Power Grid Analysis with Hierarchical Support Graphs

Xueqian Zhao, Jia Wang, Zhuo Feng, and Shiyan Hu
Department of Electrical and Computer Engineering
Michigan Technological University, Houghton, MI, 49931
Email: {xueqianz, jiaw, zhuofeng, shiyan}@mtu.edu

***Abstract*— It is increasingly challenging to analyze present day large-scale power delivery networks (PDNs) due to the drastically growing complexity in power grid design. To achieve greater runtime and memory efficiencies, a variety of preconditioned iterative algorithms has been investigated in the past few decades with promising performance, while incremental power grid analysis also becomes popular to facilitate fast re-simulations of corrected designs. Although existing preconditioned solvers, such as incomplete matrix factor-based preconditioners, usually exhibit high efficiency in memory usage, their convergence behaviors are not always satisfactory. In this work, we present a novel hierarchical support-graph preconditioned iterative algorithm that constructs preconditioners by generating spanning trees in power supply networks for fast power grid analysis. The support-graph preconditioner is efficient for handling complex power grid structures (regular or irregular grids), and can facilitate very fast incremental analysis. Our experimental results on IBM power grid benchmarks show that compared with the best direct or iterative solvers, the proposed support-graph preconditioned iterative solver achieves up to $3.6X$ speedups for DC analysis, and up to $22X$ speedups for incremental analysis, while reducing the memory consumption by a factor of four.**

I. Introduction

In nowadays IC designs, power supply networks (PDNs) have become increasingly critical, since circuit performances such as delay and power consumption may strongly depend on the voltage supply levels. While existing semiconductor technology evolutions enable integrating much more and faster transistors on chip, the resistive on-chip interconnect has not scaled well accordingly and thus become a major performance limiting factor. However, it is not always trivial to design the most economic yet reliable power grids that can meet all design specifications (e.g. maximum voltage supply noise, power dissipations, etc.) and meanwhile avoid over-design issues, considering complex and time-consuming design iterations (that are mainly due to costly circuit simulations) in which incremental grid corrections are typically performed by carefully optimizing/inserting metal wires/vias. Therefore, fast power grid simulations and incremental analysis have become very important and increasingly popular research topics in the past decades [1]–[5].

A majority of power grid simulation algorithms falls into the following two categories: direct methods such as LU and Cholesky matrix factorizations, and iterative methods such as preconditioned conjugate gradient (PCG) methods [5], [6], and multigrid methods [4], [7]. Direct methods exactly solve the simulation problems but require much higher memory to store the matrix factors, while iterative methods usually have more favorable memory requirements but may suffer from poor convergence issues. For instance, incomplete Cholesky preconditioned CG (ICPCG) algorithm creates approximate Cholesky matrix factor by removing none-critical (much smaller) elements according to a pre-defined threshold value, but effectively trading off the memory consumption (proportional to the number of elements stored in the incomplete matrix factor) with the overall convergence rate remains challenging. For the power grids with good regularities and full grid structure known in advance, multigrid preconditioned CG solver [5] can converge very fast and reliably once well designed and optimized.

In this work, we propose a hierarchical support-graph preconditioner for fast power grid analysis and incremental analysis, which is much more efficient than existing direct and iterative solvers. By creating hierarchical spanning trees (support graphs) in the power supply network and ordering the nodes according to topology, existing direct matrix algorithms (such as Cholmod [8]) can be directly applied to generate high quality yet memory efficient preconditioners with guaranteed convergence [9] for iterative solvers. We show that the condition numbers associated with power grid simulations can be significantly reduced, which leads to a much faster convergence rate compared with traditional preconditioned algorithms. As a result, highly effective preconditioners can be created with little memory consumption specifically for power grid analysis. More importantly, this novel support-graph preconditioning technique is inherently suitable for incremental power grid analysis, since the support graph topology needs to be extracted only once from the original power grid structure, while subsequent matrix factorizations after power grid corrections can be more efficiently performed than re-factorizing the original denser conductance matrix. Furthermore, the proposed support-graph preconditioner can also be easily and effectively extended to RC networks. Additionally, we emphasize that the proposed preconditioned power grid simulation algorithms can be easily parallelized on nowadays energy-efficient multi/many-core computing platforms [10], making them more appealing to large-scale power grid simulations. It has been demonstrated that for all industrial test cases [11], the proposed hierarchical support-graph preconditioning method achieves up to $3.6X$ runtime improvement for DC analysis and $22X$ runtime improvement for incremental power grid analysis when compared against the state-of-the-art direct solver [8] and preconditioned iterative solvers. Additionally, around $4X$ reduction in peak memory usage is observed for most test cases.

