Randomized Cholesky Factorization With Threshold-Based Multisampling for Power Grid Simulation

Zhiqiang Liu and Wenjian Yu, *Senior Member, IEEE*

***Abstract*—Transient simulation of large power grids (PGs) can be extremely challenging because linear equation systems with millions of unknowns need to be solved at each time step. The iterative equation solvers can be more scalable and efficient than direct solvers, thanks to the preconditioning approaches. Recently, randomized Cholesky factorization (RChol) was proposed, showing promising performance in preconditioning symmetric diagonally dominant M-matrices (SDDMs). However, it does not allow to include more fill-ins, making it less flexible and less efficient in problems like PG transient simulation. In this work, a RChol with a threshold-based multisampling strategy (RCholT) is proposed. RCholT allows to control the sparsity of preconditioners by a user-defined threshold and can construct more effective preconditioners than RChol. As the result, the RCholT-based transient simulator is 1.7× faster than the RChol-based one and 2.3× faster than the graph sparsification-based one on SDDM and PG benchmarks.**

***Index Terms*—Iterative solver, power grid (PG) transient simulation, preconditioning, randomized Cholesky factorization (RChol).**

I. Introduction

Transient simulation of power grids (PGs) is a must for modern very large-scale integrated (VLSI) circuit design. Due to the increasing complexity of on-chip PGs, PG transient analysis can be extremely time consuming because linear equations with millions or even billions of unknowns need to be solved at each time step. There are typically two methods for solving linear equations: 1) direct solver and 2) iterative solver. Traditionally, direct solvers such as CHOLMOD [1] are thought to be very efficient for transient simulation of small or medium PGs [2], [3], because Cholesky factorization can be seen as a preprocessing step and only backward/forward substitutions are performed at each time step. However, direct solvers do not scale well to large problems due to the large amount of memory usage.

Iterative solvers based on preconditioned conjugate gradient (PCG) method are more scalable, and suitable for varied time steps because different linear systems with similar coefficient matrices can share the same preconditioner [4]. Thus, the effectiveness of preconditioner is critical to the performance of the PG transient simulators using iterative solver. An effective preconditioner should be sparse (low-cost at each iteration) and largely reduce the relative condition number (fast convergence of iteration). Several preconditioning approaches have been proposed to construct effective preconditioner for PG simulation or equivalently, solving an SDDM [4], [5], [6], [7], [8]. Among them, the RChol [8], derived from the theoretical work of "Laplacian Paradigm 2.0," has shown promising performance in preconditioning large SDDMs. However, RChol lacks an effective scheme to trade-off the sparsity of the preconditioner matrix against the preconditioning quality. In other words, it does not allow users to include more fill-ins to construct a more effective preconditioner. This weakens its efficiency in problems with multiple right-hand-sides (RHS), such as PG transient simulation.

In this work, we aim to develop a more effective preconditioning approach based on RChol. Our contributions are as follows.

1) A RChol with a threshold-based multisampling strategy (RCholT) is proposed. RCholT allows to control the sparsity of preconditioner by a user-defined threshold and can construct more effective preconditioners than RChol [8], leading to faster convergence.
2) Based on RCholT and the strategy of varied time steps, an efficient transient simulator of large PGs is developed.

Extensive experiments have been conducted to verify the effectiveness of the proposed RCholT and the transient PG simulator based on it. The results show that RCholT with a threshold $\varepsilon = 0.02$ achieves 1.7× speedup on average over RChol [8] in the PCG iteration time. RCholT is also superior to the state-of-the-art graph sparsification approach [4], leading to 4.4× reductions in the number of PCG iterations on average with comparable sparsity of preconditioners. While simulating IBMPG and THUPG benchmarks [9], [10] for 5 ns time, the RCholT-based transient simulator is 1.7× faster than the RChol-based one on average, and achieves 2.3× speedup on average over the best results of graph sparsification-based methods.

II. Background

The PG is modeled as an RLC network for transient simulation. With modified nodal analysis, the following differential algebra equations (DAEs) is obtained [2], [4], [7]

$$\begin{bmatrix} G & B_l^T \\ -B_l & O \end{bmatrix}\begin{bmatrix} x \\ i_l \end{bmatrix} + \begin{bmatrix} C & O \\ O & L_0 \end{bmatrix}\begin{bmatrix} \frac{dx}{dt} \\ \frac{di_l}{dt} \end{bmatrix} = \begin{bmatrix} u \\ 0 \end{bmatrix} \tag{1}$$

where G and C are the conductance matrix and the capacitance matrix, respectively. L_0 is a diagonal matrix with inductance values and B_l is the incidence matrix corresponding to inductors. x, i_l, and u denote the vector of node voltages, inductor branch currents, and current sources, respectively. The initial node voltages can be obtained by performing DC analysis. Then with time integration schemes like backward Euler scheme, the DAEs are converted to a sequence of linear equation systems for the solution at consecutive time points

$$\begin{bmatrix} G + \frac{C}{h} & B_l^T \\ -B_l & \frac{L_0}{h} \end{bmatrix}\begin{bmatrix} x(t+h) \\ i_l(t+h) \end{bmatrix} = \begin{bmatrix} \frac{C}{h}x(t) + u(t+h) \\ \frac{L_0}{h} i_l(t) \end{bmatrix}. \tag{2}$$

