Stochastic and Hybrid Linear Equation Solvers and their Applications in VLSI Design Automation

A Dissertation
Submitted to the Faculty of the Graduate School
of the University of Minnesota
By

Haifeng Qian

in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

Sachin S. Sapatnekar, Advisor

May 2006

Acknowledgments

Thanks above all to, and half of this thesis belongs to, my advisor, Professor Sachin S. Sapatnekar. Throughout the four years of doctoral study, his intelligence, knowledge, and most importantly his keen insight in identifying potential problem and potential breakthrough, have guided me every step of the way. It has been a pleasure and an honor working with him.

Thanks to Dr. Sani R. Nassif, who has been an industrial mentor to me. Through multiple collaborative projects and arranging an internship for me, he patiently helped me understand real-life problems, and inspired and contributed to a significant portion of this thesis work.

Thanks to Dr. Haihua Su and Dr. Gi-Joon Nam for the industrial benchmarks used in all experiments in this thesis. Thanks to Dr. Joseph N. Kozhaya for his contribution to the ESD chapter.

Thanks to my PhD committee members, Professor George Karypis, Professor Kia Bazargan, Professor Gerald E. Sobelman and Dr. Sani R. Nassif, for reviewing my thesis and giving valuable feedbacks.

Thanks to my colleagues in the VEDA lab, Shrirang Karandikar, Rupesh Shelar, Jaskirat Singh, Vidyasagar Nookala, Sanjay Kumar, Brent Goplen, Tianpei Zhang, Yong Zhan, Yan Feng, Hongliang Chang, and many others, for their help and friendship.

Abstract

This thesis presents two new linear equation solvers, and investigates their applications in VLSI design automation. Both solvers are derived in the context of a special class of large-scale sparse left-hand-side matrices that are commonly encountered in engineering applications, and techniques are presented that can potentially extend the theory to more general cases.

The first is a stochastic solver that performs the computation by establishing the equivalence between linear equations and random walks. It has a desirable locality feature: a single unknown variable can be evaluated without solving the entire system. For complete solutions, it is competitive in applications with moderate accuracy requirement.

The second is a hybrid solver: it combines the random walk technique and traditional iterative approaches. Given a set of linear equations, if the left-hand-side matrix satisfies certain conditions, it is proven that an incomplete triangular factorization can be obtained from random walks, and these factors can be used as a preconditioner in a traditional iterative linear equation solver to accelerate its convergence. In other words, the proposed hybrid solver is a stochastically preconditioned iterative solver. It is argued that our factor matrices have better quality, i.e., better accuracy-size tradeoffs, than preconditioners produced by existing incomplete factorization methods. Therefore the hybrid solver requires less computation than traditional preconditioned iterative solvers to solve a set of linear equations with the same error tolerance, and the advantage increases for larger and denser sets of linear equations.

The application of these solvers on several problems in VLSI design is illustrated in this thesis, namely, power grid analysis, chip-level electrostatic discharge (ESD) simulation, and quadratic placement. We not only demonstrate the efficiency of direct usage of the solvers, but also devise a set of application-specific techniques that are often based on indirect usage of the stochastic solver, due to the fact that the localized computation process of random walks carries meaningful information in various scenarios.

For power grid analysis, the locality feature of the stochastic solver enables the calculation of a single node voltage in DC analysis or a single node voltage at a single time point in transient analysis, which may potentially translate to dramatic efficiency improvements in an incremental design environment. Special variations of the stochastic solver, by introducing hierarchies that fit naturally with power grid structure, are also derived to speed up complete simulation. To handle early-stage analysis with uncertainty, the locality feature of random walks is utilized to relate a single node voltage to the working modes of circuit blocks, and thereby achieve a fast solution.

For ESD simulation, which requires solving a power grid for up to a few thousand times with different excitations, a flexible network reduction algorithm based on random walks is applied to reduce the computational complexity. The locality feature of random walks naturally leads to the sparsity of the reduced network, and enables incremental update when the design is modified.

For quadratic placement, the hybrid linear equation solver is applied in each iteration to compute new locations for circuit modules and cells. The hybrid solver is compared against traditional preconditioned iterative solvers by measuring the actual amount of computation needed to solve sets of linear equations with the same error tolerance, and exhibits significant speedup. A trend is observed that the larger and denser the left-hand-side matrix is, the more the hybrid solver outperforms traditional methods, as predicted by our theory.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis presents two new linear equation solvers. The problem statement is simply to solve a set of linear algebraic equations $A\mathbf{x} = \mathbf{b}$, where A is a given square nonsingular matrix, often referred to as the left-hand-side matrix, $\mathbf{b}$ is a given vector, often referred to as the right-hand-side vector, and $\mathbf{x}$ is the unknown solution vector to be computed. The first solver is a stochastic solver based on random walks. The second solver is a hybrid solver: it combines the random walk technique and traditional iterative approaches. Both solvers are presented and proven in the context of a special class of left-hand-side matrices, and techniques are provided that can potentially extend the theory to more general cases.

The solution of linear equations has been studied for hundreds of years, and the majority of the solvers fall into two main categories: direct solvers that are based on matrix factorization, and iterative solvers that begin with an initial guess and repeatedly refine the solution vector. Although the basic techniques in both categories can be dated back to the time of C. F. Gauss, the last few decades have seen significant progress in computational efficiency. For direct solvers, dramatic performance improvements have been achieved by matrix ordering techniques [2]

[15] [19] [24] [25]; for iterative solvers, powerful Krylov subspace methods [3] [70], as well as multigrid methods [29], have been developed.

These mathematical innovations have been driven by engineering applications. For example, partial differential equations are involved in many engineering problems, and their discretization is a major source of large-scale systems of linear equations that constantly demand more efficient linear solvers. In recent years, with the exponential growth in circuit complexity, VLSI design automation has emerged as another application domain for large-scale linear solvers. Efficient solutions to systems of linear equations are required at several steps of the design cycle, such as circuit simulation (especially power grid analysis), thermal analysis, and quadratic placement. All of these problems involve large-scale sparse left-hand-side matrices: the dimension of A can be millions or more, but the average number of non-zero entries per row is typically below a hundred. All discussion in this thesis is limited to such large-scale sparse matrices, and any reference to the density of a matrix implies a relative density metric defined as the average number of non-zero entries per row. When a matrix is said to be denser than another, such a statement refers to the relative density comparison between two matrices that are both sparse.

In VLSI design automation applications, it is often needed to solve $A\mathbf{x} = \mathbf{b}$ multiple times, with the same left-hand-side matrix A, but with different right-hand-side vectors $\mathbf{b}$. Such multiple solutions are referred to as re-solves.

The competition between direct solvers and iterative solvers is an open argument, and there is much disagreement over which solvers are suitable for what scenarios. However, Figure 1.1 is an attempt to discuss the relative strength and weakness of the solvers, and provides a rough guideline for the choice among them given different applications. The x-axis represents the density of the left-hand-side matrix A, and can be measured by the average number of non-zero entries per row.

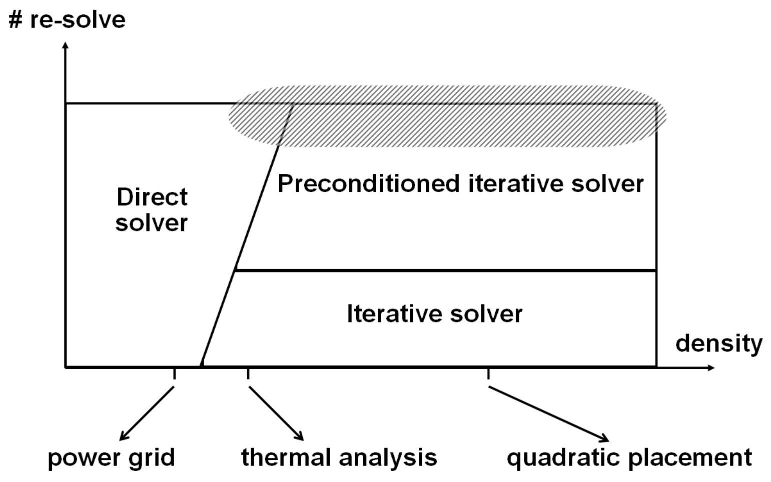


Figure 1.1: The choice of linear solver for different applications, as a function of the density of A and the number of re-solves.

Three applications in VLSI design automation are marked on the x-axis: the average number of non-zeroes per row in power grid matrices is typically around 4; for thermal matrices under finite difference discretization, this number is about 7; for placement matrices, this number ranges from 7 to 17 in our benchmarks. The y-axis is the number of re-solves needed: the number of times that the linear equations are to be solved with the same left-hand-side matrix A, but with different right-hand-side vectors $\mathbf{b}$. When the density of the matrix is low, direct solvers are the preferred method; when the density is above a certain point, iterative solvers become superior to direct solvers, due to the fact that the time and space complexity of a direct solver increases dramatically with the matrix density. The convergence of iterative solvers can be improved if the equation set is multiplied by a preconditioner matrix on both sides, at the overhead cost of building the preconditioner [70]. Such an extra computational cost can be justified when the system of equations is to be solved for multiple right-hand-side vectors, and therefore, when the number of re-solves goes above a certain point[1], preconditioned iterative solvers become the preferred choice: after paying the overhead of building a preconditioner, it can be used repeatedly for different right-hand-side vectors to achieve lower re-solve runtimes. The shaded region in Figure 1.1 represent applications that involve a relatively dense matrix A and require a relatively large number of re-solves. For these applications, it is worthwhile to build a more computationally expensive preconditioner with higher quality, as the overhead is easily amortized. In fact, the target applications of our proposed hybrid solver lie in the shaded region[2].

We now define some terms and notations that underpin the discussion throughout this thesis[3]. Let us first consider various factorizations of the left-hand-side matrix A:

- **LU factorization**: The equation $A = LU$ is referred to as the LU factorization of matrix A, where L is a lower triangular matrix with all diagonal entries being 1, and U is an upper triangular matrix. This specific form of LU factorization is unique when it exists, and is also known as Doolittle factorization in the literature, and in this thesis, any reference to LU factorization always implies Doolittle factorization.
- **Cholesky factorization**: If A is symmetric and positive definite, the equation $A = CC^{\mathrm{T}}$ is referred to as the Cholesky factorization of A, where C is a lower triangular matrix referred to as the *complete* Cholesky factor.

[1]One may argue that preconditioning is beneficial, and sometimes required, even for a single right-hand-side vector. For this reason, the region labeled "Iterative solver" in Figure 1.1 should be read not as iterative solvers without any preconditioning, but rather as ones with a simple preconditioner: for example, a diagonal preconditioner is commonly used in practice.

[2]The shaded region covers a corner of the direct solver domain because the hybrid solver may outperform direct solvers for those applications with marginal density.

[3]The readers are referred to [19] [25] [70] [83] for a more complete treatment of the basics.

- **LDL factorization**: A slight variation of Cholesky factorization, the equation $A = LDL^{\mathrm{T}}$ is referred to as the LDL factorization of A, where L is the same matrix L in the LU factorization, and D is a diagonal matrix with all positive diagonal values.
- **Incomplete Cholesky factor**: If A is symmetric and positive definite and C is its Cholesky factor, a lower triangular matrix B is referred to as an *incomplete* Cholesky factor of A if it satisfies the conditions that[4] $BB^{\mathrm{T}} \approx A$, and that if $B_{i,j} \neq 0$ then $C_{i,j} \neq 0$, $\forall i, j$.

There are simple relations among these factor matrices: $U = DL^{\mathrm{T}}$ and $C = L\sqrt{D}$, where $\sqrt{D}$ is a diagonal matrix in which each entry is the square root of the corresponding entry in D.

Definition 1 *Matrix A is referred to as an R-matrix if it satisfies the following four properties:*

1. $A_{i,i} > 0$, $\forall i$.
2. $A_{i,j} \leq 0$, $\forall i \neq j$.
3. $A_{i,j} = A_{j,i}$, $\forall i \neq j$.
4. *Matrix A is irreducibly diagonally dominant.*

To explain the last condition, let us define the correspondence between matrices and graphs. Given a symmetric matrix A of dimension N, let G be a undirected graph with N nodes labeled $1, 2, \cdots, N$, such that an undirected edge exists between two nodes $i \neq j$ if and only if $A_{i,j} \neq 0$. This graph G is referred to as the *matrix graph* of A, and each node in G corresponds to a specific row

[4]Here, $BB^{\mathrm{T}} \approx A$ in the numerical sense, for example, if matrix $\left(I - (BB^{\mathrm{T}})^{-1}A\right)$ has small absolute eigenvalues.

and a specific column in A. The i^{th} row of matrix A is said to be *diagonally dominant* if $|A_{i,i}| \geq \sum_{j \neq i} |A_{i,j}|$; it is said to be *strictly diagonally dominant* if $|A_{i,i}| > \sum_{j \neq i} |A_{i,j}|$. Matrix A is said to be *irreducibly diagonally dominant* if all rows are diagonally dominant, and in every connected component of the matrix graph of A, at least one node corresponds to a row that is strictly diagonally dominant. The first and fourth conditions together also imply that an R-matrix is positive definite [83]. Due to the third and fourth conditions, R-matrices form a subset of M-matrices[5] that are well studied in the literature [70] [83].

For clarity of the presentation, most of the discussion in this thesis is limited to R-matrices, with the exception that Section 3.5 is dedicated to extending the theory to more general matrix types.

Again, the shaded region in Figure 1.1 contains the challenging problems that are of interest. For such relatively dense R-matrices A, the most widely used linear equation solver is the Incomplete Cholesky factorization preconditioned Conjugate Gradient (ICCG) method [3] [42] [70], which uses $\left(BB^{\mathrm{T}}\right)^{-1}$ as the preconditioner, where B is an incomplete Cholesky factor of A. There are various existing techniques to produce B for ICCG. All of these perform Gaussian elimination on A, and use a specific strategy to drop insignificant entries during the process. The following are two widely used methods.

- ILU(0) applies a pattern-based strategy, and allows $B_{i,j} \neq 0$ only if $A_{i,j} \neq 0$ [70].
- ILUT applies a value-based strategy, and drops an entry from B if its value

[5]A nonsingular matrix is said to be an M-matrix if it satisfies the first two conditions in Definition 1, and all entries of its inverse matrix are nonnegative. It is provable based on the first, second and fourth conditions in Definition 1 that an R-matrix must be an M-matrix. An M-matrix is not necessarily an R-matrix, because it may not satisfy the third and fourth conditions in Definition 1.

is below a threshold, which is typically determined by multiplying the norm of the corresponding row of A by a small constant [70].

A more advanced strategy can be a combination of pattern, threshold and other size limits such as maximum number of entries per row. In later chapters, the proposed hybrid solver is compared against these traditional incomplete factorization methods using both theoretical argument and numerical tests.

The content of this thesis is organized as follows:

- Chapter 2 presents a stochastic linear equation solver. It is neither a direct solver nor an iterative solver, but belongs to an often-ignored third category that took root in [22] [87], and has a desirable locality feature: a single unknown variable can be evaluated without solving the entire system. We propose two new techniques that dramatically improve the performance, and properties of the resulting solver are investigated.
- Chapter 3 presents a hybrid linear equation solver, the centerpiece of this thesis. The solver is derived by combining the stochastic solver with traditional iterative techniques. A mathematical proof is presented that an incomplete LDL factorization of an R-matrix can be obtained by random walks, and used to precondition any iterative solver. It is argued that the resulting incomplete LDL factors have better quality, i.e., better accuracy-size tradeoffs, than the incomplete Cholesky factors produced by traditional methods, and that the improvement increases for larger and denser matrices. We also provide techniques that can potentially extend the theory to more general cases beyond R-matrices, and speculate on future challenges.
- Chapter 4 applies the stochastic solver on power grid analysis. The locality feature of the stochastic solver enables the calculation of a single node voltage in DC analysis or a single node voltage at a single time point in transient

analysis, which may potentially translate to dramatic efficiency improvement in an incremental design environment. Special variations of the stochastic solver are also derived, by introducing hierarchies that fit naturally with power grid structure, to speed up complete simulation. For example, DC analysis of a 71K-node power grid with C4 packaging takes 4.16 seconds; a 348K-node wire-bond DC power grid is solved in 93.64 seconds; RKC analysis of a 642K-node power grid takes 2.1 seconds per timestep.

- Chapter 5 applies the stochastic solver on early-stage power grid analysis. To handle uncertainty, an exact integer linear programming (ILP) method is first developed, and then the locality feature of random walks is utilized to relate a single node voltage to the working modes of circuit blocks, and thereby achieve an effective heuristic. A circuit of 43K nodes is analyzed within 70 seconds, and the worst-case scenarios found correlate well with the results from an ILP solver.
- Chapter 6 applies the stochastic solver on chip-level electrostatic discharge (ESD) simulation. A flexible network reduction algorithm based on random walks is applied to reduce the computational complexity. The locality feature of random walks naturally leads to the sparsity of the reduced network, and enables incremental update when the design is modified. A complete ESD check of a benchmark with a 2.3M-node VDD net and 1000 I/O pads is performed in 13 minutes, and 10 resimulations for incremental changes take a total of 9 minutes.
- Chapter 7 applies the hybrid solver on quadratic placement, which is an example application that lies in the shaded region in Figure 1.1. The hybrid solver computes new locations for circuit modules and cells in each placement iteration, and its performance is compared against both ICCG with ILU(0) and

ICCG with ILUT, by measuring the actual amount of computation needed to solve linear equations with the same error tolerance. The hybrid solver shows a speedup of up to 7.1 times over ICCG, and a trend is observed that the larger and denser the left-hand-side matrix is, the more the hybrid solver outperforms traditional methods, as predicted by our theory.

Chapter 2

Stochastic Linear Equation Solver

In this chapter, we study a stochastic solution to a set of linear equations, $A\mathbf{x} = \mathbf{b}$, given that A is an R-matrix. The basic framework is rooted in previous mathematical works of [22] [87], and we develop two new techniques that significantly improve the efficiency of a stochastic solver.

This chapter is organized as follows. Section 2.1 provides a summary of previous works that use random walks to solve systems of linear equations. Section 2.2 presents the generic algorithm of the stochastic solver, followed by a simple illustrative example in Section 2.3. Section 2.4 presents the two important new techniques, and Section 2.5 analyzes the complexity of the stochastic solver. Some implementation aspects of the stochastic solver are deferred to Section 3.4 in the next chapter.

2.1 A Brief History

A random walk is also known as a discrete-time discrete-state Markov chain, and is often viewed as a discrete abstraction of the physical phenomenon of Brownian motion. It forms one category of the general Monte Carlo methods for numerical computation. In this chapter, random walks are employed to solve systems of linear equations where the left-hand-side matrix A is an R-matrix. Historically, the theory was developed on two seemingly independent tracks, related to the analysis of potential theory [14] [33] [35] [43] [44] [58] and to the solution of systems of linear equations [22] [33] [78] [82] [87]. However, the two applications are closely related and research along each of these tracks has resulted in the development of analogous algorithms, some of which are equivalent. These mathematical works have found meaningful applications in electrical engineering [5] [7] [49] [69].

Along the first track, the goal has been to solve Laplace's equation in a closed region with given boundary values (i.e., under Dirichlet conditions), and it was proven that the potential value at a location can be estimated by observing a number of Brownian particles that start from this location and travel until they hit the boundary, and taking the average of the boundary values at the end points [14] [35] [44]. An important improvement was proposed in [58], which proved that, instead of simulating tiny movements of a Brownian particle, the particle can leap from a location to a random point on a sphere that is centered at this location, and that shapes other than a sphere can be used, given the corresponding Green function[1]. Another important development was [43], which extended the theory to solving Poisson's equation under Dirichlet conditions, as well as more general elliptic differential equations (under certain restrictions).

The second parallel track, which considered the solution of systems of linear equations, will be discussed in greater detail here, since it is directly related to this thesis. The first work that proposed a random-walk based linear equation solver is [22], although it was presented as a solitaire game of drawing balls from urns. It

[1]Many years later, this evolved to [49], a successful Monte Carlo algorithm in VLSI design automation.

was proven in [22] that, for any matrix A such that $\rho\,(I - A) < 1$, where ρ is the spectral radius[2] of a matrix, a game can be constructed and a random variable[3] X can be defined such that $E[X] = (A^{-1})_{ij}$, where $(A^{-1})_{ij}$ is an entry of the inverse matrix of A. In [22], the variable X is a "payment" when exiting the game. Under certain settings, the algorithm of [22] is equivalent to the "home" and "award" concepts in our theory, which is presented in the next section.

Two years later, the work in [87] continued this discussion in the formulation of random walks, and proposed the use of another random variable[4] W to replace X. A "mass" value was defined for every step in a walk, and W was defined as the total amount of "mass" carried through a walk. It was proven in [87] that $E[W] = E[X]$, and it was argued that, in certain special cases, W has a lower variance than X, and hence is likely to converge faster. Under certain settings, the algorithm of [87] is equivalent to the "motel" concept in the next section.

Both [22] and [87] have the advantage of being able to compute part of an inverse matrix without solving the whole system, in other words, localizing computation. Over the years, various descendant stochastic solvers have been developed [33] [78] [82], though some of them, e.g., [78] [82], do not have the property of localizing computation.

From a different perspective, the work in [18] aimed at investigating random walks by using electrics. It drew a parallel between resistive networks and random walks, and interpreted the relationship between conductances and probabilities. With underlying rules similar to [22], [18] proved many insightful conclusions linking

[2]The spectral radius of a matrix is defined as its maximum absolute eigenvalue. For example, here $\rho\,(I - A) = \max_l |\lambda_l\,(I - A)|$, where λ_l denotes the l^{th} eigenvalue of a matrix.

[3]The notations are different from the original ones used in [22].

[4]The notations are different from the original ones used in [87].

statistics and electrics, and inspired much of the work in Chapter 4.

Two other important works on this topic, [30] and [55], are deferred to Section 3.1.1, where they are reviewed in a more relevant context.

The basic framework of this chapter, to be presented in the next section, is mathematically a combination of [22] and [87], and it inherits the property of localizing computation. Not surprisingly, in potential theory, there is a method that can be viewed as roughly parallel to our basic framework: the counterpart is [43]. Beyond these legacies, we present two important efficiency-improving techniques in Section 2.4, which are not seen in previous works, and which not only play a crucial role in solving engineering problems in Chapters 4, 5 and 6, but also form the foundation of the hybrid solver in Chapter 3.

2.2 The Generic Algorithm

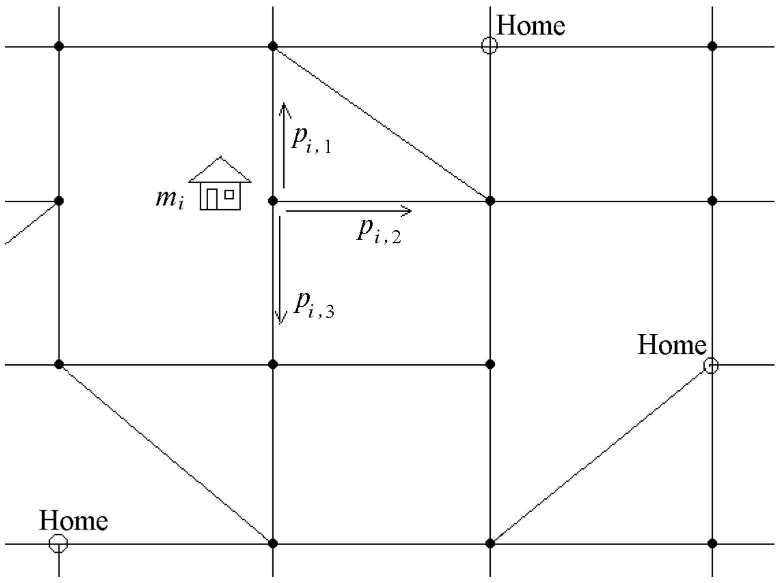


Figure 2.1: An instance of a random walk "game."

Let us consider a random walk "game" defined on a finite undirected connected graph representing a street map, for example, Figure 2.1. A walker starts from one of the nodes, and every day, he/she goes to an adjacent node l with probability $p_{i,l}$ for $l = 1, 2, \cdots, \text{degree}(i)$, where i is the current node, degree(i) is the number of edges connected to node i, and the adjacent nodes are labeled $1, 2, \cdots$ degree(i). The transition probabilities satisfy the following relationship:

$$\sum_{l=1}^{\text{degree}(i)} p_{i,l} = 1 \tag{2.1}$$

The walker pays an amount m_i to a motel for lodging everyday, until he/she reaches one of the homes, which are a subset of the nodes. Note that the motel price m_i is a function of his/her current location, node i. The game ends when the walker reaches a home node: he/she stays there and gets awarded a certain amount of money, m_0. We now consider the problem of calculating the expected amount of money that the walker has accumulated at the end of the walk, as a function of the starting node, assuming he/she starts with nothing.

The gain function for the walk is therefore defined as

$$f(i) = E[\text{total money earned } |\text{walk starts at node } i] \tag{2.2}$$

It is obvious that

$$f(\text{one of the homes}) = m_0 \tag{2.3}$$

For a non-home node i, again assuming that the nodes adjacent to i are labeled 1, 2, $\cdots$ degree(i), the f variables satisfy

$$f(i) = \sum_{l=1}^{\text{degree}(i)} p_{i,l} f(l) - m_i \tag{2.4}$$

For a random walk game with N non-home nodes, there are N linear equations similar to the one above, and the solution to this set of equations will give the exact values of f at all nodes.

In the above equations obtained from a random walk game, the set of allowable left-hand-side matrices is a superset of the set of R-matrices[5]. In other words, given a set of linear equations $A\mathbf{x} = \mathbf{b}$, where A is an R-matrix, we can always construct a random walk game that is mathematically equivalent, i.e., such that the f values are the desired solution $\mathbf{x}$. To do so, we divide the i^{th} equation by $A_{i,i}$ to obtain

$$x_i + \sum_{j \neq i} \frac{A_{i,j}}{A_{i,i}} x_j = \frac{b_i}{A_{i,i}} \tag{2.5}$$

$$x_i = \sum_{j \neq i} \left(-\frac{A_{i,j}}{A_{i,i}} \right) x_j + \frac{b_i}{A_{i,i}} \tag{2.6}$$

Equation (2.4) and equation (2.6) have seemingly parallel structures. Let N be the dimension of matrix A, and let us construct a random walk game with N non-home nodes, which are labeled $1, 2, \cdots, N$. Due to the properties of an R-matrix, we have

- $\left(-\frac{A_{i,j}}{A_{i,i}}\right)$ is a non-negative value and can be interpreted as the transition probability of going from node i to node j.
- $\left(-\frac{b_i}{A_{i,i}}\right)$ can be interpreted as the motel price m_i at node i.

However, the above mapping is insufficient due to the fact that condition (2.1) may be broken: the sum of the $\left(-\frac{A_{i,j}}{A_{i,i}}\right)$ coefficients is not necessarily one. In fact, because all rows of matrix A are diagonally dominant, the sum of the $\left(-\frac{A_{i,j}}{A_{i,i}}\right)$ coefficients is always less than or equal to one. Condition (2.1) can be satisfied if we add an extra transition probability of going from node i to a home node, by rewriting equation (2.6) as the following.

$$\begin{aligned} x_i &= \sum_{j \neq i} \left(-\frac{A_{i,j}}{A_{i,i}} \right) x_j + \frac{\sum_{\forall j} A_{i,j}}{A_{i,i}} \cdot m_0 + \frac{b'_i}{A_{i,i}} \\ \text{where } b'_i &= b_i - \sum_{\forall j} A_{i,j} \cdot m_0 \end{aligned} \tag{2.7}$$

[5]It is a superset of R-matrices because a left-hand-side matrix from a random walk game is not necessarily symmetric.