The rest of this paper is organized as follows. In Section II, we provide the background of power grid modeling methods, and preconditioned power grid simulation algorithms. In Section III, we introduce the support-graph theory and its application in solving circuit simulation problems. Section IV presents the hierarchical support-graph preconditioner for large power grid analysis. Section V demonstrates extensive experimental results for a variety of industrial power grid benchmarks to validate the proposed approach, which is followed by the conclusion of this work in Section VI.

II. Background

A. Power Grid Modeling and Analysis

Power grid analysis typically falls into the following categories: DC or steady state analysis, and transient simulations. In the steady state DC analysis, power grid is modeled as a network of pure resistors with excitation sources such as current and voltage sources. It can be shown that transient analysis can be performed by first replacing the energy storage components such as capacitors or inductors with companion models that include resistors and current/voltage sources through backward Euler (BE) or Trapezoidal (TR) approximations, and subsequently solving the equivalent DC problem. Consequently, we mainly discuss DC solution methods throughout this paper, while transient analysis can be accomplished in very similar manner. The DC problem for such a resistive power delivery network with n nodes can be formulated using the following linear system of equations

978-1-4577-1400-9/11/$26.00 ©2011 IEEE

based on nodal analysis (NA) [1], [6]:

$$Ax = b, \tag{1}$$

where A is an $n \times n$ symmetric positive definite (SPD) conductance matrix representing all interconnected resistors, x is an $n \times 1$ vector including all unknown node voltages, and b is an $n \times 1$ vector modeling excitation sources such as current or voltage sources.

Direct matrix decomposition methods such as LU or Cholesky factorization [8] algorithms can be used in solving the above linear systems with high accuracy but poor memory and runtime efficiency due to large number of new fill-ins created during the factorization process especially for large-scale power grid analysis. On the other hand, iterative methods [4]–[6] exhibit much higher memory efficiency, which involve only the sparse matrix and a few vectors during the computation.

B. Preconditioned Iterative Solvers

Preconditioning is key technique for accelerating Krylov-subspace iterative solvers, such as conjugate gradient (CG) method and generalized minimal residual (GMRES) method. The basic idea is that instead of directly solving the given linear system in (1), some effort is made to reduce the condition number of the system:

$$\kappa(A) = ||A||_2||A^{-1}||_2 = \lambda_{max}(A)/\lambda_{min}(A). \tag{2}$$

In the preconditioned iterative solver, we try to find a matrix P, called preconditioner, such that we need to solve the following alternative linear system

$$P^{-1}Ax = P^{-1}b. \tag{3}$$

If the condition number of the new system is much smaller than the original one, the linear system solution can be found in a much faster way (less number of iterations). The selection of the preconditioner has dramatic impact on convergence. A popular preconditioner in the literature is Jacobi preconditioner, which is formed through taking the inverse of the diagonal elements of matrix A. It is widely used in practice because of simplicity of the preconditioner. However, such a preconditioner may not be very useful for power grid simulations, since ill-conditioned linear systems are commonly seen in power grid analysis problems that are associated with relatively large condition numbers.

Iterative methods such as conjugate gradient (CG) methods require high quality preconditioners to improve the convergence rate [1]. A good preconditioner P should be efficient to compute (linear system $Px = y$ should be easy to solve), and effective for significantly reducing the condition number ($\kappa\left(P^{-1}A\right)$ is small). The simplest diagonal preconditioner (Jacobi preconditioner) is efficient to compute but may not be effective for reducing the number of iterations or condition number, while incomplete Cholesky matrix factorization-based preconditioners may achieve much better convergence but involve much higher memory consumption and computational cost during the preconditioning process. A stochastic preconditioner has been shown in [12] to achieve good convergence rate, but the setup procedure may still be very costly.