Eliminate current variables $i_l(t+h)$ and we can obtain that

$$\left(G + \frac{C}{h} + h\tilde{L}\right)x(t+h) = \frac{C}{h}x(t) + u(t+h) - B_l^T i_l(t). \tag{3}$$

Here, $x(t+h)$ is to be solved, h is time step and $\tilde{L} = B_l^T L_0^{-1} B_l$. The coefficient matrix $G + (C/h) + h\tilde{L}$ is a SDDM [2], [4], [7], i.e.,

Manuscript received 17 July 2023; revised 20 December 2023; accepted 16 March 2024. Date of publication 22 March 2024; date of current version 22 August 2024. This work was supported in part by the NSFC under Grant 62090025, and in part by the Beijing Natural Science Foundation under Grant Z230002. This article was recommended by Associate Editor E. R. Keiter. *(Corresponding author: Wenjian Yu.)*

The authors are with the Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing 100084, China (e-mail: yu-wj@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TCAD.2024.3381076

Algorithm 1 RChol for SDDM

```
Input: An SDDM $A \in \mathbf{R}^{N\times N}$ corresponding to Laplacian $\mathcal{G}$ of a weighted undirected graph.
Output: The lower triangular factor matrix $L \in \mathbf{R}^{N\times N}$.
1: Set $L$ to a zero matrix, and diagonal matrix $D = A - \mathcal{G}$.
2: for $k = 1$ to $N$ do
3:     Set $d_k = \mathcal{G}(k,k) + D(k,k)$.
4:     Set $L(k,k) = \sqrt{d_k}$ and $L(k+1:n,k) = \frac{\mathcal{G}(k+1:n,k)}{\sqrt{d_k}}$.
5:     Sort $N_k = \{i | i > k \ \& \ \mathcal{G}(i,k) \neq 0\}$ such that $|\mathcal{G}(n_1,k)| \le |\mathcal{G}(n_2,k)| \le \cdots \le |\mathcal{G}(n_{|N_k|},k)|$.
6:     for $j = 1$ to $|N_k|$ do
7:         Set $D(n_j,n_j) = D(n_j,n_j) - \frac{D(n_j,n_j)\mathcal{G}(n_j,k)}{d_k}$.
8:         Set $s_{k,j} = \sum_{j<i\le|N_k|} |\mathcal{G}(n_i,k)|$, and sample $n_s$ from $\{n_i : j < i \le |N_k|\}$ with probability $\frac{|\mathcal{G}(n_i,k)|}{s_{k,j}}$.
9:         Set $\mathcal{G} = \mathcal{G} + \frac{s_{k,j}|\mathcal{G}(n_j,k)|}{d_k} f_{n_j,n_s} f_{n_j,n_s}^T$.
10:    end for
11: end for
```

a symmetric diagonally dominant matrix with positive diagonal and nonpositive off-diagonal elements. It can be seen as a graph Laplacian matrix $\mathcal{G}$ plus a non-negative diagonal matrix D. Each resistor (with resistance r), each capacitor (with capacitance c), and each inductor (with inductance l) corresponds to an edge in the graph, with edge weight $1/r$, c/h, and h/l, respectively. The values of those elements adjacent to the ground node contribute to the diagonal matrix D [2].

In this work, we focus on developing more effective preconditioner for iteratively solving

$$Ax = b \tag{4}$$

where A is an SDDM which equals a graph Laplacian matrix $\mathcal{G}$ plus a non-negative diagonal matrix D. Notice that the coefficient matrix in PG simulation is a kind of SDDM.

In [8], an RChol method named RChol was proposed to construct effective preconditioner for SDDM. Instead of introducing a clique to the matrix's graph 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 [8], it is proven that the sampled spanning tree is an unbiased estimator of the clique and the whole procedure is breakdown-free. The RChol algorithm is described in Algorithm 1 [8]. Here, f_i denotes the identity matrix's ith column and $f_{i,j} = f_i - f_j$.

III. Randomized Cholesky Factorization With Threshold-Based Multisampling

The main drawback of RChol is that the number of nonzeros in the resulted Cholesky factor is fixed. Thus, RChol may be inefficient for problems with multiple RHS, due to the lack of ability of reducing the number of PCG iterations through increasing the number of nonzeros. To address this issue, an RChol with threshold-based multisampling (RCholT) is proposed in this work. RCholT allows users to include more fill-ins in the resulted factor, leading to more flexible and effective preconditioning.

Below, we present the idea of multisampling to increase the flexibility of randomized Cholesky preconditioners, followed by the core of RCholT algorithm, i.e., an effective threshold-based scheme to determine the number of samples adaptively.

