PARALLEL SIMULATION FOR VLSI POWER GRID

A Dissertation

by

LE ZHANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Vivek Sarin
Committee Members,	Duncan Walker
	Rabi Mahapatra
	Peng Li
Head of Department,	Dilma Da Silva

August 2015

Major Subject: Computer Engineering

ABSTRACT

Due to the increasing complexity of VLSI circuits, power grid simulation has become more and more time-consuming. Hence, there is a need for fast and accurate power grid simulator. In order to perform power grid simulation in a timely manner, parallel algorithms have been developed to accelerate the simulation. In this dissertation, we present parallel algorithms and software for power grid simulation on CPU-GPU platforms. The power grid is divided into disjoint partitions. The partitions are enlarged using Breath First Search (BFS) method. In the partition enlarging process, a portion of edges are ignored to make the matrix factorization light-weight. Solving the enlarged partitions using a direct solver serves as a preconditioner for the Preconditioned Conjugate Gradient (PCG) method that is used to solve the power grid. This work combines the advantages of direct solvers and iterative solvers to obtain an efficient hybrid parallel solver. Two-tier parallelism is harnessed using MPI for partitions and CUDA within each partition. The experiments conducted on supercomputing clusters demonstrate significant speed improvements over a state-of-the-art direct solver in both static and transient analysis.

DEDICATION

I dedicate this dissertation to my parents, my wife and my son.

ACKNOWLEDGEMENTS

First, I would like to thank my Ph.D advisor and committee chair Dr. Vivek Sarin. The work I have done during my Ph.D career would not be accomplished without his support and guidance. His insight in my research topic helped me solve a lot of problems. Every time when I encountered a tough problem, he always gave me useful suggestions at the first moment.

I am also grateful to my committee members, Dr. Duncan Walker, Dr. Rabi Mahapatra, and Dr. Peng Li, for their guidance in their courses and valuable suggestions for the dissertation. I appreciate their time to serve on my committee.

My gratitude also goes to the Department of Computer Science and Engineering at Texas A&M University for supporting me with teaching assistantship. I also want to extend my gratitude to Dr. Joseph Hurley, Dr. Andreas Klappenecker, Dr. Hyunyoung Lee and Dr. Teresa Leyk for their help when I worked as teaching assistant in their courses.

I thank my friends Yue Liu and Yuqing Zhang for their encouragement and friendship during my life at Texas A&M University.

Finally, I would like to thank my wife and my parents for their support and love.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

I.1 Background

In modern industry, VLSI circuits have become more and more complicated in structure. The VLSI chip design flow consists of several steps, such as architectural and logic design, circuit and layout design, and fabrication [1]. Each step needs verification and testing to guarantee the functionality of the chips. In order to ensure chips run functionally, simulation techniques are widely adopted. Circuit simulation was introduced to industry in 1950s [2]. Circuit simulation and verification, including power grid (a.k.a. power delivery network) simulation, play a critical role in VLSI design. The functionality of a circuit design can be verified by using computer simulation without fabricating real chips. Therefore, simulation provides an inexpensive and efficient way in the VLSI circuit design process.

Power grid simulation is part of circuit simulation in VLSI design. The power grid consists of metal layers that provide power to active devices in the VLSI circuit. In each metal layer, power networks are modeled as vertical and horizontal wires. Different layers are connected through vias. In the widely used flip-chip package [3], Controlled Collapse Chip Connections (C4) bumps provide connections between the external power supply and the power grid. In the power grid simulation, the voltage drop, a.k.a. IR drop, is estimated through simulation to check the functionality of a power grid. Even though the wires in the power grid are made of metal, a small resistance can lead to non-

ignorable IR drop. If the IR drop exceeds a threshold (typically 10% of V_{DD}), the chip could become defective. Therefore, detecting over-threshold IR drop is a key process in power grid design. Another factor that needs to be estimated is electromigration [4]. Electromigration is generated by the movement of electrons in metal interconnects. It can result in connection disruptions derived from heating and diffusive displacement. One way to simulate electromigration is to calculate branch currents and guarantee that the branch currents do not exceed a specific threshold. The branch currents can be obtained using the value of node voltages and the value of the resistance between two nodes.

At the present time, the size of the power grid is increasing quickly due to a rapid increase in the size and complexity of VLSI circuits. The number of nodes in a VLSI power grid can easily reach hundreds of millions. As a consequence, solving a VLSI power grid is extremely time-consuming. Furthermore, the industry is focused on low power design, which introduces additional challenges for power grid design [5]. Voltage can easily drop below safe thresholds in low power chips, resulting in a the chip that fails to operate functionally [6].

Power grid simulation can be categorized as static analysis and transient analysis. Static analysis is conducted by ignoring all the capacitors and inductors. The static power grid simulation is the basis of transient analysis. Transient analysis can be conducted by implementing static analysis at different time steps by using the companion model for capacitors and inductors.

In static analysis, power grid can be modeled as a sparse linear system. Consider a linear system

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

where $A \in \mathbb{R}^{N\times N}$, $x\ and\ b \in \mathbb{R}^{N}$, N is an integer representing the size of the system.

The matrix A is a sparse matrix, where there are a lot of zero entries in the matrix and a small portion of the entries are non-zeros. Sparse linear matrices are often stored in different formats [7]. Compressed Row Storage (CRS) and Compressed Column Storage (CCS) are widely used in linear system solvers since they are friendly to matrix and vector operations. Coordinate format (COO) is another sparse matrix storage format. It is easy for maintenance (e.g., delete, add, modify, etc). However, COO is not suitable for linear system solver operations. List of lists (LIL) is another sparse matrix storage format. The entries are often sorted by column index. This feature makes LIL faster for look up and good for incremental matrix construction.

Equation (1) can be solved using either a direct solver or an iterative solver. The most famous direct solver is Gaussian Elimination, which is also considered as LU solver. The matrix is factored as a lower triangle matrix L and an upper triangle matrix U. Then a backward or forward substitution is applied to solve the linear system. The complexity of LU factorization is $O(n^3)$, and the backward and forward substitutions cost $O(n)$ each, where n is the size of the system. With regard to sparse linear systems, the complexity of LU factorization can be reduced to $O(n^{1.5})$. From the complexity, the factorization dominates the solution process. Thus if the factorization can be parallelized, the direct solver would achieve much higher performance.

The advantage of direct solvers is that the solution is accurate and the cost is low. In current academia and industry, the state-of-the –art direct solver named CHOLMOD [8] is widely used. CHOLMOD is an excellent direct solver designed by Dr. Tim Davis. It utilizes graph theory to optimize the computation in Cholesky decomposition in direct solver. The disadvantage of direct solvers is huge memory usage and difficulty of parallelization. Direct solvers need to store the factors in memory which could cost $O(n^2)$ space for a sparse matrix unless special techniques are used. Techniques such as nested dissection ordering [9] can lower the storage requirement to $O(nlogn)$. Parallelization of direct solvers is limited since the factorization and substitution solve need to wait for results from previous row or column [10].

The sparse linear system can be solved using iterative solvers. The most well-known iterative solvers include Jacobi method, Gauss-Seidel method, Conjugate Gradient method (CG) and Generalized Minimum-Residual method (GMRES). Typically, an iterative solver needs $O(n)$ memory storage. The advantage of iterative solvers includes potential for parallelism and a low memory footprint. Iterative solvers are friendly to parallelization since the whole process only includes matrix vector operations. The vectors are reused so that the memory usage is low.

However, the solution accuracy is an issue for iterative solvers. Thus a reasonable tolerance of solution error is needed to guarantee the solution accuracy. Furthermore, iterative solvers are prone to suffer from slow convergence. Krylov subspace method is a technique to accelerate convergence of iterations. The Krylov method is based on Krylov space projection and different choice of Krylov subspace

leads to different iterative methods [11, 12]. Further acceleration can be achieved by using preconditioners. Preconditioners are often used to accelerate convergence of iterative solvers. A good preconditioner can make the iterative solver even fast than a direct solver. However, finding a good preconditioner is challenging. Also, different linear systems perform differently using different preconditioners. Thus, a particular preconditioner needs to be constructed for each linear system. Given a linear system (1), if the condition number $\kappa(A)$ is large, the system is ill-conditioned [13]. Ill-conditioned linear system could bring uncertainty to the solution. Preconditioning can reduce the condition number of a linear system, such that more reliable solution can be obtained.

Power grid simulation can be categorized as static analysis and transient analysis. Static analysis is to simulate the behavior of DC status. In static analysis, all the capacitors and inductors are ignored by opening the capacitors and shorting the inductors. The resulting power grid is a resistance network and can be modeled as a linear system as (1). Solving the power grid is to obtain the voltage value of each node. Then the branch current can be calculated using the voltages of the two end nodes. Therefore, the behavior of the power grid can be estimated from the voltage at each node. Transient analysis is to simulate the time-varying behavior of power grid. In transient analysis, the power grid is a RCL consisting of resistance, conductance and inductance. The noise in the power grid consists of IR-drop and $L \cdot di/dt$,. In order to simulate the transient behavior, static analysis is conducted at multiple time instances. The solution process at a specific moment is a process of DC analysis. One must note that, in transient analysis with constant time-step, the conductance matrix remains the

same during the transient analysis and only the right-hand-side vector changes by applying the solution of previous time step. Thus, direct solver always performs better than iterative solver in transient analysis since the matrix factor can be reused during the transient simulation.

Solving a real power grid is challenging due to the size and complexity of the power grid. The following issues need to be considered for a power grid solver:

- Memory usage:

  A real power grid can easily have hundreds of millions of nodes. The memory usage for storing the linear system and solving the linear system is quite huge. As we know, iterative solvers use less memory than direct solvers. However, the memory storage for an iterative solver can reach hundreds of Giga Bytes. This huge memory requirement makes the computation difficult to conduct since the modern computers are not equipped with large sized memory. Therefore, solving a real power grid often needs special computers with hundreds of Giga-byte memory installed. In order to solve the memory problem, distributed computing techniques can be used to distribute the burden onto multiple machines. Each machine does not need to have huge memory but the overall memory size is large enough for solving the power grid. Although distributed computing gives power grid solver a solution to meet the huge memory requirement, the management of distributed memory and optimization of data transfer between memories are very challenging.

- Time cost:

  As VLSI circuit size increases, the time to solve a power grid increases drastically. The low efficiency of power grid solver makes the power grid design process quite time-consuming. Every time a problem is found in the power grid design after simulation, the power grid needs to be modified to fix the design problem. Then another power grid simulation is conducted on the new design. If multiple such design problems are encountered in the design loop, the costly power grid simulation needs to be done several times until the design has no flaw. This design process suffers due to inefficient power grid solver. Therefore, acceleration of power grid simulation is desired in current IC industry.

- Solution Accuracy:

  Solution accuracy is required in power grid simulation. If the couputed solution is not accurate enough, the fabricated chip could malfunction. This solution inaccuracy could result in huge time and financial loss in reality. As we know, direct solvers always have high solution accuracy, whereas iterative solvers may suffer from low solution accuracy. Iterative solvers are widely used since they require less memory storage and they can beat direct solvers in speed with an effective preconditioner. The solution accuracy becomes an issue for iterative solvers, since iterative solvers often suffer from slow convergence derived from small error tolerance. If an effective and efficient preconditioner can be constructed to accelerate the convergence of iterative solvers, the power grid simulation would benefit from the acceleration.

- Scalability:

  Scalability is an important factor in power grid simulation since the size of power grids varies. A good power grid solver should be scalable and have consistent performance over different sized power grids. Parallel algorithms developed on distributed systems are ideal to meet scalability demands since the time cost and memory usage can be assigned to multiple machines to obtain good performance on runtime and memory storage.

I.2 Survey of power grid simulation

General circuit can be simulated using SPICE [14]. SPICE is not a perfect power grid solver since the characteristic of power grids is not fully addressed. Therefore, simulators developed for power grids are desired. Power grid simulation has been investigated using different techniques. The power grid problem for the static case can be modeled as an s.p.d (symmetric positive definite) linear system. In [15], a Preconditioned Conjugate Gradient (PCG) method with sub-circuit based preconditioner is presented. Good performance is reported when using a mutable preconditioner at each iteration. Multigrid methods are widely-used in power grid analysis. A multigrid based grid reduction mechanism results in less work-load [16]. More recently, an algebraic multigrid approach was presented in [17] that achieves good performance in both runtime and memory usage. A preconditioned Krylov-subspace iterative method for solving large-scale power grid problems was proposed in [18], where the authors implemented a PCG method to accelerate the rate of convergence. The random walk

approach introduced in [19, 20] helps in incremental analysis where the conductance matrix is changed to simulate the change of resistance network (length and width change of the wires). [21] presents a support graph based iterative solver. The effective preconditioner reduces the number of iterations to within an acceptable range. In [22], a relaxation iterative method is discussed. This work was presented as node-by-node traversals and row-by-row traversals and achieves good performance.

Transient analysis is an important topic in IC circuit simulation. Different time step strategies affect the performance of transient solvers. [23] presents a power grid solver for transient analysis. This work adopts a grid reduction strategy to obtain much better results compared to SPICE. In [24], a telescopic projective numerical integration method is presented. An adaptively controlled integration method is discussed in [25]. The time step is adjusted during the transient analysis and the implementation achieves automatic load balancing. The parallel implementation on multi-core machines shows encouraging results. Various transient analysis methods for power grids are presented in [26-28]. [28] presents a transient power grid solver that applies symmetric formulation and uses fast Cholesky factorizations. Model order reduction (MOR) technique is utilized in [27] to reduce the data dimensionality. In this work, a direct solver and multimode moment matching techniques are used to achieve good performance. [26] presents a transient solver based on waveform relaxation techniques. A combination of partitioning and convergence acceleration is used to achieve good performance. The method is highly parallelizable and scalable on power grids with different sizes. In [29], a transmission-line-modeling-alternating-direction method is proposed. This work first

models the power grid as a structure of transmission line meshes. Then alternating-direction-implicit method is adopted to conduct transient analysis with high stability. [30] introduces an efficient transient power grid solver. The authors developed an efficient algorithm to reduce the complexity of the power grid using regularities. An efficient solver for transient analysis of the power grid is discussed in [31]. A smart mapping approach is developed in this work to obtain accurate solutions. The performance is improved by utilizing a memorized supernode technique. [32] introduces a transient analysis method that uses a hierarchical relaxed approach. The relaxation technique help obtain significant speedups over PCG and SPICE3.

In order to solve realistic problems in a reasonable amount of time, it is necessary to develop parallel algorithms for power grid analysis. Parallel algorithms for power grid simulation need to be implemented on parallel computers to obtain the solution faster. The current widely used parallel computing technoogies include Message Passing Interface (MPI), OpenMP and GPU. MPI is a message passing library for parallel computing. The MPI standard was first introduced in 1994. MPI provides an interface for users to create multiple jobs as separate processes and run the jobs concurrently. Different processes communicate with each other using messages [33]. OpenMP is a shared memory based parallel computing interface. OpenMP utilizes threads to run multiple jobs concurrently [34]. As the modern computing clusters and multi-core techniques develop, MPI and OpenMP help gain better performance while retaining portability. In contrast, the GPU is a hardware platform developed for processing 3D graphics and visual effect [35]. Currently, researchers found GPUs have the ability of

accelerating more general computation, especially matrix computations [36], based on its many-core characteristic. Consequently, General-Purpose Computing on Graphics Processing units (GPGPU) was introduced in 2003 SIGGRAPH conference. CPU-GPU hybrid parallelization is a way to gain better performance. [37] presents a CPU-GPU hybrid approach applied in multifrontal method. Power grid simulation can benefit from those parallel computing techniques. GPGPU technology has been used in Electronic Design Automation (EDA) for years. In [38], the authors successfully accelerated two costly computations, Sparse-Matrix Vector Product (SMVP) and graph traversal in EDA using GPU. [39] introduced GPU a programming scheme for EDA using OpenCL.

In parallel power grid analysis, the biggest challenge is partitioning the grid into sub-grids such that the solution can be obtained quickly by solving sub-problems concurrently. A parallel direct solver was presented in [40], where a parallel matrix inversion algorithm was shown to achieve good time and memory performance. Partitioning based parallel solvers are presented in [41-43] and [44]. Domain decomposition method is adopted in [41] to solve power grid. However, this method suffers from costly processing of the overlapping area. [43] presents a non-overlapping domain decomposition approach. In this work, too much time is spent on forming the Schur complements. Locality driven parallel analysis was introduced in [42], where the authors divide the grid into coupled sections and use locality property to compensate the boundary effect and decouple the sections. This method could exhibit low performance if the C4 bumps are distributed sparsely throughout the power grid. The reason is that sparse distribution of C4 bumps can lead to huge window size, in which solving the

windows is expensive. [45] presents a parallel forward and back substitution method. This method computes independent variables in parallel and develops a node ordering to eliminate the data dependency. Random walk has been reported to be applied in solving power grids [46-48]. Through statistical estimation, the node voltages can be obtained in a certain number of walks. However, if the V_{DD}destination is too far from the 'home', this method can suffer from large number of walks. In [26], the authors propose a waveform-relaxation based transient power grid simulator. The performance of the method may degrade when the initial guess is not reasonable. Macro-modeling method is presented in [49, 50]. In this work, a divide and conquer strategy is adopted over geometry partitioned power grids. The power grid is divided into small sub-partitions, and ports and interfaces are constructed by applying matrix substitution on different sub-partitions. The drawback of this method is that it suffers from dense matrix operations derived from ports. Another pattern based iterative solver is presented in [51]. In this work, pattern structure help to achieve performance improvement on both time and memory. GPU is adopted in Poisson optimized solver in [52]. The GPU implementation of Poisson solver benefits from multi-threaded GPU acceleration. Although the approach performs well, lack of parallelization make these method not ideal to large scale power grids. A transform-based iterative solver using GPUs is proposed in [53, 54]. The solver performs well for 3-D multiple-layer power grids. A GPU-based multigrid PCG solver is presented in [55]. The authors adopted a multigrid preconditioned CG method and successfully accelerate the solver on GPU platform.