It is easy to verify that $\frac{\sum_{\forall j} A_{i,j}}{A_{i,i}}$ is a non-negative value for an R-matrix, and that the following mapping establishes the equivalence between equation (2.4) and equation (2.7), while satisfying (2.1) and (2.3).

- $\left(-\frac{A_{i,j}}{A_{i,i}}\right)$ is the transition probability of going from node i to node j.
- $\frac{\sum_{\forall j} A_{i,j}}{A_{i,i}}$ is the transition probability of going from node i to a home node with award m_0.
- $\left(-\frac{b'_i}{A_{i,i}}\right)$ is the motel price m_i at node i.

The choice of m_0 is arbitrary because b'_i always compensates for the m_0 term in equation (2.7), and in fact m_0 can take different values in (2.7) for different rows i. Therefore the mapping from an equation set to a game is not unique. A simple scheme can be to let $m_0 = 0$, and then $m_i = -\frac{b_i}{A_{i,i}}$.

It is worth pointing out that, if the home nodes are removed from Figure 2.1, the remaining graph is the matrix graph of A, regardless of the choice of award values. The set of nodes that are adjacent to the home nodes correspond to the set of rows in A that are strictly diagonally dominant. The fourth property of an R-matrix from Definition 1 guarantees that each node has at least one path to a home node. In this thesis, a home node is also referred to as a *terminal* or an *absorbing* node, and these three terms are used interchangeably.

To find x_i, the i^{th} entry of solution vector $\mathbf{x}$, a natural way is to simulate a certain number of random walks from node i and use the average monetary gain in these walks as the approximated entry value. If this amount is averaged over a sufficiently large number of walks by playing the "game" a sufficiently large number of times, then by the Law of Large Numbers [88], an acceptably accurate solution can be obtained. This is the idea behind the proposed *generic algorithm* that forms the most basic implementation.

According to the Central Limit Theorem [88], the estimation error of the above procedure is asymptotically a zero-mean Gaussian variable with variance inversely proportional to M, where M is the number of walks. Thus there is an accuracy-runtime tradeoff. In implementation, instead of fixing M, one may employ a stopping criterion driven by a user-specified error margin[6] Δ and confidence level[7] α:

$$P[-\Delta < x'_i - x_i < \Delta] > \alpha \tag{2.8}$$

where x'_i is the estimated i^{th} solution entry from M walks. If the standard deviation of the M sample walk-results is σ, the above criterion can be approximately converted to the following inequality that can be used in practice.

$$\begin{aligned} Q\left(\frac{\Delta\sqrt{M}}{\sigma}\right) &< \frac{1-\alpha}{2} \\ \frac{\sigma^2}{M} &< \left(\frac{\Delta}{Q^{-1}\left(\frac{1-\alpha}{2}\right)}\right)^2 \end{aligned} \tag{2.9}$$

where Q is the standard normal complementary cumulative distribution function, defined as

$$Q(\xi) = \frac{1}{\sqrt{2\pi}} \int_{\xi}^{\infty} e^{-\frac{u^2}{2}} du$$

According to condition (2.9), M is decided dynamically[8], and has different values for different nodes. It is worth noting that for each node, for a fixed confidence level, $M \propto \frac{1}{\Delta^2}$.

A desirable feature of the proposed algorithm is that it localizes the computation, i.e., it can calculate a single entry in vector $\mathbf{x}$ without having to solve the

[6]Here the error margin is defined on absolute error. A similar formulation can be derived for relative error.

[7]A typical confidence level can be, for example, $\alpha = 99\%$.

[8]Because of the fact that the estimation error has a precise Gaussian distribution only when M goes to infinity, and that σ is an estimate of its actual standard deviation, condition (2.9) is only an approximation to condition (2.8). To ensure accuracy, it is necessary to impose a lower bound on M, e.g. 20 walks.

entire set of equations; or in general, we only need to compute entries that are of interest. Such a locality property is meaningful in certain applications, and in fact Chapters 4, 5 and 6 all take advantage of the localized computation in different ways.

2.3 A Simple Example

In order to illustrate the procedure of the generic algorithm from the previous section, let us consider the following equation set, where the left-hand-side matrix is an R-matrix, and the exact solution is $\mathbf{x} = [0.6, 0.8, 0.7, 0.9]^{\mathrm{T}}$.

$$\begin{bmatrix} 1.5 & 0 & -1 & 0 \\ 0 & 2 & -1 & 0 \\ -1 & -1 & 2.25 & -0.25 \\ 0 & 0 & -0.25 & 1.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.9 \\ -0.05 \\ 0.95 \end{bmatrix}$$

Applying the mapping in equation (2.7) to this equation set, an equivalent random walk game is constructed and shown in Figure 2.2, where numbers inside circles represent motel prices and home awards, and numbers beside the arrows represent the transition probabilities from each node to a neighboring node. Note that the mapping from an equation set to a game is not unique: in this case, the award value m_0 is arbitrarily chosen to be 1; other choices may lead to different game settings.

To find out the first entry x_1, the walker starts the game at node 1 with zero balance. He/she pays the motel price of \$0.2, then either goes upwards with probability 0.33 to the terminal and end this walk, or goes downwards with probability 0.67 to node 3, then pays \$0.022, and continues from there. A random walk could be very short (for example, the walker may directly go up and end up with \$0.8), or very long (it may keep going back and forth between the four nodes, and the

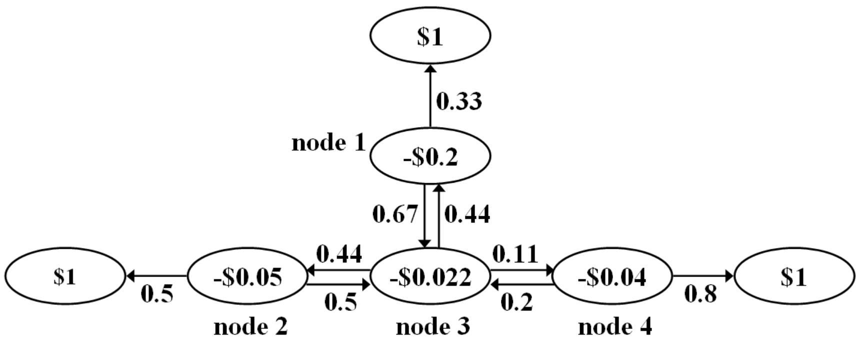


Figure 2.2: A random walk game equivalent to the example equation set.

walker could end up with very little money), although the probability of a very long walk is low. We perform M such walks, take the average of the M results, discard the dollar unit, and obtain the estimated x_1.

Table 2.1: Observed convergence behavior of solving the simple example. Δ is the error margin in equation (2.8), x_1' is the estimated value, and M is the actual number of walks used. Five tests use different seeds for the random number generator.

Δ	Test run #1		Test run #2		Test run #3		Test run #4		Test run #5	
	x_1'	M	x_1'	M	x_1'	M	x_1'	M	x_1'	M
0.05	0.6097	174	0.6067	156	0.5803	184	0.6418	103	0.6241	117
0.02	0.6087	1150	0.6075	946	0.5979	1140	0.5837	1254	0.6084	1232
0.01	0.6034	4562	0.6013	4664	0.6043	4315	0.5982	4441	0.6016	4619

Table 2.1 shows how the estimated voltage converges to the exact value of 0.6. The five columns in the table represent five different runs of the proposed algorithm, corresponding to different seeds for the random number generator.

2.4 Two Speedup Techniques

The basic framework described in the previous two sections has existed in the literature for over fifty years, yet stochastic linear equation solvers have largely failed to make any significant impact on mainstream applications.

In this section, we propose two new techniques that dramatically improve the performance of the stochastic solver. They have far-reaching significance and play a crucial role in later chapters.

2.4.1 Creating Homes

As discussed in the previous two sections, a single entry in the solution vector $\mathbf{x}$ can be evaluated by running random walks from its corresponding node in the game. To find the complete solution $\mathbf{x}$, a straightforward way is to repeat such procedure for every entry. This, however, is not the most efficient approach, since much information can be shared between random walks.

We propose a speedup technique by adding the following rule: after the computation of x_i is finished according to criterion (2.9), node i becomes a new home node in the game with an award amount equal to the estimated value x'_i. In other words, any later random walk that reaches node i terminates, and is rewarded a money amount equal to the assigned x'_i. Without loss of generality, suppose the nodes are processed in the natural ordering $1, 2, \cdots, N$, then for walks starting from node k, the node set $\{1, 2, \cdots, k-1\}$ are homes where the walks terminate (in addition to the original homes generated from the strictly-diagonally-dominant rows of A), while the node set $\{k, k+1, \cdots, N\}$ are motels where the walks pass by.

One way to interpret this technique is by the following observation about equation (2.4): there is no distinction between the neighboring nodes that are homes and the neighboring nodes that are motels, and the only reason that a random walk can terminate at a home node is that its f value is known and is equal to the award. In fact, any node can be converted to a home node if we know its f value and assign the award accordingly. Our new rule is simply utilizing the estimated $x'_i \approx x_i$ in such a conversion.

Another way to interpret this technique is by looking at the source of the value x'_i. Each walk that ends at a new home and obtains such an award is equivalent to an average of multiple walks, each of which continues walking from there according to the original game settings.

With this new method, as the computation for the complete solution $\mathbf{x}$ proceeds, more and more new home nodes are created in the game. This speeds up the algorithm dramatically, as walks from later nodes are carried out in a game with a larger and larger number of homes, and the average number of steps in each walk is reduced. At the same time, this method helps convergence without increasing M, because, as mentioned earlier, each walk becomes the average of multiple walks. The only cost[9] is that the game becomes slightly biased when a new home node is created, due to the fact that the assigned award value is only an estimate, e.g. $x'_i \neq x_i$; overall, the benefit of this technique dominates its cost.

Due to this speedup technique, the nodes computed early in the algorithm and those computed late are treated differently. Therefore, the ordering of nodes could potentially affect the performance. Random ordering is used in our implementation, because as computation proceeds, the density of home nodes is increased evenly throughout the game graph, and the performance of the algorithm is stable. Further discussion on ordering is presented in Section 3.4.4.

[9]The cost discussed here is in the context of the stochastic solver only. For the hybrid solver in Chapter 3, this will no longer be an issue.

2.4.2 Bookkeeping

For the same left-hand-side matrix A, traditional direct linear equation solvers are efficient in computing solutions for multiple right-hand-side vectors after initial matrix factorization, since only a forward/backward substitution step is required for each additional solve. Analogous to a direct solver, we propose a speedup mechanism for the stochastic linear equation solver.

The mechanism is a bookkeeping technique based on the following observation. In the procedure of constructing a random walk game discussed in Section 2.2, the topology of the game and the transition probabilities are solely determined by matrix A, and hence do not change when the right-hand-side vector $\mathbf{b}$ changes. Only motel prices and award values in the game are linked to $\mathbf{b}$.

When solving a set of linear equations with matrix A for the first time, we create a journey record for every node in the game. The following information is listed in the journey record.

- For any node i, record the number of walks performed from node i.
- For any node i and any motel node[10] j, record the number of times that walks from node i visit node j.
- For any node i and any home node[11] j, which can be either an initial home node in the original game or a new home node created by the technique from Section 2.4.1, record the number of walks that start from i and end at j.

Then, if the right-hand-side vector $\mathbf{b}$ changes while the left-hand-side matrix A remains the same, we do not need to perform random walks again. Instead, we

[10]The journey record is stored in a sparse fashion, and a motel j is included only if walks from node i visit j at least once.

[11]A home node j is included in the journey record only if at least one walk from node i ends at home j.

simply use the journey record repeatedly and assume that the walker takes the same routes, gets awards at the same locations, pays for the same motels, and only the award amounts and motel prices have been modified. Thus, after a journey record is created, new solutions can be computed by some multiplications and additions efficiently.

Practically, this bookkeeping technique is only feasible after the speedup technique from Section 2.4.1 is in use, for otherwise the space complexity can be prohibitive for a large matrix. When both techniques work together, empirically and for certain accuracy level, the space complexity of bookkeeping is less than the space complexity of a traditional direct solver, and is approximately linear in the matrix dimension[12].

In the next chapter, this bookkeeping technique serves as an important basis of the hybrid linear equation solver. There the bookkeeping scheme itself is modified in such a way that a rigorous proof is presented in Section 3.2.2 showing the fact that the space complexity of the modified bookkeeping is upper-bounded by the space complexity of the matrix factorization in a direct solver.

2.5 Runtime Trends

In this section, we argue[13] that the average-case or typical runtime of the stochastic linear equation solver is linear in the dimension of the matrix, given the same required error margin Δ and confidence level α, for matrices A of similar structures and right-hand-side vectors $\mathbf{b}$ of similar magnitude. Artificial test cases are used

[12]Actual memory consumption numbers of using this in power grid simulation are listed in Chapter 4 to validate this claim.

[13]This is a qualitative argument, not a rigorous proof, and only applies under the notion of similar matrix structure.

to validate this argument.

First, let us define the notion of "similar structure" for R-matrices: this implies that the average number of non-zero entries per row is similar, that the average value of diagonal entries is of similar magnitude, that the percentage of strictly-diagonally-dominant rows is similar, and that the average value of $\frac{A_{i,i}+\sum_{j\neq i} A_{i,j}}{A_{i,i}}$ over strictly-diagonally-dominant row i's is similar. As examples of matrices of similar structures, four artificial matrices are constructed such that they can all be mapped to a random walk game where the street map is a two-dimensional rectangular grid as follows: the four grid sizes are 50×50, 100×100, 500×500 and 1000×1000 respectively; nodes at the intersections of the $(25+50k)^{\text{th}}$ horizontal grid lines and the $(25+50l)^{\text{th}}$ vertical grid lines are home nodes, where k and l are nonnegative integers. Note that home nodes do not have corresponding rows or columns in matrix A. Let all diagonal entries of the four A matrices be 4, and let all non-zero off-diagonal entries be -1. It can be verified that these four matrices are all R-matrices, and that the rows that are strictly diagonally dominant correspond to the immediate neighbors of home nodes, and for such a row i, $\frac{A_{i,i}+\sum_{j\neq i} A_{i,j}}{A_{i,i}}$ is always 0.25.

The notion of "similar magnitude" for vectors is straightforward. For example, four right-hand-side vectors $\mathbf{b}$ can be defined for the above four matrices, such that the length of each vector is equal to the dimension of the corresponding matrix, and that all entries are the same value 0.0005.

Let us use the above-defined four pairs of left-hand-side matrices A and right-hand-side vectors $\mathbf{b}$ to show numerical results first, and then look at the reason behind the linear complexity. In all four cases, the value range of the entries in the exact solution $\mathbf{x}$ is roughly the same interval $(0, 1)$. They are all solved by the stochastic solver using a random ordering of nodes, and with the same error margin

$\Delta = 0.05$ and the same confidence level $\alpha = 99\%$; in other words, the accuracy requirement is $P[-0.05 < \text{error} < 0.05] > 99\%$. The metric of complexity is the total number of steps over all the random walks, where each step corresponds to one random-number generation, a few logic operations and one addition. Table 2.2 lists the total step numbers of solving the four test cases, and shows a sublinear complexity for small sizes, and more strictly linear complexity for larger sizes.

Table 2.2: The time complexity of the stochastic solver on artificial test cases.

Test case	Dimension	Total step number
#1	2.5e3	2.4e7
#2	1.0e4	7.5e7
#3	2.5e5	1.7e9
#4	1.0e6	6.4e9

The roughly linear complexity is due to the fact that the average amount of computation per node is independent of the graph size. Let us consider two games with node counts N_1 and N_2, which are each solved using a random ordering of nodes. Recall that due to the speedup technique in Section 2.4.1, solved nodes become home nodes. At the same stage of the computation, for example, when $5\%N_1$ nodes are solved in the first game and $5\%N_2$ nodes are solved in the second game, because the densities of "homes" are the same (roughly 5%) in both games, the average lengths of walks are the same in both games. Since the vectors $\mathbf{b}$ are of similar magnitude, and hence motel prices in both games are of similar magnitude, the typical σ values in equation (2.9) are also similar in both games. Thus, according to the stopping criterion (2.9), the typical M values are similar in both games at this stage, i.e., 5% complete. Therefore, the amount of computation for the $(5\%N_1 + 1)^{\text{th}}$ node in the first game, is close to the amount of computation for the $(5\%N_2 + 1)^{\text{th}}$ node in the second game, in the average case, and this is true for other percentage values as well. Therefore, the overall average amount of computation per node, which is equal to the average over all percentages, should be the same value for both games, and independent of their different graph sizes. Therefore, for matrices with similar structures, the complexity of the stochastic solver is linear in the matrix dimension.

Chapter 3

Hybrid Linear Equation Solver

The stochastic linear equation solver presented in the previous chapter, with the two new speedup techniques, is useful in applications that require localized or incremental solution of systems of linear equations, or when a complete solution is needed with moderate accuracy requirements. Chapters 4, 5 and 6 demonstrate its efficiency in several applications in VLSI design automation.

For a complete solution under high accuracy requirements, however, the stochastic solver retains an inherited weakness: according to equation (2.9), $M \propto \frac{1}{\Delta^2}$ for a fixed confidence level. For example, if we halve the error margin Δ, the time complexity of the stochastic solve increases by a factor of four. Such scaling behavior renders the stochastic solver inefficient when high accuracy is required for the complete solution vector. Empirically, we have found that 3% relative error needs to be tolerated in order for the stochastic solver to be faster than a state-of-the-art direct or iterative solver in an initial complete solve, or to be faster than a direct solver in a complete re-solve.

In this chapter, this accuracy restriction is eliminated by exploring the full potential of the two speedup techniques from the previous chapter. We prove that for an R-matrix A, an incomplete LDL factorization can be obtained from random walks, and used as a preconditioner for an iterative solver, e.g., conjugate gradient. Since the eventual proposed solver is a stochastically preconditioned iterative solver, which combines two traditionally separate categories of linear solvers, we call it a hybrid solver.

This chapter is organized as follows. Section 3.1 presents an initial prototype hybrid solver. Section 3.2 provides a mathematical proof that an incomplete LDL factorization of an R-matrix is formed by random walks. Section 3.3 summarizes the procedure of the final hybrid solver for R-matrices, and argues that its performance is superior to traditional ICCG. Section 3.4 discusses several implementation aspects of the hybrid solver. Section 3.5 extends the theory to more general matrices beyond R-matrices.

3.1 An Initial Hybrid Solver

This section is a first attempt at combining stochastic and iterative techniques in a linear equation solver. The presented prototype algorithm is different from the final hybrid solver, and serves as a stepping-stone: it helps to relate to two existing works in literature, and helps to explain the final theory from the perspective of a stochastic solver.

3.1.1 The Sequential Monte Carlo Method

One basis of the proposed prototype algorithm is the sequential Monte Carlo method, which was initiated in [55] and developed in [30] [32], and is a remedy that improves the accuracy of a stochastic solver.

Let $\mathbf{x}'$ be an approximate solution to $A\mathbf{x} = \mathbf{b}$ found by a stochastic solver, such as the one described in the previous chapter. Define the residual vector:

$$\mathbf{r} = \mathbf{b} - A\mathbf{x}' \tag{3.1}$$

Define the error vector:

$$\mathbf{z} = \mathbf{x} - \mathbf{x}' \tag{3.2}$$

For the system $A\mathbf{x} = \mathbf{b}$, it is easy to verify the following relation.

$$A\mathbf{z} = \mathbf{r} \tag{3.3}$$

The idea of the sequential Monte Carlo method is to iteratively solve (3.3) using the stochastic solver. In each iteration, an approximate error vector $\mathbf{z}$ is computed and then used to correct the current solution $\mathbf{x}'$. Algorithm 1 shows the pseudocode[1], and the process ends when the desired accuracy is achieved.

Algorithm 1 *Sequential Monte Carlo algorithm:*

```
Stochastic solve $A\mathbf{x} = \mathbf{b}$, find $\mathbf{x^0}$;
For $j = 0, 1, 2, \cdots$, until convergence, do {
    $\mathbf{r^j} = \mathbf{b} - A\mathbf{x^j}$;
    Stochastic solve $A\mathbf{z} = \mathbf{r^j}$, find $\mathbf{z^j}$;
    $\mathbf{x^{j+1}} = \mathbf{x^j} + \mathbf{z^j}$;
}
```

3.1.2 A Prototype Solver

Although the sequential Monte Carlo method has existed for over forty years, it has not resulted in any powerful solver that can compete with direct and iterative

[1]In both Algorithm 1 and Algorithm 2, superscripts are used to signify indices of iterations.

solvers, due to the fact that random walks are needed in every iteration, resulting in a relatively high overall time complexity. By integrating the two new speedup techniques from Section 2.4, we can finally build a stochastic solver with significant computational efficiency, and it is referred to as a *prototype hybrid solver*.

The algorithm is based on the observation that, in Algorithm 1, the initial solution to $A\mathbf{x} = \mathbf{b}$ and the subsequent solutions to the equations $A\mathbf{z} = \mathbf{r^j}$ deal with the same left-hand-side matrix A, but with multiple right-hand-side vectors. Therefore, according to the bookkeeping technique from Section 2.4.2, random walks are only needed in the initial step of Algorithm 1, with a journey record created at the same time, and the journey record can be used to perform computation inside the iterations without running a single extra walk. This results in the solver described in the pseudocode in Algorithm 2. Note that this is not yet the eventual hybrid solver that we propose, and is only an initial prototype algorithm.

Algorithm 2 *An initial hybrid algorithm:*

```
Solve $A\mathbf{x} = \mathbf{b}$, find $\mathbf{x^0}$, create journey record;
For $j = 0, 1, 2, \cdots$, until convergence, do {
  $\mathbf{r^j} = \mathbf{b} - A\mathbf{x^j}$;
  Apply record on $A\mathbf{z} = \mathbf{r^j}$, find $\mathbf{z^j}$;
  $\mathbf{x^{j+1}} = \mathbf{x^j} + \mathbf{z^j}$;
}
```

A precise representation of the journey record is investigated in the next section; at this time, it suffices to note that the calculations that are used to apply the journey record are purely linear operations. Therefore, for the approximate solution of $A\mathbf{z} = \mathbf{r^j}$, the overall effect of applying the journey record can be written as a matrix-vector multiplication in the form of the following equation.

$$\mathbf{z^j} = T\mathbf{r^j} \tag{3.4}$$

where T is a square matrix that represents the process of applying the record of random walks. The matrix T has a special structure to be discussed in the next section, and the above equation is used here for clarity of presentation. Note that T is built by an approximate solution of $A\mathbf{x} = \mathbf{b}$, and thus $T \approx A^{-1}$. The computation during one iteration of Algorithm 2 can be represented as follows.

$$\begin{aligned} \mathbf{x^{j+1}} &= \mathbf{x^j} + T\mathbf{r^j} \\ &= \mathbf{x^j} + T\left(\mathbf{b} - A\mathbf{x^j}\right) \\ &= \left(I - TA\right)\mathbf{x^j} + T\mathbf{b} \end{aligned} \tag{3.5}$$

Equation (3.5) is in exactly the same form as a preconditioned Gauss-Jacobi iterative solver, where the preconditioner is T. Its convergence condition is [70]:

$$\rho\left(I - TA\right) < 1 \tag{3.6}$$

where ρ is the spectral radius, i.e., the maximum absolute eigenvalue, of a matrix. Condition (3.6) is satisfied since $T \approx A^{-1}$, assuming that the initial stochastic solution that builds T is accurate enough. It is well known that an iterative process of the type in (3.5) converges to [70]:

$$\begin{aligned} \mathbf{x^\infty} &= \left(I - \left(I - TA\right)\right)^{-1} T\mathbf{b} \\ &= A^{-1}\mathbf{b} \end{aligned} \tag{3.7}$$

which is the exact solution.

By now, we have shown that Algorithm 2 is an iterative solver with a preconditioner built by an stochastic solver. As mentioned earlier, this is only a first cut at the hybrid approach. To refine it, we observe that the iterative mechanism does not have to be Gauss-Jacobi as in equation (3.5), and potentially can be *any* iterative solver. Since R-matrices are symmetric and positive definite, a better choice than Gauss-Jacobi is likely to be the preconditioned conjugate gradient (PCG) method. However, the PCG method requires its preconditioner to be a symmetric matrix, which is a condition that may not be met by the matrix T in this prototype algorithm.

The next section investigates the structure of a journey record, and presents a mathematical proof that, using only a fraction of the journey record and hence with lower storage than matrix T, a symmetric incomplete LDL factorization can be obtained. Our final hybrid solver for R-matrices will use such an incomplete LDL factorization with PCG.

3.2 Proof of Incomplete Factorization

Referring back to the terminology and notation defined in Chapter 1, and suppose that the exact LDL factorization of an R-matrix A is $A = LDL^{\mathrm{T}}$, the goal of this section is, by extracting information from the journey record of random walks, to construct a lower triangular matrix L' and a diagonal matrix D' such that

$$\begin{aligned} & L'_{i,i} = 1, \quad \forall i \\ & \text{if } L'_{i,j} \neq 0 \text{ then } L_{i,j} \neq 0, \quad \forall i, j \\ & L' \approx L \\ & D' \approx D \end{aligned} \tag{3.8}$$

The proof is described in two stages: Section 3.2.1 proves that the journey record contains an approximate L factor, and then Section 3.2.2 proves that its non-zero pattern is a subset of that of the exact L factor. The formula of the diagonal D factor is derived in Section 3.2.3.

The procedure of finding this factorization is independent of the right-hand-side vector $\mathbf{b}$. Throughout this section, any appearance of $\mathbf{b}$ is symbolic: its entries do not participate in actual computation, and the involved equations are true for any possible vector $\mathbf{b}$.

3.2.1 The Approximate Factorization

Suppose the dimension of matrix A is N, and its k^{th} row corresponds to node k in Figure 2.1, $k = 1, 2, \cdots, N$. Without loss of generality, assume that in the stochastic solution, the nodes are processed in the natural ordering $1, 2, \cdots, N$. According to the speedup technique in Section 2.4.1, for random walks that start from node k, the nodes in the set $\{1, 2, \cdots, k-1\}$ are already solved and they now serve as home nodes where a random walk ends. The awards for reaching nodes $\{1, 2, \cdots, k-1\}$ are the estimated values of $\{x_1, x_2, \cdots, x_{k-1}\}$ respectively. Suppose that in equation (2.7), we choose $m_0 = 0$, and hence the motel prices are given by $m_i = -\frac{b_i}{A_{i,i}}$, for $i = k, k+1, \cdots, N$. Define the following notations.

- Let M_k be the number of walks carried out from node k.
- Let $H_{k,i}$ be the number of walks that start from node k and end at node $i \in \{1, 2, \cdots, k-1\}$.
- Let $J_{k,i}$ be the number of times that walks from node k pass the motel at node $i \in \{k, k+1, \cdots, N\}$.