III. Support-Graph Preconditioner

Support-graph preconditioning is to first construct a graph according to a given matrix A, and then compute the preconditioner P as a reduced matrix based on the support of the graph. The support is usually a specific subgraph extracted from the full graph. Take the maximum weighted spanning tree as an example of support as described in [13]–[15], and refer to Fig. 1. Suppose for a linear system, matrix A is shown in Fig. 1 (a). We can transform all the off-diagonal elements to a graph as shown in Fig. 1 (b). Then, each

[1]CG needs $O\left(\kappa^{1/2}(A)\right)$ iterations to solve $Ax = b$

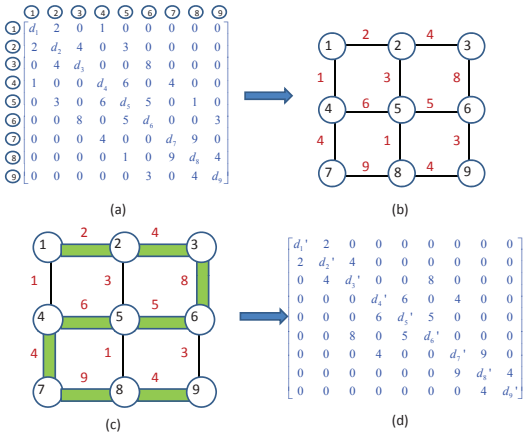

Fig. 1. Maximum spanning tree preconditioner.

off-diagonal element (wire conductance) in the matrix corresponds to a weighted edge in the mesh. Next, a maximum spanning tree is derived as illustrated in Fig. 1(c). Finally, we can construct the preconditioner matrix P according to the extracted spanning tree as shown in Fig. 1(d). It has been shown that by applying matrix P as the preconditioner, the linear system in (3) will have a bounded condition number. It has been proven that such preconditioning technique can be applied to any SPD diagonally dominant matrices.

Formally, support-graph preconditioning is to compute a preconditioner P such that the generalized eigenvalues and the condition number of the matrix pencil (A, P) are bounded [9]. If both A and P are SPD matrices, the convergence depends on the condition number $\kappa(A, P)$ computed as follows [9], [13]:

$$\kappa(A, P) = \frac{\lambda_{max}(A, P)}{\lambda_{min}(A, P)}, \tag{4}$$

where $\lambda(A, P)$ denotes the generalized eigenvalue. A stronger theoretical result on convergence can be derived as follows. Define the support of (A, P), denoted by $\sigma(A, P)$, as follows [9], [13]:

$$\sigma(A, P) = min\{\tau \in \mathbb{R} | x^T(\tau P - A)x \geq 0 \ for \ all \ x \in \mathbb{R}^n\}. \tag{5}$$

Subsequently, if one can split A and P into $A = A_1 + A_2 + ... + A_m$ and $P = P_1 + P_2 + ... + P_m$ such that all $\tau P_i - A_i$ are SPD matrices, one can show that the generalized eigenvalue of (A, P) is bounded by τ. This is proven as splitting lemma in [9], [13].

To more effectively trade off the the memory consumption and convergence rate of iterative algorithms, support-graph preconditioner can be naturally applied to accelerate large-scale power grid analysis by extracting the maximum spanning tree from the original power grid structure. Existing direct matrix solvers can be applied to generate the support-graph preconditioner based upon the spanning-tree theory. Since the support graph maintains a tree-structure, during the matrix factorization process, the number of new fill-ins can be very well controlled, especially when appropriate node ordering techniques, such as the reverse Cuthill McKee ordering (RCM) [16], are adopted.

As a result, compared with existing black-box incomplete matrix factorization-based preconditioners such as incomplete Cholesky preconditioner, the matrix factor associated with the support graph can serve as a more efficient preconditioner with guaranteed convergence [9] in power grid analysis. To achieve higher efficiency in large-scale power grid analysis, hierarchical support graphs can be created by using a divide-and-conquer strategy. We would like to emphasize that extracting such spanning trees from large graphs can be efficiently

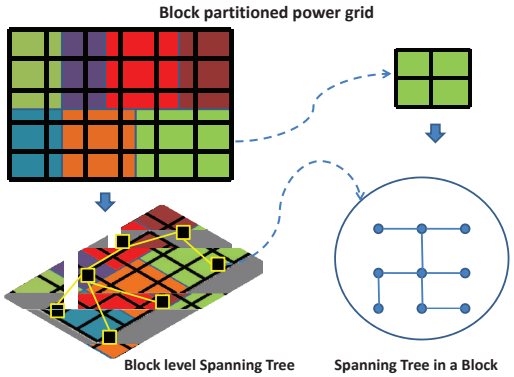

Fig. 2. Hierarchy spanning tree construction.

parallelized on recent multi/many-core parallel platforms [10]. Therefore, the overhead in building the support-graph preconditioner can be negligible when compared to the complete power grid analysis cost, especially when many incremental simulations are performed. Throughout this paper and our experiments, fast/parallel spanning tree construction techniques are not of our focus, and therefore have not been implemented.