A. RChol With Naive Multisampling

Now we take a deeper look at RChol. Let N_k denote the neighbors of node k in the graph, i.e., $N_k = \{i | i > k \ \& \ \mathcal{G}(i,k) \neq 0\}$ and $t = |N_k|$. In RChol, the neighbors $n_i \in N_k$ are first sorted such that $|\mathcal{G}(n_1,k)| \le |\mathcal{G}(n_2,k)| \le \cdots \le |\mathcal{G}(n_t,k)|$. The clique introduced by eliminating node k in Cholesky factorization corresponds to the Laplacian matrix

$$\Theta_k = \sum_{1\le j\le|N_k|} \sum_{j<i\le|N_k|} \frac{\mathcal{G}(n_i,k)\mathcal{G}(n_j,k)}{d_k} f_{n_i,n_j} f_{n_i,n_j}^T. \tag{5}$$

It is actually the sum of many star graphs' Laplacian. For each neighbor $n_j \in N_k$, the following Laplacian matrix of star graph is added to the graph Laplacian $\mathcal{G}$

$$\Gamma_{k,j} = \sum_{j<i\le|N_k|} \frac{\mathcal{G}(n_i,k)\mathcal{G}(n_j,k)}{d_k} f_{n_i,n_j} f_{n_i,n_j}^T \tag{6}$$

which also satisfies

$$\Theta_k = \sum_{1\le j\le|N_k|} \Gamma_{k,j}. \tag{7}$$

In RChol, each star graph is replaced by a random edge, which is sampled from the star graph with probability proportional to edge weights. The weight of the sampled edge is set to the sum of the edge weights in the corresponding star graph. The approximation error of the randomized Cholesky factor with respect to the classical Cholesky factor stems from sampling. Thus a naive approach to reduce the approximation error is to perform multisampling, i.e., replacing each star graph by α random edges and setting the weight of each random edge to $(1/\alpha)$ of the weight of the star graph. This turns steps 8 and 9 of Algorithm 1 to

```
8-1:   Set $s_{k,j} = \sum_{j<i\le|N_k|} |\mathcal{G}(n_i,k)|$.
8-2:   for $m = 1$ to $\alpha$ do
8-3:       Sample $n_s$ from $\{n_i : j < i \le |N_k|\}$ with probability $\frac{|\mathcal{G}(n_i,k)|}{s_{k,j}}$.
8-4:       Set $\mathcal{G} = \mathcal{G} + \frac{s_{k,j}|\mathcal{G}(n_j,k)|}{\alpha d_k} f_{n_j,n_s} f_{n_j,n_s}^T$.
9:     end for
```

We call the derived algorithm as the RChol algorithm with naive multisampling, denoted by RChol(α).

B. RCholT: RChol With Threshold-Based Multisampling

Using a fixed number of samplings for all star graphs may be inadequate. So, we derive a scheme for determining the number of samplings based on an analysis of the approximation error of the star graph.

Consider the approximation error of the star graph introduced by the jth neighbor of the kth node in Cholesky factorization, whose Laplacian matrix is $\Gamma_{k,j}$ as defined in (6). In RChol, the star graph is replaced by a random edge (n_j, n_s), which is sampled from the star graph with probability proportional to edge weights. Let $F_{n_i,n_j} = f_{n_i,n_j} f_{n_i,n_j}^T$. The approximation error of $\Gamma_{k,j}$, denoted as $\Delta_{k,j}^s$ which relies on the random variable n_s, can be written as

$$\begin{aligned}\Delta_{k,j}^s &= \frac{s_{k,j}|\mathcal{G}(n_j,k)|}{d_k} F_{n_s,n_j} - \Gamma_{k,j} \\ &= \frac{|\mathcal{G}(n_j,k)|}{d_k}\left(\sum_{i\neq s} |\mathcal{G}(n_i,k)| F_{n_s,n_j} - \sum_{i\neq s} |\mathcal{G}(n_i,k)| F_{n_i,n_j}\right).\end{aligned} \tag{8}$$

Because the n_s is a random variable and $\Delta_{k,j}^s$ is a random matrix. To analyze the approximation error, we analyze the expectation of the 2-norm of $\Delta_{k,j}^s$.

It can be easily shown that $\|F_{n_i,n_j}\|_2 = \|f_{n_i,n_j} f_{n_i,n_j}^T\|_2 = 2$. So we have

$$\|\Delta_{k,j}^s\|_2 \le 4 \frac{|\mathcal{G}(n_j,k)| \sum_{j<i\le|N_k|} |\mathcal{G}(n_i,k)|}{d_k}. \tag{9}$$

The probability of n_s being sampled is

$$P_{k,j,s} = \frac{|\mathcal{G}(n_s,k)|}{\sum_{j<i\le|N_k|} |\mathcal{G}(n_i,k)|}. \tag{10}$$