Taking the average of the results of the M_k walks from node k, we get the following equation for the estimated solution entry.

$$x'_k = \frac{\sum_{i=1}^{k-1} H_{k,i} x'_i + \sum_{i=k}^{N} J_{k,i} \frac{b_i}{A_{i,i}}}{M_k} \tag{3.9}$$

where x'_i is the estimated value of x_i for $i \in \{1, 2, \cdots, k-1\}$. Note that the awards received at the initial home nodes are ignored in the above equation since $m_0 = 0$. Moving the $H_{k,i}$ terms to the left side, we obtain

$$-\sum_{i=1}^{k-1} \frac{H_{k,i}}{M_k} x'_i + x'_k = \sum_{i=k}^{N} \frac{J_{k,i}}{M_k A_{i,i}} b_i \tag{3.10}$$

By writing the above equation for $k = 1, 2, \cdots, N$, and assembling the N equations together into a matrix form, we obtain

$$Y\mathbf{x}' = Z\mathbf{b} \tag{3.11}$$

where $\mathbf{x}'$ is the approximate solution produced by the stochastic solver; Y and Z are two square matrices of dimension N such that

$$\begin{aligned}
Y_{k,k} &= 1, & \forall k \\
Y_{k,i} &= -\frac{H_{k,i}}{M_k}, & \forall k > i \\
Y_{k,i} &= 0, & \forall k < i \\
Z_{k,i} &= \frac{J_{k,i}}{M_k A_{i,i}}, & \forall k \le i \\
Z_{k,i} &= 0, & \forall k > i
\end{aligned} \tag{3.12}$$

These two matrices Y and Z are the journey record built by the bookkeeping technique in Section 2.4.2. Obviously Y is a lower triangular matrix with unit diagonal entries, Z is an upper triangular matrix, and their entries are independent of the right-hand-side vector $\mathbf{b}$. Once Y and Z are built from random walks, given any $\mathbf{b}$, one can apply equation (3.11) and find $\mathbf{x}'$ efficiently by a forward substitution. The matrix T defined in (3.4) in the previous section is simply $Y^{-1}Z$.

It is worth pointing out the physical meaning of the entries in matrix Y: the negative of an entry, $(-Y_{k,i})$, is asymptotically equal to the probability that a random walk from node k ends at node i, when M_k goes to infinity. Another property of matrix Y is that the sum of every row is zero, except for the first row where only the first entry is non-zero.

From equation (3.11), we have

$$Z^{-1}Y\mathbf{x}' = \mathbf{b} \tag{3.13}$$

Since the vector $\mathbf{x}'$ in the above equation is an approximate solution to the original set of equations $A\mathbf{x} = \mathbf{b}$, it follows that[2]

$$Z^{-1}Y \approx A \tag{3.14}$$

Because the inverse of an upper triangular matrix, Z^{-1}, is also upper triangular, equation (3.14) is in the form of an approximate "UL factorization" of A. The following definition and lemma present a simple relation between UL factorization and the more commonly encountered LU factorization.

Definition 2 *The operator* $\mathrm{rev}(\cdot)$ *is defined on square matrices: given matrix A of dimension N,* $\mathrm{rev}(A)$ *is also a square matrix of dimension N, such that*

$$\mathrm{rev}(A)_{i,j} = A_{N+1-i,N+1-j}, \quad \forall i,j \in \{1,2,\cdots,N\}$$

In simple terms, the operator $\mathrm{rev}(\cdot)$ merely inverts the row and column ordering of a matrix. A simple example of applying this operator is as follows:

$$\mathrm{rev}\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right) = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

[2]For any vector $\mathbf{b}$, we have $\left(Z^{-1}Y\right)^{-1}\mathbf{b} = \mathbf{x}' \approx \mathbf{x} = A^{-1}\mathbf{b}$. Therefore, $A\left(Z^{-1}Y\right)^{-1}\mathbf{b} \approx \mathbf{b}$, and then $\left(I - A\left(Z^{-1}Y\right)^{-1}\right)\mathbf{b} \approx 0$. Since this is true for any vector $\mathbf{b}$, it must be true for eigenvectors of the matrix $\left(I - A\left(Z^{-1}Y\right)^{-1}\right)$, and it follows that the eigenvalues of the matrix $\left(I - A\left(Z^{-1}Y\right)^{-1}\right)$ are all close to zero. Thus we claim that $Z^{-1}Y \approx A$.

Lemma 1 *Let $A = LU$ be the LU factorization of a square matrix A, then* $\mathrm{rev}(A) = \mathrm{rev}(L)\mathrm{rev}(U)$ *is true and is the UL factorization of* $\mathrm{rev}(A)$.

Lemma 1 is self-evident, and the proof is omitted. It states that the reverse-ordering of the LU factors of A are the UL factors of reverse-ordered A.

Applying Lemma 1 on equation (3.14), we obtain

$$\mathrm{rev}(Z^{-1})\mathrm{rev}(Y) \approx \mathrm{rev}(A) \tag{3.15}$$

Since A is an R-matrix and is symmetric, $\mathrm{rev}(A)$ must be also symmetric, and we can take the transpose of both sides, and have

$$(\mathrm{rev}(Y))^{\mathrm{T}} \left(\mathrm{rev}(Z^{-1})\right)^{\mathrm{T}} \approx \mathrm{rev}(A) \tag{3.16}$$

The above equation has the form of an LU factorization: matrix $(\mathrm{rev}(Y))^{\mathrm{T}}$ is lower triangular and has unit diagonal entries; matrix $(\mathrm{rev}(Z^{-1}))^{\mathrm{T}}$ is upper triangular.

Lemma 2 *The (Doolittle) LU factorization of a square matrix is unique.*

The proof of Lemma 2 is provided in Appendix A.

Let the exact LU factorization of $\mathrm{rev}(A)$ be $\mathrm{rev}(A) = L_{\mathrm{rev}(A)}U_{\mathrm{rev}(A)}$, and its exact LDL factorization be $\mathrm{rev}(A) = L_{\mathrm{rev}(A)}D_{\mathrm{rev}(A)}\left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}}$. Since equation (3.16) is an approximate LU factorization of $\mathrm{rev}(A)$, while the exact LU factorization is unique, it must be true that:

$$(\mathrm{rev}(Y))^{\mathrm{T}} \approx L_{\mathrm{rev}(A)} \tag{3.17}$$

$$\left(\mathrm{rev}(Z^{-1})\right)^{\mathrm{T}} \approx U_{\mathrm{rev}(A)} = D_{\mathrm{rev}(A)}\left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}} \tag{3.18}$$

The above two equations indicate that from the matrix Y built by random walks, we can obtain an approximation to factor $L_{\mathrm{rev}(A)}$, and that the matrix Z contains redundant information. Section 3.2.3 shows how to estimate matrix $D_{\mathrm{rev}(A)}$ utilizing only the diagonal entries of matrix Z, and hence the rest of Z is not needed at all. According to equation (3.12), matrix Y is the award register in the journey record and keeps track of end nodes of random walks, while matrix Z is the motel-expense register and keeps track of all intermediate nodes of walks. Therefore matrix Z is the dominant portion of the journey record, and by removing all of its off-diagonal entries, the modified journey record is significantly smaller than that in the original bookkeeping technique from Section 2.4.2. In fact, an upper bound on the number of non-zero entries in matrix Y is proven in the next section.

3.2.2 The Incomplete Non-zero Pattern

The previous section proves that an approximate factorization of an R-matrix A can be obtained by random walks. However, it does not constitute a proof of incomplete factorization, because an incomplete factorization implies that the non-zero pattern of the approximate factor must be a subset of the non-zero pattern of the exact factor. Such a proof is the task of this section: to prove that an entry of $(\text{rev}(Y))^{\text{T}}$ can be possibly non-zero only if the corresponding entry of $L_{\text{rev}(A)}$ is non-zero.

For $i \neq j$, the (i, j) entry of $(\text{rev}(Y))^{\text{T}}$ is as follows, after applying Definition 2 and equation (3.12).

$$\left((\text{rev}(Y))^{\text{T}}\right)_{i,j} = Y_{N+1-j,N+1-i} = -\frac{H_{N+1-j,N+1-i}}{M_{N+1-j}} \tag{3.19}$$

This value is non-zero if and only if $j < i$ and $H_{N+1-j,N+1-i} > 0$. In other words, at least one random walk starts from node $(N + 1 - j)$ and ends at node $(N + 1 - i)$.

To analyze the non-zero pattern of $L_{\text{rev}(A)}$, certain concepts from the literature of LU factorization are used here, and certain conclusions are cited without proof. More details can be found in [2] [19] [24] [25] [34]. Figure 3.1 illustrates one step in

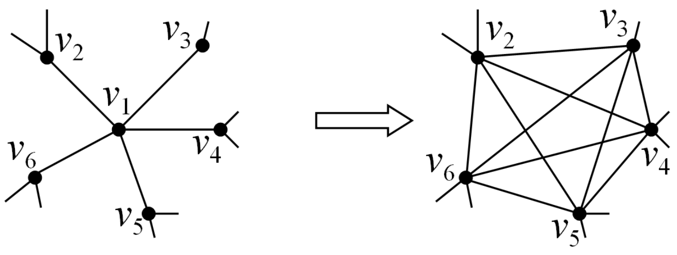


Figure 3.1: One step in Gaussian elimination.

the exact Gaussian elimination[3] of a matrix: removing one node from the matrix graph, and creating a clique among its neighbors. For example, when node v_1 is removed, a clique is formed for $\{v_2, v_3, v_4, v_5, v_6\}$, where the new edges correspond to fills added to the remaining matrix. At the same time, five non-zero values are written into the L matrix, at the five entries that are the intersections[4] of node v_1's corresponding column and the five rows that correspond to nodes $\{v_2, v_3, v_4, v_5, v_6\}$.

Definition 3 *Given a graph $G = (V, E)$, a node set $S \subset V$, and nodes $v_1, v_2 \in V$ such that $v_1, v_2 \notin S$, node v_2 is said to be reachable from node v_1 through S if there exists a path between v_1 and v_2 such that all intermediate nodes, if any, belong to S.*

Definition 4 *Given a graph $G = (V, E)$, a node set $S \subset V$, a node $v_1 \in V$ such that $v_1 \notin S$, the reachable set of v_1 through S, denoted $R(v_1, S)$, is defined as:*

$$R(v_1, S) = \{v_2 \notin S | v_2 \textit{ is reachable from } v_1 \textit{ through } S\}$$

[3]The procedure of LU factorization of a matrix is a sequence of Gaussian elimination steps. From the perspective of the matrix graph, it is a sequence of graph operations that remove nodes one by one.

[4]In this section, rows and columns of a matrix are often identified by their corresponding nodes in the matrix graph, and matrix entries are often identified as intersections of rows and columns. The reason is that such references are independent of the matrix ordering, and thereby avoid confusion due to the two orderings involved in the discussion.

Note that if v_1 and v_2 are adjacent, there is no intermediate node on the path between them, then Definition 3 is satisfied, and v_2 is reachable from v_1 through any node set. Therefore, $R(v_1, S)$ always includes the direct neighbors of v_1 that do not belong to S.

Given an R-matrix A, let G be its matrix graph, let L be the complete L factor in its exact LDL factorization, and let v_1 and v_2 be two nodes in G. Again, every node in G has a corresponding row and a corresponding column in A and in L. The following lemma can be derived from [25] [34].

Lemma 3 *The entry in L at the intersection of column v_1 and row v_2 is non-zero if and only if:*

1. *v_1 is eliminated prior to v_2 during Gaussian elimination*
2. *$v_2 \in R(v_1, \{\text{nodes eliminated prior to } v_1\})$*

Now we apply this lemma on $L_{\text{rev}(A)}$. Because the factorization of rev(A) is performed in the reverse ordering, i.e., $N, N-1, \cdots, 1$, the (i, j) entry of $L_{\text{rev}(A)}$ is the entry at the intersection of the column that corresponds to node $(N+1-j)$ and the row that corresponds to node $(N+1-i)$. This entry is non-zero if and only if both of the following conditions are met.

1. Node $(N+1-j)$ is eliminated prior to node $(N+1-i)$
2. $(N+1-i) \in R(N+1-j, S_j)$
   where $S_j = \{\text{nodes eliminated prior to } N+1-j\}$

Again, because the Gaussian elimination is carried out in the reverse ordering $N, N-1, \cdots, 1$, the first condition implies that

$$\begin{aligned} N+1-j &> N+1-i \\ j &< i \end{aligned}$$

The node set S_j in the second condition is simply $\{N+2-j, N+3-j, \cdots, N\}$.

Recall that equation (3.19) is non-zero if there is at least one random walk that starts from node $(N+1-j)$ and ends at node $(N+1-i)$. Also recall that according to Section 2.4.1, when random walks are performed from node $(N+1-j)$, nodes $\{1, 2, \cdots, N-j\}$ are home nodes that walks terminate, while nodes $S_j = \{N+2-j, N+3-j, \cdots, N\}$ are the motel nodes that a walk can pass through. Therefore, a walk started from node $(N+1-j)$ can possibly end at node $(N+1-i)$, only if node $(N+1-i)$ is reachable from $(N+1-j)$ through the motel node set, i.e., node set S_j.

By now it is proven that both conditions for $\left(L_{\text{rev}(A)}\right)_{i,j}$ to be non-zero are necessary conditions for equation (3.19) to be non-zero. Therefore, the non-zero pattern of $(\text{rev}(Y))^{\mathrm{T}}$ is a subset of the non-zero pattern of $L_{\text{rev}(A)}$. Together, this conclusion and equation (3.17) give rise to the following lemma.

Lemma 4 $(\text{rev}(Y))^{\mathrm{T}}$ *is the L factor of an incomplete LDL factorization of matrix* $\text{rev}(A)$.

This lemma indicates that, from random walks, we can obtain an incomplete LDL factorization of the left-hand-side matrix A in its reversed index ordering. The remaining approximate diagonal matrix D is derived in the next section.

3.2.3 The Diagonal Component

To evaluate the approximate D matrix, we take the transpose of both sides of equation (3.18), and obtain

$$\text{rev}(Z^{-1}) \approx L_{\text{rev}(A)} D_{\text{rev}(A)} \tag{3.20}$$

Lemma 5 *For a non-singular square matrix A,* $\text{rev}(A^{-1}) = (\text{rev}(A))^{-1}$.

The proof of this lemma is trivial and is omitted. Applying this lemma on equation (3.20), we have

$$\begin{aligned} (\text{rev}(Z))^{-1} &\approx L_{\text{rev}(A)} D_{\text{rev}(A)} \\ I &\approx \text{rev}(Z) L_{\text{rev}(A)} D_{\text{rev}(A)} \end{aligned} \tag{3.21}$$

Recall that $\text{rev}(Z)$ and $L_{\text{rev}(A)}$ are both lower triangular, that $L_{\text{rev}(A)}$ has unit diagonal entries, and that $D_{\text{rev}(A)}$ is a diagonal matrix. Therefore, the (i,i) diagonal entry in the above equation is simply

$$\begin{aligned} (\text{rev}(Z))_{i,i} \left(L_{\text{rev}(A)}\right)_{i,i} \left(D_{\text{rev}(A)}\right)_{i,i} &\approx 1 \\ (\text{rev}(Z))_{i,i} \cdot 1 \cdot \left(D_{\text{rev}(A)}\right)_{i,i} &\approx 1 \\ \left(D_{\text{rev}(A)}\right)_{i,i} &\approx \frac{1}{(\text{rev}(Z))_{i,i}} \end{aligned} \tag{3.22}$$

Applying Definition 2 and equation (3.12), we finally have the equation for computing the approximate D factor, given as follows.

$$\begin{aligned} \left(D_{\text{rev}(A)}\right)_{i,i} &\approx \frac{1}{Z_{N+1-i,N+1-i}} \\ &= \frac{M_{N+1-i} A_{N+1-i,N+1-i}}{J_{N+1-i,N+1-i}} \end{aligned} \tag{3.23}$$

It is worth pointing out the physical meaning of the quantity $\frac{J_{N+1-i,N+1-i}}{M_{N+1-i}}$. It is the average number of times that a walk from node $N+1-i$ passes node $N+1-i$ itself; in other words, it is the average number of times that the walker returns to his/her starting point before the game is over. Equation (3.23) indicates that an entry in the D factor is equal to the corresponding diagonal entry of the original matrix A divided by the expected number of returns.

3.3 The Hybrid Solver and its Comparison with ILU

In this section, the proposed hybrid solver for R-matrices is presented in its entirety, and we argue that it outperforms traditional ICCG methods.

Definition 5 *The operator* $\mathrm{rev}(\cdot)$ *is defined on vectors: given vector* $\mathbf{x}$ *of length* N*,* $\mathrm{rev}(\mathbf{x})$ *is also a vector of length* N*, such that* $\mathrm{rev}(\mathbf{x})_i = \mathbf{x}_{N+1-i}, \forall i \in \{1, 2, \cdots, N\}$.

It is easy to verify that the set of equations $A\mathbf{x} = \mathbf{b}$ is equivalent to

$$\mathrm{rev}(A)\mathrm{rev}(\mathbf{x}) = \mathrm{rev}(\mathbf{b})$$

By now, we have collected the necessary pieces of the proposed hybrid solver, and it is summarized in the pseudocode in Algorithm 3.

From the perspective of an iterative solver, the hybrid solver essentially replaces the preconditioner in existing ICCG methods with the incomplete LDL factorization produced by random walks. We claim that this new preconditioner has better quality than the incomplete Cholesky factor B produced by traditional incomplete factorization approaches. In other words, if matrices Y and B have the same number of non-zero entries, and given the same target accuracy requirement, we expect the hybrid solver to converge with fewer iterations than a traditional ICCG solver preconditioned by $\left(BB^{\mathrm{T}}\right)^{-1}$.

The argument is based on the fact that, in traditional Gaussian-elimination-based methods, the operations of eliminating different nodes are correlated and the error introduced at an earlier node gets propagated to a later node, while in random walks, the operation on a node is totally independent from other nodes. We now state this in detail and more precisely.

Algorithm 3 *The final hybrid solver for R-matrices:*

```
Precondition {
    Run random walks, build matrix $Y$ and find diagonal
        entries of $Z$ using equation (3.12);
    Build $L_{\mathrm{rev}(A)}$ using equation (3.17);
    Build $D_{\mathrm{rev}(A)}$ using equation (3.23);
}
Given $\mathbf{b}$, solve {
    Convert $A\mathbf{x} = \mathbf{b}$ to $\mathrm{rev}(A)\mathrm{rev}(\mathbf{x}) = \mathrm{rev}(\mathbf{b})$;
    Apply PCG on $\mathrm{rev}(A)\mathrm{rev}(\mathbf{x}) = \mathrm{rev}(\mathbf{b})$ with the
        preconditioner $\left(L_{\mathrm{rev}(A)} D_{\mathrm{rev}(A)} \left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}}\right)^{-1}$;
    Convert $\mathrm{rev}(\mathbf{x})$ to $\mathbf{x}$;
}
```

Let us use the ILUT approach as an example of traditional preconditioning methods; similar argument can be made for other existing techniques, as long as they are based on Gaussian elimination. Suppose in Figure 3.1, when eliminating node v_1, the new edge between nodes v_2 and v_3 corresponds to an entry whose value falls below a specified threshold, then ILUT drops that entry from the remaining matrix, and that edge is removed from the remaining matrix graph. Later when the algorithm reaches the stage of eliminating node v_2, because of that missing edge, no edge is created from v_3 to the neighbors of v_2, and thus more edges are missing, and this new set of missing edges then affect later computations accordingly. Therefore, an early decision of dropping an entry is propagated throughout the ILUT process. On the one hand, this leads to the sparsity of B, which is desirable; on the other hand, there is no control over error accumulation, and later columns of B can deviate from the exact Cholesky factor C by an amount that is greater than the planned threshold of ILUT. Such error accumulation gets exacerbated for larger and denser matrices.

The hybrid solver does not suffer from this problem. When we run random walks from node k and collect the $H_{k,i}$ values to build the k^{th} row of matrix Y according to equation (3.12), we only know that the nodes $\{1, 2, \cdots, k-1\}$ are homes, and this is the only information needed. If, for some reason, the computed k^{th} row of matrix Y is of lower quality, this error does not affect other rows in any way; each row is responsible for its own accuracy, according to a criterion to be discussed in Section 3.4.1. In fact, in a parallel computing environment, the computation of each row of Y can be assigned to a different processor.

It is worth pointing out that the error accumulation discussed here is different from the cost of bias discussed at the end of Section 2.4.1. That bias in the stochastic solver, in the context of the hybrid solver, maps to the forward/backward substitution, i.e., the procedure of applying the preconditioner inside PCG. Due to the fact that forward/backward substitution is a sequential process, such bias or error propagation is inevitable in all iterative solvers as long as a factorization-based multiplicative preconditioner is in use. Our claim here is that the hybrid solver is free of error accumulation in building the preconditioner, and not in applying the preconditioner[5].

In summary, because of the absence of error accumulation in building the preconditioner, we expect the hybrid solver to outperform traditional ICCG methods,

[5]After a row of matrix Y is calculated, it is possible to add a postprocessing step to drop insignificant entries. The criterion can be any of the strategies used in traditional incomplete factorization methods, and, as discussed in Chapter 1, may be based on pattern, threshold, size limits, or a combination of them. With such postprocessing, the hybrid solver still maintains the advantage of independence between row calculations. This is not included in our implementation.

and we expect that the advantage becomes more prominent for larger and denser matrices. These claims are validated by numerical tests in Chapter 7: the hybrid solver is compared against both ICCG with ILU(0) and ICCG with ILUT, where the comparison metric is the computational complexity, measured by the total number of double-precision multiplications needed to solve $A\mathbf{x} = \mathbf{b}$ to the same accuracy requirement.

3.4 Implementation Issues

This section looks at several implementation aspects of the hybrid solver, and some are also applicable to the stochastic solver in Chapter 2. For the hybrid solver, the goal is twofold:

- To minimize the runtime of building the preconditioner. In other words, the computation given in the first part of Algorithm 3 should be performed with the fewest random walks.
- To achieve a better accuracy-size tradeoff. That is either to improve the accuracy of the preconditioner without increasing the number of non-zero entries, or to reduce the number of non-zeroes without losing accuracy[6].

3.4.1 Stopping Criterion

The topic of this section is the accuracy control of the preconditioner, that is, how should one choose M_k, the number of walks from node k, to achieve a certain accuracy level in estimating its corresponding entries in the LDL factorization. In

[6]The accuracy of a preconditioner matrix T is measured by $\rho\left(I - TA\right)$, where ρ is the spectral radius, i.e., the maximum absolute eigenvalue, of a matrix. It is often difficult to analytically quantify the spectral radius, and the discussion of accuracy in this section is mostly qualitative.

Section 2.2, the stopping criterion in the stochastic solver is chosen to be an error margin and a confidence level defined on the result of a walk; it is not applicable to the hybrid solver because here it is necessary for the criterion to be independent of the right-hand-side vector **b**. In our implementation, a new stopping criterion is defined on a value that is a function of only the left-hand-side matrix A, as follows. Let $\Xi_k = E\left[\text{length of a walk from node } k\right]$, and let Ξ'_k be the average length of the M_k walks. The stopping criterion is chosen as

$$P[-\Delta < \frac{\Xi'_k - \Xi_k}{\Xi_k} < \Delta] > \alpha \tag{3.24}$$

where Δ is a relative error margin, and α is a confidence level, for example $\alpha = 99\%$. Practically, this criterion is checked by the following inequality:

$$\frac{\Delta \Xi'_k \sqrt{M_k}}{\sigma_k} > Q^{-1}\left(\frac{1-\alpha}{2}\right) \tag{3.25}$$

where σ_k is the standard deviation of the lengths of the M_k walks, and Q is the standard normal complementary cumulative distribution function. Thus, M_k is decided on the fly, and random walks are run from node k until condition (3.24) is satisfied. Similar to the discussion in Section 2.2, it is also necessary to impose a lower bound on M_k, e.g. 20 walks.

Condition (3.24) can be interpreted as follows: suppose all motel prices are one dollar per night, then Ξ_k is the expected cost of a trip, and the stopping criterion requests an estimation of this value with a certain error tolerance. Note that this is not the only way to design the stopping criterion: it can also be defined on quantities other than Ξ_k, for example the expected number of returns, as long as this quantity does not depend on **b**.

3.4.2 Exact Computations for One-step Walks

The implementation technique in this section is a special treatment for the random walks with length 1, which we refer to as one-step walks. Such a walk occurs when an immediate neighbor of the starting node is a home node, and the first step of the walks happens to go there. The idea is to place stochastic computations performed by one-step walks with their deterministic limits.

Without loss of generality, assume that the node ordering in the hybrid solver is the natural ordering $1, 2, \cdots, N$. Let us consider the M_k walks from node k, and suppose at least one of its immediate neighboring nodes is a home node, which could be either an initial home node if the k^{th} row of matrix A is strictly diagonally dominant, or a node j such that $j < k$. Among the M_k walks, let $M_{k,1}$ be the number of one-step walks, and let $H_{k,i,1}$ be the number of one-step walks that go to node i, where node i is an arbitrary node such that $i < k$. For the case that node i is not adjacent to node k, $H_{k,i,1}$ is simply zero. For the case that node i is adjacent to node k, note that $H_{k,i,1}$ may not be equal to $M_{k,1}$, as there can be other immediate neighbors of k that are home nodes. The $Y_{k,i}$ formula in (3.12) can be rewritten as

$$Y_{k,i} = -\frac{H_{k,i}}{M_k} = -\frac{H_{k,i,1}}{M_k} - \left(\frac{M_k - M_{k,1}}{M_k}\right) \cdot \left(\frac{H_{k,i} - H_{k,i,1}}{M_k - M_{k,1}}\right) \tag{3.26}$$

Applying the mapping between transition probabilities and matrix entries in equation (2.7), the following equations can be derived.

$$\begin{aligned} \lim_{M_k \to \infty} \frac{H_{k,i,1}}{M_k} &= P\left[\text{first step goes to node } i\right] \\ &= -\frac{A_{k,i}}{A_{k,k}} \end{aligned} \tag{3.27}$$

$$\begin{aligned}\lim_{M_k\to\infty}\frac{M_k-M_{k,1}}{M_k} &= P\left[\text{first step goes to a non-absorbing node}\right]\\ &= \sum_{j>k} P\left[\text{first step goes to node } j\right]\\ &= -\frac{\sum_{j>k} A_{k,j}}{A_{k,k}} \end{aligned}\tag{3.28}$$

We modify equation (3.26) by replacing the term $\frac{H_{k,i,1}}{M_k}$ and the term $\frac{M_k-M_{k,1}}{M_k}$ with their limits given by the above two equations, and obtain the following new formula for evaluating $Y_{k,i}$.

$$Y_{k,i} = \frac{A_{k,i}}{A_{k,k}} + \left(\frac{\sum_{j>k} A_{k,j}}{A_{k,k}}\right)\cdot\left(\frac{H_{k,i}-H_{k,i,1}}{M_k-M_{k,1}}\right)\tag{3.29}$$

The remaining stochastic part of this new equation is the term $\frac{H_{k,i}-H_{k,i,1}}{M_k-M_{k,1}}$, which can be evaluated by considering only random walks whose length is at least two; in other words, one-step walks are ignored. In implementation, this can be realized by simulating the first step of walks by randomly picking one of the non-absorbing neighbors of node k; note that then the number of random walks would automatically be $(M_k - M_{k,1})$, and no adjustment is needed.