IV. Hierarchical Support Graph in Power/Ground Networks

Many preconditioners have been designed for symmetric positive definite matrices in literature. However, only a few of existing preconditioning algorithms explore the problem specific knowledge of power grid simulations [5]. In this section, we describe the details for constructing the hierarchical support-graph preconditioner specifically for power grid analysis, and the hierarchical support-graph preconditioned CG algorithm (HSGPCG).

Since realistic power grids can include a large number of nodes, such that building a support graph for the complete power/ground network can be extremely time consuming. In this work, to gain high runtime efficiency, we first geometrically partition the original power/ground network into many blocks. Subsequently, for each block, an inner-block support graph is created, which will later form a block-level support graph. The final preconditioner is then created based on the resultant hierarchical support graph. This divide-and-conquer strategy can help control the computational cost in generating the support-graph preconditioners for large-scale power grid analysis problems.

A. Hierarchical Support Graph

In this work, instead of using the low-stretch spanning tree suggested in [17], we use a maximum spanning tree to approximate the low-stretch spanning tree. The algorithm works as follows. Arbitrarily pick a starting node, add it to the visited node list, choose the maximum weighted outgoing edge from the visited node list, and add the new node in. Repeat the above procedure until all nodes are visited. Our experiments show that such an approximation can produce a good preconditioner and can be computed efficiently.

At a high level, our geometric hierarchical support-graph preconditioner is constructed as follows. We partition the whole circuit mesh into blocks with equal size where at least one VDD pad is guaranteed for each block. Refer to Fig. 2 where mesh grid lines represent the power grid network and the colored region represents the partitioned blocks. The black filled squares are the abstract blocks and those abstract blocks are connected to form the *block-level graph*. Inside a block, a maximum spanning tree will be extracted from the mesh graph, called *inner-block graph*. Between adjacent blocks, there can be multiple connecting edges before partitioning. We merge them into a single virtual edge and its weight will be computed as the sum of weights of all connecting edges. In this way, we can compute the block-level graph (with virtual edges) and the maximum spanning tree will be extracted from it. Suppose that there is a virtual edge linking two blocks in the computed maximum spanning tree, the maximum weighted edge linking them in the original graph will be selected to replace the virtual edge. After this computation, the maximum spanning trees in all inner-block graphs and the maximum spanning tree in the block-level graph can be easily combined to form the hierarchical maximum spanning tree for the whole circuit mesh. Note that the preconditioner resulted from the above hierarchical maximum spanning tree only contains off-diagonal elements and the diagonal elements are updated accordingly such that the row sums of the preconditioning matrix are same as the original matrix. Compared to directly computing the maximum spanning tree on the original circuit mesh, our hierarchical computation is much more efficient since our algorithm is performed in a divide and conquer fashion.

B. Hybrid Node Ordering

A good node ordering can reduce the overall matrix bandwidth, the condition number of the matrix, and dramatically reduce memory cost during the matrix factorization. Thus, a hybrid node index reordering method is applied.

1) Partition-Based Node Ordering: Since node indices can be randomly assigned after loading the circuit netlist, it is not convenient to perform node traversal during the maximum spanning tree construction. The main idea behind block partition-based node reordering is that, the node indices in each partition are continuous. For instance, a power grid with N nodes is partitioned into two blocks A and B with n and m nodes, respectively, where $N = n + m$. The node indices in block A range from 0 to $n - 1$, while the node indices in block B range from n to $n + m - 1$. It can be observed that a majority of matrix elements are located more closer to the diagonal than before. Such that, the block node reordering method would reduce the number of new fill-ins created in the matrix factorization, as well as the memory cost.

2) RCM Node Reordering: Although partition-based node reordering can help improve the memory efficiency and condition number, new fill-ins can not be eliminated. To further improve the memory efficiency, we applied the RCM algorithm [16] to re-order the node indices of the support graph in each block. In the best case, during the matrix factorization, there will be no new fill-ins.