So the expectation of the 2-norm of $\Delta_{k,j}^s$ satisfies

$$\begin{aligned}\mathbb{E}\Big(\|\Delta_{k,j}^s\|_2\Big) &= \sum_{j<s\le|N_k|} P_{k,j,s} \Big\|\Delta_{k,j}^s\Big\|_2 \\ &\le \sum_{j<s\le|N_k|} \frac{|\mathcal{G}(n_s,k)|}{\sum_i |\mathcal{G}(n_i,k)|} \times 4 \frac{|\mathcal{G}(n_j,k)| \sum_i |\mathcal{G}(n_i,k)|}{d_k} \\ &= 4 \frac{|\mathcal{G}(n_j,k)| \sum_{j<s\le|N_k|} |\mathcal{G}(n_s,k)|}{d_k}. \end{aligned} \tag{11}$$

Let $e_{k,j} = [(|\mathcal{G}(n_j,k)| \sum_{j<s\le|N_k|} |\mathcal{G}(n_s,k)|)/d_k]$ and (11) implies that the approximation error of $\Gamma_{k,j}$ depends on $e_{k,j}$. The larger $e_{k,j}$ is, the larger the deviation between the randomly sampled edge and the star graph is. For those star graphs with smaller $e_{k,j}$, single sampling as in RChol suffices to produce good approximations to original star graphs. While for those star graphs with larger $e_{k,j}$, the strategy of multisampling should be employed to reduce approximation errors.

In this work, we adopt the following formula to determine the number of samples (denoted as $\alpha_{k,j}$), which is based on a user-defined threshold ϵ

$$\alpha_{k,j} = \begin{cases} 1, & \frac{|\mathcal{G}(n_j,k)| \sum_{j<s\le|N_k|} |\mathcal{G}(n_s,k)|}{d_k^2} \le \epsilon \\ 1 + \log\left(\frac{|\mathcal{G}(n_j,k)| \sum_{j<s\le|N_k|} |\mathcal{G}(n_s,k)|}{d_k^2 \epsilon}\right), & \text{otherwise.} \end{cases} \tag{12}$$

Here, we use the $(e_{k,j}/d_k)$, instead of $e_{k,j}$, to determine $\alpha_{k,j}$, because it can be shown that the $(e_{k,j}/d_k)$ is a normalized number which lies in $(0, 1)$. Therefore, the number of samplings can be effectively controlled by the user-defined threshold ϵ, which also lies in $(0, 1)$. To avoid using too many samples for one star graph, we utilize the log function in (12). Otherwise, the $\alpha_{k,j}$ can be very large when a small ϵ is employed and the density of resultant factors will increase rapidly as ϵ decreases. A smaller ϵ results in a more effective and denser preconditioner. If $\epsilon = 1$, the resulted factor is identical to the factor produced by RChol. The whole RCholT algorithm is described as Algorithm 2.

IV. Experimental Results

We have implemented the proposed RCholT algorithm, compared with the original codes of RChol.[1] Three graph sparsification approaches are also employed as baselines, including GRASS [6],[2] feGRASS [5],[3] and the approximate trace reduction based graph sparsification (atrSpar) [4]. The relative tolerance for PCG convergence is always set to 10^{-6}. All these programs are written in C++. 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.

[1]https://github.com/ut-padas/rchol

[2]https://sites.google.com/mtu.edu/zhuofeng-graphspar/home

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

Algorithm 2 RChol With Threshold-Based Multisampling for SDDM (RCholT)

```
Input: An SDDM $A \in \mathbf{R}^{N\times N}$ corresponding to Laplacian $\mathcal{G}$, a threshold to control the density of preconditioner: $\epsilon$.
Output: The lower triangular factor matrix $L$.
1:  Set $L$ to a zero matrix, and diagonal matrix $D = A - \mathcal{G}$.
2:  for $k = 1$ to $N$ do
3:      Set $d_k = \mathcal{G}(k,k) + D(k,k)$.
4:      Set $L(k,k) = \sqrt{d_k}$ and $L(k+1:n, k) = \frac{\mathcal{G}(k+1:n,k)}{\sqrt{d_k}}$.
5:      Sort $N_k = \{i | i > k \ \& \ \mathcal{G}(i,k) \ne 0\}$ such that $|\mathcal{G}(n_1,k)| \le |\mathcal{G}(n_2,k)| \le \cdots \le |\mathcal{G}(n_{|N_k|},k)|$.
6:      for $j = 1$ to $|N_k|$ do
7:          Set $D(n_j,n_j) = D(n_j,n_j) - \frac{D(n_j,n_j)\mathcal{G}(n_j,k)}{d_k}$.
8:          Set $s_{k,j} = \sum_{j<i\le|N_k|} |\mathcal{G}(n_i,k)|$.
9:          Compute $\alpha_{k,j}$ using Equation (12).
10:         for $m = 1$ to $\lfloor\alpha_{k,j}\rfloor$ do
11:             Sample $n_s$ from $\{n_i : j < i \le |N_k|\}$ with probability $\frac{|\mathcal{G}(n_i,k)|}{s_{k,j}}$.
12:             Set $\mathcal{G} = \mathcal{G} + \frac{s_{k,j}|\mathcal{G}(n_j,k)|}{\alpha_{k,j} d_k} f_{n_j,n_s} f_{n_j,n_s}^T$.
13:         end for
14:     end for
15: end for
```