With a similar derivation, the $Z_{k,k}$ formula[7] in (3.12) can be modified to the following.

$$Z_{k,k} = \frac{1}{A_{k,k}} + \frac{\sum_{j>k} A_{k,j}}{A_{k,k}^2} - \left(\frac{\sum_{j>k} A_{k,j}}{A_{k,k}^2}\right)\cdot\left(\frac{J_{k,k}-J_{k,k,1}}{M_k-M_{k,1}}\right)\tag{3.30}$$

where $J_{k,k,1}$ is the number of times that one-step walks pass node k. Obviously $J_{k,k,1} = M_{k,1}$, and therefore

$$Z_{k,k} = \frac{1}{A_{k,k}} + \left(\frac{\sum_{j>k} A_{k,j}}{A_{k,k}^2}\right)\cdot\left(1-\frac{J_{k,k}-M_{k,1}}{M_k-M_{k,1}}\right)\tag{3.31}$$

[7]Recall that we only need diagonal entries of matrix Z.

The remaining stochastic part of this new equation, the term $\frac{J_{k,k}-M_{k,1}}{M_k-M_{k,1}}$, again can be evaluated by considering only random walks with length being at least two. Practically, such computation is concurrent with evaluating $Y_{k,i}$'s based on equation (3.29).

The benefit of replacing equation (3.12) with equations (3.29) and (3.31) is twofold:

- Part of the evaluation of $Y_{k,i}$ and $Z_{k,k}$ entries is converted from stochastic computation to its deterministic limit, and the accuracy is potentially improved. For the case when all neighbors of node k have lower indices, i.e., when all neighbors are home nodes, equations (3.29) and (3.31) become exact: they translate to the exact values of the corresponding entries in the complete LDL factorization.
- By avoiding simulating one-step walks, the amount of computation in building the preconditioner is reduced. For the case when all neighbors of node k are home nodes, the stochastic parts of (3.29) and (3.31) disappear, and hence no walks are needed.

This technique is also applicable to the stochastic solver in Chapter 2. The only difference is that, in the stochastic solver case, the one-step walks are singled out in counting their actual monetary gain, instead of counting their portion in $Y_{k,i}$ and $Z_{k,k}$ entries.

3.4.3 Reusing Walks

Without loss of generality, assume that the node ordering in the hybrid solver is the natural ordering $1, 2, \cdots, N$. A sampled random walk is completely specified by the node indices along the way, and hence can be viewed as a sequence of integers $\{k_1, k_2, \cdots, k_\Gamma\}$, such that $k_1 > k_\Gamma$ and $k_1 \le k_l, \forall l \in \{2, \cdots, \Gamma - 1\}$. If a sequence of integers satisfy the above requirements, it is referred to as a legal sequence, and can be mapped to an actual random walk.

Due to the fact that a segment of a legal sequence may also be a legal sequence, it is possible to extract multiple legal sequences from a single simulated random walk, and use them also as random walks in the evaluation of equation (3.12) or its placement, (3.29) and (3.31). However, there are rules that one must comply with when extracting these legal sequences. A fundamental premise of both the stochastic solver and the hybrid solver is that random samples must be independent of each other. If two walks share a segment, they become correlated. In the hybrid solver, if two walks have different starting nodes, they never participate in the same equation (3.29) or (3.31), and hence are allowed to share segments; if two walks have the same starting nodes, however, they are prohibited from overlapping. Moreover, due to the technique in the previous section, any one-step walk should be ignored.

$$\begin{array}{ll} \text{(a)} & \{2,4,6,4,5,7,6,3,2,5,8,1\} \\ \text{(b)} & \{4,6,4,5,7,6,3\} \\ & \{5,7,6,3\} \\ & \{5,8,1\} \end{array}$$

Figure 3.2: An example of (a) the legal sequence of a simulated random walk and (b) three extra walks extracted from it.

Figure 3.2 shows an example of extracting multiple legal sequences from a single simulated random walk. The sequence $\{2,5,8,1\}$ cannot be used because it has the same starting node as the entire sequence; the sequence $\{4,5,7,6,3\}$ cannot be used because it has the same starting node as $\{4,6,4,5,7,6,3\}$ and the two sequences overlap[8]. On the other hand, $\{5, 7, 6, 3\}$ and $\{5, 8, 1\}$ are both extracted because they do not overlap and hence are two independent random walks.

Considering all of the above requirements, the procedure is shown in Algorithm 4, where the extracted legal sequences are directly accounted for in the M, H and J accumulators, which are defined the same as in all equations in this chapter. Note that the simulated random walk is never stored in memory, and the only extra storage due to this technique is the stacks, which contain a monotonically increasing sequence of integers at any moment.

This technique reduces the preconditioning runtime by fully utilizing the information contained in a single simulated random walk, such that it contributes to equations (3.29) and (3.31) as multiple random walks. It also guarantees that no two overlapping walks have the same starting node, and hence does not hurt the accuracy of the produced preconditioner. The only cost of this technique is that the node ordering of the hybrid solver must be determined beforehand, and hence pivoting is not allowed during the incomplete factorization[9].

[8]It is also legitimate to extract $\{4, 5, 7, 6, 3\}$ instead of $\{4, 6, 4, 5, 7, 6, 3\}$. However, the premise of random sampling must be fulfilled: the decision of whether to start a sequence with $k_2 = 4$ must be made without the knowledge of numbers after k_2, and the decision of whether to start a sequence with $k_4 = 4$ must be made without the knowledge of numbers after k_4. The strategy in Algorithm 4 is to start a sequence as early as possible, and hence produces $\{4, 6, 4, 5, 7, 6, 3\}$ instead of $\{4, 5, 7, 6, 3\}$.

[9]For R-matrices, or in general for diagonally dominant matrices, pivoting is not needed. For more general matrices to be discussed in Section 3.5, however, the usage of this technique may be limited.

Algorithm 4 *Extract multiple random walks from a single simulation:*

```
stack1.push( $k_1$ );
stack2.push( 1 );
For $l = 2, 3, \cdots$ , until the end of walk, do {
    While( $k_l <$ stack1.top() ){
        If( $l >$ stack2.top()+1 ){
            $k' =$ stack1.top();
            $M_{k'} = M_{k'} + 1$;
            $H_{k',k_l} = H_{k',k_l} + 1$;
            $J_{k',k'} = J_{k',k'} + 1$;
        }
        stack1.pop();
        stack2.pop();
    }
    If( $k_l >$ stack1.top() ){
        stack1.push( $k_l$ );
        stack2.push( $l$ );
    }
    else $J_{k_l,k_l} = J_{k_l,k_l} + 1$;
}
```

3.4.4 Matrix Ordering

In traditional factorization-based preconditioning techniques, matrix ordering can affect the performance, i.e., the accuracy-size tradeoff, of the preconditioner. The same statement is true for the hybrid solver. In general, since the hybrid solver performs an incomplete LDL factorization of the reverse ordering of matrix A, we can apply any existing ordering method on A, reverse the ordering that it produces, and then use the resulting ordering in the hybrid solver. In this way, any benefit of that ordering method can be inherited by us. The following are a few examples of practical ordering schemes for the hybrid solver.

- Approximate minimum degree ordering (AMD) from [2] is one of the state-of-the-art ordering techniques to reduce the number of non-zero entries in a complete LU factorization, or LDL factorization for an R-matrix. Since the complete L factor has a smaller size, it is likely that with the same size, the incomplete L factor may have better quality. Therefore, using a reversed AMD ordering in the hybrid solver may improve the accuracy-size tradeoff.
- Reverse Cuthill-McKee ordering (RCM) from [15] is a simple but useful ordering technique to reduce the bandwidth of both the original matrix A and the complete LU factors, and thereby improve cache efficiency. The physical CPU time of applying the LU factors on a particular right-hand-side vector is reduced due to less cache misses. For the hybrid solver, this means that, with the same preconditioner size, the actual CPU time of applying the preconditioner may be reduced. Of course, the ordering to use should be reversed RCM, which becomes the original Cuthill-McKee ordering.
- Random ordering is used in our implementation in Chapters 4–7. With random ordering, home nodes are relatively evenly distributed at all stages of the game, and for walks from any node, the most viable home nodes are of similar distances. Empirically, we have observed a stable performance.

3.4.5 Random Number Generator

In implementing both the stochastic solver and the hybrid solver, a fundamental element is a random number generator[10], because every step in random walks is simulated by a random number evenly distributed between 0 and 1. The choice of random number generator is a tradeoff: one that produces high quality random numbers is typically computationally expensive, while low quality random numbers may cause bias in the computation.

In implementations of both the stochastic solver and the hybrid solver throughout this thesis, the random number generator on page 279 of [60] is used. Empirically, we observe that its quality is sufficient for all the applications investigated in this thesis. In fact, the hybrid solver has lower quality requirement on random numbers than the stochastic solver.

Finally, a reference implementation of both the stochastic solver and the hybrid solver is available to the public [68].

3.5 Extension

So far the discussion on both the stochastic solver and the hybrid solver is limited to R-matrices. This section presents techniques aimed at extending the theory to more general matrices, and speculates on potential challenges in future research on this topic.

[10]An alternative to a random number generator is using pre-generated quasirandom numbers [33]. The benefit is removing the computation of generating random numbers, and the cost is extra storage, a few logic operations, and some CPU time due to cache misses.

3.5.1 Asymmetric Matrix

Let us first remove the symmetry requirement on matrix A. Recall that the construction of the random walk game and the derivation of equation (3.15) does not require A to be symmetric. Therefore, matrices Y and Z can still be obtained from random walks, and equation (3.15) remains true for an asymmetric matrix A. Suppose $\text{rev}(A) = L_{\text{rev}(A)} D_{\text{rev}(A)} U_{\text{rev}(A)}$, where $L_{\text{rev}(A)}$ is a lower triangular matrix with unit diagonal entries, $U_{\text{rev}(A)}$ is an upper triangular matrix with unit diagonal entries, and $D_{\text{rev}(A)}$ is a diagonal matrix. This is called the LDU factorization [19], which is a slight variation of the LU factorization, and is also unique: based on Lemma 2, the proof is trivial. Substituting the factorization into equation (3.15), we have

$$\text{rev}(Z^{-1})\text{rev}(Y) \approx L_{\text{rev}(A)} D_{\text{rev}(A)} U_{\text{rev}(A)} \tag{3.32}$$

Based on the uniqueness of LDU factorization, it must be true that

$$\text{rev}(Y) \approx U_{\text{rev}(A)} \tag{3.33}$$

$$\text{rev}(Z^{-1}) \approx L_{\text{rev}(A)} D_{\text{rev}(A)} \tag{3.34}$$

By equation (3.33), we can approximate $U_{\text{rev}(A)}$ based on Y; by equation (3.34), and through the same derivation as in Section 3.2.3, we can approximate $D_{\text{rev}(A)}$ based on the diagonal entries of Z. The remaining question is how to obtain $L_{\text{rev}(A)}$.

Suppose we construct a random walk game based on A^{T} instead of A, and suppose we obtain matrices $Y_{A^{\text{T}}}$ and $Z_{A^{\text{T}}}$ based on equation (3.12). Then according to equation (3.33), we have

$$\text{rev}(Y_{A^{\text{T}}}) \approx U_{\text{rev}(A^{\text{T}})} \tag{3.35}$$

where $U_{\text{rev}(A^{\text{T}})}$ is the U factor in the LDU factorization of $\text{rev}(A^{\text{T}})$. It is easy to derive the following

$$\mathrm{rev}(A^{\mathrm{T}}) = (\mathrm{rev}(A))^{\mathrm{T}} = \left(U_{\mathrm{rev}(A)}\right)^{\mathrm{T}} D_{\mathrm{rev}(A)} \left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}} \tag{3.36}$$

Therefore,

$$L_{\mathrm{rev}(A^{\mathrm{T}})} D_{\mathrm{rev}(A^{\mathrm{T}})} U_{\mathrm{rev}(A^{\mathrm{T}})} = \left(U_{\mathrm{rev}(A)}\right)^{\mathrm{T}} D_{\mathrm{rev}(A)} \left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}} \tag{3.37}$$

Based on the uniqueness of the LDU factorization, it must be true that

$$\left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}} = U_{\mathrm{rev}(A^{\mathrm{T}})} \tag{3.38}$$

By (3.35) and (3.38), we finally have

$$\mathrm{rev}(Y_{A^{\mathrm{T}}}) \approx \left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}} \tag{3.39}$$

In other words, we can approximate $L_{\mathrm{rev}(A)}$ based on $Y_{A^{\mathrm{T}}}$.

In summary, when matrix A is asymmetric, we need to construct two random walk games for A and A^{T}, and then based on the two Y matrices and the diagonal entries of one of the Z matrices[11], we can approximate the LDU factorization of rev(A) based on equations (3.23), (3.33), and (3.39). The proof of non-zero pattern is similar to Section 3.2.2, and with the same conclusion: the non-zero patterns of the resulting approximate L and U factors are subsets of those of the exact factors. Both the time complexity and space complexity of preconditioning become roughly twice those of the symmetric case: this is the same behavior as a traditional incomplete LU factorization.

3.5.2 Random Walk Game with Scaling

By now, the symmetry restriction on matrix A has been removed, and the remaining requirements on A are the following.

[11]Due to the uniqueness of the LDU factorization, it does not matter the diagonals of which Z are used.

- The diagonal entries must be positive.
- The off-diagonal entries must be negative or zero.
- A must irreducibly diagonally dominant, both row-wise and column-wise.

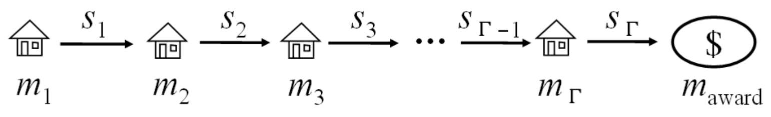


Figure 3.3: A random walk in the modified game with scaling.

To remove these constraints, a new game is designed by defining a scaling factor s on each direction of every edge in the original game from Section 2.2. Such a scaling factor becomes effective when a random walk passes that particular edge in that particular direction, and remains effective until this random walk ends. Let us look at the stochastic solver first. A walk is shown in Figure 3.3: it passes a number of motels, each of which has its price m_l, $l \in \{1, 2, \cdots, \Gamma\}$, and ends at a home node with certain award value m_{award}. The monetary gain of this walk is defined as follows.

$$\text{gain} = -m_1 - s_1 m_2 - s_1 s_2 m_3 - \cdots - \prod_{l=1}^{\Gamma-1} s_l \cdot m_\Gamma + \prod_{l=1}^{\Gamma} s_l \cdot m_{\text{award}} \tag{3.40}$$

In simple terms, this new game is different from the original game in that each transaction amount during the walk gets scaled by the product of the currently active scaling factors. Define the expected gain function f to be the same as in equation (2.2), and it is easy to derive the replacement of equation (2.4):

$$f(i) = \sum_{l=1}^{\text{degree}(i)} p_{i,l} s_{i,l} f(l) - m_i \tag{3.41}$$

where $s_{i,l}$ denotes the scaling factor associated with the direction $i \to l$ of the edge between i and l, and the rest of the symbols are the same as defined in (2.4).

Due to the degrees of freedom introduced by the scaling factors, the allowable left-hand-side matrix A is now any matrix with non-zero diagonal entries. In other words, given any matrix A with non-zero diagonal entries, a random walk game with scaling can be constructed such that the f values, if they uniquely exist, satisfy a set of linear equations where the left-hand-side matrix is A.

A corresponding hybrid solver can be derived for this new random walk game, by redefining the H and J values in equations (3.12), (3.29), and (3.31) to be the sum of products of scaling factors.

If every scaling factor in the game has an absolute value less or equal to 1, there is no numerical problem in the above new stochastic and hybrid solvers. This can be achieved as long as matrix A is diagonally dominant, in which case we can simply assign scaling factors to be $+1$ or -1, or, if matrix A is complex-valued, assign complex-valued scaling factors with unit magnitude. If there exist scaling factors with absolute values over 1, however, numerical problems may potentially occur since the product of scaling factors may be unbounded. How to quantify this effect and to analyze the corresponding convergence rate, is an open question for future research.

Therefore, the conclusion of this section is as follows.

- If the left-hand-side matrix A is irreducibly diagonally dominant both row-wise and column-wise, the generalized stochastic solver and the generalized hybrid solver are guaranteed to work, and according to the argument in Section 3.3, the hybrid solver is expected to outperform traditional incomplete factorization methods.
- If the left-hand-side matrix A is not diagonally dominant, as long as its diagonal entries are non-zero, a random walk game exists such that the f values, if they uniquely exist, satisfy a set of linear equations where the left-hand-side

matrix is A. However, no claim is made about the convergence of the resulting stochastic and hybrid solvers, and this is open for further investigation.

Chapter 4

Application in Power Grid Analysis

This chapter examines the application of the stochastic linear equation solver from Chapter 2 on power grid analysis of VLSI design. Specifically, we investigate the mapping between a power grid and a random game, the advantage brought by the locality property of random walks, and a few variations of the stochastic solver specially designed for power grid analysis. In terms of the historical development of the work in this thesis, this was the problem that led us to investigate the use of random walks for solving systems of linear equations. The successful use of random walks in this and other applications resulted in the theoretical developments described in Chapters 2 and 3. Some of the ideas in this chapter are problem-specific, while others, such as the hierarchy in Section 4.3, may be generalized to other applications; however the generalization is beyond the scope of this thesis.

Power grid analysis is an indispensable step in high-performance VLSI design. In successive technology generations, the VDD voltage decreases, resulting in narrower noise margins. Meanwhile, integrated circuits are rapidly growing more and more power-intensive. For example, [80] reports an Itanium processor with a worst-case power dissipation of 130W, and a power dissipation of 110W for the average case; [46] reports an Alpha processor with an estimated power consumption of 100W. Such power numbers imply increase in the average current carried by the power grid, which, in combination with increasing wire resistances due to the reduced interconnect wire widths, causes IR drops on power grids to worsen. Therefore, power grid noise is becoming a larger fraction of VDD in successive technology generations, and since the delivered VDD values significantly affect circuit performance, it is critical to analyze power grids accurately and efficiently to check for signal integrity.

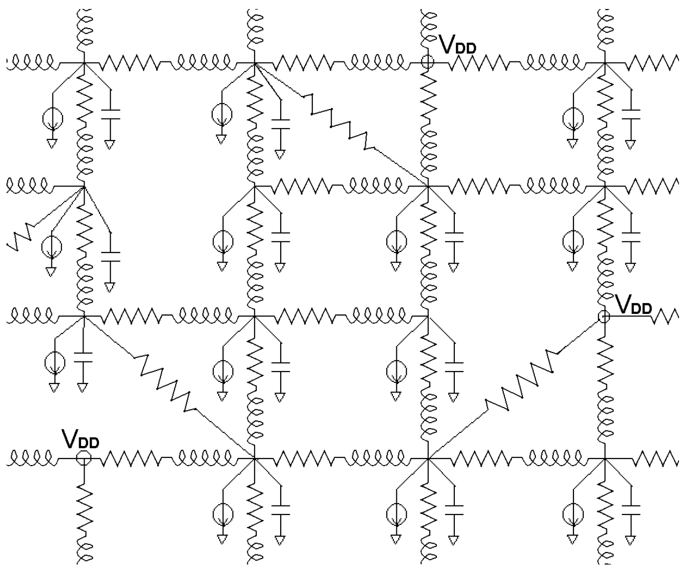


Figure 4.1: A part of a typical power grid model.

4.1 Static and Transient Analysis

A typical power grid may be represented by the model illustrated in Figure 4.1, consisting of wire resistances, wire inductances, wire capacitances, decoupling capacitors, VDD or ground pads, and current sources that correspond to the currents drawn by logic gates or functional blocks. There are two sub-problems to power grid analysis: *DC analysis* to find steady-state node voltages, and *transient analysis* which is concerned with finding voltage waveforms considering the effects of capacitors, inductors and time-varying current waveform patterns.

The DC analysis problem is formulated as:

$$G\mathbf{V} = \mathbf{b} \tag{4.1}$$

where G is the conductance matrix for the interconnected resistors, $\mathbf{V}$ is the vector of node voltages, and $\mathbf{b}$ is a vector of independent sources. Traditionally, direct solvers have been used to exploit the sparse and positive definite nature of G to solve this system of linear equations for $\mathbf{V}$. However, the cost of doing so can become prohibitive for a modern-day power grid with up to hundreds of millions of nodes, and this gets worse as the circuit size is ever growing from one technology generation to the next. The more difficult transient analysis problem involves the solution of an equation similar to (4.1) at each time point in the analysis. In recent technologies, inductive effects in the top few metal layers can no longer be ignored, and when mutual inductances are taken into consideration, the left-hand-side matrix, which contains the contribution of capacitors and inductors, is significantly denser than that for DC analysis, making it expensive even at a single time point.

Different circuit models and simulation techniques have been developed for power grid analysis, to handle large problem size, and to incorporate capacitances and inductances efficiently [6] [11] [12] [17] [40] [47] [59] [71] [74] [79] [81] [89].

Among them, several methods are proposed to achieve a lower time and space computational complexity by sacrificing a certain degree of accuracy. For example, [47] proposes a grid-reduction scheme to coarsen the circuit recursively, solves a coarsened circuit, and then maps back to find the solution to the original circuit. The approach in [89] utilizes the hierarchical structure of a power grid, divides it into a global grid and multiple local grids, and solves them separately.

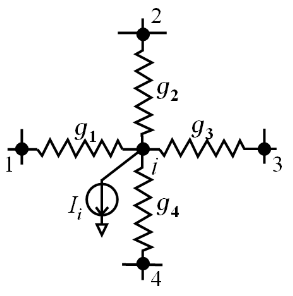


Figure 4.2: A representative node in a power grid.

4.1.1 DC Analysis

Our discussion is focused on the analysis of a VDD grid, pointing out the difference for a ground grid where applicable. For the DC analysis of a power grid, let us look at a single node i in the circuit, as illustrated in Figure 4.2. Applying Kirchoff's Current Law, Kirchoff's Voltage Law and the device equations for the conductances, we can write down the following equation:

$$\sum_{l=1}^{\text{degree}(i)} g_l(V_l - V_i) = I_i \tag{4.2}$$

where the nodes adjacent to i are labeled $1, 2, \cdots$, degree(i), V_i is the voltage at node i, V_l is the voltage at neighbor l, g_l is the conductance between node i and node l, and I_i is the current load connected to node i. Equation (4.2) can be reformulated as follows:

$$V_i = \sum_{l=1}^{\text{degree}(i)} \frac{g_l}{\sum_{j=1}^{\text{degree}(i)} g_j} V_l - \frac{I_i}{\sum_{j=1}^{\text{degree}(i)} g_j} \tag{4.3}$$

This implies that the voltage at any node is a linear function of the voltages at its neighbors. Another observation is that the sum of the linear coefficients associated with the V_l's is 1. For a power grid problem with N non-VDD nodes, we have N linear equations similar to the one above, one for each node. Solving this set of equations gives the exact solution vector $\mathbf{V}$.

It is easy to draw a parallel between the random walk game in Section 2.2 and power grid analysis. Equation (2.4) becomes identical to (4.3), and equation (2.3) reduces to the condition of perfect VDD nodes if

$$\begin{aligned} p_{i,l} &= \frac{g_l}{\sum_{j=1}^{\text{degree}(i)} g_j} \qquad l = 1, 2, \cdots, \text{degree}(i) \\ m_i &= \frac{I_i}{\sum_{j=1}^{\text{degree}(i)} g_j} \qquad m_0 = V_{DD} \qquad f(i) = V_i \end{aligned} \tag{4.4}$$

The formulation for ground net analysis is analogous; the major differences are that (i) the I_i's have negative values, (ii) VDD is replaced by zero. As a result, the walker earns money in each step, but gets no award at home.

Based on equation (4.4), for any power grid problem, we can construct a random walk problem that is mathematically equivalent, i.e., characterized by the same set of equations. The equation set has and only has one unique solution, and it is both the solution to the random walk problem, and the solution to power grid DC analysis. Therefore, the stochastic solver from Chapter 2, including the speedup techniques, is applicable on power grid.

It is worth noting that we may not need the complete solution vector and only need to compute node voltages that are of interest. For example, for checking VDD drop violations, only node voltages on the bottom metal layer are needed. Another observation is that the stopping criterion (2.8) and (2.9) do not have to be fixed, and can be adaptive to different node voltages: for a node with high estimated voltage drop, i.e., a relatively dangerous node in terms of signal integrity, the criterion can be switched to higher confidence level, lower Δ, or both; for a node with low estimated voltage drop, i.e., a safe node, the computation can stop after satisfying a relaxed criterion with lower confidence level or larger Δ. In other words, the computation for a node voltage may start with a low-accuracy criterion; when this accuracy level is met, a decision is made based on the estimated voltage drop at this time: if this value is below a certain threshold, the computation stops, otherwise the algorithm switches to a higher-accuracy criterion and continues running; when the new accuracy level is met, another decision is made based on the new estimated voltage drop, and even higher accuracy can be used if necessary, and so on. Using this adaptive strategy, more runtime would be spent on potential failure nodes, to get more accurate voltages for them, while safe nodes only get coarse estimation.

4.1.2 RC Transient Analysis

This and the next sections extend the discussion to transient analysis, where voltage waveforms are to be found while considering the effects of capacitances, inductances and time-varying current waveforms. Throughout this chapter, and in the implementation, the backward Euler approximation with timestep size h is used to convert differential equations to linear equations. We assume that the timestep size h is kept constant in a transient analysis.

Let us first incorporate capacitors into the framework. The equations to be solved for RC transient analysis may be written as follows [36]:

$$G\mathbf{V}(t) + C\mathbf{V}'(t) = \mathbf{b}(t) \tag{4.5}$$

where G is a conductance matrix, C is the matrix introduced by capacitors, $\mathbf{V}(t)$ is the vector of node voltages, and $\mathbf{b}(t)$ is the vector of independent sources. Applying the backward Euler formula with timestep size h, the equations become

$$\left(G + \frac{C}{h}\right)\mathbf{V}(t) = \mathbf{b}(t) + \frac{C}{h}\mathbf{V}(t-h) \tag{4.6}$$

This transformation translates the problem to solving a set of linear equations. Let us consider a single node i, at one time step at time t, and by Kirchoff's Current Law:

$$\sum_{l=1}^{\text{degree}(i)} g_l \left(V_l(t) - V_i(t)\right) = \frac{C_i}{h}\left(V_i(t) - V_i(t-h)\right) + I_i(t) \tag{4.7}$$

where V_i, V_l, I_i and g_l are as defined in equation (4.2), and C_i is the capacitance between node i and ground.

For a RC network with capacitors between two non-ground nodes, those capacitors can be replaced by resistors and current sources, while a current source between two nodes can be replaced by two current sources between the two nodes and ground. Then the following algorithm is applicable. Here we only discuss the case described by equation (4.7).

