou. 2014. PowerRush: An efficient simulator for static power grid analysis. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 22, 10 (2014), 2103–2116.
[15] W. Yu, T. Zhang, X. Yuan, and H. Qian. 2013. Fast 3-D thermal simulation for integrated circuits with domain decomposition method. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 32, 12 (2013), 2014–2018.
[16] A. Zharmagambetov and M. A. Carreira-Perpinan. 2022. Semi-supervised learning with decision trees: Graph Laplacian tree alternating optimization. In *Proc. NeurIPS*, Vol. 35. 2392–2405.