Power grid simulation is a step in power grid design [56]. The behavior of a power grid needs to be simulated multiple times to obtain a design for fabrication. The power grid design is a discussed in the following literatures. A parallel design method of power grid is presented in [57]. In this work, a macro-modeling strategy is adopted to implement partition based optimization. [58] discusses the use of linear programming to improve power grid wire size optimization.

I.3 Proposed work

In this work, we adopt a divide-and-conquer strategy and propose an enlarged partition based parallel iterative solver for large-scale flip-chip power grid simulation. This approach divides the power grid into multiple partitions. In order to obtain an approximate solution for each partition, an enlarged partition is generated to enclose the original partition. Two partition enlarging methods are discussed in this dissertation. One is a naïve approach, in which a large area is specified to include all the nodes and edges as extension. The idea of this approach is simple. However, this naïve approach could have the following drawbacks. First, isolated nodes may be generated since the boundary of the new enlarged partition does not rely on the connection between two nodes. Isolated node may result in singular linear system that cannot be solved. Second, the naïve approach includes all the nodes and edges into the enlarged partition which may not be useful to obtain more accurate solutions. This makes the solving process of enlarged partition over-burdened and downgrades the parallel performance. Third, in a real chip, the whole power grid could consist of disconnected sub-grids. The naïve

approach could bring nodes and edges that do not belong to the partition. In this case, the solution accuracy is lowered and computation on unnecessary nodes affects the parallel performance. In order to overcome these drawbacks, a Breadth First Search (BFS) based partition enlarging algorithm is proposed. The enlarged partition first includes all the edges and nodes in the original partition. Then starting from the boundary nodes, a BFS method is applied to include more nodes and edges. When a new node is encountered, the edge with the largest conductance value is retained. All the other edges from this new node are ignored. This strategy trims most of the edges that have low influence on the solution and only retains the edges that are more useful to the solution. In order to control the solution accuracy, all the edges are retained up to a certain level. This step is necessary in our approach since the solution error is large if no such edges are retained outside the original partition. By solving the enlarged partitions one by one, an approximate solution for each partition is obtained. An approximation to the global solution is constructed from partition solutions. Since the global error is large, one can use PCG to reduce the error below the specified threshold. The enlarged-partition solver acts as a preconditioner for PCG.

The solver can be parallelized using MPI on a multi-processor supercomputer. Each enlarged partition is factored and solved by an MPI process. Besides, other computations within the PCG algorithm are assigned to different MPI processes to obtain good runtime performance. Between MPI processes on different cores/machines, data transfer rate play an important role in the time performance. MPI_Allgather and MPI_Reduction are two routine used in data transmission. The parallelization of EPPCG

can be separated into two parts: factorization and matrix-vector operations. Factorization is parallelized by assigning each partition matrix to a computing node/core. Data transmission between nodes/cores is not necessary since the factorization is conducted on each enlarged partition itself. Other computations include matrix-vector multiplication, vector-vector inner produce, vector summation, etc. Matrix vector multiplication is computed on the enlarged partition since at least one more layer outside the original partitions is needed to obtain the correct results. Other vector related computations are implemented on partitions since no additional data is needed outside the original partition. The GPU is adopted to accelerate the factorization within the direct solver CHOLMOD. A two-tier parallelization is used to obtain the best performance.

The experiments are conducted on IBM P/G benchmarks and artificial large-scale benchmarks. Both static and transient analysis is implemented. In static analysis, all the capacitors and inductors are ignored and the power grid is simplified as a resistant network. Only a sparse linear system is solved in static analysis. Our experimental results show that the enlarged partition based preconditioner is effective. The fully parallelizable preconditioner is computed and stored on different machines, such that the time and space cost are distributed onto different machines. The parallelization of our implementation is efficient. The best results obtained on the largest benchmark demonstrate a speedup improvement of 130X over the state-of-the-art direct solver CHOLMOD when using 16 cores. The effective preconditioner and efficient parallelization makes our approach much faster than CHOLMOD. In transient analysis,

the same enlarged partition based preconditioner is used. Static analysis is conducted multiple times at different time instances. In general, a direct solver is ideal for transient analysis, since the most time-consuming process of factorization only needs to be conducted once in fixed time-step scenario. In constant time-step transient analysis, our approach is slower than direct solver due to the costly PCG solving process. In variable time-step scenario, our approach performs better than direct solver since the factorization needs to be computed at each time step.

The main contributions of this work can be summarized as follows:

- The proposed enlarged-partition based preconditioner is effective. It reduces the number of iterations significantly compared to classical preconditoners. As a consequence, the proposed EPPCG method converges in fewer iterations.
- Our proposed enlarged-partition based preconditioner is fully parallelized on enlarged partitions. The experimental results show the parallelization of preconditioner achieves super-linear speedups in some cases.
- In order to reduce the size of the linear system derived from the enlarged partitions, Breadth-First Search method is adopted when enlarging the partition. This strategy makes the enlarged partitions lightweight and eliminates the potential for isolated nodes.
- A two-tier parallelism is adopted on CPU-GPU hybrid platforms. MPI implementation on CPU is much faster than CHOLMOD. Using GPU acceleration, matrix factorization is further accelerated over dense matrix operations in CHOLMOD. Although GPU results in several times speedup, the

combination with efficient MPI implementation makes the speedup significantly improved. The efficient implementation of MPI-GPU help the proposed approach run 130X faster over the state-of-the-art direct solver on large benchmarks.

- Previous partition based power grid solvers suffer from costly processing of boundary areas. In [42] and [43], the conductance matrix associated with the boundary area can be either a dense matrix of the graph network or of huge size. Our proposed partition enlarging method extends the partition through edges and ignores edges that have relatively less information. The method makes the generated conductance lightweight for computations.
- Transient analysis is conducted using EPPCG method. Direct solvers are hard to beat in runtime due to the reuse of factorization. Our approach combines the advantages of direct solver and iterative solver. The factorization in EPPCG can also be reused in transient analysis. In fixed time-step scenario, we observed comparable time with CHOLMOD. In variable time-step scenario, our approach performs better than CHOLMOD.

CHAPTER II

ENLARGED-PARTITION BASED PARALLEL SOLVER FOR POWER GRID*

II.1 Modeling and mathematical background

II.1.1 Modeling the power grid

Power grid plays an important role by providing power to active devices in electronic chips. IR drop would be significant if only single or multiple wires are used for power delivery. Power grids are designed to minimize the IR drop and guarantee the functionality of chips.

Power grid simulation is part of VLSI circuit simulation, and consists of simulating the behavior of the power network. Power grid simulation is a necessary step before fabricating actual chips to examine faults in the design. The low-cost and easy-to-change properties make the power grid simulation a necessary step in power grid design.

A power grid can be modeled as an RCL network consisting of metal wires. In our work, we conduct both static analysis and transient analysis. Transient analysis can be implemented as static analysis on multiple time points. Therefore, the proposed algorithm for solving power grids is described over static analysis. The strategy for transient analysis will be discussed based on static analysis approach. The power grid is modeled as a resistance network by ignoring the capacitors and inductors. The structure

---

* Part of this chapter is reprinted with permission from "An enlarged-partition based preconditioned iterative solver for parallel power grid simulation" by L. Zhang and V. Sarin, in *Quality Electronic Design, 15th International Symposium on*, pp. 715-722, 2014, Copyright[2014] IEEE. [44]

of the power grid can be irregular, in which the nodes in the power grid are not well-connected. The static power grid is a resistance network, which can be described as a weighted graph where nodes are connected to each other through edges with weight (resistance value). In the following discussion, we call the resistance between two nodes 'edge'. Typically, a power grid contains multiple layers including vertical and horizontal layers, and consists of V_{DD} and GND networks. Figure 1 shows a V_{DD} power network. The red circles represent C4 bump connections, where the external power supply (V_{DD}) is connected to the power grid. Active devices that pull current from the power grid are modeled as current drains. Different layers are connected through vias. The power grid can be considered as a 3D structure that is more difficult to solve than a 2D structure

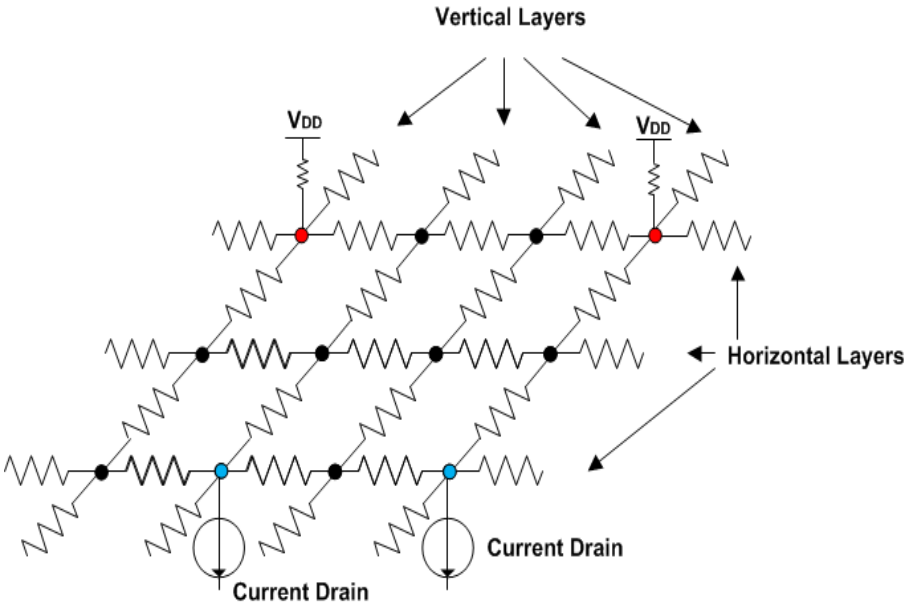

Figure 1. Modeling of a power grid

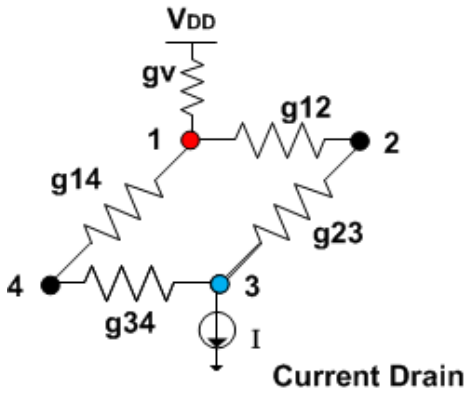

Figure 2. A simple example of power grid

[59]. GND network is similar to V_{DD} network except the C4 bumps are connected to GND and the direction of current drains is opposite.

In power grid simulation, the voltage of each node is unknown. Solving a power grid is a process to obtain the values of those node voltages. Moreover, branch currents can be calculated using the obtained node voltages. Nowadays, power grid simulation is very costly in both time and space due to the huge size of VLSI circuits. Therefore, fast power grid solver is in great need.

Based on Kirchhoff's current law, the current flowing out of a node equals that flowing into a node. Figure 2 shows a simple example of power grid. Taking node 3 as example, we have

$$g_{34}(V_4 - V_3) + g_{23}(V_2 - V_3) = I \tag{2}$$

A more general equation for an arbitrary node x is

$$\sum_i g_{ix}(V_i - V_x) = \sum_m I_{xm} \tag{3}$$

where node i denotes a neighbor of node x, g_{ix} denotes the conductance between node i and node x, and I_{xm} denotes one of the current drains on node x.

In matrix form, based on Modified Nodal Analysis (MNA) [60], the overall power grid system can be modeled as a linear system.

$$GV = I \tag{4}$$

Assume there are N nodes and M C4 bumps in the power grid, G is an $(N + M) \times (N + M)$ matrix which represents a conductance matrix with information about V_{DD}, V is an $(N + M) \times 1$ vector which represents the unknown node voltages, and I is an $(N + M) \times 1$ vector which represents the current load of active devices and V_{DD}values. Now that V_{DD} is connected between GND and the power grid, the linear system can be reduced by eliminating M rows/columns as illustrated below.

Since V_{DD} is a known constant, the size of the matrix can be reduced by eliminating columns and rows that correspond to V_{DD} nodes. The revised linear system can be described as

$$\begin{pmatrix} g_{12} + g_{14} + g_v & -g_{12} & 0 & -g_{14} \\ -g_{12} & g_{12} + g_{23} & -g_{23} & 0 \\ 0 & -g_{23} & g_{23} + g_{34} & -g_{34} \\ -g_{14} & 0 & -g_{34} & g_{14} + g_{34} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} V_{DD} g_v \\ 0 \\ -I \\ 0 \end{pmatrix} \tag{5}$$

Based on the discussion above, the power grid system is modeled as an improved linear system

$$G'V' = I' \tag{6}$$

Assume there are N nodes in the power grid, Here, G' is an $N \times N$ matrix which represents the conductance matrix, V' is an $N \times 1$ vector which represented the unknown

node voltages, I' is an $N \times 1$ vector which stands for the current load of active devices and V_{DD}/g_v for C4 bumps. G' is a symmetric positive-definite system.

The sparse linear system (6) can be solved using many methods consisting of direct solvers and iterative solvers. The direct solver usually first factorizes the matrix of the linear system, then backward/forward substitution is adopted to obtain the solutions. When solving a sparse linear system, the factorization costs the most of the runtime compared to backward/forward substitution. The benefit of direct solver is that direct solver can solve the linear system more accurately than iterative solver. The most widely used direct solver is LU factorization and backward substitution. Usually, factorization such as LU factorization and Cholesky factorization are used to store the factors for reuse. Iterative methods, including GMRES and Conjugate Gradient methods, solve the linear system in an iterative way to reduce the error step by step. Generally, direct solver is faster but requires more space storage. Iterative solvers use less memory but depend on a good preconditioner to achieve fast convergence. Our algorithm described in next section is a combination of direct solver and iterative solver that combines the advantage of both solvers.

II.1.2 Preconditioned conjugate gradient (PCG) method

Conjugate Gradient (CG) method is a widely used iterative method for solving s.p.d. linear system $\boldsymbol{Ax} = \boldsymbol{b}$ [61]. An advantage of the CG method is that it requires much less memory compared to direct methods. Although convergence of CG is guaranteed for

s.p.d systems, it may still need large number of iterations to converge. Most of the time, in order to reduce the number of iterations, preconditioning method is used.

Preconditioning is an approach to reduce the computational time when solving a linear system $\boldsymbol{Ax} = \boldsymbol{b}$. A preconditioner, say $\boldsymbol{M}$, is a matrix such that the condition number of $\boldsymbol{M^{-1}A}$ is small. Typically, the preconditioner $\boldsymbol{M}$ is applied to the linear system equation as $\boldsymbol{M^{-1}Ax} = \boldsymbol{M^{-1}b}$. If the preconditioner $\boldsymbol{M}$ is close enough to $\boldsymbol{A}$ (i. e., $\boldsymbol{M^{-1}A} \approx \boldsymbol{I}$), the linear system will become much easier to solve. Each iteration of PCG requires a preconditioning step. The preconditioning step involves computing $\boldsymbol{Mz} = \boldsymbol{r}$.

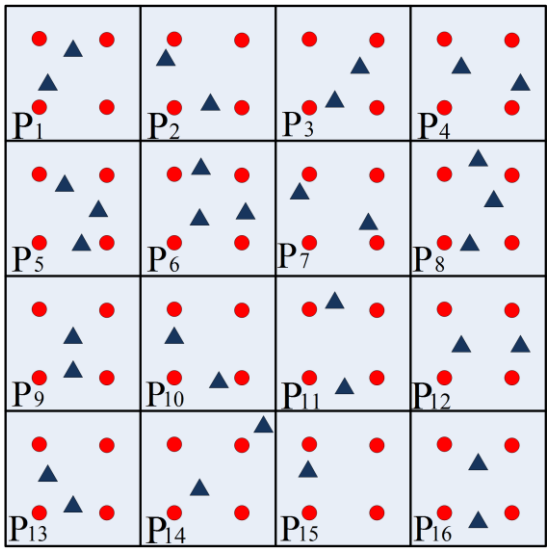

Figure 3. Partitioning of a power grid into $\boldsymbol{4 \times 4}$ subgrids.
Red circles represent C4 bumps; blue triangles represent current drains

A good preconditioner is key to accelerating the rate of convergence of the PCG method. If the preconditioner is easy to set up, but the condition number of $\boldsymbol{M^{-1}A}$ is large, the number of iteration will not drop and the time required to solve the system can be high. On the other hand, if the condition number of $\boldsymbol{M^{-1}A}$ is small, but the cost of computing and applying the preconditioner is large, solving the system will still be costly. Setting up a preconditioner often requires a lot of computation time. If the process of setting up a preconditioner can be parallelized, the total run time will be significantly reduced.