Equation (4.7) can be converted to the following form

$$\begin{aligned} V_i(t) &= \sum_{l=1}^{\text{degree}(i)} \frac{g_l}{\sum_{j=1}^{\text{degree}(i)} g_j + \frac{C_i}{h}} V_l(t) + \frac{\frac{C_i}{h}}{\sum_{j=1}^{\text{degree}(i)} g_j + \frac{C_i}{h}} V_i(t-h) \\ &\quad - \frac{I_i(t)}{\sum_{j=1}^{\text{degree}(i)} g_j + \frac{C_i}{h}} \end{aligned} \tag{4.8}$$

The rules of the random walk game are changed to accommodate the changes in the above equation. As shown in Figure 4.3, each node i has an additional connection, and the walker could end the walk and be awarded the amount $V_i(t-h)$ with probability

$$\frac{\frac{C_i}{h}}{\sum_{j=1}^{\text{degree}(i)} g_j + \frac{C_i}{h}}$$

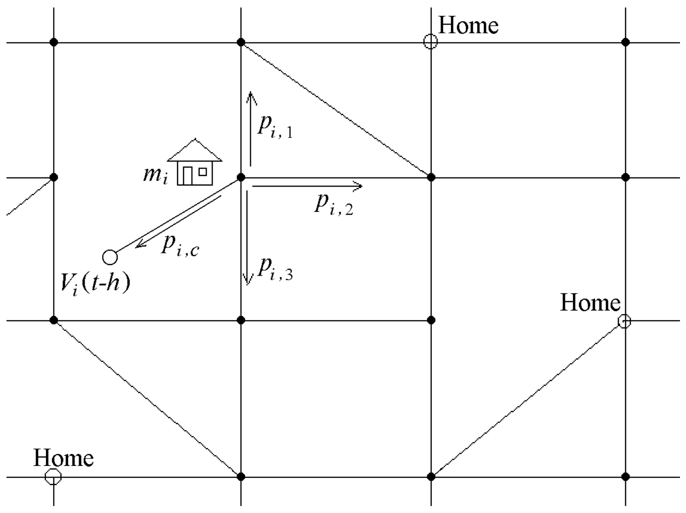


Figure 4.3: Rules for the transient RC analysis "game."

Intuitively, this rule is equivalent to replacing each capacitor by a resistor and a voltage source.

Under this new rule, the random walk game is mathematically equivalent to the equation set (4.6), and the stochastic solver can perform transient analysis of a RC network, timestep by timestep. In each timestep, the $V(t-h)$ values are updated with the node voltage values solved from the previous timestep.

The bookkeeping technique from Section 2.4.2 is a crucial ingredient in transient power grid analysis. With the timestep size h being constant, the left-hand-side matrix in equation (4.6) remains the same for all timesteps. Therefore the bookkeeping technique is applicable: random walks are needed only in the first transient timestep, and in all subsequence timesteps, equation (4.6) is solved by applying the journey record repeatedly, without any random walk.

4.1.3 RKC Transient Analysis

Inductances include self inductances and mutual inductances. Under the backward Euler approximation, a self inductance becomes a resistor and a current source in parallel, and can be easily added to the random walk game. Mutual inductances in RLC circuit formulation, however, are difficult to incorporate into the proposed framework, because of their induced extra unknown variables: the currents through the inductors.

Therefore, instead of the partial inductance matrix L, the inverse inductance, or susceptance, matrix K [16] [39] is used in this chapter to model inductors. The K matrix is defined as the inverse of the inductance matrix L, and has been shown to have better locality than L, and hence reduces the problem size of circuit simulation [16] [39]. The device equations under the backward Euler approximation are

$$K\mathbf{V}(t) = \frac{\mathbf{I}(t) - \mathbf{I}(t-h)}{h} \tag{4.9}$$

where $\mathbf{V}(t)$ is the vector of voltage drops over the inductors, $\mathbf{I}(t-h)$ is the vector of known currents through the inductors from the previous timestep, $\mathbf{I}(t)$ is the vector of unknown currents through the inductors in the present timestep, and h is again the timestep size. Equation (4.9) can be written as

$$\mathbf{I}(t) = hK\mathbf{V}(t) + \mathbf{I}(t-h) \tag{4.10}$$

and the corresponding companion model is illustrated in Figure 4.4, where only a pair of coupled inductors are shown.

In the transient simulation, a lumped π model is used for each wire segment, and is composed of a resistor and an inductor in series, and capacitors at two ends. Figure 4.5(a) shows this model, where capacitors are not drawn. By substituting the companion model of the inductor, the circuit in Figure 4.5(b) is obtained, where

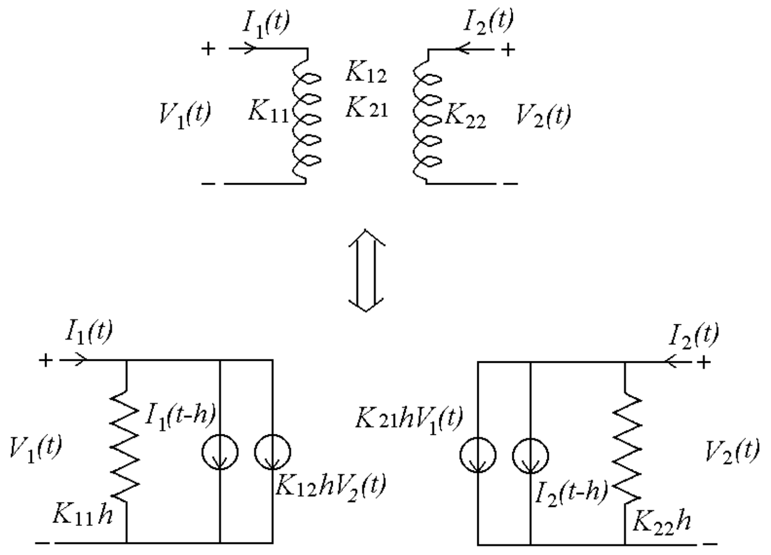


Figure 4.4: Companion model of a pair of inductors, adapted from [39].

the inductor is replaced by a resistor and two current sources in parallel. One of the two current sources is equal to the current from the last timestep, which is a known constant; the other source is a voltage-controlled current source which corresponds to the current induced by other inductors, i.e., a function of V_B's and V_C's from a number of other wire segments. The model in (b) can be further converted to (c) and then to (d), which is a circuit structure that can be directed added to the random walk game.

One complication caused by mutual inductances is that the current sources in Figure 4.5(d) is a function of not only V_A's and V_B's, but also V_C's, while V_C's are not among the system variables when we solve the circuit in form (d). In other words, the voltage-controlled current sources cannot be expressed as a linear function of node voltages.

To resolve the above problem, we propose an iterative approach to compute

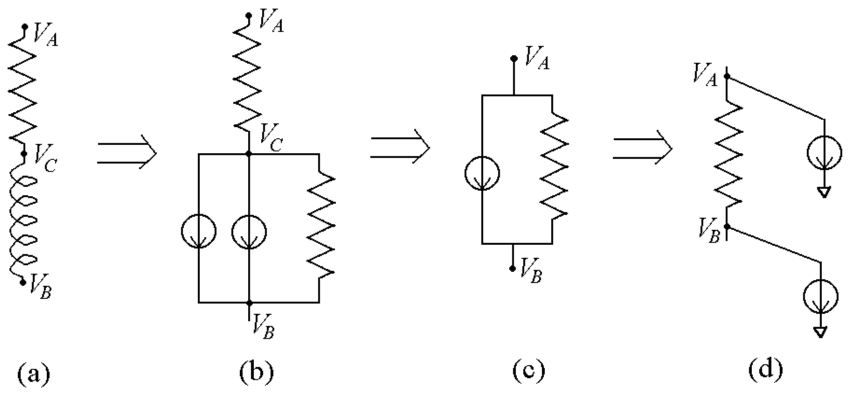


Figure 4.5: A wire segment model.

node voltages in each timestep, and in each iteration, the voltage-controlled current sources are assumed to have constant values. First, V_B's and V_C's of all wire segments from the previous timestep are used as the initial guess to compute the voltage-controlled portion of current sources in Figure 4.5(b)(c)(d). Next, by assuming these current sources to be constant, the stochastic solver solves the circuit in form (d), and obtain new V_A and V_B values. Then, we update V_C's and hence current sources in (b)(c)(d), and solve the circuit in form (d) again. This process iterates until voltages converge.

The above iterative approach is guaranteed to converge, and the theoretical proof is provided in Appendix B. Empirically, it is observed that the iterative process converges within three iterations, when the convergence criterion is maximum voltage difference being less than 10^{-5}V. The results are reported in Section 4.4.

Again, the bookkeeping technique from Section 2.4.2 is used here: random walks are needed only in the first iteration of the first transient timestep, and in all subsequent iterations of all timesteps, the journey record is used instead to solve the circuit in form (d).

4.2 Locality

A desirable feature of the stochastic solver is that it localizes the computation, and for power grid analysis, this translates to calculating a single node voltage without having to solve the entire circuit. Such a property is especially meaningful when the designer knows which part of the power grid is problematic, or in the scenario of incremental design when the designer makes a minor change in the design and wishes to see the impact. Compared to a conventional approach that must solve the full set of matrix equations to find the voltage at any one node, the computational advantage of this method could be tremendous.

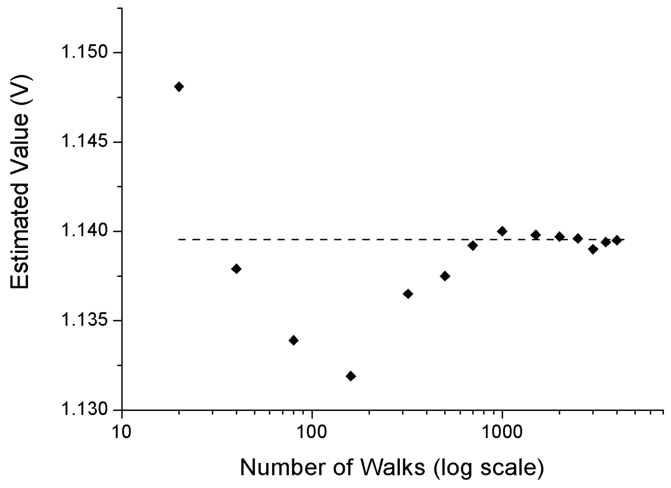


Figure 4.6: Estimated voltages at a single node for various values of M.

Figure 4.6 shows the results of computing the DC solution for only one node in an industrial power grid model, using the stochastic solver. The markers are estimated voltage values for different M's, where M is the number of walks used, and the dashed line is the true voltage. The ultra-accurate right-most point, for which $M = 4000$, takes only 0.42 second runtime, and thus shows the efficiency of using the stochastic solver to solve individual nodes.

The above locality in DC analysis is still valid in RC transient analysis: the stochastic solver can compute a single node voltage at a single time point, without solving any other nodes or any other timesteps. If we want to compute the voltage at node i at time t, the walks start at node i in the random walk game for time t; some walks may reach $V(t-h)$ terminals, and then they continue in the random walk game for time $(t-h)$; some of these may reach $V(t-2h)$ terminals, and then they continue in the random walk game for time $(t-2h)$, and so on. The real terminals where random walks end are those from physical voltage sources, which are present at all times. The farthest a random walk can go in time is the time point zero, which is a DC analysis game. In short, "travelling back time" makes such complete locality feasible in both space domain and time domain, and this is inspired by [5].

4.3 Hierarchies

In this section, the power grid analyzer from Section 4.1 is combined with a divide-and-conquer strategy to form hierarchical variations, which are faster and more robust on certain types of power grid. To distinguish between them, the method from Section 4.1, which is a direct application of the stochastic solver from Chapter 2, is referred to as the generic algorithm, and the new methods in this section are referred to as the hierarchical algorithms. The part of building hierarchy can also be a stand-alone network reduction algorithm, and is used in Chapter 6.

Section 4.3.1 and Section 4.3.2 present the hierarchical algorithms for DC analysis and transient analysis respectively. Section 4.3.3 discusses the advantages of these methods for power grid analysis, and Section 4.3.4 shows a few variations of hierarchy that are useful in practice.

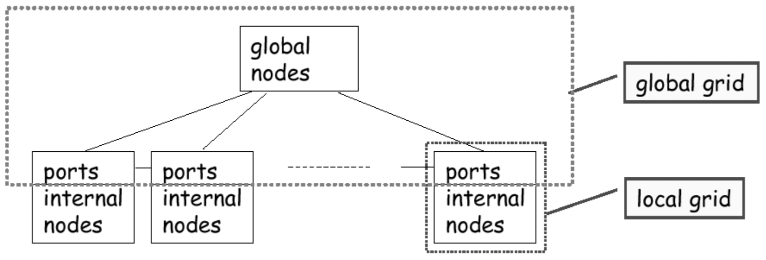


Figure 4.7: Hierarchical strategy in [89].

4.3.1 Principles

The hierarchical strategy in [89] is illustrated in Figure 4.7. The power grid is divided into a global grid and multiple local grids, and interfacing nodes are defined as ports. From the global perspective, the behavior of a local grid is completely described by the following equation.

$$\mathbf{I_{ports}} = A\mathbf{V_{ports}} + \mathbf{S} \tag{4.11}$$

where $\mathbf{I_{ports}}$ is the vector of currents flowing from the global grid into this local grid, $\mathbf{V_{ports}}$ is the vector of port voltages, A is a square matrix, and $\mathbf{S}$ is a constant vector. In DC analysis, matrix A represents the effective conductances between the ports, and vector $\mathbf{S}$ represents current sources inside the local grid.

The algorithmic flow of [89] is shown in Figure 4.8. First, macromodels, i.e., the A matrices and $\mathbf{S}$ vectors, are extracted from local grids. Next, the set of linear equations for the global grid is solved and port voltages obtained. Finally, local grids are solved individually.

An exact method for calculating A and $\mathbf{S}$ is provided in [89], and 0-1 integer linear programming (ILP) is used to make A sparse, at the expense of a bounded loss in accuracy. We now demonstrate a stochastic alternative to build A and $\mathbf{S}$, and to achieve sparsity naturally as a part of this procedure.

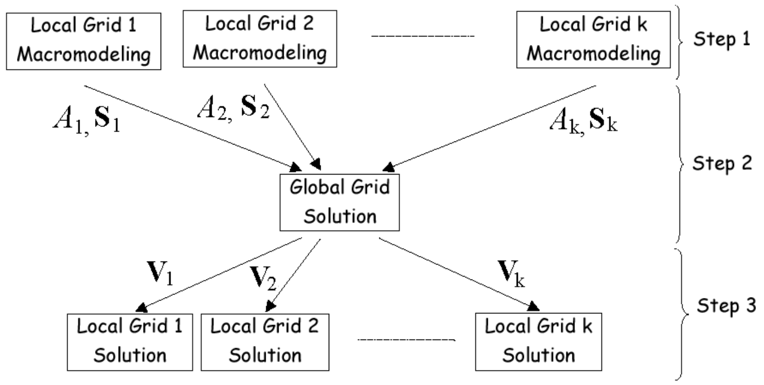


Figure 4.8: Algorithm flow in [89].

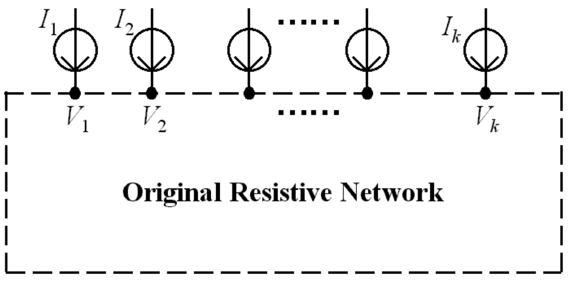


Figure 4.9: The original resistive network with external connections replaced by current sources.

Macromodeling is essentially a network reduction procedure applied on a local grid. Figure 4.9 illustrates a representation of the resistive network to be reduced. This network is composed of resistors, and only the ports have external connections, with port voltages $\mathbf{V_{ports}} = [V_1, V_2, \cdots, V_k]^T$ and port currents $\mathbf{I_{ports}} = [I_1, I_2, \cdots, I_k]^T$. In Figure 4.9, the external connections are replaced by symbolic current sources. This is justified by the fact that $\mathbf{V_{ports}}$ and $\mathbf{I_{ports}}$ are treated as algebraic symbols throughout the derivation of A and $\mathbf{S}$, and the equations apply to all possible values of $\mathbf{V_{ports}}$ and $\mathbf{I_{ports}}$. In fact, the goal of network reduction is to find a square matrix A and a constant vector $\mathbf{S}$ such that equation

(4.11) holds for all possible $\mathbf{V_{ports}}$ and $\mathbf{I_{ports}}$.

A random walk game is set up as follows:

- A set of M walks are run from a port i, $i \in \{1, 2, \cdots, k\}$, inside the network shown in Figure 4.9.
- The motel price at port i is $\frac{-I_i}{\sum_{l=1}^{\text{degree}(i)} g_l}$, where I_i is the port current, degree(i) is the number of resistors connected to port i inside the network, and g_l's are the conductances of these resistors. Note that I_i is symbolic, and any computation regarding this motel price is carried out symbolically.
- All of the ports are home nodes where random walks end, except for port i itself. In other words, these ports as absorbing nodes, while port i and non-port nodes are non-absorbing nodes. Therefore, a random walk cannot end at port i, and has to reach a port other than i to stop. The award for reaching a port is its port voltage. This value is unknown, and computation is carried out symbolically.
- The port currents other than I_i are at absorbing nodes, and therefore do not have corresponding motels. For constant current sources inside the network, each of them becomes a motel with the price $\frac{I_{\text{source}}}{\sum g}$, where I_{source} is the value of the current source flowing from a node to ground, $\sum g$ is the sum of conductances connected to that node.

For each individual walk in the above game, the money earned at the end of the walk is composed of an award, which is a port voltage, minus a sequence of motel expenses. The result of the q^{th} walk from node i (which has a current source of I_i) is therefore in the following form.

$$\chi_q = V_{\text{end}\,q} + J_{i,i,q} \frac{I_i}{\sum_{l=1}^{\text{degree}(i)} g_l} - \xi_q \tag{4.12}$$

where $q \in \{1, 2, \cdots, M\}$ is the index of the walk, $V_{\text{end}\,q}$ is the voltage at the port where the random walk ends, $J_{i,i,q}$ is the number of times that the walk passes port i, and ξ_q is the sum of all expenses paid at motels corresponding to internal (non-port) constant current sources. Note that ξ_q is a constant number, i.e., it is independent of $\mathbf{I_{port}}$ and $\mathbf{V_{port}}$.

Taking the average of the results from the M random walks, an estimate of V_i is obtained in the following form:

$$V_i = \frac{\sum_{q=1}^{M} \chi_q}{M} = \sum_{j\in\{1,\cdots,k\}, j\neq i} \frac{H_{i,j}}{M} V_j + \frac{J_{i,i}}{M} \cdot \frac{I_i}{\sum_{l=1}^{\text{degree}(i)} g_l} - \xi \tag{4.13}$$

$$\text{where} \qquad J_{i,i} = \sum_{q=1}^{M} J_{i,i,q} \qquad \xi = \frac{\sum_{q=1}^{M} \xi_q}{M}$$

$H_{i,j}$ is the number of walks that end at port j. $J_{i,i}$ is the total number of times that random walks pass port i. Note that ξ is also a constant number independent of $\mathbf{I_{port}}$ and $\mathbf{V_{port}}$, and because every random walk stops at a port that is not port i, $H_{i,j}$'s must satisfy the following condition.

$$\sum_{j\in\{1,\cdots,k\}, j\neq i} H_{i,j} = M \tag{4.14}$$

After obtaining the coefficients of equation (4.13) from the random walks, it can be converted into the following format by simple algebraic transformations.

$$I_i = \frac{M}{J_{i,i}} \sum_{l=1}^{\text{degree}(i)} g_l V_i - \sum_{j\in\{1,\cdots,k\}, j\neq i} \frac{H_{i,j}}{J_{i,i}} \sum_{l=1}^{\text{degree}(i)} g_l V_j + \frac{M\xi}{J_{i,i}} \sum_{l=1}^{\text{degree}(i)} g_l \tag{4.15}$$

Comparing equation (4.15) and equation (4.11), it can be seen that (4.15) estimates the i^{th} row in matrix A and the i^{th} entry in vector $\mathbf{S}$ as:

$$A_{i,i} = \frac{M}{J_{i,i}} \sum_{l=1}^{\text{degree}(i)} g_l \tag{4.16}$$

$$A_{i,j} = -\frac{H_{i,j}}{J_{i,i}} \sum_{l=1}^{\text{degree}(i)} g_l \tag{4.17}$$

$$S_i = \frac{M\xi}{J_{i,i}} \sum_{l=1}^{\text{degree}(i)} g_l \tag{4.18}$$

The above equations estimate the entries in A and $\mathbf{S}$ that correspond to a specific port i. For each port node of the network, such procedure is repeated and the matrix A is constructed row by row, and the vector $\mathbf{S}$ entry by entry. By equations (4.16) and (4.17), the diagonal entries of the estimated matrix A are positive, while off-diagonal entries are negative or zero, and using equation (4.14), the following equation can be easily proven.

$$\sum_{j=1}^{k} A_{i,j} = 0 \tag{4.19}$$

The sparsity-accuracy tradeoff of the reduced network is controlled by M, the number of random walks used. In equations (4.16) and (4.17), $A_{i,i}$ can be viewed as the total conductance from port i to other ports, and this amount is distributed among $(-A_{i,j})$'s such that each of them gets a fraction $\frac{H_{i,j}}{M}$, proportional to $H_{i,j}$, the number of walks from i to j. Hence, M can be considered as the resolution of the estimation. When M increases, the matrix A becomes denser and closer to the exact matrix, and is more expensive to compute.

Now we move on to step 2 in Figure 4.8, solving the global grid based on the extracted macromodels. The reduced resistive network has an equivalent form that is easier to visualize, by rewriting equation (4.15) in the following form:

$$I_i = S_i + \sum_{j \in \{1,\cdots,k\}, j \neq i} (-A_{i,j})(V_i - V_j) \tag{4.20}$$

Equation (4.20) can be viewed as a circuit, in which $(-A_{i,j})$ conductance connects port i to port j, and an independent current source S_i flows out of port i. This is an imaginary circuit, because each resistor only exists for one direction (corresponding to the asymmetry of the computed A matrix), i.e., the conductance from port i to port j could be different from the conductance from j to i. Figure 4.10 illustrates this imaginary circuit composed of directed resistors.

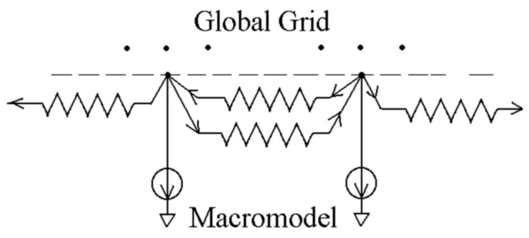


Figure 4.10: The imaginary circuit interpretation of a macromodel in DC analysis.

Based on this imaginary circuit interpretation, the global grid can be solved by running random walks from each port node, and the port voltages can be obtained.

Next, we move on to step 3 in Figure 4.8, solving the bottom-metal-layer nodes in each local grid based on the port voltages computed in step 2. The ports correspond to “homes” in this random walk game, and each walk from the bottom layer typically ends within a relatively small number of steps.

4.3.2 Transient Hierarchy

In transient analysis, due to the additional $V(t-h)$ terminals, the hierarchical algorithm from the previous section is changed in various ways. Since some random walks may stop at $V(t-h)$ terminals, equation (4.14) is no longer true, and is replaced by the following inequality.

$$\sum_{j\in\{1,\cdots,k\},j\neq i} H_{i,j} \leq M \tag{4.21}$$

Consequently, equation (4.19) is also replaced by an inequality:

$$\sum_{j=1}^{k} A_{i,j} \geq 0 \tag{4.22}$$

Equations (4.15), (4.16), (4.17), and (4.18) remain true, except that the constant ξ accounts for not only motel expenses corresponding to internal current sources, but also awards received when walks stop at $V(t-h)$ terminals.

Due to the new inequality (4.22), equation (4.15) can no longer be converted to (4.20), and equation (4.20) is replaced by the following.

$$I_i = S_i' + \sum_{j\in\{1,\cdots,k\},j\neq i} (-A_{i,j})\,(V_i - V_j) + \left(\sum_{j=1}^{k} A_{i,j}\right)\left(V_i - \frac{S_i' - S_i}{\sum_{j=1}^{k} A_{i,j}}\right) \tag{4.23}$$

The splitting of S_i into S_i' and $(S_i - S_i')$ is arbitrary. In our implementation, S_i' is chosen to be the portion from motel expenses at internal current sources, and $(S_i - S_i')$ is chosen to be the contributions from $V(t-h)$ terminals. Equation (4.23) maps to an imaginary circuit that is slightly different from Figure 4.10: conductance $\sum_{j=1}^{k} A_{i,j}$ connects port i to a voltage source with voltage value $\frac{S_i' - S_i}{\sum_{j=1}^{k} A_{i,j}}$; independent current source S_i' flows out of port i. Figure 4.11 illustrates this imaginary circuit for transient analysis.

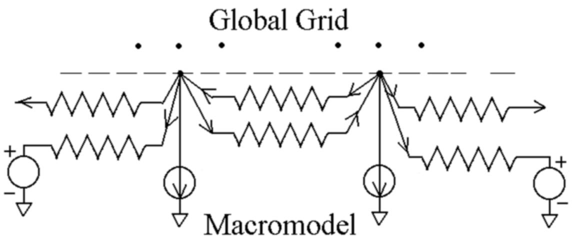


Figure 4.11: The imaginary circuit interpretation of a macromodel in transient analysis.

When applying the bookkeeping technique from Section 2.4.2, the hierarchical algorithm demands some extra storage: the journey record keeps information not only for solving the global grid and the local grids, but also for building vector **S**, because **S** needs to be updated in every timestep, whenever the current sources or $V(t-h)$ sources in the local grid change. In RKC transient analysis, **S** may need to be updated in every iteration of every timestep, depending on how detailed the inductance model is.

4.3.3 Benefits of Hierarchy

The approach in [89] requires small-cut partitioning of the power grid, since this leads to small port matrices. In our hierarchical approach, such partitioning is not necessary, and the algorithm only needs to distinguish local nodes, global nodes, and ports. Consequently, multiple local grids are not needed, and only the boundary between the global grid and the local grid needs to be defined. This can be done in various ways, and we recommend the following natural approach: given a power grid, a layer of vias is chosen as the border between the global grid and the local grid, the upper ends of these vias being ports.

Choosing such a layer of vias is a new degree of tradeoff: if a lower layer is chosen, the global grid size is larger, the number of ports is larger, and consequently solving the global grid takes more runtime; on the other hand, the local grid is smaller, there are more terminals, i.e., solved ports, and therefore solving the local grid takes less time. Empirically, a relatively good tradeoff point is choosing a layer of vias such that the global grid is roughly 10% of the entire circuit size.

Compared to the generic random-walk algorithm, the hierarchical algorithm has two major advantages:

- The hierarchical method is faster. The reason is illustrated in Figure 4.12. When solving the global grid, each random walk starts from a port and ends at a perfect voltage source; when solving the local grid, each random walk starts from a bottom layer node and ends at a port. In either case, a walk has fewer steps than a walk in the generic method that starts from a bottom layer node and has to reach a perfect voltage source at the top metal layer. Also, when random walks are shorter, the variance of the results of walks tends to be lower, and consequently, a higher accuracy can be achieved with the same number of walks, or fewer walks are needed to achieve the same

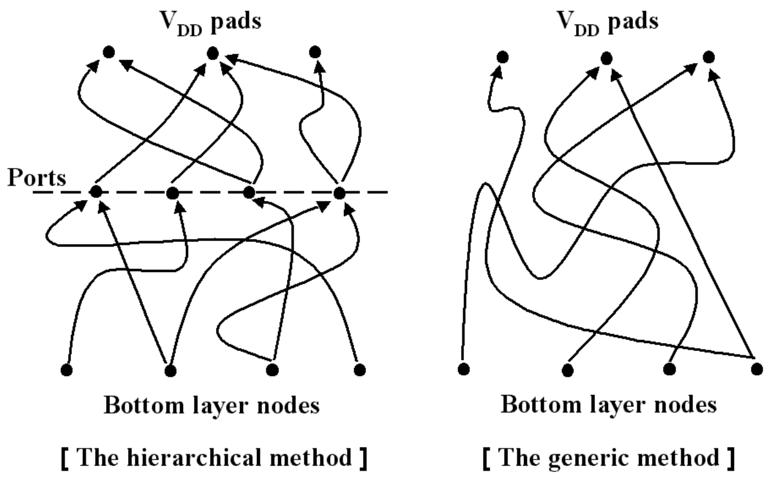


Figure 4.12: Random walks in the hierarchical algorithm are shorter than those in the generic algorithm.

accuracy level. Although the overhead of building macromodels is paid, the overall savings typically dominate this cost.