C. The HSGPCG Algorithm Flow

The algorithm details for constructing the proposed hierarchical support-graph preconditioner are as follows.

1) HSGPCG first partitions the complete power/ground network into many blocks based on the geometrical information of the grid. Within each block, the partition-based node index reordering is applied such that the node indices within corresponding sub-support-graph are continuous.
2) Next, the hierarchical support-graph technique is performed to generate the preconditioner matrix P. The maximum spanning tree is computed from each inner-block graph. Subsequently, RCM technique is applied to reorder the node indices of maximum spanning tree in each block. Subsequently, the conductance matrix A can be constructed .
3) In the block level maximum spanning tree construction, each block is considered as a single node and the block level graph can be constructed. Subsequently, a maximum spanning tree is extracted from the graph. After that, the preconditioner matrix P can be constructed for Cholesky matrix factorization [8]. Before applying PCG solver, the factorization of preconditioning matrix P needs to be performed once.

V. Experimental Results

In this section, extensive experiments have been conducted to evaluate our proposed hierarchical support-graph preconditioned CG (HSGPCG) solver for power grid simulation. Additionally, incremental power grid analysis is also performed for all test cases to show the high efficiency of HSGPCG.

A. Experiment Setup

We compare our proposed method with existing algorithms, like direct solver, CG, diagonally preconditioned CG (DPCG), and incomplete Cholesky preconditioned CG (ICPCG). External libraries are adopted for Cholesky factorization [8] and incomplete matrix factorization with zero fill-in [18]. A set of industrial power grid benchmarks has been tested [11]. All of above algorithms have been implemented in C++. All experiments are performed using a single CPU core of a computing platform running 64-bit Ubuntu 8.04 with 2.66GHz quad-core CPU and 6GB DRAM memory. All runtime results are measured in seconds. For all test cases, we assure that the maximum error is no greater than $1mV$ when using iterative solvers. The characteristics of power grid benchmarks has been concluded in Table I.

B. The Results of Power Grid Analysis

We would like to demonstrate the benefits of proposed HSGPCG solver by comparing it with the state-of-the-art direct solver [8] and existing PCG methods. Table I demonstrates the memory efficiency of proposed "HSGPCG" method w/ RCM node ordering as well as the direct method, where the memory cost for Cholesky [8] matrix factorizations of the original conductance matrix A and the support-graph matrix P are shown. By using the hybrid node ordering technique, compared with direct Cholmod solver, our proposed method can reach up to $40X$ improvement in the number of none-zero entries required during matrix factorization. Besides, our proposed method can save up to $4.8X$ peak memory consumption during the matrix factorization.

The simulation results for eight IBM power grid benchmarks using different methods are summarized in Table II. As observed, the classic "CG" solver needs thousands of iterations, making itself much slower than the high performance direct solver. By using diagonal preconditioner and incomplete Cholesky preconditioner [18], "DPCG" and "ICPCG" can dramatically reduce the number of iterations if compared to "CG". Although "ICPCG" needs less number of iterations than "DPCG", due to the costly preconditioning algorithm within each iteration, "ICPCG" can be sometimes slower than "DPCG".

The proposed hierarchical support-graph preditioned CG solver "HSGPCG" uses the same direct solver [8] to create preconditioner's matrix factor. Due to the hierarchical tree structure of the support graph, the preconditioner matrix P is much sparser and thus easier to factorize than the original conductance matrix A. As a result, the factorization time of P is much smaller than the direct Cholesky factorization time. As observed in Table II that the factorizations for the support-graph preconditioners are $3X$ to $30X$ faster than factorizing the original matrix A using the same direct solver. Since the preconditioner generated by support-graph method can accurately approximate the original power grid network and its dominant eigenvalues, "HSGPCG" usually requires much fewer iterations than other existing PCG solvers. We observe in our experiments that for smaller test cases, less than 50 iterations are required with "HSGPCG", while for other larger test cases less than 400 iterations are typically needed. Therefore, "HSGPCG" converges in $2X$-$25X$ less number of iterations than other PCG solvers.