II.2 Parallel iterative solver based on enlarged partitions

Parallelization of power grid simulation is challenging, because it is hard to separate the power grid into decoupled sub-grids. In this section, an enlarged-partition based preconditioned power grid solver is proposed.

II.2.1Power grid partitioning

In order to solve the power grid in parallel, the basic idea is to separate the power grid into multiple disjoint partitions. Figure 3 shows a power grid divided into 3×4 partitions. Our partitioning strategy is to divide the power grid into equal-sized partitions in geometry. The edges that are across the partition boundaries are neglected and all the nodes are retained. Approximate solutions within each partition can be obtained by solving those sub-grids/partitions. However, the error of the approximate solution is far beyond our tolerance due to the inaccurate solution on the boundaries. Inspired by the

'spatial locality' property of power grid [62], the partitions are enlarged to achieve more accurate solutions.

Spatial locality demonstrates that C4 bumps modeled in a power grid have a limited spatial range of influence on the node voltages. In other words, if a node is close to a C4 bump, the IR drop from the C4 bump at that node is subtle; if a node is far from a C4 bump, the IR drop from the C4 bump at that node is significant. If we solve a larger area which enclose the original partition and retain the solutions within the partition, we can obtain more accurate solutions within the partition including the boundary. Based on the discussion above, we enlarge the partitions to include more nodes and edges such that the solution errors can be reduced.

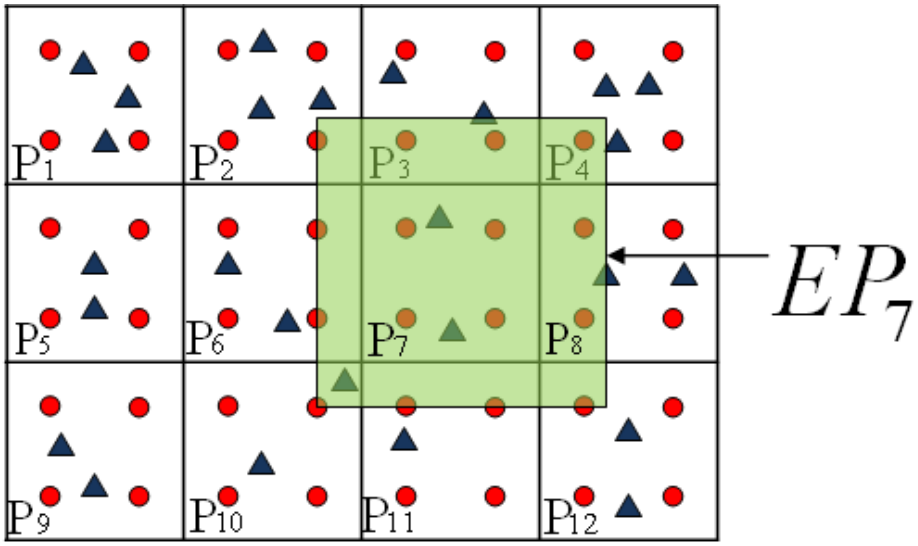

Figure 4. Constructing the enlarged partition

II.2.2 Naive partition-enlarging approach

Since the power grid is separated into partitions geometrically, a straightforward way to enlarge the partition is to move the border of the partition outwards to a certain distance (Figure 4). After enlarging the partition, the nodes and edges in the original partition are retained and more nodes and edges are included. Note that the edges across the new boundary are omitted.

Suppose the power grid is divided into P partitions. For each partition, we have a linear system

$$G_{ep}^{i} v_{ep}^{i} = I_{ep}^{i} , \tag{7}$$

where G_{ep}^{i} is the conductance matrix for enlarged partition i; v_{ep}^{i} is the node voltage vector; and I_{ep}^{i} is the vector representing current drains. After solving (7), we obtain v_{ep}^{i} which denotes the solution of an enlarged partitions. In v_{ep}^{i}, only the voltage solutions at the nodes that belongs to the original partition are retained and other voltage solutions are ignored. Therefore, solving all the enlarged partitions provides voltage solutions on each original partition. The voltage solution derived from the enlarged partition is more accurate compared to the linear system from the original partition because the influence of external C4 bumps near the boundary of the partition is captured.

The straightforward enlarging method is effective and easy to implement. However, it may introduce difficulties for the solution process. It has the following drawbacks. First, the irregular power grid can consist of disjoint sub-grids. The straightforward partitioning method could include disconnected nodes and edges from other sub-grids. Second, in a real power grid, a node is not always connected to all its

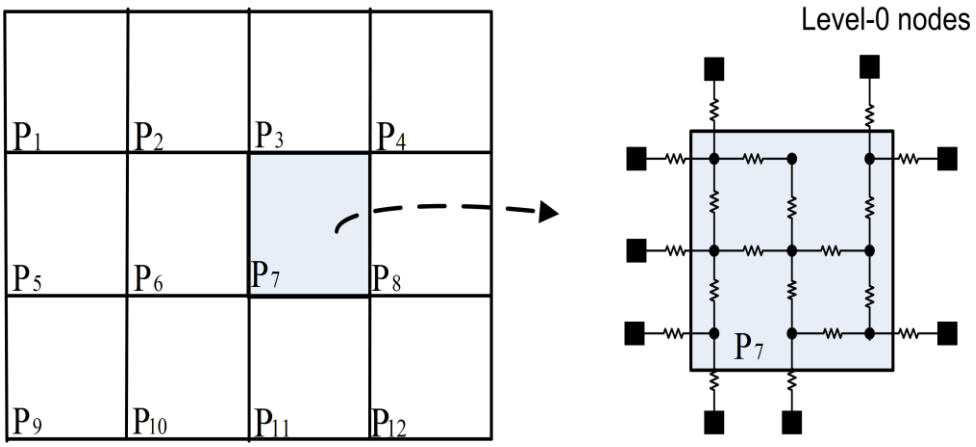

Figure 5. Including inner nodes and edges

neighbors. If such a node happens to lie next to the boundary and all its edges are across the boundary, this node will be an isolated node that is not connected to any other nodes. Isolated nodes cause the power grid matrix to become singular. In order to overcome these drawbacks and make the solving process more efficient, we propose an improved partitioning method based on Breadth-First Search (BFS) algorithm.

II.2.3 Improved partition-enlarging approach using breadth-first search

The approach described in II.2.2 is too expensive, since unrelated nodes and edges are also included in the enlarged partition. If we consider the area outside the partition as layers, we can extend the partition up to a certain layer and we can select edges to be included in the enlarged partitions. In this way, the number of nodes and number of

edges are fewer than those of the naive approach. As a consequence, the size of the generated linear system matrix and the number of non-zeros will be reduced. This size reduction helps improve the efficiency of the solver.

Since the partition is enlarged layer by layer, we adopt a BFS method to extend the original partition. The resulting enlarged partition is a tree-like structure outside the original partition. Note that we do not include all the edges in the BFS tree, but select the edge with the largest conductance that connects to a node.

To enlarge a partition using BFS our approach consists of the following steps:

1. Retain all the nodes and edges in partitions. All the boundary edges across the partition borders as well as the nodes connected by the edges are retained. The nodes that lie outside the partitions and are connected by boundary edges are called level-0 nodes (Figure 5).

2. Starting with those level-0 nodes, apply breadth first search algorithm to extend the partitions. The level of an extended node is the shortest distance from level 0. The level of an unvisited node is set to -1. If a node is encountered during BFS, the largest edge (defined as the largest conductance) from its parent in the preceding level is retained in the BFS tree (Figure 6). The enlarged partition size (EP-Size) is defined as the number of levels extended out of the original partitions.

3. In order to have better control of the solution accuracy, additional edges are added to the BFS tree. The number of levels for which all edges will be included is defined as retained-level size (RL-Size). For example, if RL-Size is k, then all edges incident on nodes at levels 0 through k are retained. RL-Size is an adjustable parameter

to make the partition solution more accurate. Experiments suggest that RL-Size is a necessary parameter to control the solution accuracy and ensure convergence to the solution.

4. Return resulting enlarged partition.

The overall algorithm for enlarging a partition is described in Algorithm 1.

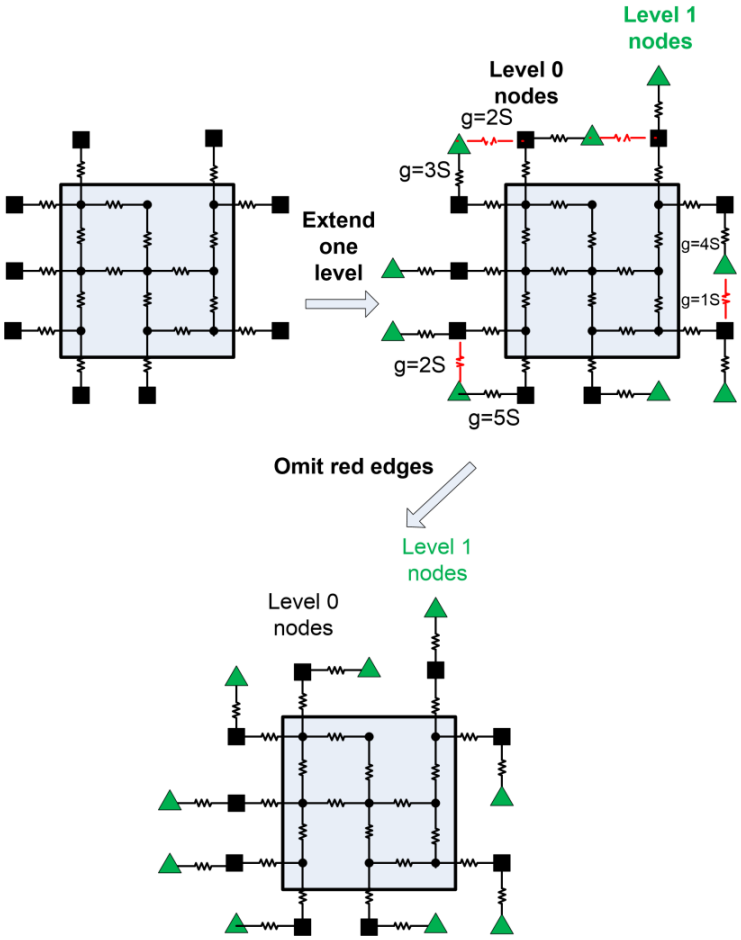

Figure 6. Extending a partition

ALGORITHM 1. Enlarging a Partition

Input: An partition with node and edge information; boundary information of the partition; EP-size; RL-size

Output: An enlarged partition with node and edge information.

1: Mark all the internal nodes and edges as 'retained', set the level of those nodes = 0.

2: Mark all the boundary edges and the nodes connected by boundary edges as 'retained'.

3: Mark all the nodes in Step 1 and 2 as 'visited'.

4: Create a queue-BFSQueue and push the nodes connected by boundary edges into BFSQueue, and set the level of those nodes = 1;

5: ***while****(BFSQueue not empty)*

6: CurrentNode=BFSQueue.pop();

7: ***if****(node.level>EP_size) then*

8: return;

9: ***end if***

10: ***for****(each neighbor node-NeiNode of CurrentNode)*

11: ***if****(NeiNode not visited)*

12: BFSQueue.push(NeiNode)

13: mark NeiNode as 'visited' and set NeiNode's level to CurrentNode.level+1

14: ***if****(NeiNode.level<=RL_size) then*

15: retain all the edges connecting NeiNode and 'visited' nodes

16: ***end if***

17: *check NeiNode's edges connecting to nodes one level lower, retain the edges that holds the largest conductance value and ignore the other edges.*

18: ***end if***

19: ***end for***

20: ***end while***

One must note that each enlarged partition must have at least one C4 bump. Otherwise, based on the discussion in II.1, the matrix of the generated linear system will be singular. A singular matrix makes the linear system unsolvable. Thus, in our approach, if there are no C4 bumps in the resulting enlarged partition, the enlarged partition is further extended to enclose at least one C4 bump.

Compared to the naive enlarging strategy, the BFS-based partition enlarging approach overcomes a number of drawbacks and reduces the size of the enlarged partition.

1. In a real chip, the power grid can consist of several disjoint sub-grids. During the enlarging process, the naive approach does not consider this situation but extends the border outwards. In this case, it is possible that the enlarged partition includes nodes and edges from other unrelated sub-grids that do not improve the accuracy of the solution on the primary partition. The BFS-based approach does not include these nodes and edges in the enlarged partition, making the partition lightweight and efficient for computations.

2. The naive approach retains all the edges in the enlarged partition. In contrast, the BFS-based partition enlarging method only includes the edges that have the most useful information (largest conductance) and neglect other edges. This strategy reduces the time to solve the enlarged partition system since the conductance matrices have fewer non-zeros due to the inclusion of fewer edges.

3. In a power grid, there exist nodes that are not connected to every neighbor. The naive approach to extending the partitions ends up including some nodes that may not be connected to any other nodes inside the enlarged partition. These resulting isolated nodes make the conductance matrix singular since they have no incident edges. In the BFS-based approach, since each included node is visited from its neighbor and at least one edge from this node is retained, there does not exist any isolated nodes. The BFS-based approach make the enlarged partition friendly to linear system solvers.

After constructing the enlarged partitions using BFS, we have linear systems for enlarged partitions as

$$\overline{G_{ep}^{i}}\,\overline{V_{ep}^{i}} = \overline{I_{ep}^{i}} \tag{8}$$

where $\overline{G_{ep}^{i}}$ is the conductance matrix for enlarged partition i; $\overline{V_{ep}^{i}}$ denotes the associated node voltage vector; and $\overline{I_{ep}^{i}}$ represents the current source vector. After (8) is solved, the solutions belonging to the original partitions are extracted. The solutions are more accurate than the ones obtained from the original partition.

Table I. Solution errors of enlarged partitions with different EP-size

EP-size	**Max error**	**Average error**
0	153.2%	7.9%
5	42.8%	4.3%
15	10.9%	1.2%
200	0.0%	0.0%

To verify that the enlarged partitions lead to more accurate solutions, we compare the average error and maximum error in Table I. The results are obtained on the benchmark ckt3 (introduced in III.2) with 3×4 partitions. As described before, after solving the enlarged partition, only the solutions on the nodes that belong to the original partition are retained. Max error denotes the maximum relative error in node voltage by solving the enlarged partitions; Average error denotes the relative average error. The solution was obtained using a direct solver to guarantee computational accuracy. RL-size is set to be 0 to better demonstrate the comparison. The trend of error changes illustrates that the larger the enlarged partition is, the more accurate the solution we obtain. Two

extreme cases are that EP-size equals 0 and EP-size is large enough to make the enlarged partition cover the whole power grid. As for the first case, the error is huge as seen in the first row of Table I. In the second case, the whole power grid is solved and only the solutions within the original partition are retained. The error is 0 because the exact solutions are obtained.

II.2.4 Error reduction using preconditioned conjugate gradient method

As presented in II.2.3, larger enlarged partitions lead to more accurate solutions. However, larger enlarged partitions are more costly to solve. Using modest size of enlarged partitions, we obtain solution errors that are still larger than our tolerance. In order to reduce the solution error in an efficient way, we adopt PCG method to conduct the error reduction.

Next, we first briefly review the CG method and preconditioning technique. Then, the proposed Enlarged-Partition based Preconditioned Conjugate Gradient (EPPCG) method is presented.

CG is an iterative Krylov subspace method for solving s.p.d linear systems [12]. Compared to direct solvers, CG requires less memory storage and is friendly to extremely large sparse linear systems. CG method is often enhanced by applying a preconditioner to achieve fast convergence.

Preconditioning is a method to accelerate the solution of linear systems. Suppose we have a linear system $\mathrm{Ax} = \mathrm{b}$. Preconditioning is to apply a matrix M to the linear system as $\mathrm{M}^{-1}\mathrm{Ax} = \mathrm{M}^{-1}\mathrm{b}$, such that the condition number of $\mathrm{M}^{-1}\mathrm{A}$ is less than that of

A. By applying an effective preconditioner to a linear system, the linear system becomes easier to solve. One extreme case is where $\mathrm{M} = \mathrm{A}$, then the left-hand side of the linear system becomes x. As a result, the linear system has been solved by applying such a preconditioner. Usually, a preconditioner is used in iterative solver to accelerate the convergence. In PCG, a preconditioner M is used in each iteration involving computing a matrix-vector product $\mathrm{z} = \mathrm{M}^{-1}\mathrm{r}$. Note that z can be computed by solving an equivalent linear system $\mathrm{Mz} = \mathrm{r}$. The preconditioner M can be explicit or implicit. In PCG, an explicit preconditioner is applied by computing the matrix-vector product as $\mathrm{z} = \mathrm{M}^{-1}\mathrm{r}$. An implicit preconditioner does not have an explicit expression. It outputs a vector z using a given input vector r after a process of computations. This process refers to a preconditioning process.