- The hierarchical method is more robust. In certain power grids, a highly resistive metal layer forms a barrier that makes it difficult for the walker to go up to the top layer, and the runtime of generic method can be long. The hierarchical method solves such circuits simply by defining ports right on this barrier. In other words, instead of relying on the random walker to pass this barrier, a walk is cut into two segments, and the barrier nature is preserved in the macromodel. This can also be viewed as an extreme case of the speedup shown in Figure 4.12.

Finally, we want to point out a drawback of hierarchy. In the hierarchical algorithm, it is no longer possible to solve a single bottom-metal-layer node only: the overhead of building and solving the hierarchy has to be paid first. In other words, the algorithm does not have the *complete* locality anymore. One way to maintain a *partial* locality is to use multiple local grids: when a change is made in the design, only the macromodel of the local grid containing the change needs to be rebuilt and re-solved.

4.3.4 Variations of Hierarchy

A natural extension of the hierarchical algorithm is to use multi-level hierarchy. Making use of all available vias, we can build macromodel on top of macromodel. After this bottom-up traversal, the circuit is reduced to a global grid, then port voltages are solved in a top-down order, and the bottom layer voltages are obtained in the end. Compared with the single-level method, the extra cost of the multi-level method is building multiple macromodels, while the benefit is shorter walks in each level. Hence there is a tradeoff in choosing the number of levels. Since the single-level hierarchical method is better than generic method, the multi-level method is expected to be even faster and more robust. Test results in Section 4.4 show that the multi-level method has a similar accuracy-runtime tradeoff as the single-level method.

Another extension of the hierarchical method leads to the concept of a "virtual-layer," when ports are chosen such that the global grid physically does not exist. In other words, there are no direct connections between these ports in the original circuit. This can be considered to be similar in flavor to grid coarsening in [47]. When all connections of these ports are abstracted into a macromodel, this macromodel provides imaginary connections between ports, and the global grid is totally composed of such virtual connections, as illustrated in Figure 4.13.

For example, in a large power grid where the number of voltage sources is very limited and they are located at periphery, a random walk from a center node typically needs a very large number of steps. We may traverse the graph (for

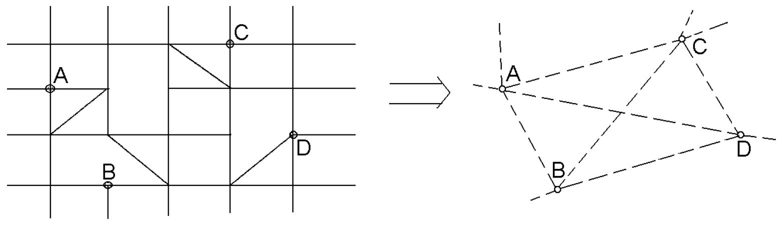


Figure 4.13: The original graph with ports marked, and the extracted virtual layer.

example, a breadth-first search in the implementation), and mark one port in every l nodes. For example, if $l = 10$, the sampling rate is 1/10. Special arrangements must be made such that each home is surrounded by ports, because edges leading to a home should not be abstracted into the macromodel. Then all connections of these ports are abstracted into a macromodel, except for those leading to a home. Thus the virtual layer is constructed and the size is roughly 10% of the original graph. After solving it, we go back to the local grid, i.e., the original graph, and because there are solved ports all over the graph, it can be solved efficiently.

This virtual-layer method will be shown useful when solving a wire-bond power grid in the next section, and for ESD simulation in Chapter 6.

4.4 Simulation Results

In this section, three industrial benchmarks are used to evaluate the proposed algorithms for DC analysis. Then, artificial RC and RKC circuits generated based on real-life structures are used to test the performance of transient analysis. Computations are carried out on a Linux workstation with 2.8GHz CPU frequency.

The three industrial power grids are:

- Industry1 is a 70,729-node circuit, and we solve for the 15,876 bottom-metal-layer VDD nodes and 15,625 bottom-metal-layer ground nodes. The voltage range of VDD bottom layer is 1.1324–1.1917V.

- Industry2 has 218,947 nodes, in which 25,137 bottom-metal-layer VDD nodes and 18,803 bottom-metal-layer ground nodes are to be solved. The voltage range of VDD bottom layer is 1.61248–1.79822V, that of ground bottom layer being 0.000334–0.066505V.
- Industry3 is a wire-bond ground net with 347,566 nodes, and the bottom layer has a voltage range of 0.024347–0.110860.

One implementation issue is that, in order to avoid any possible deadlock, a limit is put on the number of steps in a walk. Any walk that fails to end within this limit is forced to end, and be awarded VDD if inside the VDD net, be awarded 0 if inside the ground net. This operation is optimistic and will results in a bias in the estimated voltage; however, if the limit is chosen appropriately, the error will be very small as the probability of an overlength walk is minute. Thus a new degree of accuracy-runtime tradeoff is introduced, and this limit is empirically chosen to be 10,000 steps for power grid analysis as a good tradeoff point. For hierarchical methods, there are typically no or only a few violations of the step limit.

The above tradeoff only affects runtime indirectly, while the error margin Δ in Equation (2.8) decides M, which is directly proportional to runtime and needs careful investigation. Figure 4.14 plots the relation between Δ and runtime for solving the complete Industry1, i.e., finding all bottom-metal-layer voltages, using the generic random-walk method. The runtime is always larger than 8 seconds because the minimum value of M is set to be 40. The lower part of this curve shows the quadratic relation between M and Δ: $M \propto \frac{1}{\Delta^2}$. For example, the runtime is around 15 seconds when Δ is 4mV, and roughly 60 seconds when Δ is 2mV.

Figure 4.15 plots the tradeoff between average error and runtime in solving

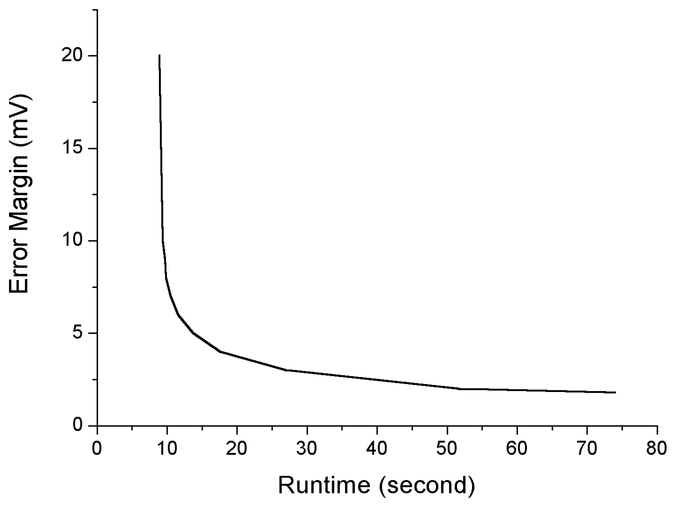


Figure 4.14: Runtime-Δ tradeoff for the computation of all bottom-metal-layer nodes in Industry1.

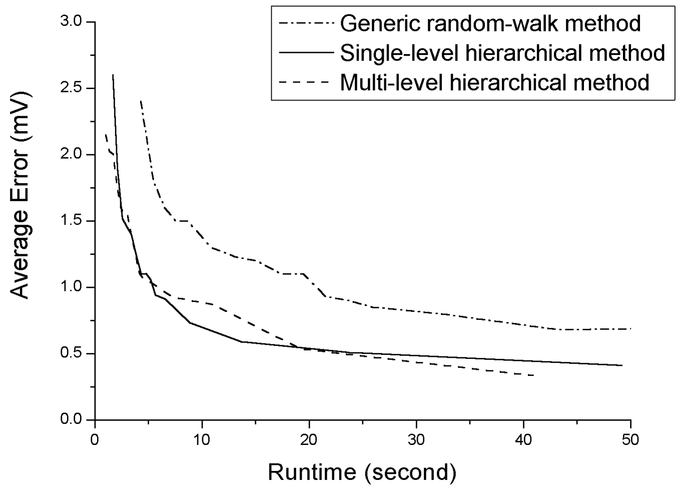


Figure 4.15: Accuracy-runtime tradeoff curves for solving Industry1 using the generic random-walk method, the single-level hierarchical method, and a two-level hierarchical method.

Industry1, where the three curves are for the generic random-walk method, the single-level hierarchical method, and a two-level hierarchical method, respectively. All hierarchies are divided at vias. All three methods use pre-determined and fixed M in each run, and points on the curves correspond to different M values. Both hierarchical methods achieve roughly 3–4 times speedup over the generic method, with the same average error.

In practice, the user decides the tradeoff point by choosing M values according to the needs of the analysis. Here for runtime comparison purpose, we choose a reasonable tradeoff point on each of the three curves, and list them in Table 4.1.

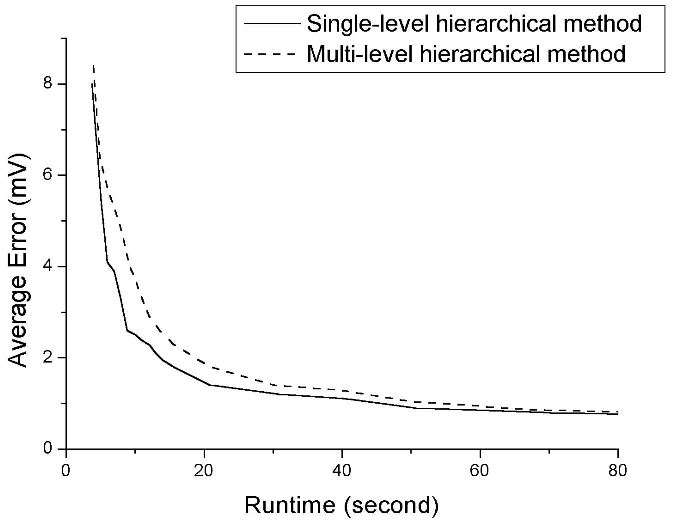


Figure 4.16: Accuracy-runtime tradeoff curves for solving Industry2, using the single-level hierarchical method and a three-level hierarchical method.

Figure 4.16 plots the tradeoff between average error and runtime in solving Industry2, using the single-level hierarchical method and a three-level hierarchical method. All hierarchies are divided at vias. Both methods use pre-determined and fixed M in each run, and points on the curves correspond to different M values. The curve for the generic random-walk method is omitted because its runtime is unacceptably high for this circuit. The reason is that a highly resistive metal layer on top of low-resistance vias forms a barrier structure. This circuit shows an example of the robustness introduced by hierarchical methods. Again, the tradeoff point should be decided by the designer. Here we choose a reasonable tradeoff point on each of the curves, and list them in Table 4.1. One tradeoff point of the generic method is also listed.

Table 4.1: DC analysis comparison. N is the circuit size, E1 is the average error, E2 is the max error, T is the runtime, NT is the normalized runtime, defined as the runtime per thousand nodes, P is the peak memory, and NP is the normalized peak memory, defined as the peak memory per thousand nodes. G denotes the generic random-walk method, S denotes the single-level hierarchical method, and M denotes the multi-level hierarchical method.

Benchmark		N	E1(mV)	E2(mV)	T	NT(sec)	P(MB)	NP(MB)
Industry1	G	71K	**1.1**	**9.8**	**17.40 sec**	**0.245**	**10.7**	**0.15**
	S		**1.1**	**6.6**	**4.34 sec**	**0.061**	**11.4**	**0.16**
	M		**1.1**	**9.4**	**4.16 sec**	**0.059**	**16.8**	**0.24**
Industry2	G	219K	10.9	142.2	329.57 sec	1.50	27.3	0.12
	S		**1.4**	**30.7**	**20.82 sec**	**0.095**	**37.0**	**0.17**
	M		**1.4**	**35.3**	**30.12 sec**	**0.138**	**41.4**	**0.19**
Industry3	G	348K	4.3	7.6	71 min	12.2	57.7	0.17
	S		4.4	18.8	498.02 sec	1.43	72.4	0.21
	M		**3.6**	**17.0**	**93.64 sec**	**0.27**	**84.6**	**0.24**
Chip2 by the method of [89]		*2.7M*	*N/A*	*N/A*	*25 min*	*0.56*	300	0.11

The runtime comparison is shown in Table 4.1. The three rows Industry2-G, Industry3-G and Industry3-S, are results with robustness problems, while the boldface rows are results without the problems, or with them overcome. The numbers for chip2 in [89] are listed as a baseline. In viewing the numbers, it is important to note that our computer is approximately 3 times faster than those used by [89], according to SPEC benchmarks [77]. Runtimes reported by [89] show superlinear time complexity; chip2 is their smallest circuit, and therefore has the smallest normalized runtime. Since the time complexity of random-walk algorithms is linear in circuit size (for circuits with similar structure, according to Section 2.5), as power grid size increases, they will outperform [89] more. Note that due to factors such as benchmark structure, coding, compiling, and platform difference, this is only an approximate comparison, even after considering the speed factor of 3.

For Industry1, both hierarchical methods show a 4 times speedup over the generic method. For Industry2, the speedup is dramatic, and shows the robustness introduced by hierarchical methods.

The multi-level hierarchical method does not show a runtime advantage over the single-level method for Industry2. The reason is that, the benefit of the multi-level hierarchy, which is easier access to home nodes, is not worth the cost of building multiple macromodels, for the Industry2 case with C4 packaging. However, it is worthwhile for Industry3, a similar-sized circuit with wire-bond packaging.

Industry3 is a wire-bond power grid, a difficult circuit type to solve. Even after it is reduced to its top metal layer only, there are still 80K nodes, yet there are only 20 perfect voltage sources distributed on four sides of the top metal layer. Thus it requires high runtimes if using the generic method or the single-level method, as listed in Table 4.1. We employ a two-level hierarchical method, the top level being a virtual layer, as discussed in Section 4.3.4. This scheme solves this benchmark in a reasonable amount of time, with acceptable error. The results are listed in Table 4.1, and the normalized runtime is seen to be higher than solving other circuit types.

In order to evaluate the transient analysis, since we were unable to obtain real-life RC/RLC power grid circuits, we generated four circuits with realistic parameters. RC1 and RC2 listed in Table 4.2 are RC networks based on the structure of Industry1. RKC1 and RKC2 listed in Table 4.3 are RKC networks based on the structure of Industry2. Inductances are assumed to be only in the top two metal layers, and are estimated using formulas provided by [27]. Then K matrices are constructed by the method proposed by [16], using 7-by-7 and 7-by-5 window sizes for the two metal layers. Current-load waveforms are designed such that inductive effect is visible: simulation using a direct solver shows that if inductors in circuit RKC1 are ignored, the induced error is up to 21mV.

Table 4.2: RC transient analysis results. N is the circuit size, TS is the number of timesteps, T is CPU time per timestep for subsequent timesteps, E1 is the average error, E2 is the max error, and P is the peak memory. G denotes the generic random-walk method, and S denotes the single-level hierarchical method.

Ckt		N	TS	T(sec)	E1(mV)	E2(mV)	P(MB)
RC1	G	3.7K	500	0.0026	1.6	11.9	–
	S			0.0014	2.0	13.7	–
RC2	G	2.3M	1000	0.65	N/A	N/A	680
	S			0.64	N/A	N/A	854

The results of RC analysis using both the generic method and the hierarchical method are shown in Table 4.2. CPU times are measured for the timesteps that follow the initial DC analysis and the first transient step. The solution for circuit RC1 is compared with HSPICE, while circuit RC2 is too large to be simulated in HSPICE. Note that E1 is the average over all nodes at all timesteps, and E2 is the maximum over all nodes at all timesteps. The peak memory numbers are small for

RC1, and are omitted. The runtimes are several times faster than traditional direct solver runtimes reported in [89], even after normalization by the speed factor of 3. The space complexity is higher for the hierarchical method, because bookkeeping is needed not only for the bottom-metal-layer nodes, but also for building and solving the global grid. However, the peak memory of the hierarchical method is still lower than that of traditional methods reported in [89], in terms of memory consumption per million nodes.

Table 4.3: RKC transient analysis results. N is the circuit size, TS is the number of timesteps, T is CPU time per timestep for subsequent timesteps, E1 is the average error, E2 is the max error, and P is the peak memory.

Ckt	N	TS	T(sec)	E1(mV)	E2(mV)	P(MB)
RKC1	6.4K	1000	0.0165	0.8	13.9	–
RKC2	642K	1000	2.1	N/A	N/A	837

The results of RKC analysis are shown in Table 4.3. The single-layer hierarchical method is used, and the algorithm discussed in Section 4.1.3 is used when solving the global grid with inductors. Note that inductances are assumed to be only in the top two metal layers, and hence only in the global grid. CPU times are measured for the timesteps that follow the initial DC analysis and the first transient step. The solution for circuit RKC1 is compared with a direct solver, while circuit RKC2 is too large to be simulated by a direct solver. Note that E1 is the average over all nodes at all timesteps, and E2 is the maximum over all nodes at all timesteps. The peak memory is small for RKC1, and is omitted. Comparing with Table 4.2, it is apparent that RKC analysis has higher time and space complexity than RC analysis. This is due to the extra storage for mutual inductances, and the extra iterations of computation.

When viewing Tables 4.2 and 4.3, one common concern is error accumulation: although the error of one timestep is low, it could add up to large error over many timesteps. This concern drives us to measure E1 and E2. Note that E1 is the average over all timesteps, and E2 is the maximum over all timesteps. They suggest that the errors are acceptable after 500/1000 timesteps. Practically, this implies that errors tend to cancel each other, and that the error accumulation has a very slow rate.

Chapter 5

Application in Early-stage Power Grid Analysis

This chapter investigates the application of the stochastic linear equation solver from Chapter 2 on early-stage power grid analysis. This is a different problem from Chapter 4 in that the current loads, i.e., the right-hand-side vector, are unknown. In fact, the goal of such early-stage analysis is to identify the worst possible current load pattern in terms of VDD loss.

5.1 Problem Statement

As discussed at the beginning of Chapter 4, as integrated circuits growing more and more power-intensive, power grid analysis is needed at all stages of the design cycle. Most of the existing works on this subject, [12] [47] [71] [81] [89], including the methods from Chapter 4, deal with the deterministic analysis of a power grid for a complete design. In other words, they assume that the current loads at bottom-layer nodes are given, and power grid analysis is performed subsequent to this. On the other hand, [45] proposes to perform analysis without deterministic current loads, and instead, uses current constraints to limit possible working modes, formulating a linear programming problem to find the worst voltage drops.

This chapter is motivated by two issues that have not been adequately addressed by prior works:

- To efficiently model *uncertain working modes.* Modern designs operate under a number of power modes, in each of which a different set of blocks may be on. This uncertainty can exert a large influence on power grid performance. The current loads in our work are modeled not as constants, but as functions of the working mode of the circuit, and we look at power grid analysis for these uncertain loads, to find the worst-case scenario associated with the largest voltage drop. To do so, we utilize information that was not used in deterministic analysis. For example, power budget is an increasingly useful piece of information that can be used during analysis when circuits operate under multiple power modes.
- To perform *early-stage analysis.* The most effective fixes to the power grid must be made *early* in the design cycle, when much of the details of the design are unknown. If one waits until later in the design flow, the number of available degrees of freedom for optimization reduces dramatically. This implies that it is important to analyze the grid early in the design process; however, the side-effect of this is that such analyses must operate under some uncertainty.

The information that is available at an early stage, e.g. after floorplanning, is that the circuit is composed of a number of functional blocks whose positions are known. For example, an SRAM block on the circuit can be modeled by one current source distributed over hundreds of power grids nodes. One may determine a reasonable estimate for current consumed by each block, and based on the position of a block, its proximity to VDD/GND pads is known. In different working modes, some of the blocks are active and consuming current, while others are standing by. The number of working modes for the circuit may be very large (potentially exponential in the number of blocks), and it is often not possible to enumerate all such modes.

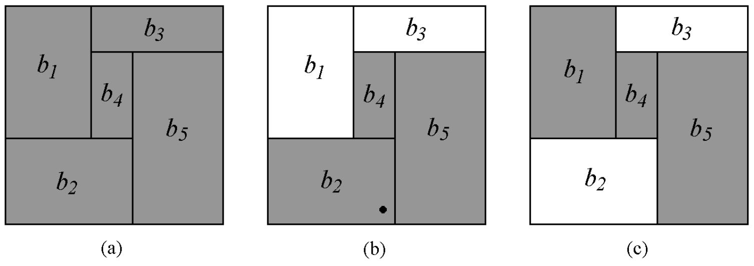


Figure 5.1: A small example: (a) the floorplan with five blocks, (b) the working mode that causes the largest single-node voltage drop, with the location of the largest voltage drop marked with a black dot, and (c) the working mode that causes the largest average voltage drop.

One way to deal with this uncertainty is to perform a worst-case analysis assuming that every block is on. This is too pessimistic and produces false alarms, since such a working mode may never occur. Figure 5.1 shows a small illustrative example to show the necessity of analyzing realistic working modes instead of making this pessimistic assumption. It is an artificial regular-structured power grid with 5884 nodes, four metal layers, and four VDD pads at the top metal layer. Each of the 2601 bottom-metal-layer nodes has a 2mA current load. The current loads are drawn by five functional blocks, as shown in Figure 5.1(a), and each current load is on when the corresponding block is active. If we assume all five blocks are on, the total power is 6.2W (the nominal VDD being 1.2V), the largest single-node voltage drop is 135mV, and the average voltage drop over all bottom-metal-layer nodes is 114mV. However, if we have the knowledge that the maximum power of the circuit is 4W, then not all working modes can occur. By enumerating all possible working modes that obey the 4W constraint, Figure 5.1(b) is found to be the working mode that causes the largest single-node voltage drop, 99mV at the location indicated by the black dot, and Figure 5.1(c) is found to be the working mode that causes the largest average voltage drop, 73mV. Hence, for this small example, the pessimistic analysis overestimates the voltage drop by 36mV, and could produce false alarms.

Constraints are used to limit the analysis to working modes that are more likely to occur, and to find the worst among them. Examples of these constraints are:

- A power-limit constraint indicates that a design cannot consume more than a certain amount of power $P_{\max}$. This number can be provided by the power budget that is set as a constraint early in the design process.
- A synchronization constraint demands that two blocks always work together.
- An exclusivity constraint provides that only one of two RAM blocks may be accessed at a time, or that only one of three ALUs is active at a time, etc.

Under such constraints, the worst case working mode needs to be found, in terms of either the largest single-node voltage drop, or in terms of the largest average voltage drop. If the specified voltage drop design goal is violated, this must be fixed by assigning more routing resources to the power grid and/or moving certain blocks apart from each other. This type of early-stage optimization can substantially reduce the risk of later optimizations that may require expensive rip-up-and-reroutes.

Another example where the analysis of a power grid under uncertainty is also meaningful is the case when there is a critical noise-sensitive block in the design. For example, a phase-locked loop is sensitive to VDD noise, i.e., sensitive to the working mode of circuit units around it, and requires careful analysis [20]. In this case, the scenario that causes the largest voltage drop at these specific nodes must be found to guarantee correct analysis of the unit.

The discussion is based on DC analysis, as little is known about circuit waveform details at the early stage, and it is impractical to perform transient analysis. The DC analysis of a GND net is formulated as [36]:

$$G\mathbf{V} = \mathbf{I} \tag{5.1}$$

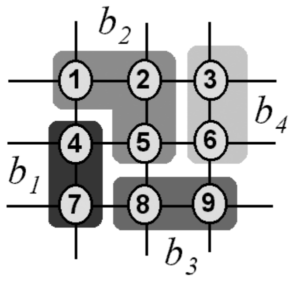
where G is the conductance matrix for the interconnected resistors, $\mathbf{V}$ is the vector of node voltages, and $\mathbf{I}$ is the vector of current loads. For a VDD net, the right-hand-side vector also contains perfect VDD sources, but if we look at the voltage drops, i.e., if we subtract every entry in $\mathbf{V}$ by VDD and reverse its sign, the formulation becomes the same as equation (5.1).

To investigate variations in load vector $\mathbf{I}$, we must account for the origin of current loads. In reality, vector $\mathbf{I}$ is composed of contributions from functional blocks, and can be formulated as:

$$\mathbf{I} = F \cdot \mathrm{diag}(\mathbf{w}) \cdot \mathbf{I_b} \tag{5.2}$$

where $\mathbf{I_b}$ is a vector of block currents with length K, $\mathbf{I}$ is the vector of current loads with length N, F is an N-by-K matrix, $\mathbf{w}$ is a vector with length K and entries being 0 or 1, and diag($\mathbf{w}$) is a K-by-K diagonal matrix with diagonal entries equal to the entries in $\mathbf{w}$.

At an early stage of the design, only block-level estimates of the currents are available. Since these blocks are large and may cover many nodes of the power grid, typically $K \ll N$. The matrix F is an incidence matrix that describes the distribution of block currents, with each column corresponding to a block, such that the sum of all entries in a column is one. Figure 5.2 shows a small illustrative



$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7 \\ I_8 \\ I_9 \end{bmatrix} = \begin{bmatrix} & 0.3 & & \\ & 0.3 & & \\ & & & 0.2 \\ 0.5 & & & \\ & 0.4 & & \\ & & & 0.8 \\ 0.5 & & & \\ & & 0.3 & \\ & & 0.7 & \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 0 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} I_{b_1} \\ I_{b_2} \\ I_{b_3} \\ I_{b_4} \end{bmatrix}$$

Figure 5.2: A small example of equation (5.2).

example with the number of blocks being $K = 4$ and the number of power grid nodes being $N = 9$. The first column of F indicates that, the current drawn by block b_1 is distributed among node 4 and node 7 in the power grid, each with 50% and 50% of the total block current I_{b_1} respectively; the second column indicates that block b_2 is supplied by nodes 1, 2 and 5 by 30%, 30% and 40% respectively, etc. In reality, the block size is much larger: for instance, an SRAM would be distributed over hundreds of sink nodes for the power grid. In the early design stage, matrix F can be constructed by assuming uniform distribution of block currents among nodes of each block, or, if we have more specific knowledge of the structure of a block, certain patterns can be assumed in the corresponding column of matrix F.

Each entry in $\mathbf{I}$ could consist of contributions from more than one block, because each bottom-layer node in the power grid typically provides power for multiple logic gates that could belong to different functional modules. Therefore, different columns of matrix F can overlap with each other. Also, leakage current contributions can be considered as a block that is always on and contributes to every entry in $\mathbf{I}$.

If all blocks were always on, then all entries in $\mathbf{w}$ are 1. However, such mode is typically not feasible, as dictated by the power-limit constraint and exclusivity constraints. In a more realistic case, $\mathbf{w}$ is a switch vector with an entry being 1 if the corresponding block is on and 0 otherwise. Different $\mathbf{w}$ vectors represent different working modes of the circuit, and hence model the source of uncertainty. In Figure 5.2, $\mathbf{w} = [1, 1, 0, 1]^{\mathrm{T}}$, describing a working mode that blocks b_1, b_2 and b_4 are active, while block b_3 is off. For Figure 5.1(b), $\mathbf{w} = [0, 1, 0, 1, 1]^{\mathrm{T}}$; for Figure 5.1(c), $\mathbf{w} = [1, 0, 0, 1, 1]^{\mathrm{T}}$.