Compared with the state-of-the-art direct solver based on Cholesky decomposition [8], the proposed "HSGPCG" algorithm can achieve up to $3.6X$ improvement in runtime. For smaller test case, the direct matrix factorization can fully utilize the on-chip cache memory thus provide very good results, for which our "HSGPCG" solver is $1.4X$ slower. For $ibmpg5$ and $ibmpg6$ test cases, the "HSGPCG" convergence becomes a bit worse since there are many isolated grids in the power/ground network that would result in broken graphs.

It should be noted that extracting spanning trees from large graphs can be efficiently parallelized on recent multi/many-core parallel platforms [10]. Therefore, the overhead in building the support-graph preconditioner can be negligible if compared to the complete power grid analysis or incremental analysis costs.

C. The Results of Incremental Power Grid Analysis

Finally, the results of incremental power grid analysis are illustrated in Table III. In the incremental analysis, 20% of total resistors are randomly modified with an average of 50% amplitude change of the conductance values. To quickly perform the simulations considering significant resistance changes, the conductance matrices of support-graph preconditioners are updated with the latest resistance values, while the factorizations are performed after each power grid modification. Then we run incremental analysis using the previous simulation solution as the initial guess. In Table III, we compare the incremental analysis results of our HSGPCG solvers with direct solver, where "Non-updated HSGPCG" denotes the incremental analysis without updating the preconditioner matrix factor, and "Updated HSGPCG" denotes the incremental analysis with updated one. Since we do not change the topology of maximum spanning tree during the incremental analysis, the runtime of updating the P matrix is negligible. The re-factorization time for "Updated HSGPCG" is the same as the one of "HSGPCG" in Table II. From the table, we observe that, both "Non-updated HSGPCG" and "Updated HSGPCG" are much faster than the direct solver in the incremental analysis: up to $20X$ speedups can be achieved. It is also observed that in the incremental analysis using the updated preconditioner, HSGPCG solver usually needs much fewer iterations and much less runtime compared to the "Non-updated HSGPCG"case. As shown, for $ibmpg7$, without updating the support-graph preconditioner, HSGPCG may have serious convergence issues, since the condition number of $P^{-1}A$ may become much greater than before after significant grid modifications.

When power grid topology changes are made locally, spanning trees for the corresponding blocks can be reconstructed and subsequently combined with other sub-support graphs to form the updated preconditioner, which can be also efficiently accomplished.

VI. Conclusions

We propose a support-graph preconditioning method for fast power grid analysis and incremental simulations. By constructing spanning trees in the power supply network, hierarchical support-graph preconditioner can be efficiently created for accelerating the Krylov-subspace iterative algorithms such as conjugate gradient method. Since support graph helps significantly reduce the condition numbers of the preconditioned system, much fewer iterations are required for reaching the desired accuracy level. Our experimental results on IBM power grid benchmarks show that compared with the best direct or iterative solvers, the proposed support-graph preconditioned iterative solver is very runtime and memory efficient, achieving up to $3.6X$ speedups for DC analysis, up to $22X$ speedups for incremental analysis, and around $4X$ memory reduction.

References

[1] H. Qian, S. R. Nassif, and S. S. Sapatnekar, "Power grid analysis using random walks," *IEEE Trans. on Computer-Aided Design*, vol. 24, no. 8, pp. 1204–1224, 2005.

[2] Y. Zhong and M. D. F. Wong, "Fast algorithms for IR drop analysis in large power grid," in *Proc. IEEE/ACM ICCAD*, 2005, pp. 351–357.

[3] Y. Fu, R. Panda, B. Reschke, S. Sundareswaran, and M. Zhao, "A novel technique for incremental analysis of on-chip power distribution networks," in *Proc. IEEE/ACM ICCAD*, 2007, pp. 817–823.

TABLE I

EXPERIMENTAL SETUP OF IBM BENCHMARKS AND MEMORY COMPARISON BETWEEN DIRECT SOLVER AND THE PROPOSED SUPPORT-GRAPH BASED METHODS. "N" DENOTES THE TOTAL NUMBER OF NODES IN THE POWER GRID, "I" FOR CURRENT SOURCES, "R" FOR RESISTORS, "S" FOR SHORTS, AND "V" FOR VOLTAGE SOURCES [11]. "N_A"("N_P") DENOTES THE NUMBER OF NONE-ZERO ELEMENTS IN THE MATRIX FACTOR OF THE ORIGINAL MATRIX A (PRECONDITIONER MATRIX P). "Mem_A " ("Mem_P") DENOTES THE PEAK MEMORY USAGE DURING THE FACTORIZATION OF MATRIX A (PRECONDITIONER MATRIX P), AND "IMP." DENOTES THE MEMORY CONSUMPTION REDUCTION.