A good preconditioner is key to the performance of the PCG method. Ideally, a preconditioner is inexpensive to construct and effective for reducing the number of iterations. In our proposed EPPCG method, we apply the implicit preconditioner by solving the enlarged partitions and extracting the solutions belonging to the original partitions as described in Algorithm 2. In other words, given an input vector r, we approximate the solution of the linear system $Mz = r$ by solving the enlarged partitions.

The proposed preconditioner has the following advantages.

The preconditioner is parallelizable. The preconditioner is conducted by solving the enlarged partitions. Solving the enlarged partitions can be done in parallel. In EPPCG, preconditioning costs most of the runtime. A parallelizable preconditioner can help reduce the runtime drastically.

Our experimental results discussed in Chapter III indicate that the enlareged-partition based preconditioner is effective for achieving fast convergence in EPPCG. The proposed preconditioner reduces the number of iterations significantly.

ALGORITHM 2. Enlarged Partition Based Preconditioner

Input: N × 1vector r, enlarged partition information containing retained node IDs and edges.

Output: Preconditioned N × 1 vector z

1: Separate r into r_{ep}^{i} based on enlarged partition information

2: ***for****(each enlarged partition)*

3: solve $G_{ep}^{i} z_{ep}^{i} = r_{ep}^{i}$ using a direct solver;

4: ***end for***

5: ***for****(each original partition)*

6: extract z^{i} of original partitions from z_{ep}^{i};

7: ***end for***

8: combine z^{i} into the global vector z

9: ***return*** *z*

ALGORITHM 3. EPPCG

Define: tol is the error tolerance; $K_{max}$is the maximum number of iterations;

Input: The global conductance matrix G; the global current drain vector I; the error tolerance tol; the maximum number of iterations K_{max}

Output: The node voltage vector v.

1: Factorize the conductance matrix G_{ep}^{i} for each enlarged partition

2: Obtain an initial solution v_0 by applying EP preconditioner on vector I .

3: $r_0 = I - Gv_0$ *//matrix-vector multiplication is conducted on enlarged partitions*

4: Obtain the preconditioned vector z by applying EP preconditioner on r_0.

5: $t_0 = r_0$

6: ***while*** *(* $k \leq K_{max}$*)*

7: *do* $\alpha_k = \frac{<r_k, z_k>}{<t_k,\ Gt_k>}$

8: $v_{k+1} = v_k + \alpha_k t_k$

9: $r_{k+1} = r_k - \alpha_k G t_k$ *//matrix-vector multiplication is conducted on enlarged partitions*

10: ***If*** $(\|r\| < tol,)$ *return* v_{k+1}

11: *Obtain* z_{k+1}*by applying EP perconditioner on* r_{k+1}

12: $\beta_k = \frac{<r_{k+1}, z_{k+1}>}{<r_k,\ z_k>}$

13: $t_{k+1} = z_{k+1} + \beta_k t_{k+1}$

14: $k = k + 1$

15: ***end while***

The overall EPPCG algorithm is described as Algorithm 3. The enlarged partitions are solved using a state-of-the-art direct solver, CHOLMOD [8, 63]. CHOLMOD consists of matrix factorization and triangular solve. The runtime of matrix factorization dominates the whole process. The conductance matrices of enlarged partitions do not change in EPPCG. Thus, we factor the matrix before entering the PCG loop, so that the factors are re-used in each iteration. This strategy helps improve performance by eliminating unnecessary repeated factorization.

II.3 Computational complexity and performance modeling

II.3.1 Complexity analysis

As described in Algorithm 3, the EPPCG method consists of two main computational parts. One is the factorization and triangular solve of the conductance matrices for enlarged partitions. The other one consists of remaining computations in PCG method such as matrix-vector multiplications, vector inner-products and other vector operations.

Suppose the power grid is divided into P partitions, enlarging the P partitions results in P enlarged partitions. The number of nodes in the partitions are $n_1, n_2, \dots, n_P$. For enlarged partition i, suppose the factorization of conductance matrix for enlarged partition costs $T_f(n_i)$ and applying the precondtioner costs $T_p(n_i)$. The runtime of matrix-vector multiplication is defined as $T_{mv}(n_i)$ and the runtime of vector inner-

product is defined as $T_{vi}(n_i)$. Each of the vector operations, such as vector additions, scalar-vector multiplications, is defined as $T_v(n_i)$, since they have the same complexity. The overall sequential runtime of EPPCG can be described as

$$T = \sum_{i=1}^{P} T_f(n_i) + 2\sum_{i=1}^{P} T_p(n_i) + \sum_{i=1}^{P} T_{mv}(n_i) + \sum_{i=1}^{P} T_v(n_i)$$
$$+ m\left(\sum_{i=1}^{P} T_p(n_i) + \sum_{i=1}^{P} T_{mv}(n_i) + 9\sum_{i=1}^{P} T_v(n_i)\right) \quad (9)$$

where T denotes the overall runtime and m denotes the number of iterations. Based on our experiments, m is in the range of [2-6]. Note that $T_p(n_i)$ includes triangular solve and vector extraction operations as illustrated in Algorithm 2. For each enlarged partitions with n_i nodes, $T_f(n_i) = O(n_i^{1.5})$, $T_p(n_i) = O(n_i \log n_i)$, and $T_v(n_i) = O(n_i)$ [9]. To estimate $T_{mv}(n_i)$, we consider the structure of the power grid. In a power grid, a node is connected to up to four neighbor nodes. As a result, there are at most 4 non-zeros at each row/column in the conductance matrix. For a power grid with n nodes, a matrix-vector multiplication runs at most 7n operations. Thus, $T_{mv}(n_i) = O(n_i)$.

More interesting analysis is the parallel complexity estimation. Assume the number of processors is identical to the number of partitions. Each item in (9) is determined by the slowest process. The overall parallel runtime can be approximated as an equation

$$\overline{T} = \max_{1\le i\le P} T_f(n_i) + 2\max_{1\le i\le P} T_p(n_i) + \max_{1\le i\le P} T_{mv}(n_i) + \max_{1\le i\le P} T_v(n_i)$$
$$+ m\left(\max_{1\le i\le P} T_p(n_i) + \max_{1\le i\le P} T_{mv}(n_i) + 9\max_{1\le i\le P} T_v(n_i)\right) \quad (10)$$

where $\overline{T}$ denotes the overall runtime of parallel EPPCG. Since power grid is divided into partitions with equal size and the partitions are enlarged to the same level, the resulting enlarged partitions are of similar size. This work load balance helps the parallel implementation achieve significant speedup. Let $\tilde{n}$ denote the number of nodes of the largest enlarged partition. Based on the estimate of each item discussed above, $\overline{T} = O(\tilde{n}^{1.5} + m\tilde{n}log\tilde{n})$. In the complexity estimation, the factorization of conductance matrices dominates the whole algorithm for extremely large power grids. The parallelization of factorization step helps reduce the runtime drastically.

II.3.2 Performance modeling

EP-size and RL-size are two parameters to control the speed of convergence of EPPCG. On one hand, if EP-size and RL-size are too large, the factorization time increase due to the expansion of conductance matrices. On the other hand, if EP-size and RL-size are too small, the convergence becomes slow due to the inaccuracy of the solutions to the enlarged partitions. Therefore, finding optimal EP-size and RL-size helps maximize the efficiency of EPPCG. RL-size is found to be 0.75×EP-size approximately in our experiments. Thus, in the following discussion, we are focused on finding the optimal EP-size.

The parallel runtime is approximated in (10). We divide the power grid into equal-sized partitions and the enlarged partitions are formed by extending the partition to the same level. Therefore, the work load of each enlarged partition is similar. We can obtain the following equation derived from (10).

$$\overline{T} \approx T_f(\bar{n}) + 2\,T_p(\bar{n}) + T_{mv}(\bar{n}) + T_v(\bar{n}) + m\left(T_p(\bar{n}) + T_{mv}(\bar{n}) + T_v(\bar{n})\right) \quad (11)$$

where $\bar{n}$ denotes the average number of nodes in an enlarged partition.

Based on our discussion on time complexity in II.3.1, $T_f(\bar{n})$, $T_p(\bar{n})$, and $T_{mv}(\bar{n})$ can be approximated as follows.

$$T_f(\bar{n}) \approx C_1\bar{n}^{1.5} + K_1 \quad (12)$$

$$T_{mv}(\bar{n}) \approx C_2\bar{n} + K_2 \quad (13)$$

$$T_p(\bar{n}) \approx C_3\bar{n}\,log\bar{n} + K_3 \quad (14)$$

where C_1, C_2, C_3, K_1, K_2, and K_3 are constants.

Suppose there are n nodes and p partitions, the average number of nodes in a partition is n/p. If we consider the partition as a square, each side of the square contains $\sqrt{n/p}$ nodes. Since the original partition is large and the number of levels of extension is not significant, we can ignore the difference of number of nodes between levels. Note that the partition is extended on all the four sides of the square. Therefore, the number of nodes added by extending the partition can be approximated as $4Ep\sqrt{n/p}$, where Ep denotes EP-size. The total number of nodes $\bar{n}$ in an enlarged partition can be approximated as follows.

$$\bar{n} \approx 4Ep\sqrt{\frac{n}{p}} + \frac{n}{p} \quad (15)$$

Based on our experimental observation, $T_f(\bar{n})$ and $T_{mv}(\bar{n})$ dominates the overall runtime, and the typical value of m is 3. Therefore, $\overline{T}$ can be approximated using the following equation.

$$\overline{T} \approx C_1\left(4Ep\sqrt{\frac{n}{p}}+\frac{n}{p}\right)^{1.5} + K_1 + \quad C_2\left(4Ep\sqrt{\frac{n}{p}}+\frac{n}{p}\right) + K_2 \tag{16}$$

The unknown constants C_1 and C_2 can be obtained using linear regression and observation data in the experiments. Using the data in the table on Page 51, we get the values as $C_1 = 3.0 \times 10^{-8}$ and $C_2 = 3.9 \times 10^{-5}$ in our experiment setup. K_1 and K_2 are not necessary for computing the optimal EP-size, as discussed as follows.

In order to find the optimal Ep (EP-size), we take derivatives in (16) and solve the following equation.

$$\frac{d\overline{T}}{dEp} \approx 1.5\, C_1\left(4Ep\sqrt{\frac{n}{p}}+\frac{n}{p}\right)^{0.5}\left(4\sqrt{\frac{n}{p}}\right) + \quad C_2\left(4\sqrt{\frac{n}{p}}\right) = 0 \tag{17}$$

II.4 Parallelism

The EPPCG algorithm is designed to parallelize both the preconditioning process and other computation steps (e.g., vector inner products, matrix-vector multiplications, etc.) in a straightforward way.

In this dissertation, we employ a two-tier parallelization scheme as shown in Figure 7. The power grid is divided into overlapping sub-grids (enlarged partitions). Correspondingly, sub-grid linear systems are generated. Each of the enlarged partition is mapped to an MPI process. The first tier MPI parallelization can be described as two parts.

Factorization: In EPPCG method, the conductance matrices of enlarged partitions are factored concurrently by MPI processes that own the partition.

Vector-related computations: The remaining computations of EPPCG, including matrix-vector multiplications, vector inner products, and vector additions, are also computed at the partition level by respective MPI processes. These computations may involve data exchanges and synchronizations among MPI processes. A gather-scatter strategy is adopted for vector operations. An index vector is constructed for vectors of each partition and enlarged partition (Figure 8). The index vector denotes the position of elements in the global vector. Matrix-vector multiplication is computed using the enlarged partitions since it needs at least one more level outside of the original partition to obtain the correct results. Vector inner products and other vector operations are done on original partitions. The MPI routine MPI_Allreduce is used when calculating the residual. MPI_Allgatherv routine is used to combine the local vectors into a global

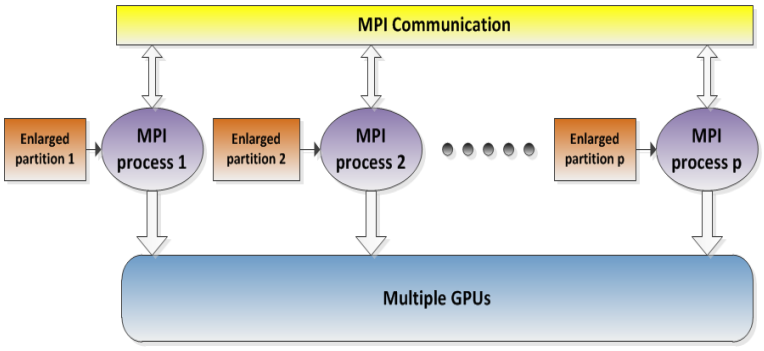

Figure 7. Parallelization scheme

vector. Vectors of enlarged partitions can be extracted from this global vector.

GPUs are used to accelerate sub-grid matrix factorizations and triangular solves. CHOLMOD is adopted as the direct solver for the sub-grid linear systems. CHOLMOD uses a supernodal method to obtain smaller dense matrices in the process of factoring a sparse linear system. The task of factoring these dense matrices is passed on to the GPU, which uses the highly optimized cudablas library to deliver high parallel performance. The overall performance of CHOLMOD on GPU varies depending on the structure of matrices and the amount of data transferred between the CPU and GPU.

In the parallel implementation of EPPCG, the number of partitions is always identical to the number of MPI processes. To obtain the best parallel performance, one

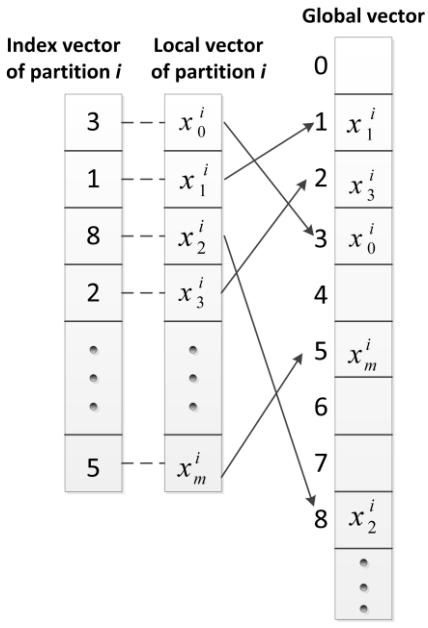

Figure 8. Local and global vector projection

should use the maximum number of GPUs available. For instance, a parallel implementation with 6 partitions/processes can use up to 6 GPUs. Each MPI process is assigned to a GPU card. When the number of MPI processes in an implementation exceeds the number of available GPU cards, multiple MPI processes share a GPU. For instance, 2 MPI processes are mapped to 1 GPU in the scenario where 12 MPI processes are created.

CHAPTER III

STATIC ANALYSIS OF POWER GRIDS

III.1 Static analysis of power grid problem

In static analysis of power grids, all the capacitors are open and all the inductors are short. A linear system $GV = I$ is constructed based on the power grid structure. EPPCG can be applied to static analysis directly. Some factors that need to be considered in static analysis using EPPCG are as below:

- EP-size and RL-size are two parameters to control the convergence of EPPCG. If these parameters are too large, the whole process suffers from heavy computation of matrix factorization since the matrices are larger. If the two parameters are too small, more iterations are expected such that the performance degrades due to slow convergence of the iterative solver.
- The number of partitions is a factor that affects the performance of EPPCG. Too many partitions make the solution of the enlarged partition inaccurate. This makes the convergence of EPPCG slow. If too few partitions are constructed, the parallel performance will not be significant due to insufficient coarse level parallelism, although the efficiency of the parallel implementation could be high.

III.2 Experimental results

To investigate the performance of EPPCG algorithm on static analysis, we conducted experiments on the supercomputer cluster Eos and Ada at Texas A&M Supercomputer center. The hardware specification of Eos and Ada are presented as below:

Eos supercomputing cluster:

- Number of nodes: 372
- Number of cores per node: 8
- CPU: 2.8GHz Nehalem
- Memory: 24GB DDR3 DRAM
- Operating system: 64-bit RedHat Linux
- GPU card per node: two computing nodes are equipped with two Nvidia M2050 GPU cards per node and two nodes are equipped with one Nvidia M2050 GPU card per node.
- GPU speed: 575 MHz
- GPU cores: 14 cores of GF100 architecture

Ada supercomputing cluster:

- Number of nodes: 845
- Number of cores per node: 8
- CPU: 2.5GHz IveBridge
- Memory: 64GB DDR3 DRAM
- Operating system: 64-bit CentOS 6.5 Linux

- GPU card per node: 2 Nvidia K20 GPUs
- GPU speed: 706 MHz
- GPU cores: 13 cores of GK110 architecture

The implementation of the proposed method is written in C++ and the algorithm is parallelized using MPI and CUDA. A tolerance of 10^{-4} was specified on the relative norm of residual error for EPPCG implementations.

We conducted experiments on IBM P/G benchmarks [64] and several artificial benchmark circuits. The format of the benchmarks is described in [65]. The specifications of the benchmarks vary as shown in Table II. The number of nodes (Num_node), number of resistance edges (Num_edge), number of C4 bumps (Num_C4) and number of current sources (Num_I) of each benchmark circuit are listed. Num_layer denotes the number of layers. Different layers are connected through vias. The experimental results will show the comparison of EPPCG and CHOLMOD, the comparison of EPPCG and other iterative solvers, the MPI-based parallel performance, the GPU-accelerated parallel performance, the impact of partitioning, the impact of varying parameters, the performance on different clusters, the performance of different parallel implementations, and the scalability.