A. Results on Solving SDDM Linear Systems

We first compare the proposed RCholT (Algorithm 2) with RChol and graph sparsification approaches on solving general SDDM linear systems. The test cases cover a variety of real-world SDDMs [11], which is a superset of those used in [4]. For RCholT, the threshold ϵ is set to 0.02 for all cases. Here, we do not tune the threshold value carefully and the experimental results show that simply setting it to 0.02 for all cases suffices to achieve good performance in iteration time. To make a fair comparison, we manually set the sparsity parameter in graph sparsification approaches to ensure that the number of nonzeros in Cholesky factor of sparsifier is comparable to that of the factor produced by RCholT. For the RChol(α) algorithm with naive multisampling, the parameter α is set to 2.

The results are listed in Table I. For each preconditioning approach, we record the time for setting up preconditioners (T_{set}) and the time for PCG iteration (T_{iter}). For the three graph sparsification approaches, T_{set} includes the time for graph sparsification and the time for reordering (using AMD) and factorizing Laplacian matrix of sparsifier. For three RChol-based approaches, T_{set} includes the time for matrix reordering (using AMD) and the time for factorization. Since the aim of this work is to develop more effective preconditioning approaches for problems with multiple RHS, we care about T_{iter}. From the results we see that, the proposed RCholT achieves the best performance for all cases on the PCG iteration time (T_{iter}), thus making it more favourable on problems with multiple RHS.

Compared with the three graph sparsification approaches feGRASS [5], GRASS [6], and atrSpar [4], the proposed RCholT achieves 5.0×, 4.4×, and 4.3× speedups in the PCG iteration time on average, respectively. Although the numbers of nonzeros of preconditioners are comparable among the proposed RCholT and three graph sparsification approaches, the former leads to much fewer PCG iterations, showing 5.2×, 4.4×, and 4.4× reductions in the number of PCG iterations on average. This demonstrates the superiority of the proposed RCholT. Compared with RChol [8], the proposed RCholT (with the threshold ϵ set to 0.02) can construct denser and more effective preconditioners. On average, the preconditioners constructed by RCholT have 45% more nonzeros than those constructed by

TABLE I
RESULTS OF PCG SOLVERS WITH DIFFERENT PRECONDITIONING APPROACHES (TIME IN UNIT OF SECOND)

Case	N(NNZ)	feGRASS [5]			GRASS [6]			atrSpar [4]			RChol [8]			RChol(α)			RCholT (Alg. 2)				
		T_{set}	T_{iter}	$iter(nnz)$	T_{set}	T_{iter}	$iter$	T_{set}	T_{iter}	$iter$	T_{set}	T_{iter}	$iter(nnz)$	T_{set}	T_{iter}	$iter(nnz)$	T_{set}	T_{iter}	$iter(nnz)$	sp_1	sp_2
ecology2	1.0e6(5.0e6)	2.43	2.60	46(1.0e7)	33.3	2.08	36	19.3	1.79	31	1.02	1.53	39(6.1e6)	4.30	1.62	21(2.2e7)	2.11	0.85	18(1.0e7)	2.1	1.8
apache2	7.2e5(4.6e6)	3.81	6.25	123(1.2e7)	15.2	4.29	82	10.8	13.6	281	1.50	1.52	43(7.1e6)	12.9	4.10	23(6.4e7)	2.23	1.00	23(1.0e7)	13.6	1.5
parabolic	5.3e5(2.6e6)	1.17	1.07	38(5.4e6)	13.7	1.42	49	10.2	0.81	29	0.65	0.82	41(3.2e6)	1.69	0.67	20(8.0e6)	1.23	0.46	18(5.2e6)	1.8	1.8
tmt_sym	7.3e5(4.4e6)	1.96	1.91	44(8.0e6)	18.4	2.09	48	16.9	1.22	28	1.25	1.37	38(5.2e6)	4.25	1.34	19(1.9e7)	2.09	0.72	17(8.0e6)	1.7	1.9
thermal2	1.2e6(8.6e6)	4.20	3.67	49(1.4e7)	45.1	3.59	46	39.2	2.73	37	2.41	2.66	43(9.4e6)	6.46	2.41	22(2.8e7)	4.17	1.70	23(1.4e7)	1.6	1.6
G3_circuit	1.6e6(7.8e6)	2.88	6.44	75(1.6e7)	43.6	4.30	47	21.7	3.91	45	2.39	4.07	62(1.0e7)	8.79	3.39	25(3.7e7)	4.28	2.34	29(1.6e7)	1.7	1.7
M6	3.5e6(2.5e7)	20.9	17.4	85(3.9e7)	218	14.2	61	176	8.68	42	13.4	8.68	50(2.6e7)	23.7	6.87	23(7.2e7)	18.3	4.92	24(3.9e7)	1.8	1.8
NLR	4.2e6(2.9e7)	27.7	22.7	91(4.8e7)	232	18.6	68	234	10.6	42	16.3	10.6	51(3.0e7)	28.9	8.62	23(8.8e7)	21.9	5.96	24(4.6e7)	1.8	1.8
NACA0015	1.0e6(7.3e6)	3.99	2.91	48(1.2e7)	45.9	3.42	56	49.7	2.03	33	2.98	2.24	45(7.5e6)	5.65	1.77	22(1.9e7)	4.45	1.41	24(1.1e7)	1.4	1.6
333SP	3.7e6(2.6e7)	14.9	18.5	85(4.0e7)	219	12.3	52	209	8.45	37	10.7	10.2	55(2.8e7)	20.5	7.40	24(7.2e7)	16.2	5.91	26(4.3e7)	1.4	1.7
AS365	3.8e6(2.7e7)	20.8	13.7	62(4.1e7)	169	17.1	76	199	9.43	43	14.2	9.62	51(2.8e7)	24.8	7.38	23(7.5e7)	19.8	5.60	25(4.1e7)	1.7	1.7
com-Youtube	1.1e6(7.1e6)	16.3	13.6	230(1.2e7)	51.9	10.1	157	46.2	15.4	246	12.1	1.40	20(9.2e6)	32.3	4.53	14(7.1e7)	12.8	1.19	15(1.2e7)	12.9	1.2
com-Amazon	3.3e5(2.2e6)	2.66	4.60	286(3.6e6)	7.06	3.43	220	7.69	2.54	166	1.05	0.62	33(2.1e6)	2.54	0.66	21(7.4e6)	1.33	0.29	17(2.9e6)	8.8	2.1
com-DBLP	3.2e5(2.4e6)	5.58	2.61	131(5.3e6)	12.2	3.52	171	15.5	2.67	124	0.98	0.49	23(2.6e6)	5.11	0.92	17(1.7e7)	1.26	0.31	16(3.4e6)	8.6	1.6
Average	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4.3	1.7