CKT	n	i	r	s	v	N_A	N_P (IMP.)	Mem_A	Mem_P (IMP.)
ibmpg1	30,638	10,774	30,027	14,208	14,308	319,696	60,708(5.2X)	9Mb	4Mb (2.1X)
ibmpg2	127,238	37,926	208,325	1,298	330	4,415,700	253,778 (17.4X)	77Mb	17Mb (4.5X)
ibmpg3	851,584	201,054	1,401,572	461	955	58,216,200	1,700,933 (34.2X)	522Mb	117Mb (4.5X)
ibmpg4	953,583	276,976	1,560,645	11,682	962	77,317,700	1,905,036 (40.1X)	634Mb	131Mb (4.8X)
ibmpg5	1,079,310	540,800	1,076,848	606,587	539,087	38,787,800	2,157,795 (18.0X)	645Mb	148Mb (4.4X)
ibmpg6	1,670,494	761,484	1,649,002	836,107	836,239	51,537,700	3,339,879 (15.4X)	889Mb	229Mb (3.9X)
ibmpg7	1,461,036	357,930	2,352,355	461	955	103,422,000	2,919,660 (35.4X)	912Mb	201Mb (4.5X)
ibmpg8	1,461,039	357,930	1,422,830	929,722	930,216	101,399,000	2,919,663 (34.7X)	925Mb	201Mb (4.6X)

TABLE II

RUNTIME RESULTS OF EXISTING PCG ALGORITHMS AND HIERARCHICAL SUPPORT-GRAPH PRECONDITIONED CG METHOD. "CG" DENOTES THE CLASSIC CONJUGATE GRADIENT METHOD, "DPCG" FOR DIAGONALLY PRECONDITIONED CG, "ICPCG" FOR INCOMPLETE CHOLESKY PRECONDITIONED CG WITH ZERO FILL-IN, AND "HSGPCG" FOR HIERARCHICAL SUPPORT-GRAPH PRECONDITIONED CG. "NUM.ITER." DENOTES THE NUMBER OF ITERATIONS, "SOLVER" FOR THE RUNTIME OF DIRECT OR ITERATIVE SOLVERS, "FACTOR" FOR THE MATRIX FACTORIZATION TIME, AND "SPEEDUP" FOR THE RUNTIME IMPROVEMENT OF HSGPCG COMPARED TO CHOLMOD [8].

CKT	Direct Solver		CG		DPCG		ICPCG			HSGPCG			
	Factor (s)	Solver (s)	Num.Iter.	Solver (s)	Num.Iter.	Solver (s)	Factor (s)	Num.Iter.	Solver (s)	Factor (s)	Num.Iter.	Solver (s)	Speedup
ibmpg1	0.060	0.002	805	0.910	373	0.493	0.004	162	0.262	0.018	44	0.069	0.7X
ibmpg2	1.360	0.280	1,682	8.100	436	2.530	0.017	221	1.612	0.077	118	0.425	3.2X
ibmpg3	16.590	1.460	3,634	146.900	2,679	113.440	0.121	1,568	81.237	0.605	231	6.637	2.5X
ibmpg4	20.610	1.840	3,687	170.100	611	33.130	0.118	312	18.484	0.620	195	5.548	3.6X
ibmpg5	13.660	1.620	6,765	335.660	2,489	130.780	0.119	1231	80.204	0.713	376	11.913	1.2X
ibmpg6	20.670	2.540	7,534	649.920	2,808	248.920	0.220	1328	139.803	1.400	308	21.637	1.0X
ibmpg7	33.440	2.750	3,264	250.250	1,606	136.470	0.205	899	79.415	1.070	128	13.585	2.7X
ibmpg8	31.990	2.540	4,236	314.920	3,173	248.890	0.214	1809	165.493	1.078	129	12.622	2.5X

TABLE III

RUNTIME COMPARISON OF INCREMENTAL POWER GRID ANALYSIS BETWEEN DIRECT SOLVER AND HSGPCG-BASED METHODS. IN THE INCREMENTAL ANALYSIS, 20% RESISTOR VALUES OF EACH BENCHMARK HAVE BEEN CHANGED BY 50%. "NON-UPDATED HSGPCG" DENOTES THE HSGPCG RUNTIME USING THE ORIGINAL SUPPORT-GRAPH PRECONDITIONERS, AND "UPDATED HSGPCG" DENOTES THE HSGPCG METHOD USING UPDATED SUPPORT-GRAPH PRECONDITIONERS.