III.2.1 Comparison with state-of-the-art direct solver

A direct solver is widely used in power grid simulation in the industry. In our experience, CHOLMOD is among the fastest public domain single-processor direct solvers. We compare our EPPCG method parallelized using MPI and GPU (MPI-GPU-

Table II. Specification of experimental power grids

Circuit	Num_node	Num_edge	Num_C4	Num_I	Num_layer
ibmpg3	851,584	1400353	955	201,054	5
ibmpg4	953,583	1550719	962	276,976	6
ibmpg5	1,079,310	1615381	539,087	540,800	3
ibmpg6	1,670,494	2484098	836,239	761,484	3
ibmpgnew1	1,461,036	2351136	955	357,930	6
ibmpgnew2	1,461,039	2351136	930,216	357,930	6
ckt1	3,006,003	5,007,002	1,600	1,508,320	4
ckt2	4,327,203	7,208,402	2,304	1,609,834	4
ckt3	6,759,003	11,260,502	3,600	2,154,654	6
ckt4	5,077,803	8,459,102	27,040	1,465,455	6
ckt5	6,132,002	9,194,501	4,900	2,335,540	4

Table III. Speed improvement over direct solver

Circuit	CHOLMOD		MPI-GPU-EPPCG (16 processors)						**Spd-Imp**
	T(s)	Mem (GB)	EP-size	RL-size	T(s)	Mem (GB)	max-error(V)	aver-error(V)	
ckt1	282.35	7.9	45	35	4.65	8.0	2.67E-05	1.19E-05	**61X**
ckt2	406.83	12.0	30	22	5.73	12.6	2.87E-04	7.69E-05	**71X**
ckt3	1375.52	20.0	40	35	9.69	20.0	6.32E-05	2.83E-05	**142X**
ckt4	819.53	14.1	40	30	7.45	15.0	1.37E-05	3.98E-06	**110X**
ckt5	451.6	13.8	40	35	7.12	14.1	2.49E-04	7.35E-05	**63X**

EPPCG) with CHOLMOD in Table III. T denotes the overall runtime, Max_error denotes the maximum solution error and Aver_error denotes the average solution error. MPI-GPU-EPPCG method solves the power grids much faster than the state-of-the-art direct solver CHOLMOD without losing much accuracy.

The accuracy of the solution is determined by the residual norm's tolerance for the CG iterations. Desired accuracy can be achieved by choosing an appropriate value for the tolerance. A smaller value of tolerance requires additional iterations of the CG

Table IV. Speed comparison between iterative solvers and serial EP-PCG solver

Benchmark	CG		Jacobi PCG		Serial EP-PCG			
	Iter	Runtime	Iter	T(s)	EP-size	RL-size	Iter	T(s)
ckt1	1808	635.0	784	243.9	40	35	5	68.6
ckt2	1995	966.0	774	378.6	30	22	2	85.1
ckt3	1783	1514.0	761	630.9	40	35	4	230.3
ckt4	2391	1243.0	770	456.0	40	30	5	132.7
ckt5	1406	1006.0	606	430.1	40	35	3	128.7

method, which increases the cost. However, the increase in cost is relatively small when compared to the one-time cost of factoring the partitions at the start of the algorithm. A change in the number of iterations does not impact parallel efficiency of the CG method itself.

The memory usage of EPPCG is comparable to CHOLMOD. The speed improvement (Spd-Imp) of parallel EPPCG over CHOLMOD ranges from 61X to 142X.

Table V. MPI parallel performance

Circuit	EP-size	RL-size	Ite	Serial EPPCG (sec)			MPI- EPPCG (sec)			Speedup (serial EPPCG vs MPI-EPPCG)		
				factor	pcg	overall	factor	pcg	overall	factor	pcg	overall
ckt1	40	35	5	58.3	10.2	68.6	3.6	1.2	4.9	16.1X	8.3X	**14X**
ckt2	30	22	2	75.9	9.2	85.1	5.0	0.8	5.8	15.2X	11.3X	**15X**
ckt3	40	35	4	210.7	19.6	230.3	14.1	2.9	17.0	15.0X	6.8X	**14X**
ckt4	40	30	5	113.9	18.8	132.7	6.9	2.2	9.0	16.6X	8.8X	**15X**
ckt5	40	35	3	116.3	12.4	128.7	7.4	1.7	9.1	15.8X	7.2X	**14X**

III.2.2 Comparison with other iterative solvers

Table IV shows the performance comparison among classic CG method, Jacobi PCG method and serial EP-PCG method with 4×4 partitions. Classic CG performs the worst among all the methods since it requires a large number of iterations to converge. Jacobi PCG reduces runtime by applying a diagonal preconditioner. The proposed EP-

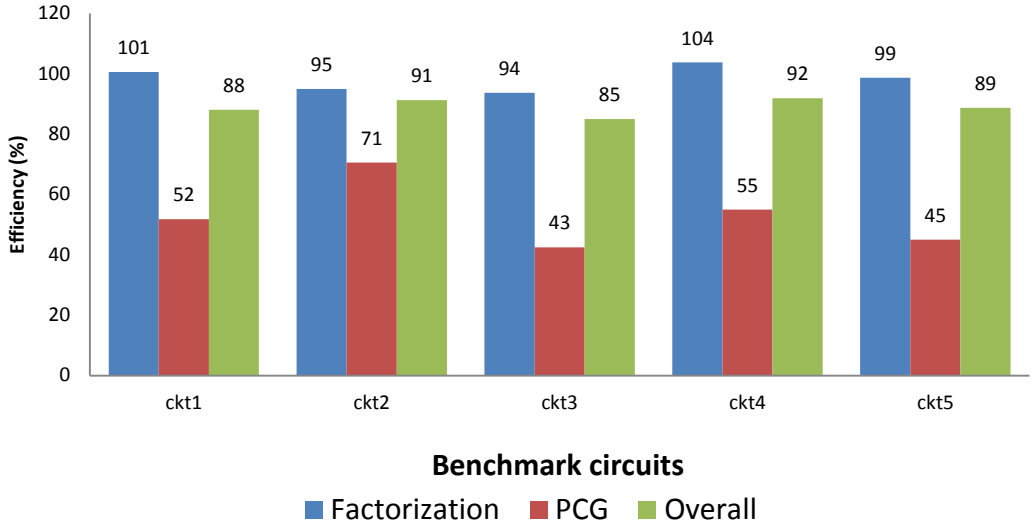

Figure 9. MPI parallel efficiency for different benchmarks

PCG method is able to reduce the number of iterations significantly due to the effectiveness of the EP-preconditioner. The effective EP-preconditioner makes the serial EP-PCG fastest among all the iterative solvers.

III.2.3 MPI-based parallel performance

The parallel performance of MPI-based EPPCG method (MPI-EPPCG) is investigated with 4 × 4 partitions using 16 MPI processes in Table V and Figure 9. Iterative solvers suffer from slow convergence rate easily, however, EPPCG achieves fast convergence by employing an effective enlarged-partition based preconditioner. The number of iterations (Ite) is in a single-digit range for all the benchmarks.

Table VI. Enhancing parallel performance with GPUs

Circuit	EP-size	RL-size	Ite	MPI- EPPCG (sec)			MPI-GPU-EPPCG (sec)			GPU Speedup		
				factor	pcg	overall	factor	pcg	overall	factor	pcg	overall
ckt1	45	35	3	9.0	1.1	10.1	6.3	1.1	7.5	1.4X	1.0X	**1.4X**
ckt2	30	25	3	11.9	1.6	13.5	7.8	1.5	9.3	1.5X	1.1X	**1.4X**
ckt3	30	25	3	36.4	4.7	41.1	13.3	3.8	17.2	2.7X	1.2X	**2.4X**
ckt4	50	45	3	18.5	1.9	20.4	9.1	2.0	11.1	2.0X	0.9X	**1.8X**
ckt5	45	40	2	21.8	1.5	23.3	9.9	1.6	11.5	2.2X	1.0X	**2.0X**

In Table V, “factor” denotes the runtime of matrix factorization for enlarged partition conductance matrices using CHOLMOD, “pcg” denotes the accumulated runtime other than factorization step in EPPCG, and “overall” denotes the total runtime of EPPCG. As seen in Table V, the factorization dominates the overall runtime. Thus,

the overall parallel performance is mainly determined by the factorization process. Our parallel implementation achieves significant speedup on factorization and reasonable speedup on remaining PCG computations. MPI parallelization efficiency is defined as the ratio of speedup and number of processes. Figure 9 illustrates the parallel efficiency of MPI-EPPCG. Superlinear speedup occurs in the factorization process on ckt1 and ckt4 due to the effective utilization of cache.

III.2.4 GPU-accelerated parallel performance

In order to study the effect of GPU on parallel performance, we compare MPI-EPPCG with GPU-MPI-EPPCG. Table VI shows the comparison with 3×3 partitions using 9 MPI processes and 6 GPUs. 3 GPUs are shared by 6 MPI processes and the other 3 GPUs are assigned to 3 MPI processes. The factorization speedup of MPI-GPU-EPPCG over MPI-EPPCG ranges from 1.4X to 2.7X. As expected, the GPU performs better for heavy-weight work load computations (e.g. ckt3) than light-weight work load. In terms of pcg process, the GPU performance degrades for some benchmarks because of overheads in CUDA parallelization.

The overall parallel performance for each implementation is shown in Figure 10. All the benchmark circuits are divided into 3×4 partitions. It can be seen that with efficient parallelization, MPI-GPU-EPPCG reduces the runtime further.

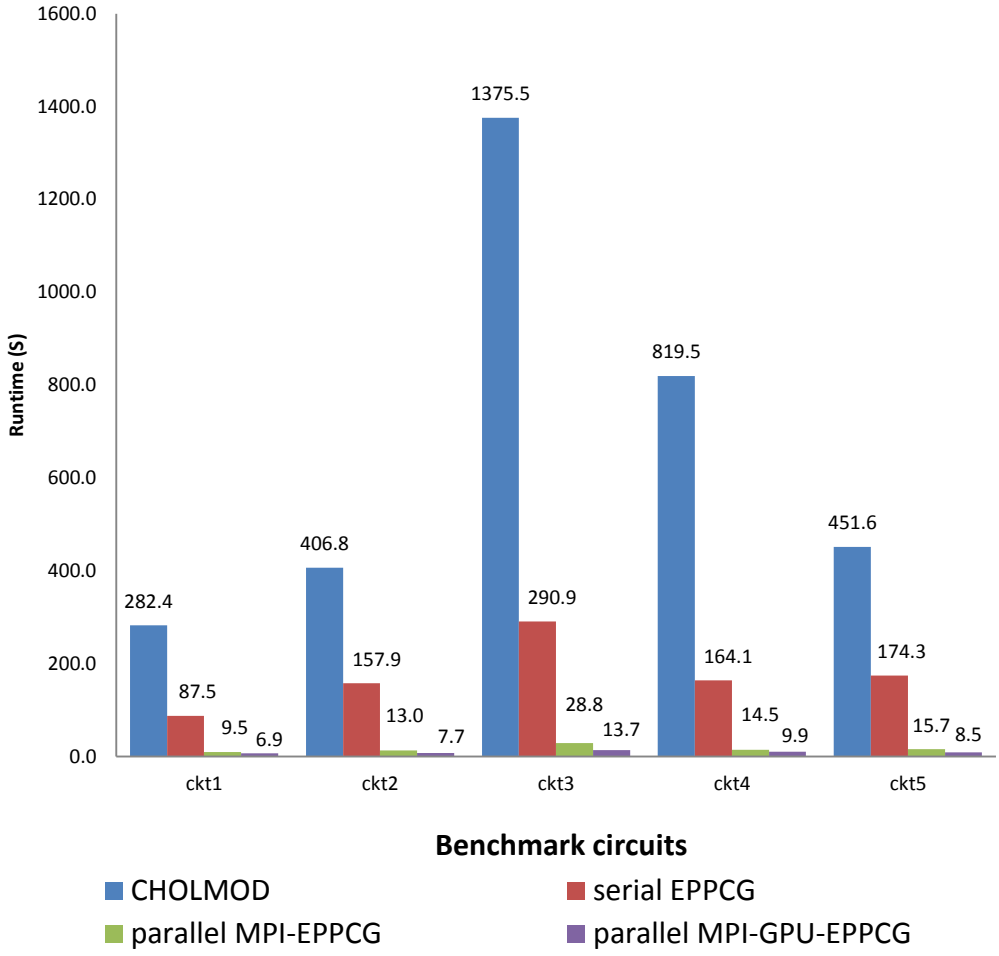

Figure 10. Overall parallel performance (time in seconds)

III.2.5 Impact of partitioning

The EPPCG method divides the power grid into partitions. The results for different number of partitions are shown in Table VII. For each benchmark circuit, the two parameters EP-size and RL-size are identical for different number of partitions. Parallelization efficiency is defined as the ratio of speedup and the number of processes.

Table VII. MPI-GPU-EPPCG performance with different number of partitions

No. of Process	ckt3				ckt4				ckt5			
	#Iter	T(S)	Speed up	Efficiency	#Iter	T(S)	Speed up	Efficiency	#Iter	T(S)	Speed up	Efficiency
9	4	17.0	7.9	87.8%	6	13.5	9.1	101.3%	3	10.9	8.9	98.4%
12	3	12.8	11.6	96.7%	4	9.5	12.1	100.7%	3	8.0	12.0	100.3%
16	4	9.7	13.8	86.3%	5	7.5	14.7	91.9%	3	7.1	14.4	90.0%
25	5	9.6	19.5	77.9%	6	8.7	18.3	73.0%	5	8.6	19.1	76.5%

Note that using too many partitions may downgrade the performance due to the overheads of MPI processes and GPU threads. If the work load is light-weight, those overheads will dominate the processing and lower the parallel performance. If the problem size is large enough, it is expected that many more partitions can be created to obtain higher speed improvement. One must note that selecting a large number of smaller sized partitions has a negative impact on the quality of the preconditioner, which may increase the number of iterations resulting in a higher solution time. The results are shown in Figure 11.

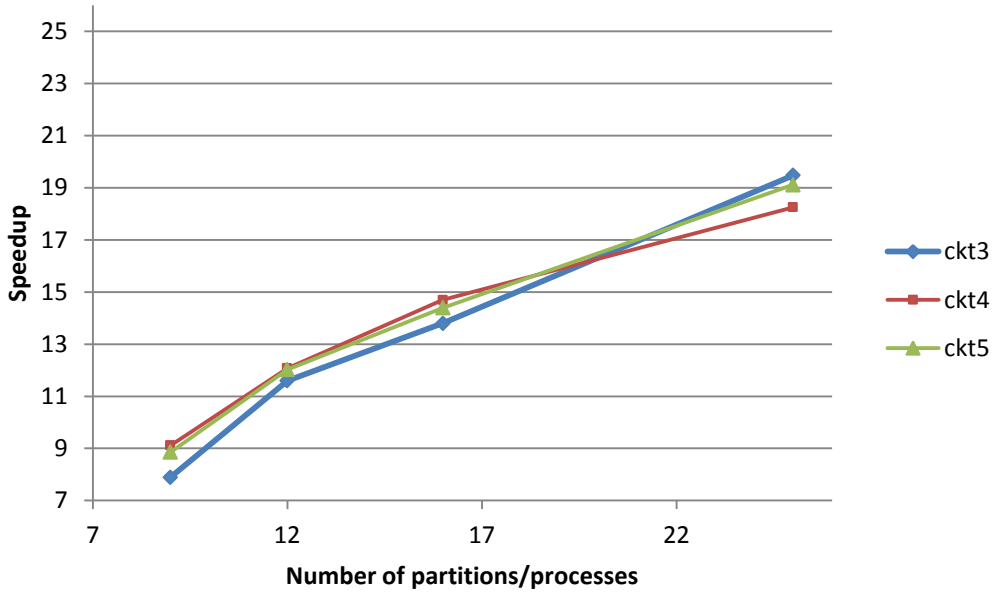

Figure 11. Parallel performance with different number of partitions

III.2.6 Impact of parameters

The effect of varying parameters EP-size and RL-size was also investigated. Table VIII shows the runtime results for ckt1 with 3×4 partitions.

Different EP-size and RL-size lead to different performance. A smaller value of EP-Size reduces factorization time and PCG time by lowering the cost of factorization provided the number of iterations needed to converge to the solution do not grow; a larger value of RL-Size reduces the number of iterations but makes the factorization more expensive. Thus EP-Size and RL-Size need to be chosen carefully to obtain the best performance.