N and NNZ are the dimension and the number of nonzeros of the corresponding matrix. T_{set} and T_{iter} are the time for setting up preconditioners and the time for PCG iteration. $iter$ and nnz denote the number of PCG iterations and the number of nonzeros of preconditioners. The nnz of GRASS and atrSpar are comparable to that of feGRASS so they are omitted. sp_1 and sp_2 are speedups of RCholT in T_{iter} to atrSpar and RChol respectively.

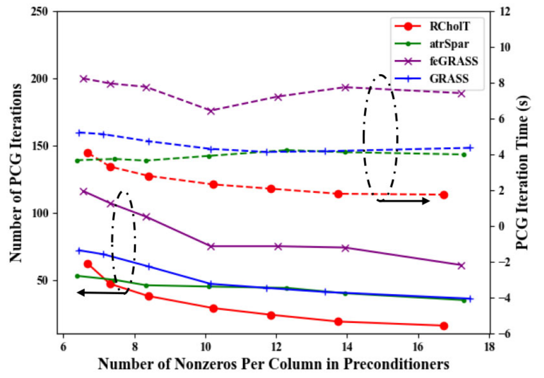

Fig. 1. Iteration numbers (solid lines) and the PCG iteration time (dashed lines) versus the density of preconditioners for the case "G3_circuit," where the results of RCholT are obtained by setting the threshold ϵ to 1.0, 0.1, 0.05, 0.02, 0.01, 0.005, and 0.002, in turn.

RChol and they lead to 2.0× reductions in the number of PCG iterations. Consequently, RCholT achieves an average 1.7× speedup over RChol in PCG iteration time. The naive multisampling strategy (i.e., RChol(α) algorithm) can also reduce the number of iterations, but it may even increase the iteration time for some cases, because the preconditioners constructed by it are much denser and the cost of each iteration is increased dramatically. The comparison between Algorithm 2 and RChol(α) algorithm shows the effectiveness of the proposed threshold-based multisampling strategy.

In order to show the effect of the threshold ϵ in RCholT, we gradually decrease the threshold in Algorithm 2 for the case "G3_circuit" and record the resultant iteration numbers and iteration time. The results are plotted in Fig. 1. As the ϵ decreases from 1.0 to 0.002, the number of nonzeros (per column) in resultant factors increases from 6.7 to 16.7, while the number of iterations decreases from 62 to 16. The results of graph sparsification approaches are obtained by setting different sparsity parameters of sparsifiers. RChol (RCholT with ϵ set to 1.0) is less effective than the atrSpar algorithm [4]. However, as preconditioners become denser, RCholT can lead to the fastest PCG convergence. This demonstrates the effectiveness of the proposed multisampling technique.

B. Results on Transient Analysis of Large Power Grids

We have implemented the transient simulators based on feGRASS [5], GRASS [6], atrSpar [4], RChol [8], direct solver CHOLMOD [1], and the proposed RCholT Algorithm (2), respectively. Test cases are from two well-known PG benchmarks [9], [10]. For the cases from [10], capacitors and inductors with random values are added (similar to IBM PG benchmarks) and periodic pulse currents are assumed. The simulation time is set to 5 ns. The strategy of fixed time step (10 ps, 500 steps totally) is adopted in the simulator based on CHOLMOD. These settings follow [4]. The strategy of varied time steps is adopted in the simulators based on iterative solvers, with time step not larger than 100 ps for error control. For RCholT, the threshold ϵ is set to 0.02.