CKT	Direct Solver		Non-updated HSGPCG			Updated HSGPCG			
	Re-factor(s)	Re-solver(s)	NumIter	Re-solver(s)	Speedup	NumIter	Re-factor(s)	Re-solver(s)	Speedup
ibmpg1	0.060	0.002	12	0.019	3.2X	6	0.011	0.012	2.5X
ibmpg2	1.360	0.280	15	0.087	18.8X	9	0.041	0.067	15.2X
ibmpg3	16.590	1.460	41	2.375	7.6X	18	0.314	0.748	17.0X
ibmpg4	20.610	1.840	29	1.313	17.1X	18	0.317	0.699	22.1X
ibmpg5	12.660	1.620	85	3.661	3.9X	32	0.383	2.079	5.8X
ibmpg6	20.67	2.540	50	4.737	4.9X	24	0.620	2.475	7.5X
ibmpg7	33.440	2.750	674	72.380	0.5X	19	0.540	1.708	16.1X
ibmpg8	31.990	2.540	39	3.320	10.4X	15	0.539	1.379	18.0X

[4] Z. Feng and P. Li, "Multigrid on GPU: tackling power grid analysis on parallel SIMT platforms," in *Proc. IEEE/ACM ICCAD*, 2008, pp. 647–654.

[5] Z. Feng and Z. Zeng, "Parallel multigrid preconditioning on graphics processing units (GPUs) for robust power grid analysis," in *Proc. IEEE/ACM DAC*, 2010, pp. 661–666.

[6] T. H. Chen and C. C.-P Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods," in *Proc. IEEE/ACM DAC*, 2001, pp. 559–562.

[7] J. N. Kozhaya, S. R. Nassif, and F. N. Najm, "A multigrid-like technique for power grid analysis," *IEEE Trans. on Computer-Aided Design*, vol. 21, no. 10, pp. 1148–1160, 2002.

[8] T. Davis, *CHOLMOD: sparse supernodal Cholesky factorization and update/downdate*, [Online]. Available: http://www.cise.ufl.edu/research/sparse/cholmod/, 2008.

[9] M. Bern, J. R. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo, "Support-graph preconditioners," *SIAM J. Matrix Anal. Appl.*, vol. 27, pp. 930–951, 2006.

[10] V. Vineet, P. Harish, S. Patidar, and P. Narayanan, "Fast minimum spanning tree for large graphs on the GPU," in *Proc. HPG*, New York, NY, USA, 2009, HPG '09, pp. 167–171, ACM.

[11] S. R. Nassif, *IBM power grid benchmarks*, [Online]. Available: http://dropzone.tamu.edu/ pli/PGBench/, 2008.

[12] H. Qian and S. S. Sapatnekar, "A hybrid linear equation solver and its application in quadratic placement," in *Proc. IEEE/ACM ICCAD*, 2005, pp. 905–909.

[13] E. Boman and B. Hendrickson, "Support theory for preconditioning," *SIAM J. Matrix Anal. Appl.*, vol. 25, pp. 694–717, 2003.

[14] E. Boman, D. Chen, B. Hendrickson, and S. Toledo, "Maximum-weight-basis preconditioners," *Numerical Linear Algebra and Applications*, vol. 11, pp. 695–721, 2004.

[15] E. Boman, B. Hendrickson, and S. Vavasis, "Solving elliptic finite element systems in near-linear time with support preconditioners," *SIAM J. Numer. Anal.*, vol. 46, pp. 3264–3284, 2004.

[16] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proc. 1969 24th national conference*, New York, NY, USA, 1969, ACM '69, pp. 157–172, ACM.

[17] M. Elkin, Y. Emek, D. A. Spielman, and S. Teng, "Lower-stretch spanning trees," *SIAM J. Comput*, vol. 38, pp. 608–628, 2008.

[18] J. Dongarra, A. Lumsdaine, R. Pozo, and K. Remington, "A sparse matrix library in C++ for high-performance architectures," *[Online]. Available: http://math.nist.gov/sparselib++/*, 1996.