Table VIII. Parallel performance with varying parameters

	EP-size	RL-size	Ite	serial EPPCG			paralllel MPI-EPPCG			parallel MPI-GPU-EPPCG		
				factor	pcg	total	factor	pcg	total	factor	pcg	total
Ckt1	37	28	8	74.21	13.86	88.07	7.52	1.96	9.48	4.93	2.08	7.01
	40	30	5	77.89	9.61	87.50	8.19	1.26	9.45	5.27	1.59	6.86
	40	35	3	85.96	6.61	92.57	6.80	0.93	7.73	4.83	0.89	5.72
	45	35	3	86.00	6.69	92.69	6.80	1.01	7.81	4.82	0.95	5.77
	50	45	2	92.00	5.87	97.87	7.33	0.76	8.09	5.23	0.71	5.94

III.2.7 Performance on different machines

In order to demonstrate the consistency of EPPCG on different machines, experiments were conducted on both EOS and ADA. ADA is an IBM NeXtScale super computing cluster with 845 nodes. Each node is equipped with 20-core 2.5 GHz IvyBridge CPU. 30 of the nodes have 2 Nvidia K20 GPUs installed. The memory size of each node is 64 GB and Linux (CentOS 6.5) is running on all the nodes. The power grids are divided into 9 partitions. 9 MPI processes and 6 GPU cards are used on both EOS and ADA. The

Table IX. Comparison of EPPCG performance between machines

Circuit	EP-size	RL-size	Ite	MPI-GPU-EPPCG(EOS)		MPI-GPU-EPPCG(ADA)	
				T	Spd-Imp	T	Spd-Imp
ckt1	45	35	3	7.5	38X	9.2	32X
ckt2	30	25	3	9.3	44X	11.7	49X
ckt3	30	25	3	17.2	80X	18.4	77X
ckt4	50	45	3	11.1	74X	12.7	70X
ckt5	45	40	2	11.5	39X	11.4	45X

experimental results are shown in Table IX. From the results we observe the two clusters give us similar results. The speedup of MPI-GPU-EPPCG over CHOLMOD on EOS ranges from 38X to 80X, and on ADA the range is from 32X to 77X. These results demonstrate the consistency of EPPCG method on different machines.

Table X. Comparison between MPI and OpenMP performance

Circuit	EP-size	RL-size	Ite	serial	16-process MPI		16-thread OpenMP	
				T(s)	T(s)	Speedup	T(s)	Speedup
ckt1	40.00	35.00	5.00	68.58	4.85	14.14	6.90	9.94
ckt2	30.00	22.00	2.00	85.11	5.81	14.65	10.50	8.11
ckt3	40.00	35.00	4.00	230.31	16.98	13.56	25.60	9.00
ckt4	40.00	30.00	5.00	132.71	9.00	14.75	13.20	10.05
ckt5	40.00	35.00	3.00	128.68	9.09	14.16	15.10	8.52

III.2.8 Different parallel implementation

In EPPCG, the factorization dominates the whole process. The parallelization of factorization is the key factor that influences the overall parallel performance. In the parallelization of factorization, there is no data transfer between processes. Compared to MPI, OpenMP performs better when there is much communications between sub-

Table XI. EPPCG performance on different benchmarks

Circuit	CHOLMOD	MPI-GPU-EPPCG			Spd-Imp
	T(s)	EP-size	RL-size	T(s)	
ibmpg3	28.27	33	15	1.31	22X
ibmpg4	47.75	20	12	1.39	34X
ibmpg5	19.95	35	25	1.18	17X
ibmpg6	26.16	75	50	1.88	14X
ibmpgnew1	62.07	25	20	2.22	28X
ibmpgnew2	56.81	25	20	2.18	26X
ckt1	282.35	45	35	4.65	61X
ckt2	406.83	30	22	5.73	71X
ckt3	1375.52	40	35	9.69	142X
ckt4	819.53	40	30	7.45	110X
ckt5	451.6	40	35	7.12	63X

problems. However, since OpenMP runs on shared memory environment, the performance of OpenMP could downgrade on a distributed cluster due to the less-developed memory sharing mechanism. MPI and OpenMP implementations are compared in Table X. MPI performs better than OpenMP on all the benchmarks. The parallelization efficiency of MPI is much better than that of OpenMP. This observation can be explained by the latency of memory data sharing on distributed machines.

III.2.9 Scalability

In order to investigate the scalability of EPPCG, we ran experiments on all the available benchmarks and obtained the results in Table XI. The size of the benchmarks varies as shown in Table XI. In Table XI, we compared the performance of EPPCG on CPU-GPU hybrid platform with CHOLMOD. From the results, it is observed that the speedup achieved varies on different benchmarks. The best speedup is 140X on ckt3. The reasons for this observation can be described as the following. First, ckt3 is extremely large in size. A more complicated circuit with more nodes makes the parallel implementation of EPPCG more efficient since the overhead of MPI is much less than the actual computation cost. Second, the direct solver CHOLMOD suffers on benchmarks with more layers. In other words, direct solver does not perform as well on 3D circuits. More layers means larger portion of nodes has more neighbors than 2D circuits. CHOLMOD does not perform well on these linear systems.

CHAPTER IV

PARALLEL TRANSIENT ANALYSIS OF POWER GRID

IV.1 Transient analysis problem

In transient analysis, the power grid can be modeled as a RCL network as shown in Figure 12. Transient analysis is conducted over a certain amount of time. To simulate the transient response on computers, the period of time is discretized into time points. The interval between two time points is defined as time step. The time step size can be fixed or varying in the transient analysis. The process for transient power grid simulation with

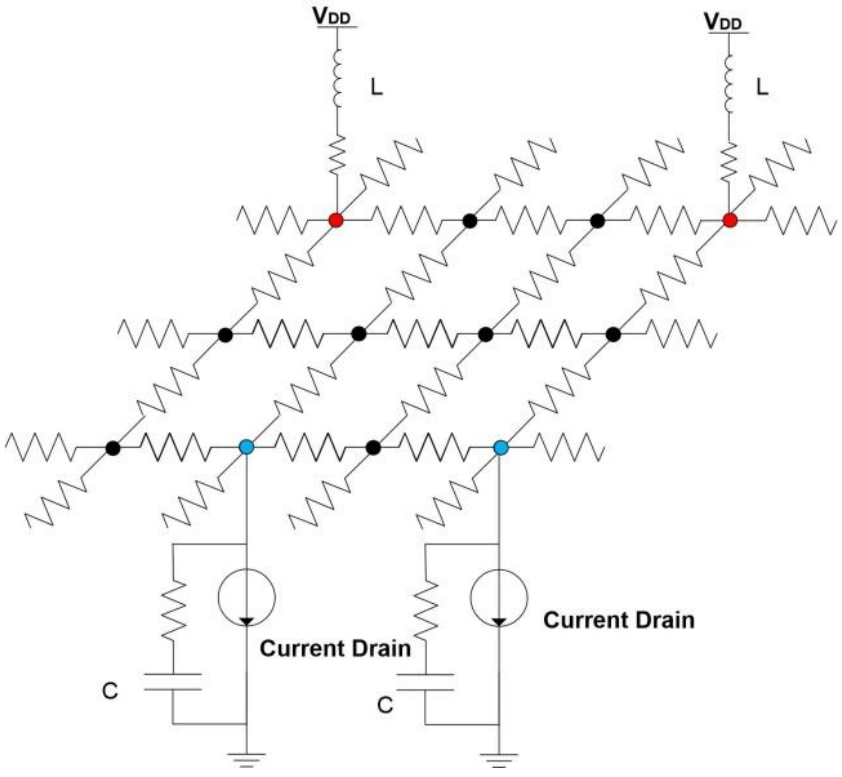

Figure 12. Modeling power grid in transient analysis

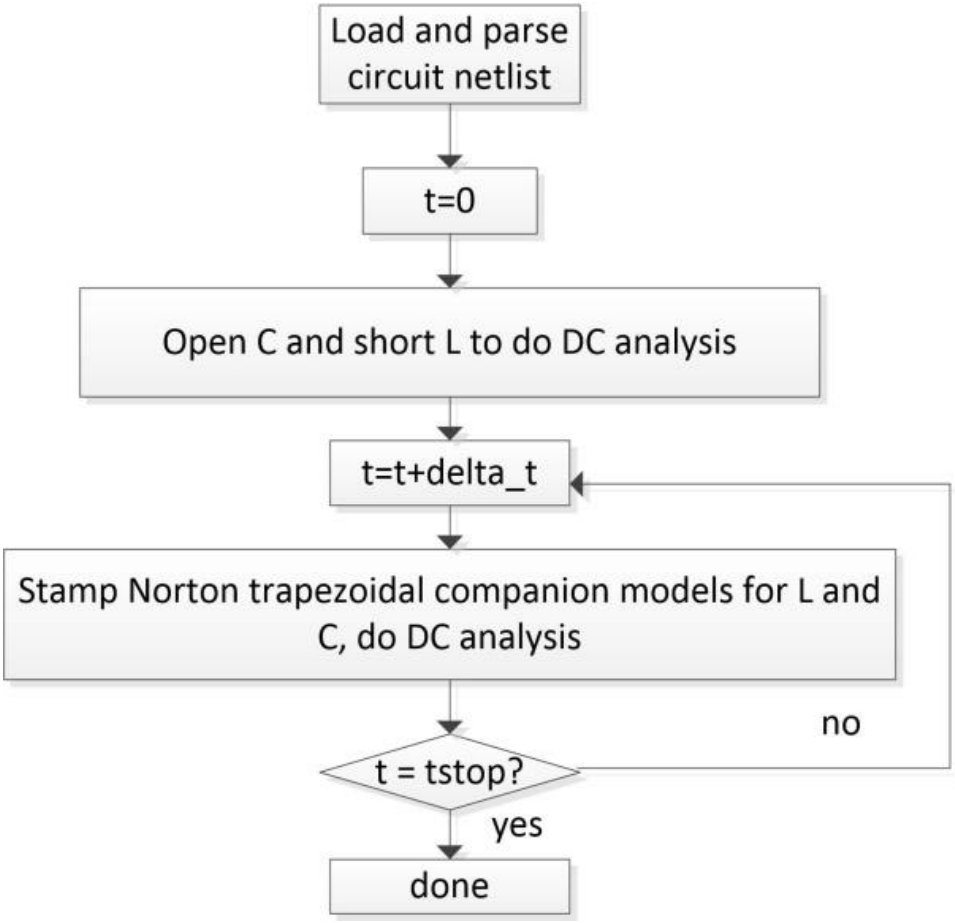

Figure 13. Flow of transient analysis with a fixed time step

a fixed time step is described in Figure 13. A power grid circuit contains only linear components, such as resistors, linear capacitors, linear inductors, voltages sources and current sources. The value of each component is stamped in the resulting linear system. Stamping each component is discussed below.

Resistors are stamped in the G matrix. If the value of the resistor is R and the indices of the two terminals are *k* and *j*, the resistor stamps a 1/R at row *j* and *k* on the

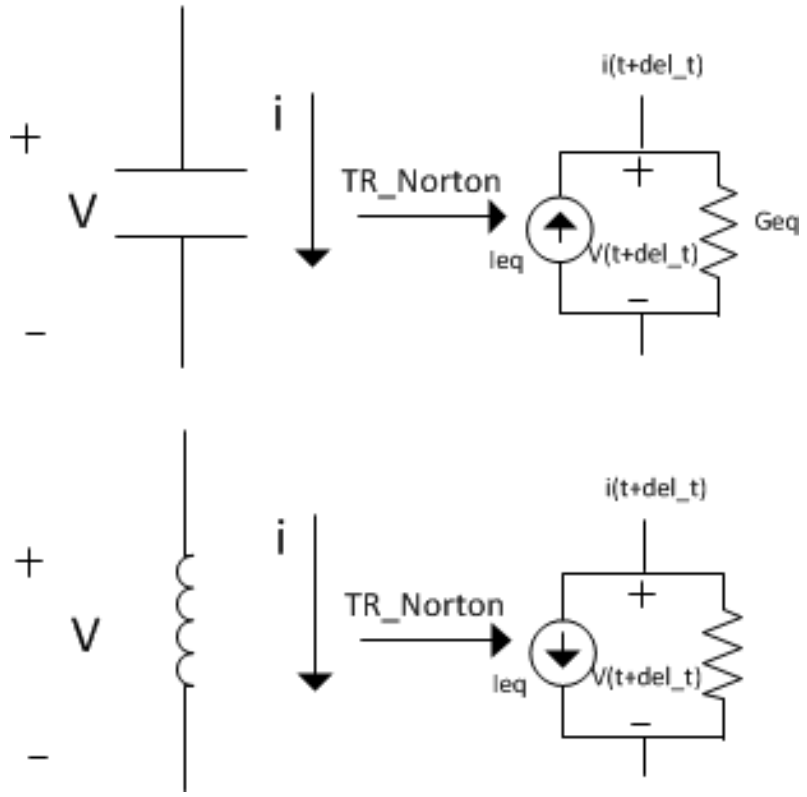

Figure 14. Companion model for *C* and *L*

diagonal and -1/R at (*k, j*) and (*j, k*) in the conductance matrix. In static and transient analysis, resistors give the same stamping results.

In transient analysis, voltages sources are stamped in both G matrix and I vector. Basically, for a voltage source V across node k and l, an equation $V_k - V_l = V$ is added to the linear system. Each time a voltage source is encountered, a row and column need to be created in G matrix. The new row or column has 1 and -1 stamped in G, depending

on the index of the two terminal nodes. In the vector I, the value of the voltage source V is added at the new added row.

Current drains are easier to handle. The only step is to add the value of the current source at each terminal node's row. The positive or negative sign needs to be added depending the on flowing direction of flow of the current. In transient analysis, the current drain is often time varying. Therefore, we need to re-stamp current drains which require updating the right hand side of the linear system at each time step.

Capacitors are considered open and inductors are considered short in static analysis. In transient analysis, a companion model is often used for each of the two components. To construct the companion model, multiple methods can be used. Forward Euler, Backward Euler and Trapezoidal methods are single-step methods to obtain the companion model. Trapezoidal method is the most stable method among the three. In power grid simulation, Norton equivalent circuit is often used to replace the capacitors and inductors [60]. Figure 14 shows the Norton trapezoidal companion model for capacitors and inductors.

In Figure 14, the values of equivalent current source and conductance for capacitors are computed using the following equation.

$$I_{eq}^{C}(t+\Delta t) = \frac{2C}{\Delta t} v^{C}(t) + v^{C}(t) G_{eq}^{C} - I_{eq}^{C}(t) \tag{18}$$

$$G_{eq}^{C} = \frac{2C}{\Delta t} \tag{19}$$

where Δt is the time step, $I_{eq}^{C}(t+\Delta t)$is the value of equivalent current source for the capacitor at time $t+\Delta t$, $I_{eq}^{C}(t)$ is the value of equivalent current source for the

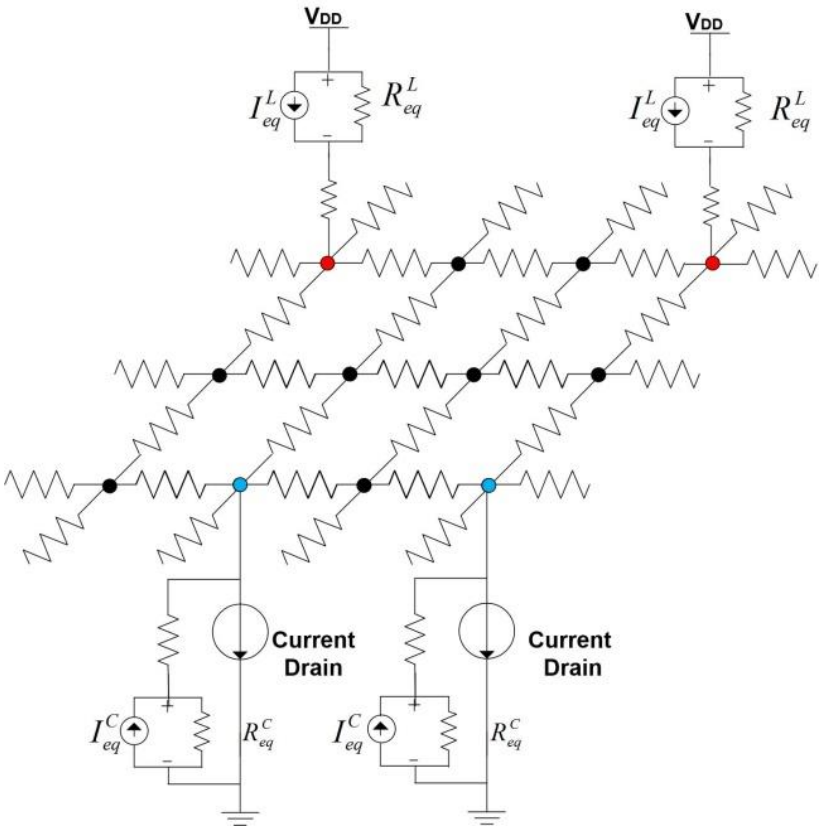

Figure 15. Companion model of power grid

capacitor at time t, G_{eq}^{C} is the value of the equivalent conductance, C is the value of the capacitor, and $v^{C}(t)$ is the voltage across the capacitor at time t.

The values of equivalent current source and conductance for inductors are computed using the following equation.

$$I_{eq}^{L}(t+\Delta t) = \frac{\Delta t}{2L} v^{L}(t) + v^{L}(t) G_{eq}^{L} + I_{eq}^{L}(t) \tag{20}$$

$$G_{eq}^{L} = \frac{\Delta t}{2L} \tag{21}$$

where Δt is the time step, $I_{eq}^{L}(t+\Delta t)$is the value of equivalent current source for the inductor at time $t+\Delta t$, $I_{eq}^{L}(t)$ is the value of equivalent current source for the inductor at time t, G_{eq}^{L} is the value of the equivalent conductance, L is the value of the capacitor, $v^{L}(t)$ is the voltage across the inductor at time t.