The above model is similar to [45] in terms of using upper bounds to constrain the maximum current drawn, but differs in the following: [45] uses a matrix to model current constraints provided by the designer, and formulates a continuous linear programming problem, where the variables are N normalized node voltages; our model accounts for the origin of uncertainty and use the K variables in $\mathbf{w}$ as the 0-1 integer variables to be optimized.

The solution to the system of equations (5.1) is therefore:

$$\mathbf{V} = G^{-1} \cdot F \cdot \mathrm{diag}(\mathbf{w}) \cdot \mathbf{I_b} \tag{5.3}$$

The objective is to find the vector $\mathbf{w}$ that causes the largest value in solution vector $\mathbf{V}$, in terms of either its maximum entry or the average of its entries, under certain constraints. The i^{th} entry in equation (5.3) can be written as

$$V_i = \sum_{j=1}^{K} c_j w_j I_{b_j} \tag{5.4}$$

where K is the number of blocks; w_j is the j^{th} entry of vector $\mathbf{w}$, with value 1 when the j^{th} block is on, 0 when it is off; I_{b_j} is the j^{th} entry of vector $\mathbf{I_b}$, i.e., the total current of the j^{th} block; c_j's are constant coefficients from equation (5.3).

The power-limit constraint becomes:

$$\mathbf{w}^{\mathrm{T}} \cdot \mathbf{I_b} \leq \frac{P_{\max}}{V_{DD}} \tag{5.5}$$

A synchronization constraint of multiple blocks being on and off together can be incorporated by considering these blocks as one single block, although they might be physically apart from each other. An exclusivity constraint that specifies that at most l' out of l blocks is active can be written under this notation as

$$w_{j_1} + w_{j_2} + \cdots + w_{j_l} \leq l' \tag{5.6}$$

where $j_1, j_2, \cdots, j_l$ are indices of those blocks.

The early-stage analysis can now be set up as an integer linear programming (ILP) problem as follows:

$$\begin{array}{rrcl} \text{maximize} & V_i & = & \displaystyle\sum_{j=1}^{K} c_j w_j I_{b_j} \\ \text{subject to} & \mathbf{w}^{\mathrm{T}} \cdot \mathbf{I_b} & \leq & \dfrac{P_{\max}}{V_{DD}} \\ & w_{j_1} + w_{j_2} + \cdots + w_{j_l} & \leq & l' \end{array} \tag{5.7}$$

Note that synchronization constraints are already implicitly included in assigning the blocks.

So far we have been dealing with the situation where each block has only two modes: it is either off or consuming a current amount given by the corresponding entry in $\mathbf{I_b}$. If we consider the case where each block has multiple working modes, when some blocks are consuming maximum currents, others may be also on, but in a low-consumption working mode. This can be modeled by multiple $\mathbf{I_b}$ vectors that represent possible patterns. By constructing and solving an ILP problem (5.7) for each $\mathbf{I_b}$ vector, we have a set of worst case V_i values, and the largest one among them is the real worst case for this node.

Conceptually, the worst case working mode of the entire circuit can be determined as follows. After constructing and solving the ILP formulation for every entry in vector $\mathbf{V}$, a worst-case $\mathbf{w}$ vector can be found for every node in the circuit, as well as its worst-case voltage drop. Then, if we pick the largest among these voltage-drop values, the corresponding $\mathbf{w}$ vector is the worst-case working mode for the whole circuit, in terms of the largest single-node voltage drop. If we are interested in the average voltage drop, we can use the sum of equation (5.4) from all nodes as the object function, and solve the resulting ILP problem for the worst case $\mathbf{w}$ vector.

The large size of G is one reason that affects the evaluation of equation (5.4), since the coefficients c_j require a knowledge of G^{-1} and are expensive to compute. Secondly, when the number of blocks is large, the dimension of $\mathbf{w}$ is correspondingly large, and the number of integer variables may be prohibitive for an ILP solver. For these reasons, it is impractical to construct equation (5.4) for every node and use an ILP solver to find the exact solution. In the next section, we propose a heuristic method to find a near-worst $\mathbf{w}$ vector.

5.2 Proposed Solution

As mentioned earlier, there are two issues that need to be resolved in order to find a fast solution:

- The cost function, equation (5.4), needs to be constructed without the knowledge of G^{-1}. In other words, we need to find V_i, the voltage drop at node i, as a linear function of block currents, without inverting matrix G. The generic power grid analyzer from Chapter 4 is capable of such computation, and is the basis of the proposed algorithm.
- The worst-case **w** vector needs to be found without using an ILP solver. A greedy heuristic is used in the proposed algorithm for this purpose.

Apply the mapping in Section 4.1.1 on equation (5.1), the award for reaching home is zero, and the estimated $f(i)$ is essentially the average motel expenses in one walk from node i. Thus, the estimated voltage V_i can be written as

$$V_{\text{estimate}} = \frac{\sum_{\text{motel}\,k} J_{i,k} m_k}{M} \tag{5.8}$$

where M is the number of random walks, m_k is the price of motel k, and $J_{i,k}$ is the number of times that walks pass node k. Applying equation (4.4), we can rewrite equation (5.8) as a linear function of current loads:

$$\begin{aligned} V_{\text{estimate}} &= \frac{\sum_{\text{node}\,k} \xi_k I_k}{M} \\ \text{where } \xi_k &= \frac{J_{i,k}}{\sum_{l=1}^{\text{degree}(k)} g_l} \end{aligned} \tag{5.9}$$

Then we substitute equation (5.2) into equation (5.9), and equation (5.9) becomes a linear combination of block currents:

$$\begin{aligned} V_{\text{estimate}} &= \frac{\sum_{j=1}^{K} \beta_j w_j I_{b_j}}{M} \\ \text{where } \beta_j &= \sum_{\text{node}\,k} \xi_k F_{k,j} \end{aligned} \tag{5.10}$$

Here, w_j and I_{b_j} are as defined in equation (5.4), $F_{k,j}$ is the (k, j) entry of matrix F. Equation (5.10) is an approximation to (5.4), the cost function of the ILP formulation.

Intuitively, the above computation enumerates current paths and compute the influence of the blocks on the voltage drop at node i. Not all current paths are considered, and those with larger influence on the voltage drop are more likely to be chosen. The resulting equation (5.10) essentially describes the importance of each block.

Now the goal of the ILP problem is to find the $\mathbf{w}$ vector that maximizes equation (5.10) under the constraints (5.5) and (5.6). Since the coefficients $\{\beta_1, \beta_2, \cdots, \beta_K\}$ are weights in equation (5.10), activating blocks with large β's is likely to cause large voltage drop at node i. However, the "likely" may not be true when competing blocks are involved in an exclusivity constraint (5.6): for example, if only one of the two blocks b_1 and b_2 can be on at a time, and if $\beta_1 > \beta_2$, but $\beta_1 I_{b_1} < \beta_2 I_{b_2}$, then the choice is complicated.

The flow of proposed heuristic algorithm is as follows:

1. Run a certain number of, e.g. 10, random walks from node i. Instead of calculating the walk results, we keep track of the power grid nodes visited.
2. By the procedure from equation (5.8) to (5.9) (5.10), obtain coefficients $\{\beta_1, \beta_2, \cdots, \beta_K\}$.
3. Sort $\{\beta_1, \beta_2, \cdots, \beta_K\}$. Repeat the above process until this sorted sequence does not change any more, according to a stopping criterion described at the end of this section.
4. Greedily activate blocks one by one. Each time, activate the block with the largest β coefficient, which does not violate constraint (5.5) or (5.6), and which falls into one of the following three categories:

- The block has no exclusivity constraint.
- The block has exclusivity constraint(s), but activating it does not close any constraint, i.e., the constraint(s) allows at least one more later block.
- The block has exclusivity constraint(s), and it has the largest βI_b product among all inactive blocks that are involved in the constraint(s).

The greedy heuristic stops when no more blocks can be added to the active block list.

For the example in Figure 5.1, if the node marked with the black dot is chosen to be node i, and the above algorithm is performed, the final β ordering from step 3 is $\{\beta_2, \beta_5, \beta_4, \beta_1, \beta_3\}$. Because there is no exclusivity constraint (5.6) in this case, step 4 activates block b_2, then block b_5, then block b_4, and stops because no more block can be added without violating (5.5). Thus, the pattern in Figure 5.1(b) is found. (Because the black-dot location is not known beforehand, the algorithm has to be performed for every node in order to find the largest single-node voltage drop.)

Note that the first three steps of the algorithm are independent of the actual current loads of the blocks. In the case where multiple $\mathbf{I_b}$ vectors are considered, the algorithm only needs to go through the first three steps once, and simply repeats step 4 for each $\mathbf{I_b}$ vector. Thus the extra computational cost is low.

The generated $\mathbf{w}$ vector is a near-worst-case $\mathbf{w}$ vector, in terms of the voltage drop at node i. When entries in vector $\mathbf{I_b}$ have different values, this problem is similar in flavor to the NP-hard bin-packing problem [13], in the sense of finding a number of blocks with various current consumptions to fit into a fixed power budget, although there is a different optimization goal that is to maximize another linear function (5.10) of the chosen block currents. The proposed heuristic does not guarantee optimality. However, since at the floorplanning stage, entries in vector

$\mathbf{I_b}$, i.e., block currents, have similar order of magnitude, it is likely that the degree of suboptimality is minor.

The above process provides a heuristic that aims to find the worst-case $\mathbf{w}$ vector for a specific node in the power grid. This procedure can be adapted for several global objectives:

- If the objective is to find the worst-case $\mathbf{w}$ vector that causes the largest *maximum* voltage drop in the whole circuit over all working modes, we can apply the heuristic to every node, and then, among the $\mathbf{w}$ vectors and voltage-drop values obtained, we pick the largest voltage-drop and its associated $\mathbf{w}$ vector.
- If the objective is to find the worst-case $\mathbf{w}$ vector that causes the largest *average* voltage drop in the circuit, we can modify step 1 of the heuristic to run a random walk from every node in the circuit, and the outcome would be the near-worst-case $\mathbf{w}$ vector for average voltage drop.

In any case, a stopping criterion is required for step 3 of the proposed heuristic. The implementation checks convergence after every 10 random walks, and look at a portion of the sorted β sequence, starting from the largest β's, such that the sum of the corresponding block currents is equal or less than $\frac{2P_{\max}}{V_{DD}}$. If this portion of the sorted sequence does not change after 10 walks, the algorithm claims convergence. When the number of blocks is large, it takes a long time to converge when there is no change in the sorted sequence. However, our primary interest is which blocks are active, and not the precise significance ranking of each block. Therefore, we loosen the stopping criterion to save unnecessary runtime, by defining a tolerance as $\lceil \frac{K}{20} \rceil$. If the position change of every block after 10 walks is less than $\lceil \frac{K}{20} \rceil$, the algorithm claims convergence.

5.3 Simulation Results

We use an industrial power grid, GSRC floorplans [28], and MCNC floorplans [57] to evaluate the proposed heuristic. The results are compared against exact solutions produced by an ILP solver, and results from a pessimistic analysis. Computations are carried out on a Linux workstation with 2.8GHz CPU frequency.

The power grid benchmark is the top three layers of an industrial V_{DD} net. It has 43,473 nodes, among which 19,395 third-layer nodes are to be analyzed. The total power is 26W if all circuit components are switching and consume maximum current, which includes 8W of leakage power that is assumed to be always on. The actual power limit is assumed to be 16W; since this includes 8W of leakage power, this implies that the active blocks cannot consume more than 8W switching current. The nominal V_{DD} is 1.2V. Six GSRC floorplans [28] and five MCNC floorplans [57] are mapped onto this power grid, and third-layer current loads are grouped into blocks accordingly in each floorplan. Block boundaries are adjusted such that there are no white space with uncovered current loads. After we obtain one $\mathbf{I_b}$ vector for each floorplan, by multiplying random numbers between 0.5 and 1.5 to the entries of $\mathbf{I_b}$, we generate four extra $\mathbf{I_b}$ patterns for each of the six cases. Because we are unable to obtain functional description of floorplan blocks, we arbitrarily assign exclusivity constraints. The number of constraints for each floorplan is up to 21.

The comparison of largest single-point voltage-drop analysis using different strategies is shown in Table 5.1, where the first six benchmarks are GSRC floorplans, and the rest are MCNC floorplans. In order to study the performance of the proposed heuristic method in finding $\mathbf{w}$ vector without interference of error from any other estimation step, we substitute the produced $\mathbf{w}$ into equation (5.3), use a direct linear solver to solve (5.3), and list the maximum entry of the solution vector in the fourth column of Table 5.1. For this circuit size, it is already impractical to

Table 5.1: Comparison of analysis methods for the largest single-point voltage-drop.

Floorplan	Number of blocks	Heuristic runtime(s)	Heuristic result(mV)	ILP exact result(mV)	Pessimistic result(mV)
n10a	10	48.14	234.4	234.4	250.8
n30a	30	48.79	274.2	274.3	304.3
n50a	50	49.44	190.8	191.0	247.4
n100a	100	51.05	223.4	223.5	236.7
n200a	200	55.87	238.2	240.4	266.0
n300	300	63.38	282.3	283.4	303.4
apte	9	48.18	193.2	193.3	214.5
xerox	10	48.13	225.1	225.1	243.1
hp	11	48.44	225.7	226.0	278.3
ami33	33	48.70	245.6	245.6	279.1
ami49	49	49.19	239.9	240.0	269.3

construct and evaluate the ILP equation (5.4) for every node. The ILP results listed in the fifth column are the exact answers by ILP analysis for the 50 highest-drop nodes found by the proposed heuristic. The last column is the result by solving equation (5.3) with diag($\mathbf{w}$) being an identity matrix, i.e., assuming that all blocks are active. All three methodologies consider the five $\mathbf{I_b}$ patterns for each floorplan, and report the worst among the five results.

In Table 5.1, there are noticeable differences between constrained analysis and pessimistic analysis. This difference depends on the details of the most power-intensive region of the circuit. For floorplan n50a, the largest voltage-drop node happens to be close to a corner of its block, and these two neighboring blocks have an exclusivity constraint. Consequently, we see a 56mV overestimate by the pessimistic analysis. Although there may not be an exclusivity constraint in the power-intensive region of every circuit design, the possibility of existence of such constraints makes the proposed heuristic superior to pessimistic analysis. Figure 5.3 shows the near-worst-case working mode found for floorplan n30a.

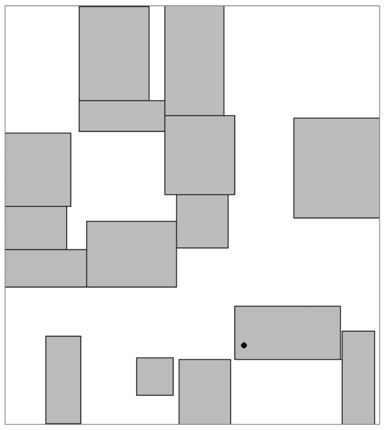


Figure 5.3: Near-worst-case working mode of GSRC floorplan n30a found by the proposed heuristic. The black dot marks location of the largest voltage drop.

Table 5.2: Numbers of node voltage violations reported by the proposed heuristic and the pessimistic analysis.

Floorplan	n10a	n30a	n50a	n100a	n200a	n300
Heuristic	113	122	44	71	102	91
Pessimistic	181	163	72	95	124	137
Floorplan	apte	xerox	hp	ami33	ami49	
Heuristic	91	120	109	130	91	
Pessimistic	112	133	150	184	106	

Table 5.2 shows the comparison of number of node voltage violations reported by different strategies, when the voltage-drop threshold is 80mV. In most cases, about one third of the violating nodes reported by pessimistic analysis are found

legal by the proposed heuristic.

Table 5.3: Comparison of analysis methods for the largest average voltage-drop.

Floorplan	Number of blocks	Heuristic runtime(s)	Heuristic result(mV)	ILP exact result(mV)	Pessimistic result(mV)
n10a	10	36.17	6.81	7.57	14.49
n30a	30	46.47	7.72	7.73	13.59
n50a	50	51.66	7.24	7.66	12.76
n100a	100	56.69	6.99	7.80	12.86
n200a	200	66.99	7.55	7.85	12.88
n300	300	67.03	7.47	7.86	12.96
apte	9	30.87	7.33	7.58	13.80
xerox	10	31.09	7.51	7.55	13.66
hp	11	36.12	7.43	7.60	14.73
ami33	33	36.21	7.34	7.69	13.71
ami49	49	46.41	7.43	7.68	12.99

Table 5.3 shows the comparison of average voltage-drop analysis using different strategies. All results are for the average of 19,395 third-layer nodes. In this case, because only one ILP is required to be formulated and solved for each floorplan with each $\mathbf{I_b}$ vector, the fifth column is the exact solution.

In both Table 5.1 and Table 5.3, results from the proposed heuristic correlate well with those from the ILP solver. The difference between the two solutions is due to the fact that the proposed heuristic finds only a near-worst case, and does make mistakes on certain not-very-significant blocks. Consequently, the results are always slightly optimistic. One remedy is to use the power budget $P_{\max}(1+\delta)$ instead of $P_{\max}$, where δ is a small positive value, but this is not included in the implementation.

Chapter 6

Application in Chip-level Electrostatic Discharge Simulation

This chapter applies the stochastic linear equation solver from Chapter 2 and the network reduction algorithm from Section 4.3.1 on the problem of Electrostatic Discharge (ESD) simulation.

ESD is an important issue in VLSI manufacturing, and unless adequate ESD protection is built into a chip, mechanisms such as contact between the chip with an assembly-line probe, or with a human body, could cause a surge of discharge current that damages the chip permanently. The average product losses due to ESD were reported to be 16-22% in electronic component manufacturing as early as 1990 [23]. In recent years, as feature sizes reduce, thinner gate oxides come into use, and design complexity grows, circuits are becoming increasingly vulnerable to ESD damage. To protect against these problems, a modern design usually employs a full-chip ESD protection strategy. Before a design is sent for manufacturing, simulations are needed to make sure that the design can sustain a certain amount of ESD stress, specified by industrial standards.

6.1 ESD Modeling

Figure 6.1 illustrate the schematic of a typical chip that contains the circuitry that implements its functionality (the blocks labeled "internal circuit") and the following ESD protection mechanisms [41] [56] [72]:

- I/O protection circuitry is introduced for each signal I/O pad. As can be seen in Figure 6.1, this circuitry has the capability to divert the discharge current into the VDD net and/or the ground net.
- ESD voltage clamps are placed between the VDD net and the ground net, and are distributed among the internal devices on the chip, as shown in Figure 6.1. Each such clamp is effectively a very large transistor that is turned off during normal operation. In the presence of an ESD event, however, this transistor turns on, creating a path for the charge to be drained into the ground network, thus allowing the safe discharge of the ESD event and avoiding damage to the on-chip circuitry.
- Diode strings are placed between the different power nets in chips that have multiple power domains. These strings provide discharge paths to protect interface circuits between the power domains. These are not explicitly shown in Figure 6.1, which illustrates a single VDD domain.

The ESD discharge path in a chip may go through either the primary chip power supply net, or the secondary supplies that are used for the I/O circuitry. We generically refer to each of these two as the VDD net, even though they typically operate at different voltages and have vastly different sizes.

An important verification task that must be carried out before a design is sent for manufacturing is to determine, through design-rule checks and simulations, whether these protective devices are adequate to ensure that the chip can withstand a specified level of ESD stress. The purpose of the design-rule checks is to

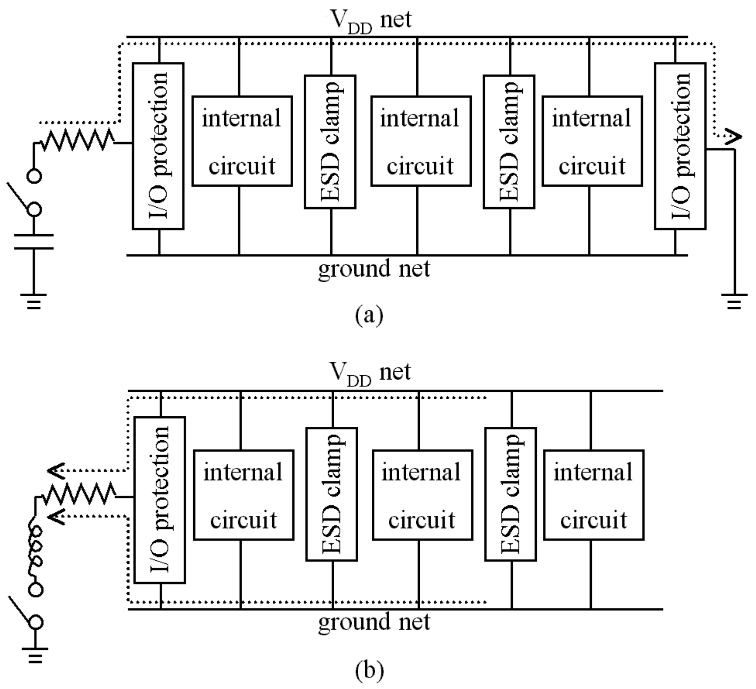


Figure 6.1: ESD simulations for: (a) an HBM event (b) a CDM event. The dotted lines represent a desirable set of discharge paths.

ensure functionality and to avoid unexpected parasitic discharge paths [51], while the simulations are intended to imitate physical tests specified by industrial standards, e.g. [38]. These physical tests describe different discharge scenarios, and are designed based on three primary models: the human body model (HBM), the charged device model (CDM), and the machine model (MM). The first two models usually cover most extreme scenarios and are the most widely used:

- An HBM event is simulated for pairs of pads: one of these is the source of the event, simulated by discharging an external capacitor at that pad through a resistor, while the other is grounded. As a result, a surge of current flows through a resistor into one pad, and out through the grounded pad, as shown in Figure 6.1(a).

- In a CDM event, the chip itself accumulates charge as it passes through the manufacturing process. As shown in Figure 6.1(b), this is simulated for each pad separately, where a charge is switched into the chip through the pad when it is touched by a grounded conductor. In practice, it is seen that CDM accounts for a majority of ESD damage during chip manufacturing [50].

SPICE-like models have been developed for the protective devices under ESD stress, and several techniques have been proposed for circuit-level simulation of ESD circuitry [4] [53] [54]. These works perform detailed transient simulation for one I/O at a time, with certain assumptions about the boundary conditions of the VDD/ground nets. As the design complexity grows, especially for chips with multiple power domains, unexpected discharge paths often cause failures that are not visible in circuit-level simulations, and consequently methodologies have been proposed to address chip-level ESD simulation [50] [51] [75]. A common hurdle faced by all of these approaches is the large size of the simulation problem, which prohibits a full SPICE-like simulation, and therefore, these efforts all apply techniques to reduce the amount of computation. For example, the work in [50], which is targeted at CDM simulation, builds a macromodel for each power domain, where the charge source in each macromodel is represented by a pair of lumped capacitors. A detailed transient analysis is then performed for the reduced full-chip model with SPICE-like device models.

This section presents a different model for simulating ESD events due to CDM. The circuit-level simulation and the chip-level simulation are separated, and the chip-level simulation is formulated as a DC analysis problem of finding the voltage at a stressed I/O node. This voltage is used as an indicator of potential ESD failure.

The purpose of a chip-level CDM simulation is to compute the voltage drop, caused by resistances in the power distribution network(s), along the discharge paths for ESD. As pointed out in [8] [75], a strong correlation has been observed between hardware ESD failures and the wire resistance of the discharge paths, and may be explained as follows. Although ESD voltage clamps are utilized to ensure that the voltage level is constrained to be no more than a specified value V_{clamp}, these devices are only fully effective at the points where they are connected. During an ESD event, discharge current flows through the resistance of the VDD grid, and a voltage drop is induced along its path: the higher the wire resistance of the discharge path, the higher this voltage drop is. Hence for I/O pads that are placed far away from the ESD clamps, a high resistance in the power grid may result in a high voltage on the VDD net, and possibly an ESD failure if this exceeds certain threshold.

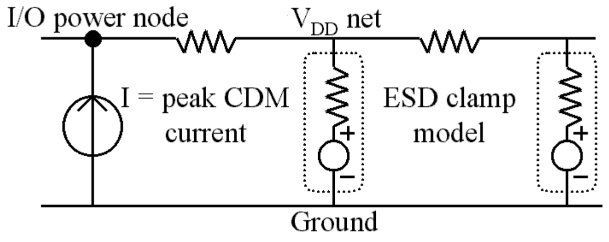


Figure 6.2: A DC model for chip-level CDM simulation of the circuit shown in Figure 6.1(b).

A DC formulation that captures the required chip-level CDM simulation, to verify whether a chip is immune to ESD failures, is illustrated in Figure 6.2 and Figure 6.3. The circuit is modeled as follows:

- The VDD net is extracted from the layout and modeled as a resistive network. The resistance of ground net is ignored in this model. The reason is that the VDD net corresponds to the secondary power net for I/O or the power net of a

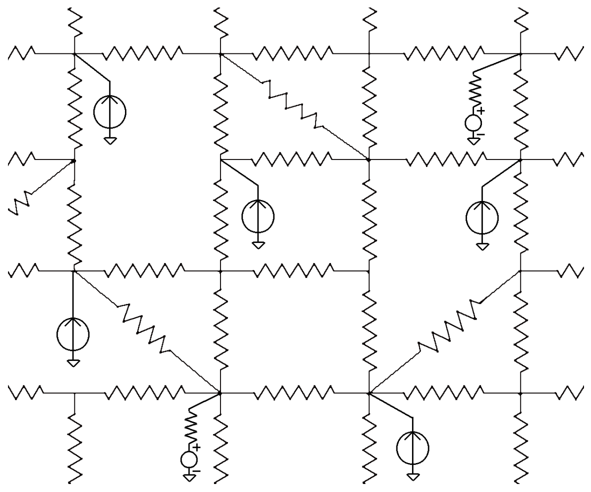


Figure 6.3: A full view of the DC model for chip-level CDM simulation. Each current source represents an I/O, and only one of these is on during each simulation.

single power domain, and this typically has higher resistance than the global ground net which serves all power domains and is the lowest impedance net on chip. However, if the ground net has a significant resistance, then it can be extracted and modeled as a resistive network in a similar manner within the framework.

- An ESD event at an I/O pad is modeled as a current source placed at the location of that I/O. The value of this current source is assigned to be the peak CDM surge current specified in the JEDEC standard [38].
- An ESD voltage clamp is modeled as a voltage source in series with a resistor, placed at the physical location of the ESD clamp. The values of the voltage source and the resistor are obtained from simulated I-V curves of clamps under a stressed situation.

The linear network model thus extracted is excited by the current and voltage sources as defined above. The I/O pads are simulated one at a time for a CDM simulation. An implicit assumption, which is commonly made in ESD analysis, is that the discharge is a single event, i.e., it occurs at only one I/O pad at a time. Therefore, the number of simulations to be carried out equals the number of pads. In Figure 6.3, for the j^{th} simulation, a current value is assigned for the current source modeling the j^{th} I/O pad, zero current is assigned for all the other current sources, and the simulation is carried out. The computed voltage at the j^{th} I/O is checked against the allowable threshold V_{limit} to determine whether the ESD specification is met or not. The simulation is repeated for all other I/O pads. Note that for every simulation, there are many voltage source-resistor elements representing the multiple ESD clamps, since all of these clamps help in providing a path to the ground network during an ESD event.