The results are listed in Table II. We record the total time for solving linear equations in the entire transient simulation (T_{tot}). The time for PCG iteration (T_{iter}) and the time for backward/forward substitution (T_{sol}) are also listed. Note that if more time than 5 ns is simulated, T_{iter} or T_{sol} will increase while the other parts of T_{tot} will remain unchanged. Compared with the three graph sparsification approaches feGRASS [5], GRASS [6], and atrSpar [4], the proposed RCholT achieves 2.3×, 2.9× and 2.5× speedups in T_{tot}, respectively, on average. The speedups come from two aspects. First, the proposed RCholT can construct more effective preconditioners leading to fewer PCG iterations with almost the same density, thus it can reduce the iteration time T_{iter}. Second, it can also reduce the time for setting up preconditioners. Compared with RChol [8], the proposed RCholT shows an average 1.7× speedup in T_{tot}. We notice that RCholT takes more time to set up preconditioners than RChol but the resulted preconditioners are more effective, resulting in an average 1.8× reduction in iteration time T_{iter}. The comparison of T_{iter} reflects the comparison of T_{tot} when simulating longer time. We see that RChol [8] is inferior to atrSpar [4] in T_{iter} and with the proposed multisampling technique, RCholT achieves the best performance in T_{iter} for all cases. This demonstrates the effectiveness of the proposed technique for PG transient simualtion problem. Compared with the direct method CHOLMOD [1], the proposed RCholT achieves 10.8× speedups on average. Meanwhile, the maximum voltage error is below 1.2 mV for all cases, which is satisfactory since the supply voltage is 1.8 V. We adopt fixed time step in the CHOLMOD-based simulator as in [7] and [4]. Note that for CHOLMOD the result of $T_{\text{tot}} - T_{sol}$ in Table II

TABLE II
RESULTS ON TRANSIENT ANALYSIS OF LARGE PGS WITH DIFFERENT SOLVERS (TIME IN UNIT OF SECOND)

Case	N	feGRASS [5]			GRASS [6]			atrSpar [4]			RChol [8]			CHOLMOD [1]			RCholT (Alg. 2)						
		T_{iter}	T_{tot}	$iter(nnz)$	T_{iter}	T_{tot}	$iter$	T_{iter}	T_{tot}	$iter$	T_{iter}	T_{tot}	$iter(nnz)$	T_{sol}	T_{tot}	nnz	$\#pts$	$Err(V)$	T_{iter}	T_{tot}	$iter(nnz)$	sp_1	sp_2
ibmpg3t	8.5e5	87.3	89.7	24.6(5.5e6)	60.8	75.0	17.2	53.6	62.1	14.3	67.9	69.8	20.5(4.6e6)	81.1	120	5.9e7	96	1.2e-3	47.8	50.0	13.7(5.2e6)	1.8	1.4
ibmpg4t	9.5e5	24.7	27.1	9.0(7.4e6)	28.9	46.2	10.2	29.5	39.8	10.8	36.8	38.7	14.8(5.4e6)	109	181	8.4e7	60	6.8e-5	20.1	22.8	7.47(7.3e6)	1.2	1.7
ibmpg5t	1.1e6	168	171	36.2(7.7e6)	80.8	102	16.9	128	140	27.4	105	107	25.9(5.7e6)	58.4	75.1	4.0e7	96	5.4e-4	59.2	61.8	13.8(7.4e6)	2.8	1.7
ibmpg6t	1.7e6	245	250	33.7(1.1e7)	122	154	17.7	170	192	23.4	161	165	26.1(8.5e6)	81.1	95.9	5.2e7	96	9.0e-4	88.6	93.1	13.1(1.1e7)	2.7	1.9
thupg1t	5.0e6	207	246	10.9(4.5e7)	193	356	10.3	156	269	8.18	160	176	10.6(2.9e7)	466	933	3.0e8	71	9.3e-4	95.0	115	5.38(4.2e7)	2.1	1.5
thupg2t	9.0e6	361	437	10.7(7.6e7)	389	738	10.1	276	527	7.82	292	319	10.5(5.3e7)	857	1607	5.1e8	71	9.6e-4	169	205	5.21(7.6e7)	2.1	1.6
thupg3t	1.2e7	474	610	10.6(1.0e8)	494	863	10.9	366	685	7.92	405	443	11.2(6.9e7)	1005	2149	6.6e8	71	8.3e-4	224	275	5.34(9.8e7)	2.2	1.6
thupg4t	1.5e7	650	804	11.1(1.4e8)	683	1164	11.9	486	923	8.04	580	626	12.3(9.0e7)	1434	3424	9.5e8	71	6.7e-4	298	358	5.47(1.3e8)	2.2	1.7
thupg5t	1.9e7	800	1013	10.6(1.7e8)	849	1484	11.5	632	1249	8.13	699	762	11.7(1.1e8)	1848	4609	1.2e9	71	5.9e-4	378	460	5.38(1.6e8)	2.2	1.7
thupg6t	2.4e7	1039	1318	11.2(2.1e8)	1105	1892	11.8	778	1574	8.11	965	1043	12.9(1.4e8)	2388	6222	1.6e9	71	7.1e-4	467	569	5.34(2.0e8)	2.3	1.8
thupg7t	2.8e7	1344	1762	11.4(2.7e8)	1311	2346	11.5	928	1945	7.76	1133	1224	12.5(1.7e8)	4435	13722	2.3e9	71	5.5e-4	599	717	5.27(2.4e8)	2.5	1.7
thupg8t	4.0e7	1856	2478	10.9(3.9e8)	2022	3513	12.1	1333	2806	8.03	1450	1589	11.2(2.4e8)	5423	15917	2.5e9	71	5.5e-4	780	961	5.13(3.4e8)	2.6	1.7
thupg9t	5.2e7	2441	3346	10.8(5.2e8)	2482	4421	11.4	1801	3655	8.01	1967	2152	11.5(3.1e8)	8913	25285	3.9e9	71	7.7e-4	986	1227	4.75(4.5e8)	2.7	1.8
thupg10t	6.0e7	2875	3906	11.0(5.9e8)	3113	5573	12.5	2038	4566	7.65	2204	2421	11.1(3.6e8)	11627	39726	4.9e9	71	6.4e-4	1168	1448	4.89(5.1e8)	2.7	1.7
Average	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2.3	1.7