Figure 15 shows an equivalent circuit of power grid for transient analysis. For both capacitors and inductors, I_{eq} varies at each time point of the transient analysis. Therefore, I_{eq} needs to be re-calculated and re-stamped at each time point. G_{eq} is a function of time step Δt. As a consequence, if the time step Δt is fixed, the conductance matrix remains the same during the transient analysis; if the time step Δt varies, the conductance matrix needs to be re-constructed.

IV.2 Applying EPPCG in transient analysis

Although EPPCG is designed for one-step solve of a power grid, it extends naturally to transient case. Transient analysis requires many-step solves on different time points. When applying EPPCG in transient power grid analysis, there are several issues that need to be addressed in the implementation.

1. The node at which the inductor and V_{DD} resistor are interconnected must be included in the enlarged partition. Otherwise, the resulting enlarged partition may lose important information from the V_{DD} such that the solution accuracy cannot be guaranteed.

2. The solution at each time point must be accurate enough. If the error of the solution at each time point is large, the accumulated error would be too large to obtain accurate solution in the simulation.

ALGORITHM 4. Transient analysis with variable time step

1: ***while****(t<t_end)*

2: *perform a static analysis at time t using timestep to obtain solution* x_t

3: *compute Local Truncation Error(LTE) and timestep_upperbound*

4: ***if****(timestep_upperbound < 0.9·timestep)*

5: *reject* x_t *and let timestep = timestep_upperbound*

6: *recomputed solution* x_t

7: ***else***

8: *accept* $\boldsymbol{x_t}$

9: *update timestep = min(timestep_upperbound,* $2 \cdot$ *timestep) and update t=t+timestep*

10: ***end if***

*11:****end while***

IV.2.1 Fixed time step vs. variable time step

The time step size is constant in fixed time step analysis of the transient case, whereas the time step size changes in variable time step analysis. Fixed time step analysis is easy to implement. However, fixed time step analysis could be slow if the chosen time step is too small. Furthermore, fixed time step schemes can be unstable if the time step is not sufficiently small. Variable time step analysis needs more effort on the implementation but it guarantees stability and accuracy of the solution without compromising speed. The time step is maximized at each time point without degradation of solution accuracy and stability.

In fixed time step analysis, the conductance matrix remains unchanged. Direct solvers usually perform better than iterative solvers. Since the expensive step of matrix factorization is computed only once and the factor is reused at the remaining time points. On the other hand, the whole solving process of an iterative solver needs to be conducted at each time point. This “pre-processing” feature is a significant advantage that makes the direct solver faster than the iterative solver in fixed time step analysis.

In variable time step analysis, direct solver loses the advantage over iterative solvers. Since the conductance matrix changes at each time step, it needs to be re-factored. Thus, the whole solving process including factorization and triangle solve need to be conducted at each time point. The total time cost of the direct solver is a summation of time cost at each time point. As a consequence, the performance comparison of direct solvers and iterative solvers depends on the runtime of a single solve at each time point independent of the number of time steps.

For a linear circuit such as a power grid, a direct solver with fixed time step is usually adopted. The matrix factorization that dominates the solving process only needs to be conducted once and the triangle solving process whose cost is negligible is conducted at each time point. This makes direct solver with a fixed time step a good choice in transient power grid simulation.

IV.2.2 Step size control for EPPCG

The EPPCG method is a combination of direct solver and iterative solver. It consists of factorization process and iterative solve process. In order to exploit the advantage of EPPCG method in transient analysis, we developed a time step control algorithm using variable time steps. The time step control algorithm is based on the strategy used in SPICE [66]. The aim of this algorithm is to reduce the number of time points, such that the number of solves can be reduced. The idea of the algorithm is to reduce the step size when the solution error is larger than a threshold but to maximize the step size at each time point. The step control algorithm is presented in Algorithm 4. The solution error is estimated using Local Truncation Error (LTE). LTE is the error estimated by assuming the solution at the previous time point is accurate. In the Trapezoidal model, LTE can be estimated using the following formula.[67]

$$LTE_n \approx -\frac{1}{12} h_n^3 DD_3(t_n) \tag{22}$$

where $h_n = t_n - t_{n-1}$ is the step size at time point *n*, $DD_3(t_n)$ is the divided difference at time point t_n. $DD_3(t_n)$ can be calculated using the following formulas.

$$DD_1(t_n) = \frac{v_n - v_{n-1}}{h_n} \tag{23}$$

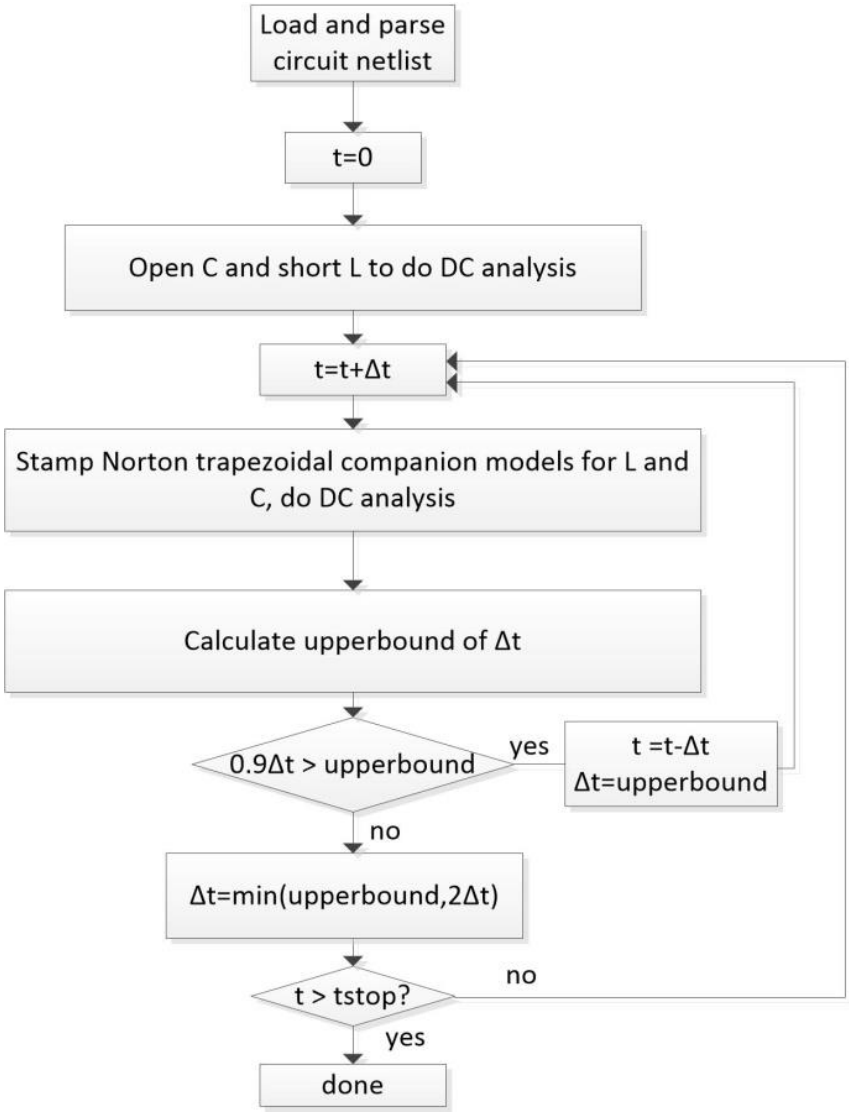

Figure 16. Flow of variable time step analysis

$$DD_2(t_n) = \frac{DD_1(t_n) - DD_1(t_{n-1})}{h_n + h_{n-1}} \tag{24}$$

$$DD_3(t_n) = \frac{DD_2(t_n) - DD_2(t_{n-1})}{h_n + h_{n-1} + h_{n-2}} \tag{25}$$

where h_n is the time step size at step n, h_{n-1} is the time step size at step $n-1$ and h_{n-2} is the time step size at step $n-2$.

In Algorithm 4, the timestep_upperbound can be calculated as below.

$$\bar{h}_n = \left(\frac{ErrorBound}{-\frac{1}{12} DD_3(t_n)} \right)^{\frac{1}{3}} \tag{26}$$

where *ErrorBound* is the LTE error bound that can be entered by the user.

Table XII. Specification of power grids for transient analysis

Circuit	**Num_node**	**Num_edge**	**Num_C4**	**Num_I**	**Num_layer**
ibmpg3t	1,039,624	1,589,351	955	201,054	5
ibmpg4t	1,210,440	1,808,541	962	265,944	6
ibmpg5t	1,552,230	2,088,581	277	473,200	4
ibmpg6t	2,366,420	3,180,408	381	761,484	4

The overall flow of variable time step analysis is shown in Figure. 16. In this variable time step algorithm, EPPCG only factorizes the conductance matrix once with the initial time step size. The factor is reused in the remaining time points. The reduction of number of time points make EPPCG performs better than the state-of-the-art direct solver.

IV.3 Experimental results

The experiments of transient analysis were conducted on IBM benchmarks. The specification of benchmark circuits is shown in Table XII. We used fixed time stepping scheme and variable time stepping schemes to obtain the solutions. The experiments were conducted on supercomputer cluster Ada. The detailed specification of Ada is described in III.2. The implementation is in C++ and the algorithm is parallelized using MPI. A tolerance of 10^{-3} was specified as the LTE error bound in the variable time step analysis.

IV.3.1 Comparison with CHOLMOD with fixed time step

Table XIII shows the performance comparison between EPPCG and CHOLMOD with fixed time step. The power grid is divided into 3*3 partitions. CHOLMOD is faster than EPPCG on most of benchmarks since the solving process of CHOLMOD is faster than that of EPPCG. In fixed time step analysis, the conductance matrix remains the same and the factorization of both CHOLMOD and EPPCG only needs to be conducted once. The right-hand-side vector *I* is changed using the solution of previous time point. The

Table XIII. Parallel EPPCG vs CHOLMOD with constant time step

Circuit	# TimeSteps	CHOLMOD			EPPCG		
		factor	solve	overall	factor	solve	overall
ibmpg3t	1000	14.2	341.3	355.5	0.9	375.4	376.3
ibmpg4t	1000	26.0	552.6	578.6	1.8	635.5	637.3
ibmpg5t	1000	18.2	420.3	438.5	1.1	363.8	364.9
ibmpg6t	1000	19.0	735.8	754.8	1.2	753.2	754.4

aggregate time for the solve process and vector *I* update is much larger than the factorization time. Therefore, factorization does not dominate in the whole process of transient analysis. The solve time of CHOLMOD is less than that of EPPCG. This makes CHOLMOD faster than EPPCG in fixed time step case. The parallelized right-hand-side change in EPPCG helps to gain acceleration in this process. Thus the difference of overall time cost of EPPCG and CHOLMOD is not very large.

Table XIV. EPPCG with variable time step vs CHOLMOD with fixed time step

Circuit	CHOLMOD with constant time step				EPPCG with variable time step						Sp-imp
	# Time Steps	factor	solve	overall	#Time steps	EP-size	RL-size	factor	solve	overall	
ibmpg3t	1000	14.2	341.3	355.5	358	30	25	0.9	138.2	139.1	2.6
ibmpg4t	1000	26	552.6	578.6	274	30	25	1.8	177.6	179.4	3.2
ibmpg5t	1000	18.2	420.3	438.5	370	30	25	1.0	138.5	139.5	3.1
ibmpg6t	1000	19	735.8	754.8	363	75	60	1.2	301.7	302.9	2.5

IV.3.2 Comparison of EPPCG with variable time step and CHOLMOD with fixed time step

As mentioned in IV.2, transient analysis of power grids is often manipulated using direct solvers with fixed time step. In order to show demonstrate the strength of EPPCG, we developed variable time step control strategy in IV.2.2. Table XIV shows the performance comparison between EPPCG with variable time step and CHOLMOD with fixed time step. The power grid is divided into 3*3 partitions. In the table, EPPCG performs better than CHOLMOD. There are two factors that make EPPCG faster than

Table XV. EPPCG performance with different time step strategies

Circuit	EP-size	RL-size	EPPCG(fixed time step)		EPPCG(variable time step)		Sp-imp
			#Timesteps	T	#Timesteps	T	
ibmpg3t	30	25	1000	376.3	358	139.1	2.7
ibmpg4t	30	25	1000	637.3	274	179.4	3.6
ibmpg5t	30	25	1000	364.9	370	139.5	2.6
ibmpg6t	75	60	1000	754.4	363	302.9	2.5

CHOLMOD. First, the variable time step strategy reduces the number of time points. Second, the factorization of the EPPCG matrix is only computed once and the matrix factor is reused at each time point during the whole process. In the PCG process, solving the enlarged partition acts as a preconditioner which is a good approximation of the conductance matrix. The factorization of the preconditioner is conducted at the initial time step and stored for reuse. In the transient analysis, the factor does not need to be computed at each time point although the conductance matrix changes. This makes

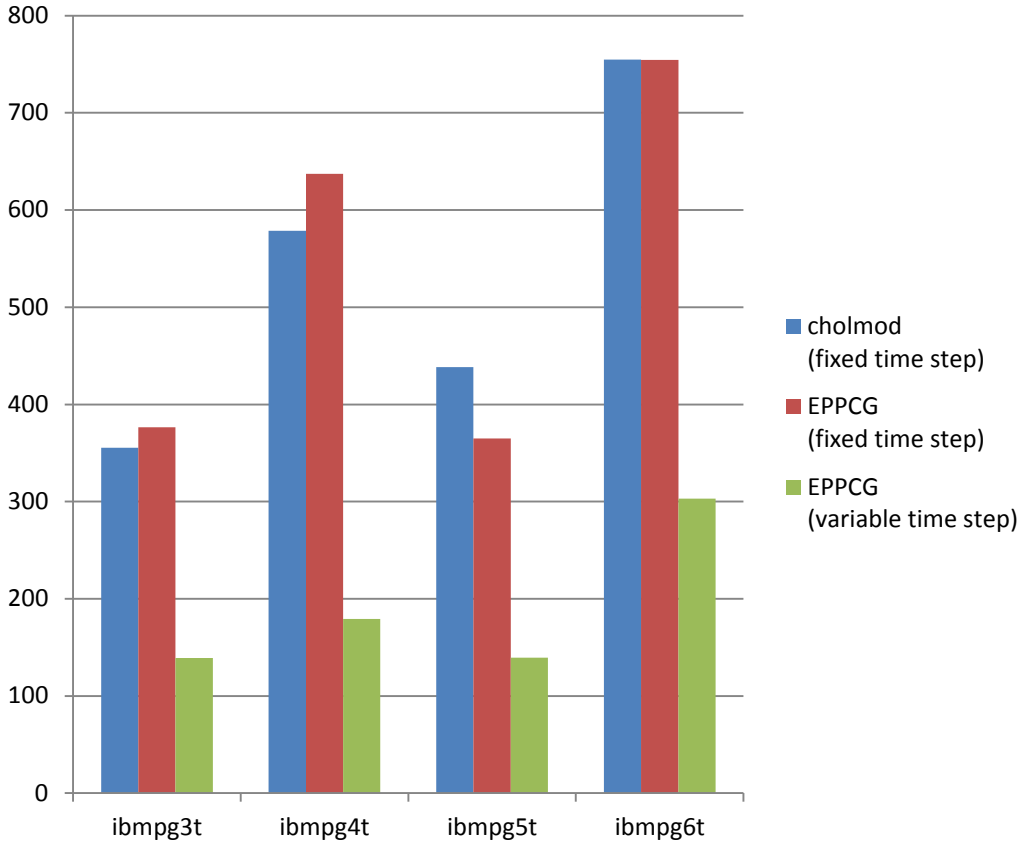

Figure 17. Performance comparison among CHOLMOD, EPPCG(fixed step) and EPPCG(variable step)

EPPCG perform well since the factorization is the most expensive process at each transient iteration.

IV.3.3 Comparison of EPPCG with fixed time step and EPPCG with variable time step

In order to investigate the performance of EPPCG in constant step analysis and variable step analysis, we present the results in Table XV and Figure 17. The EPPCG with variable time step benefits from reduced number of time points and reuse of matrix

Table XVI. MPI parallelization performance

Circuit	EP-size	RL-size	#steps	EPPCG(serial)	EPPCG(parallel)	speedup
				T	T	
ibmpg3t	30	25	358	987.6	139.1	7.1
ibmpg4t	30	25	274	1309.6	179.4	7.3
ibmpg5t	30	25	370	1046.3	139.5	7.5
ibmpg6t	75	60	363	2090.0	302.9	6.9

factors. The overall speed improvement obtained over fixed time step analysis ranges from 2.5X to 3.6X.

The comparison among CHOLMOD, EPPCG with fixed time step and EPPCG with variable time step is presented in Figure 17. From the figure, it can be seen that EPPCG with variable time step is the fastest.