T_{tot} denotes the total time for solving linear equations in transient simulation, including the time of setting up preconditioners and the time of PCG iteration (T_{iter}) for iterative solvers, and the time of matrix factorization and the time of backward/forward substitution (T_{sol}) for CHOLMOD. nnz and $iter$ are the number of nonzeros of preconditioners (or Cholesky factors) and the resulted average number of iterations. The nnz of three graph sparsification approaches are comparable so only that of feGRASS is listed. $\#pts$ is the number of time points for all iterative solvers. $Err(V)$ is the maximum voltage error compared to the results of CHOLMOD. sp_1 and sp_2 are speedups of RCholT in T_{tot} to feGRASS and RChol respectively.

corresponds to twice matrix factorizations (for DC and transient analysis, respectively). For the largest case "thupg10t," one factorization takes more than 10 000 s and thus T_{tot} would be much larger if varied time steps are utilized due to the expensive refactorization cost.

V. CONCLUSION

An RChol approach (RCholT) based on a novel threshold-based multisampling strategy is proposed. The RCholT allows to control the sparsity of preconditioners via a threshold and can construct more effective preconditioners, leading to an average 1.7× speedup in PCG iteration time over RChol for tested SDDMs. Moreover, based on RCholT, an efficient PG transient simulator is developed, showing an average 1.7× speedup over the simulator based on RChol and an average 2.3× speedup over the best results of graph sparsification approaches for PG benchmarks.

REFERENCES

[1] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, "Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate," *ACM Trans. Math. Softw.*, vol. 35, no. 3, p. 22, 2008.

[2] J. Yang, Z. Li, Y. Cai, and Q. Zhou, "PowerRush: Efficient transient simulation for power grid analysis," in *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, 2012, pp. 653–659.

[3] T. Yu and M. D. F. Wong, "PGT_SOLVER: An efficient solver for power grid transient analysis," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2012, pp. 647–652.

[4] Z. Liu and W. Yu, "Pursuing more effective graph spectral sparsifiers via approximate trace reduction," in *Proc. Des. Autom. Conf. (DAC)*, 2022, pp. 613–618.

[5] Z. Liu, W. Yu, and Z. Feng, "feGRASS: Fast and effective graph spectral sparsification for scalable power grid analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 3, pp. 681–694, Mar. 2022.

[6] Z. Feng, "GRASS: Graph spectral sparsification leveraging scalable spectral perturbation analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4944–4957, Dec. 2020.

[7] C. Li, C. An, F. Yang, and X. Zeng, "ESPSim: An efficient scalable power grid simulator based on parallel algebraic multigrid," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, no. 1, Dec. 2022.

[8] C. Chen, T. Liang, and G. Biros, "RCHOL: Randomized Cholesky factorization for solving SDD linear systems," *SIAM J. Sci. Comput.*, vol. 43, no. 6, pp. C411–C438, 2021.

[9] S. R. Nassif. "IBM power grid benchmarks." Accessed: 2023. [Online]. Available: https://web.ece.ucsb.edu/ lip/PGBenchmarks/ibmpgbench.html

[10] J. Yang and Z. Li. "THU power grid benchmarks." 2012. [Online]. Available: http://tiger.cs.tsinghua.edu.cn/PGBench/

[11] "SuiteSparse matrix collection." Accessed: 2023. [Online]. Available: https://sparse.tamu.edu/