IV.3.4 MPI-based parallel performance

In Table XVI, the MPI parallel performance is shown. The power grid is divided into 3*3 partitions and 9 MPI processes are created. The comparison between serial and parallel runtime with variable time step is presented. In our variable time step scheme, the matrix factors are reused at each time point. As a result, the solving process is much more costly than the matrix factorization. Therefore, the parallelization of solving process determines the performance of MPI implementation. In our experiments, the speedup obtained ranges from 6.9X-7.5X.

Table XVII. EPPCG performance with different number of partitions

Circuit	EP-size	RL-size	3*3 (S)	4*4 (S)	5*5 (S)
ibmpg3t	30	25	139.1	121.5	151.3
ibmpg4t	30	25	179.4	147.6	181.4
ibmpg5t	30	25	139.5	119.8	142.6
ibmpg6t	75	60	302.9	279.5	276.3

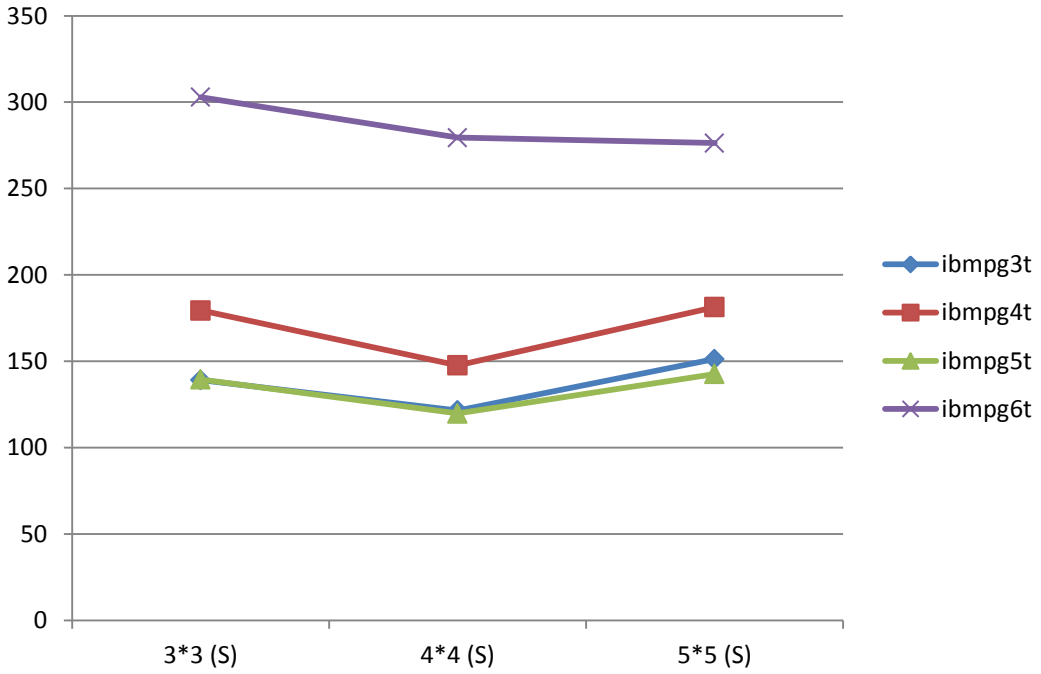

Figure 18. Transient performance with different partitioning scheme

IV.3.5 Impact of partitioning

In order to demonstrate the influence of different partitioning methods, Table XVII and Figure 18 are constructed. In table XVII, all the partitioning methods share the same EP-size and RL-size. The experiments were conducted using variable time step strategy. From the table, more partitions do not always imply higher speed. This is caused by MPI more expensive communication overheads when more partitions are created.

CHAPTER V

CONCLUSION

In this dissertation, a preconditioned iterative solver for large power grid simulation on multicore CPU-GPU platforms is presented. The algorithm divides the power grid into partitions and extends the partitions to obtain more accurate voltage solution. Breadth first search is employed to extend the partition. In order to further reduce the boundary solution error, Preconditioned Conjugate Gradient (PCG) method is adopted. Solving the enlarged partitions acts as an implicit preconditioner for PCG. We outlined a two-tier parallel scheme that utilizes MPI-GPU hybrid approach. In static analysis, the experimental results show that the implementation achieves high parallel efficiency using MPI on multicore-CPUs. GPU helps in accelerating the parallel implementation by an additional factor in the range of 1.4X-2.4X. The results also indicate that higher GPU speedup can be achieved by using larger benchmarks. The combination of an effective preconditioner and efficient parallelization allows the EPPCG to achieve speed improvements in the range of [61X-142X] over a state-of-the-art direct solver in static analysis and [2.5X-3.2X] in transient analysis. The main contribution of this work can be summarized as follows. Firstly, an efficient preconditioner is constructed such that the iterative solver converges very fast. Secondly, the preconditioning process that dominates the solver runtime can be fully parallelized. Thirdly, a breadth-first search method to enlarge the partitions is adopted to ensure the enlarged-partition used for preconditioning is light-weight.

In future research, the improvement of preconditioner can be considered. A potential improvement would utilize the positions of the C4 bumps. In detail, C4 bumps influence the nodes that are close to them more than the nodes that are far away. First, when growing the BFS tree, the paths that contain at least one C4 bump would be retained; the paths that contain no C4 bumps would be omitted. Second, the growth of BFS tree can stop when encountering a C4 bump for a path. This method will guarantee each path has only one C4 bump. For both methods above, the size of the conductance can be reduced significantly. In this case, the runtime of the factorization will be reduced. For transient analysis, a strategy can be designed to better exploit the advantage of EPPCG method. For fixed time-step settings, EPPCG does not have significant advantage over CHOLMOD. In varying time-step settings, CHOLMOD suffers from costly factorization at each time point. Multi-step methods can be used on EPPCG to obtain more accurate and faster implementations.

REFERENCES

[1] R. J. Baker, *CMOS Circuit Design, Layout, and Simulation*: Wiley-IEEE Press, 2010.

[2] F. Najm, *Circuit Simulation*: Wiley-IEEE Press, 2010.

[3] J. Darnauer, D. Chengson, B. Schmidt, E. Priest, D. A. Hanson, and W. G. Petefish, "Electrical evaluation of flip-chip package alternatives for next generation microprocessors," *Advanced Packaging, IEEE Transactions on,* vol. 22, pp. 407-415, 1999.

[4] Q. K. Zhu, *Power Distribution Network Design for VLSI*: Wiley-Interscience Press, 2004.

[5] S. Borkar, "Low power design challenges for the decade," in *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific*, pp. 293-296, 2001.

[6] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, "Reducing power in high-performance microprocessors," in *Design Automation Conference, 1998. Proceedings*, pp. 732-737, 1998.

[7] Y. Saad, "SPARSKIT: a basic tool kit for sparse matrix computations," University of Illinois, Urbana, IL, 1994.

[8] T. A. Davis and W. W. Hager, "Dynamic Supernodes in Sparse Cholesky Update/Downdate and Triangular Solves," *ACM Trans. Math. Softw.,* vol. 35, pp. 1-23, 2009.

[9] A. George, "Nested Dissection of a Regular Finite Element Mesh," *SIAM Journal on Numerical Analysis*, vol. 10, pp. 345-363, 1973.

[10] M. T. Heath, E. Ng, and B. W. Peyton, "Parallel Algorithms for Sparse Linear Systems," *SIAM Review*, vol. 33, pp. 420-460, 1991.

[11] H. A. van der Vorst, "Efficient and reliable iterative methods for linear systems," *Journal of Computational and Applied Mathematics*, vol. 149, pp. 251-265, 2002.

[12] Y. Saad, *Iterative Methods for Sparse Linear Systems*: Society for Industrial and Applied Mathematics, 2003.

[13] J. Demmel, *Applied Numerical Linear Algebra*: Society for Industrial and Applied Mathematics, 1997.

[14] *The Spice Page*. Available: http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/

[15] C. Chung-Han, T. Nien-Yu, Y. Hao, L. Che-Rung, S. Yiyu, and C. Shih-Chieh, "On the preconditioner of conjugate gradient method: A power grid simulation perspective," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pp. 494-497, 2011.

[16] J. N. Kozhaya, S. R. Nassif, and F. N. Najm, "A multigrid-like technique for power grid analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, pp. 1148-1160, 2002.

[17] Y. Jianlei, L. Zuowei, C. Yici, and Z. Qiang, "PowerRush: A linear simulator for power grid," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pp. 482-487, 2011.

[18] C. Tsung-Hao and C. C. Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods," in *Design Automation Conference, 2001. Proceedings*, pp. 559-562, 2001.

[19] B. T. Boghrati and S. S. Sapatnekar, "Incremental power network analysis using backward random walks," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pp. 41-46, 2012.

[20] B. T. Boghrati and S. Sapatnekar, "Incremental solution of power grids using random walks," in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pp. 757-762, 2010.

[21] Z. Xueqian, W. Jia, F. Zhuo, and H. Shiyan, "Power grid analysis with hierarchical support graphs," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pp. 543-547, 2011.

[22] Z. Yu and M. D. F. Wong, "Fast algorithms for IR drop analysis in large power grid," in *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, pp. 351-357, 2005.

[23] S. R. Nassif and J. N. Kozhaya, "Fast power grid simulation," in *Design Automation Conference, 2000. Proceedings 2000*, pp. 156-161, 2000.

[24] D. Wei and L. Peng, "Parallelizable stable explicit numerical integration for efficient circuit simulation," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pp. 382-385, 2009.

[25] W. Dong and P. Li, "Parallel circuit simulation with adaptively controlled projective integration," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, pp. 1-24, 2011.

[26] R. Achar, M. S. Nakhla, H. S. Dhindsa, A. R. Sridhar, D. Paul, and N. M. Nakhla, "Parallel and Scalable Transient Simulator for Power Grids via Waveform Relaxation (PTS-PWR)," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, pp. 319-332, 2011.

[27] W. Jia and X. Xuanxing, "Scalable power grid transient analysis via MOR-assisted time-domain simulations," in *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pp. 548-552, 2013.

[28] Y. Jianlei, L. Zuowei, C. Yici, and Z. Qiang, "PowerRush : Efficient transient simulation for power grid analysis," in *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, pp. 653-659, 2012.

[29] L. Yu-Min and C. C. P. Chem, "Power grid transient simulation in linear time based on transmission-line-modeling alternating-direction-implicit method," in *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*, pp. 75-80, 2001.

[30] Z. Pan, C. Yici, S. X. Tan, L. Zuying, and H. Xianlong, "Transient analysis of on-chip power distribution networks using equivalent circuit modeling," in

Quality Electronic Design, 2004. Proceedings. 5th International Symposium on, pp. 63-68, 2004.

[31] Y. Ting and M. D. F. Wong, "PGT_SOLVER: An efficient solver for power grid transient analysis," in *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, pp. 647-652, 2012.

[32] Y. Cai, J. Shi, Z. Pan, X. Hong, and S. X. D. Tan, "Large scale P/G grid transient simulation using hierarchical relaxed approach," *Integration, the VLSI Journal*, vol. 41, pp. 153-160, 2008.

[33] B. Barney. *MPI tutorial*. Available: https://computing.llnl.gov/tutorials/mpi/

[34] *The OpenMP API*. Available: http://openmp.org/wp/

[35] W. Enhua and L. Youquan, "Emerging technology about GPGPU," in *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, pp. 618-622, 2008.

[36] T. George, V. Saxena, A. Gupta, A. Singh, and A. R. Choudhury, "Multifrontal Factorization of Sparse SPD Matrices on GPUs," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pp. 372-383, 2011.

[37] C. D. Yu, W. Wang, and D. l. Pierce, "A CPU-GPU hybrid approach for the unsymmetric multifrontal method," *Parallel Comput.*, vol. 37, pp. 759-770, 2011.

[38] D. Yangdong, B. D. Wang, and M. Shuai, "Taming irregular EDA applications on GPUs," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pp. 539-546, 2009.

[39] R. O. Topaloglu and B. Gaster, "GPU programming for EDA with OpenCL," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pp. 63-66, 2011.

[40] S. Cauley, V. Balakrishnan, and K. Cheng-Kok, "A Parallel Direct Solver for the Simulation of Large-Scale Power/Ground Networks," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, pp. 636-641, 2010.

[41] Z. Yu and M. D. F. Wong, "Fast block-iterative domain decomposition algorithm for IR drop analysis in large power grid," in *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pp. 277-283, 2010.

[42] Z. Zeng, Z. Feng, P. Li, and V. Sarin, "Locality-Driven Parallel Static Analysis for Power Delivery Networks," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, pp. 1-17, 2011.

[43] S. Kai, Z. Quming, K. Mohanram, and D. C. Sorensen, "Parallel domain decomposition for simulation of large-scale power grids," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pp. 54-59, 2007.

[44] L. Zhang and V. Sarin, "An enlarged-partition based preconditioned iterative solver for parallel power grid simulation," in *Quality Electronic Design (ISQED), 2014 15th International Symposium on*, pp. 715-722, 2014.

[45] X. Xuanxing and W. Jia, "Parallel forward and back substitution for efficient power grid simulation," in *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, pp. 660-663, 2012.

[46] Q. Haifeng, S. R. Nassif, and S. S. Sapatnekar, "Power grid analysis using random walks," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 24, pp. 1204-1224, 2005.

[47] B. T. Boghrati and S. Sapatnekar, "A scaled random walk solver for fast power grid analysis," in *Design, Automation & Test in Europe Conference & Exhibition (DATE),* 2011, pp. 1-6, *2011*.

[48] S. T. Weikun Guo, Zuying Luo, Xialong Hong, "Partial random walks for transient analysis of large power distribution networks," *IEICE transactions on fundamentals of electronics, communications and computer science,* vol. 87, p. 8, 2004.

[49] Z. Min, R. V. Panda, S. S. Sapatnekar, T. Edwards, R. Chaudhry, and D. Blaauw, "Hierarchical analysis of power distribution networks," in *Design Automation Conference, 2000. Proceedings 2000*, pp. 150-155, 2000.

[50] Z. Min, R. V. Panda, S. S. Sapatnekar, and D. Blaauw, "Hierarchical analysis of power distribution networks," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, pp. 159-168, 2002.

[51] S. Jin, C. Yici, S. X. D. Tan, J. Fan, and H. Xianlong, "Pattern-Based Iterative Method for Extreme Large Power/Ground Analysis," *Computer-Aided Design of*

Integrated Circuits and Systems, IEEE Transactions on, vol. 26, pp. 680-692, 2007.

[52] S. Jin, C. Yici, H. Wenting, M. Liwei, S. X. Tan, H. Pei-Hsin, *et al.*, "GPU friendly Fast Poisson Solver for structured power grid network analysis," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pp. 178-183, 2009.

[53] K. Daloukas, N. Evmorfopoulos, P. Tsompanopoulou, and G. I. Stamoulis, "A 3-D Fast Transform-based preconditioner for large-scale power grid analysis on massively parallel architectures," in *Quality Electronic Design (ISQED), 2014 15th International Symposium on*, pp. 723-730, 2014.

[54] K. Daloukas, N. Evmorfopoulos, G. Drasidis, M. Tsiampas, P. Tsompanopoulou, and G. I. Stamoulis, "Fast Transform-based preconditioners for large-scale power grid analysis on massively parallel architectures," in *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, pp. 384-391, 2012.

[55] Z. Zhiyu, X. Tong, F. Zhuo, and L. Peng, "Fast static analysis of power grids: Algorithms and implementations," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pp. 488-493, 2011.

[56] A. Dalal, L. Lev, and S. Mitra, "Design of an efficient power distribution network for the UltraSPARC-I microprocessor," in *Computer Design: VLSI in Computers and Processors, 1995. ICCD '95. Proceedings., 1995 IEEE International Conference on*, pp. 118-123, 1995.

[57] J. Singh and S. S. Sapatnekar, "Partition-based algorithm for power grid design using locality," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 25, pp. 664-677, 2006.

[58] S. X. D. Tan, C. J. R. Shi, and L. Jyh-Chwen, "Reliability-constrained area optimization of VLSI power/ground networks via sequence of linear programmings," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 22, pp. 1678-1684, 2003.

[59] Z. Cheng, V. F. Pavlidis, and G. De Micheli, "Voltage propagation method for 3-D power grid analysis," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pp. 844-847, 2012.

[60] L. Pillage, *Electronic Circuit and System Simulation Methods (SRE)*: McGraw-Hill, Inc., 1999.

[61] D. F. Richard Burden, *Numerical Analysis, Fourth Edition*: PWS-KENT, 1988.

[62] E. Chiprout, "Fast flip-chip power grid analysis via locality and grid shells," in *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pp. 485-488, 2004.

[63] 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, pp. 1-14, 2008.

[64] *IBM P/G benchmarks*. Available: http://dropzone.tamu.edu/~pli/PGBench/

[65] S. R. Nassif, "Power grid analysis benchmarks," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, pp. 376-381, 2008.

[66] A. Vladimirescu, *The Spice Book*: John Wiley and Sons, Inc., 1994.

[67] T. Quarles, "The SPICE3 implementation guide," EECS Department, University of California, Berkeley, Berkeley, California, 1989.