When viewing Figure 6.3, it is important to note that the density of sources in reality is much lower than what is shown in the schematic. The VDD net can have up to millions of nodes, and the number of I/O pads may be up to a few thousands. There are usually tens of ESD clamps on the chip: this number is typically between 30 and 40, and varies depending on the physical and electrical constraints of the chip.

During the simulation, if the threshold V_{limit} is exceeded at an I/O, then this I/O is considered a potential ESD failure, and one of the following methods may be used by the designer to fix this ESD violation:

- Reduce the effective resistance of the discharge paths by using wider wires to connect this I/O to power grid, or move the I/O circuitry closer to a power bus.
- Add an ESD clamp at a nearby location.

Such design changes require incremental resimulations, and it is important for the analysis to be able to do so rapidly.

Admittedly, a DC model is used here to capture an ESD event that is fundamentally a transient phenomenon. Such a formulation is justified as conservative by using the peak of the CDM current waveform as the input excitation, while brings the benefit of a much faster simulation than a complete transient analysis. The computed voltage at the VDD node of the stressed I/O represents the worst-case voltage drop along the discharge paths, and has been shown to be a good indicator of potential ESD failure: in a 90nm ASIC, I/O failures start to occur when the path voltage drop exceeds 13V [8]. Different I/O pads may be checked against different thresholds depending on their designs.

6.2 Proposed Solution

In the previous section, the chip-level CDM simulation is reduced to a DC analysis problem, where voltage sources represent ESD clamps, and current sources represent I/O protective circuits, one current source being on during each simulation. Such a DC analysis must be performed for every I/O. Considering that the size of the VDD net can be up to several millions of nodes, and that the number of I/O pads can be up to a few thousands, the total computational complexity of performing such a simulation is still very high.

To further reduce computation, we observe that in the DC analysis, the number of sources is limited: the number of current sources is no more than a few thousands, and the number of voltage sources is typically 30 to 40. Therefore, it is desirable to perform a network reduction and build an equivalent circuit that only contains the nodes with a current source or voltage source. Then DC analysis can be carried out for this reduced circuit only, with one current source being on during each simulation.

However, even this may not be adequate. For a circuit that can be represented by a connected graph, the exact reduced circuit is typically a clique on the set of these nodes, and such a reduction does not result in any runtime advantage. In practice, many of these edges have a very low conductance. Therefore, a practical algorithm should exploit this property in performing the network reduction to produce a reasonably sparse reduced circuit, by dropping insignificant connections, without excessive loss of accuracy.

There are a number of options to perform sparse network reduction, namely [47] [89] and our method in Section 4.3.1. The method in [89] performs well on partitionable circuits, but for ESD simulation where the active nodes are distributed all over the circuit, the procedure entails building the exact clique first and then performing sparsification, and the space complexity can be high; it is not trivially possible to force the procedure in [47] to reduce the network to only the set of nodes wanted, because of any horizontal or vertical wire that passes an active node is maintained, implying that the coarsened circuit can have a worst-case node count of K^2, where K is the number of active nodes.

Therefore, we choose the method in Section 4.3.1 to perform the network reduction. Each of the external connections in Figure 4.9, in the context of the ESD problem, corresponds to an I/O protective device or ESD voltage clamp model. Since in the ESD formulation there are no current sources inside the resistive network, ξ_q in equation (4.12) and ξ in (4.13) are both zero, and consequently the vector $\mathbf{S}$ in a macromodel is always zero.

We now look at an extreme case in which the reduced network could lose connectivity. This happens when a small group of ports are very well connected to each other, but are poorly connected to the rest of the ports, such that any random walk initiated in this small group would end in this group. Then the reduced network is not a connected graph, and if this small group does not contain any ESD clamps, the circuit is not solvable. However, the ESD clamps and I/O pads are likely to be evenly distributed all over the chip, and the above extreme case would not happen. If it does happen, this small group of I/O's would be ESD failures and need to be reported and fixed anyway.

With the reduced network, the I/O and clamp models can now be connected to it, and DC analysis can be performed with one current source being on at a time. Any linear solver can fulfill this task. Iterative solvers would be inefficient for this problem because the circuit is solved repeatedly with different excitations; direct solvers and stochastic solvers are better choices, because after the first DC analysis, followup runs can be carried out efficiently. We choose the stochastic solver from Chapter 2, because it can estimate one single node voltage without solving the whole circuit.

To further reduce runtime, we note that in the ESD simulation, only nodes with high voltages are of interest. Hence adaptive error margin Δ is used in the stochastic solver. In the implementation, we use three error margins, $\Delta_1 < \Delta_2 < \Delta_3$, and define two thresholds $V_{T1} < V_{T2}$. When estimating an I/O node voltage, the computation starts with Δ_3. After this accuracy level is achieved, if the estimated voltage is below V_{T1}, the computation stops; otherwise, the error margin is changed to Δ_2, and the computation continues. After the new accuracy level is achieved, if the estimated voltage is below V_{T2}, the computation stops; otherwise, the error margin is changed to Δ_1, and the computation continues. Using this adaptive strategy, more runtime is spent on high-voltage nodes, and get more accurate voltages for them, while safe I/O nodes only get coarse estimation.

As discussed in the previous section, if the threshold V_{limit} is exceeded at an I/O, this I/O is considered a potential ESD failure, and this ESD violation is fixed by either reducing the resistance in discharge paths or adding an ESD clamp at a nearby location. After the design is modified, a resimulation is needed to ensure that the I/O node voltage is reduced to a satisfactory level. This can be performed by rerunning the entire process of network reduction and voltage computation, but the runtime would be high. Instead, we propose to locally update the reduced circuit:

- If wider wires are used to connect the target I/O to power grid, or if it is moved closer to a power bus, we run M walks from this target I/O, as discussed Section 4.3.1, and update its connections in the reduced circuit. Then the ports adjacent to this I/O are updated as well, each with M additional walks.
- If an ESD clamp is added at a location nearby the target I/O, this clamp becomes a new port in the original network, and a new node in the reduced circuit. We first run M walks from the node where the new ESD clamp is located to find its entries in A and $\mathbf{S}$, and thereby find its connections in the reduced circuit. Next, the ports that are found to be adjacent to this new port are updated as well.

In the above local updates, we save computation by ignoring possible effect on ports that are not connected to the improved I/O node or to the new ESD clamp. In other words, if random walks from port i never visit port j, we assume that walks from port j also never visit port i. Because of the fact that the connections between two nodes in the reduced circuit are most likely mutual and with similar conductances, the error induced by the local updating is minimal. We validate this claim by results in the next section.

Finally, the proposed procedure is not limited to CDM ESD simulation, and is also applicable to HBM ESD simulation by replacing every involved resistive network by a circuit with only the nodes that tie to ESD protective devices.

6.3 Simulation Results

In this section, we use three benchmarks, described in Table 6.1, to evaluate the proposed algorithms for chip-level ESD simulation. The first two benchmarks are created by randomly assigning I/O pads and ESD voltage clamps on two industrial power grid models, with nominal VDD values 1.2V and 1.8V respectively. The third and largest benchmark is artificially generated based on the structure of the first benchmark, and with randomly assigned I/O's and clamps. Computations are carried out on a Linux workstation with 2.8GHz CPU frequency.

The results reported in Table 6.1 and Table 6.2 are from simulations using the following parameters: $M = 1000$, $\Delta_1 = 0.1$V, $\Delta_2 = 0.5$V, $\Delta_3 = 1$V, $V_{\text{limit}} = 13$V, $V_{T1} = 12$V, $V_{T2} = 12.5$V. The runtimes and accuracy for the first two benchmarks are compared against the SPARSE linear solver [48], which is a direct solver, while the third benchmark is too large to be handled by SPARSE.

Table 6.1: Benchmarks and runtimes. N1 is the number of nodes, N2 is the number of I/O pads, N3 is the number of ESD clamps, T1 is the runtime of an initial complete simulation, and T2 is the runtime of 10 resimulations after an initial simulation. RW denotes the proposed algorithm, SPARSE denotes the SPARSE linear solver.

Ckt	N1	N2	N3	T1		T2	
				RW	SPARSE	RW	SPARSE
#1	36K	500	20	23.28sec	459.15sec	49.69sec	61min
#2	101K	700	40	89.30sec	55min	61.65sec	8.5hour
#3	2.3M	1000	40	783.17sec	–	537.18sec	–

In interpreting the errors in Table 6.2, note that due to the adaptive error

Table 6.2: Accuracy of the initial complete ESD simulation.

Ckt	#1	#2
Voltage range(V)	10.35–14.40	5.61–45.13
Average error(V)	0.19	0.29
Max error(V)	0.80	1.38
Fraction of failures covered	9/9	2/2
Number of false alarms	5	0

margins employed, most of the high errors occur at the low-voltage safe nodes, and high-voltage nodes are estimated with higher accuracy. Therefore, the errors at the critical nodes are typically smaller. With the ESD threshold $V_{\text{limit}} = 13\text{V}$, and being conservative, the algorithm reports a possible ESD failure whenever an estimated I/O voltage exceeds 12.9V. As shown in the last two rows of Table 6.2, all real ESD failures reported by SPARSE are covered by the proposed algorithm. However, as a cost of being conservative, five false alarms are given for the first benchmark, at nodes with voltages below but very close to V_{limit}.

When desired, the proposed algorithm can achieve higher accuracy by increasing M, the number of walks used in network reduction. Table 6.3 shows the accuracy-runtime tradeoff when M is increased to 3000 and 10000. The error margins are also shrunk accordingly in generating the results.

When incremental changes are made to the network to fix ESD failures, the local update technique shows large advantages in resimulation, with accuracy assessment shown in Table 6.4. The voltages before and after the design change at the target I/O node are listed in each row, using both the SPARSE solver and the proposed algorithm, at a high-accuracy setting with $M = 10000$. 10 nodes with high voltages are chosen as target nodes arbitrarily in each circuit, and are not limited to those

Table 6.3: Runtime-accuracy tradeoff. E1 is average error, E2 is max error, T1 is the runtime of an initial complete simulation, T2 is the runtime of 10 resimulations after an initial simulation, M is number of walks used for each port in network reduction.

M		1000	3000	10000
Ckt #1	E1(V)	0.19	0.10	0.05
	E2(V)	0.80	0.38	0.24
	T1(sec)	23.28	69.37	231.82
	T2(sec)	49.69	194.34	842.72
Ckt #2	E1(V)	0.29	0.14	0.08
	E2(V)	1.38	0.57	0.47
	T1(sec)	89.30	328.69	1606.09
	T2(sec)	61.65	227.12	1318.00

violating the 13V threshold. The most dramatic change is the first node in Table 6.4, where the before-fix voltage is over 45V. This is due to an I/O being assigned at a node with poor connection to major power bus, which emulates the scenario of a poorly designed I/O protection. It is fixed by adding a large via at that location, and the proposed local update method captures the corresponding voltage change.

Table 6.4: Accuracy of resimulations for the first two circuits: voltage changes at I/O pads that are improved.

Ckt #1						Ckt #2				
	Initial(V)		Final(V)				Initial(V)		Final(V)	
	Exact	Est.	Exact	Est.			Exact	Est.	Exact	Est.
1	13.76	13.60	10.31	10.23		1	45.13	45.30	8.46	8.23
2	14.40	14.46	10.95	10.88		2	13.27	13.23	11.04	10.88
3	12.73	12.84	9.28	9.26		3	12.60	12.50	11.69	11.54
4	13.25	13.27	9.80	9.79		4	12.56	12.47	11.54	11.61
5	13.16	13.26	11.96	12.24		5	11.49	11.48	10.48	10.40
6	13.37	13.18	9.91	9.85		6	11.11	11.16	10.78	10.66
7	12.99	13.11	9.61	9.51		7	11.21	11.10	10.43	10.47
8	12.93	13.05	9.50	9.46		8	11.35	11.20	10.63	10.63
9	13.42	13.56	9.99	10.01		9	11.13	11.10	10.77	10.70
10	12.99	13.00	9.60	9.62		10	11.11	11.01	10.45	10.34

Chapter 7

Application in Quadratic Placement

The previous three chapters have primarily applied variations of the stochastic solver from Chapter 2 on various problems in VLSI design automation. This chapter evaluates the performance of the hybrid linear equation solver from Chapter 3 on quadratic placement benchmarks. This is an application associated with relatively dense matrices, and fits in the shaded region in Figure 1.1.

7.1 Quadratic Placement

Placement is a critical and computationally intensive step during the VLSI design cycle that must handle instances of large size. The most widely used approaches fall into the following paradigms: quadratic placement [21] [37] [84] [85] [86], simulated annealing [73], and partitioning-based placement [9]. Of these, quadratic placement, also referred to as analytical or force-directed placement, has emerged as a very popular method, and is the topic of this section. The essential idea is to define a set of attractive and repulsive (or spreading) forces between the modules, and to iteratively find an equilibrium point that corresponds to the optimal placement. Each iterative step involves the minimization of a cost function, whose components typically include an indirect measure of wire length, plus factors such as congestion, overlap, or timing, and requires the solution of a large set of linear equations to compute new locations for modules/cells. The set of linear equations can be written as $A\mathbf{x} = \mathbf{b}$, where A is a square matrix that is typically symmetric and positive definite, $\mathbf{b}$ is a given vector based on the cost function, and $\mathbf{x}$ is the vector of new coordinates of modules/cells to be computed.

For quadratic placement, the left-hand-side matrix A is relatively dense, and several hundred re-solves are often needed to converge to a final placement. Therefore, according to the discussion in Chapter 1, a preconditioned iterative solver such as ICCG is the logical choice: examples of state-of-the-art quadratic placers that use ICCG or its variants include [84] and [85]. The usage of an alternative preconditioned iterative solver based on algebraic multigrid (AMG) in placement was investigated in [10].

Recall that our hybrid solver is especially suitable for applications with large dense left-hand-side matrices and requiring many re-solves, and hence quadratic placement is a good match. This sections discusses the formulation of placement linear equations, and the next section compares the hybrid solver against two different versions of ICCG.

In quadratic placement, the cost of a two-pin net that connects module i and module j is typically defined as

$$\text{netcost} = w\left((x_i - x_j)^2 + (y_i - y_j)^2\right)$$

where w is the weight of the net, (x_i, y_i) and (x_j, y_j) are the coordinates of the two modules i and j [21] [37] [84] [85]. The net weight w depends on the optimization goal, for example, it can be a function of timing criticality [37]. The overall cost function of the placement is the sum of the net costs, and is typically in the following form.

$$\text{cost} = \sum \text{netcost} = \frac{1}{2}\mathbf{x}^{\mathrm{T}}A\mathbf{x} + \frac{1}{2}\mathbf{y}^{\mathrm{T}}A\mathbf{y} + \mathbf{d_x}^{\mathrm{T}}\mathbf{x} + \mathbf{d_y}^{\mathrm{T}}\mathbf{y} \tag{7.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the vectors of unknown x-coordinates and y-coordinates of the modules, A is a square matrix, and $\mathbf{d_x}$ and $\mathbf{d_y}$ are two given vectors. The cost function is minimized by solving the following two sets of linear equations.

$$A\mathbf{x} + \mathbf{d_x} = 0 \tag{7.2}$$

$$A\mathbf{y} + \mathbf{d_y} = 0 \tag{7.3}$$

The x-component of the cost of a net is

$$\text{netcost}_{\text{x}} = w(x_i - x_j)^2 = w{x_i}^2 + w{x_j}^2 - 2wx_ix_j \tag{7.4}$$

Its contribution to equation (7.1) can be viewed as adding $2w$ to the (i,i) entry and (j,j) entry of matrix A, and adding $(-2w)$ to the (i,j) entry and (j,i) entry of A. If one end of a net is a fixed pin, its cost is

$$\text{netcost}_{\text{x}} = w(x_i - x_{\text{pin}})^2 = w{x_i}^2 - 2wx_{\text{pin}}x_i + w{x_{\text{pin}}}^2 \tag{7.5}$$

Then the contribution to equation (7.1) is adding $2w$ to the (i,i) entry of A, and adding $(-2wx_{\text{pin}})$ to the i^{th} entry of $\mathbf{d_x}$, while the last constant term is ignored. Because matrix A is composed of contributions of the above two types, A automatically satisfy the first three conditions of an R-matrix. If a module is not connected to a fixed pin, the sum of its corresponding row in matrix A is 0, otherwise the sum of the row is positive and hence the row is strictly diagonally dominant. In every connected component of a circuit design, at least one module must be connected to an I/O pin, and thus at least one row must be strictly diagonally dominant.

This satisfies the fourth condition of an R-matrix. Thus, matrix A is always an R-matrix, and the hybrid solver can be applied on (7.2) and (7.3).

While mapping the equations to a random walk game as in Figure 2.1, one can certainly choose $m_0 = 0$ as in Chapter 3, but there is an alternative way. That is, every fixed pin can be interpreted as a home node, and the award for reaching it is equal to its coordinate x_{pin} or y_{pin}. Then, instead of having a universal m_0, the pre-existing homes can have different award values.

A typical placement flow involves solving (7.2) and (7.3) repeatedly, each iteration with different $\mathbf{d_x}$ and $\mathbf{d_y}$ vectors, which contain not only contributions from net costs, but also the cost of overlapping modules, congestion or timing criticality, depending on the placement algorithm. Matrix A may also change during the placement, for example, due to adding friction [85], in which case the preconditioning part of the hybrid solver needs to run again to update the incomplete LDL factorization.

7.2 Numerical Results

In this section, we use realistic placement matrices to evaluate the proposed hybrid solver. Table 7.1 and Table 7.2 compare the computational complexity of the hybrid solver with that of ICCG; Table 7.4 shows the performance of embedding the hybrid solver into the Waterloo placement tool [85]. Computations are carried out on a Linux workstation with 2.8GHz CPU frequency.

The first set of benchmarks are matrices generated by an industrial placement tool for 10 circuits: m1-m10 in Table 7.1. One actual right-hand-side vector instance is also available for each of them. The second set of benchmarks, shown in Table 7.2, are the ISPD02 circuits [1], and their matrices are generated by the Waterloo placement tool before the initial placement, and hence contain only the connectivity component from the original netlists. A right-hand-side vector with all entries being 1 is used with each of them.

In Table 7.1 and Table 7.2, we compare the hybrid solver against ICCG with ILU(0) and ICCG with ILUT. The complexity metric is the number of double-precision multiplications needed at the iterative solving stage for the equation set $A\mathbf{x} = \mathbf{b}$, in order to converge with an error tolerance of 10^{-6}. This error tolerance is defined as:

$$\| \mathbf{b} - A\mathbf{x} \|_2 < 10^{-6} \cdot \| \mathbf{b} \|_2 \tag{7.6}$$

LASPack [76] is used for ICCG with ILU(0). MATLAB is used for ICCG with ILUT. There are three node ordering algorithms available in MATLAB: minimum degree ordering (MMD) [24], approximate minimum degree ordering (AMD) [2], and reverse Cuthill-McKee ordering (RCM) [15]. AMD results in the best performance on the benchmarks and is used for all tests. The dropping threshold of ILUT in MATLAB is tuned, and the setting of the hybrid solver is adjusted, such that the sizes of the Cholesky factors produced by both methods are similar, i.e., the C values in the tables are close. For LASPack and MATLAB, the $M1$ values are computed using the following equation.

$$M1 = C \cdot 2 + E + N \cdot 4 \tag{7.7}$$

According to the PCG pseudo codes in [3] and [70], the above equation is the best possible implementation. The $M1$ values of the hybrid solver is obtained by a detailed count embedded in its implementation, and in fact equation (7.7) is roughly true for the hybrid solver as well.

In Table 7.1, the hybrid solver shows up to 8.4 times speedup over ICCG with ILU(0), and up to 7.1 times speedup over ICCG with ILUT. In Table 7.2, although the speedup ratios are less than the first set of benchmarks, the hybrid solver is at least 2 times faster than ICCG for any matrix with over 10^5 dimension. It is worth noting that the first set of benchmarks are larger and denser than the ISPD02 matrices, and there is an obvious trend that the larger and denser a matrix is, the more the hybrid solver outperforms ICCG. This is consistent with our argument in Section 3.3: when the matrix is larger and denser, the effect of error accumulation in traditional methods becomes stronger.

Physical runtimes $T1$ and $T2$ are shown in Table 7.3. Admittedly, the preconditioning runtime $T1$ of the hybrid solver is more than the typical runtime of a traditional incomplete factorization; however, it is not a large overhead, gets easily amortized over multiple re-solves, and is worthwhile given the speedup achieved in the solving stage.

In Table 7.4, the hybrid solver is embedded into the Waterloo placement tool [85] [86], and the new runtimes on the ISPD02 benchmarks are compared with the original runtimes. The replaced original solver is BICGSTAB preconditioned by ILU(0) with RCM ordering. This comparison is not completely fair because a slight difference in an early placement could lead to very different final placement, and the computation process can be different too. Therefore the two runtimes may not be measured on the same computational tasks. Despite such uncertainty, the results suggest 5%-10% runtime reduction. The speedup is not as dramatic as in Table 7.1 and Table 7.2, because solving linear equations is not the dominant portion of the runtime for this particular placer. Circuit ibm18 is not included due to stability issues in the placement algorithm.

Table 7.1: Computational complexity comparison of the hybrid solver, ICCG with ILU(0) (LASPACK), and ICCG with ILUT (MATLAB), for the first set of matrices, with 1e-6 error tolerance. N is the dimension of a matrix; E is the number of non-zero entries of a matrix; C is the size of the Cholesky factor; $M1$ is the number of multiplications per iteration; I is the number of iterations; $M2$ is the total number of multiplications; $R1$ is the speedup ratio of the hybrid solver over ICCG with ILU(0); $R2$ is the speedup ratio of the hybrid solver over ICCG with ILUT.

Ckt	N	E	ICCG with ILU(0)				ICCG with ILUT				Hybrid				R1	R2
			C	$M1$	I	$M2$	C	$M1$	I	$M2$	C	$M1$	I	$M2$		
m1	7.1e4	9.5e5	5.1e5	2.3e6	82	1.9e8	1.2e6	3.6e6	41	1.5e8	1.2e6	3.6e6	13	4.7e7	4.0	3.2
m2	7.3e4	1.1e6	5.9e5	2.6e6	126	3.2e8	1.2e6	3.7e6	62	2.3e8	1.3e6	3.9e6	13	5.0e7	6.4	4.6
m3	1.2e5	1.5e6	8.0e5	3.6e6	63	2.3e8	1.8e6	5.6e6	34	1.9e8	1.8e6	5.5e6	10	5.5e7	4.1	3.5
m4	2.1e5	2.1e6	1.2e6	5.3e6	136	7.2e8	3.0e6	9.0e6	64	5.7e8	2.9e6	8.5e6	12	1.0e8	7.1	5.6
m5	2.8e5	3.2e6	1.7e6	7.7e6	76	5.9e8	4.1e6	1.2e7	37	4.6e8	4.1e6	1.2e7	12	1.5e8	4.0	3.1
m6	2.7e5	3.2e6	1.7e6	7.8e6	150	1.2e9	3.9e6	1.2e7	77	9.3e8	3.9e6	1.2e7	12	1.4e8	8.3	6.7
m7	4.3e5	5.2e6	2.8e6	1.3e7	122	1.5e9	6.5e6	2.0e7	62	1.2e9	6.5e6	2.0e7	12	2.3e8	6.6	5.3
m8	3.5e5	5.5e6	2.9e6	1.3e7	82	1.0e9	5.1e6	1.7e7	27	4.6e8	5.0e6	1.6e7	12	2.0e8	5.3	2.4
m9	4.6e5	8.2e6	4.3e6	1.9e7	110	2.1e9	7.5e6	2.5e7	55	1.4e9	8.0e6	2.5e7	13	3.3e8	6.3	4.2
m10	8.8e5	9.4e6	5.2e6	2.3e7	159	3.7e9	1.3e7	3.9e7	82	3.2e9	1.2e7	3.7e7	12	4.4e8	8.4	7.1

Table 7.2: Computational complexity comparison of the hybrid solver, ICCG with ILU(0) (LASPACK), and ICCG with ILUT (MATLAB), for the ISPD02 benchmarks, with 1e-6 error tolerance. N, E, C, $M1$, I, $M2$, $R1$ and $R2$ are as defined in Table 7.1.

Ckt	N	E	ICCG with ILU(0)				ICCG with ILUT				Hybrid				R1	R2
			C	$M1$	I	$M2$	C	$M1$	I	$M2$	C	$M1$	I	$M2$		
ibm01	1.6e4	9.8e4	5.7e4	2.8e5	84	2.3e7	1.5e5	4.6e5	19	8.7e6	1.6e5	4.4e5	18	7.8e6	3.0	1.1
ibm02	2.6e4	1.7e5	9.6e4	4.6e5	68	3.1e7	2.5e5	7.8e5	25	1.9e7	3.1e5	8.3e5	20	1.7e7	1.9	1.2
ibm03	3.0e4	1.8e5	1.0e5	5.0e5	67	3.4e7	2.6e5	8.2e5	18	1.5e7	3.1e5	8.2e5	18	1.5e7	2.3	1.0
ibm04	3.5e4	2.0e5	1.2e5	5.8e5	72	4.2e7	3.0e5	9.4e5	24	2.3e7	3.4e5	9.1e5	20	1.8e7	2.3	1.2
ibm05	3.7e4	2.5e5	1.4e5	6.9e5	44	3.0e7	3.9e5	1.2e6	16	1.9e7	4.5e5	1.2e6	16	1.9e7	1.6	1.0
ibm06	4.2e4	2.6e5	1.5e5	7.2e5	68	4.9e7	4.2e5	1.3e6	22	2.8e7	4.9e5	1.3e6	19	2.5e7	2.0	1.1
ibm07	5.9e4	3.5e5	2.1e5	1.0e6	103	1.0e8	5.5e5	1.7e6	31	5.2e7	6.0e5	1.6e6	20	3.3e7	3.2	1.6
ibm08	6.6e4	4.1e5	2.4e5	1.2e6	77	8.9e7	5.9e5	1.9e6	30	5.6e7	7.4e5	2.0e6	24	4.9e7	1.8	1.1
ibm09	6.9e4	4.4e5	2.5e5	1.2e6	114	1.4e8	6.6e5	2.0e6	35	7.1e7	7.5e5	2.1e6	22	4.5e7	3.1	1.6
ibm10	9.3e4	5.9e5	3.4e5	1.7e6	130	2.1e8	9.4e5	2.9e6	42	1.2e8	1.0e6	2.9e6	19	5.4e7	4.0	2.2
ibm11	9.2e4	5.5e5	3.2e5	1.6e6	120	1.9e8	8.5e5	2.6e6	36	9.4e7	9.6e5	2.6e6	20	5.2e7	3.6	1.8
ibm12	9.5e4	6.4e5	3.7e5	1.7e6	110	1.9e8	1.0e6	3.1e6	47	1.4e8	1.1e6	3.0e6	20	6.0e7	3.2	2.4
ibm13	1.1e5	7.0e5	4.0e5	2.0e6	130	2.5e8	1.1e6	3.3e6	45	1.5e8	1.2e6	3.4e6	20	6.8e7	3.7	2.2
ibm14	1.9e5	1.1e6	6.5e5	3.2e6	145	4.6e8	1.7e6	5.3e6	53	2.8e8	2.0e6	5.3e6	26	1.4e8	3.3	2.0
ibm15	2.2e5	1.4e6	8.2e5	3.9e6	153	6.0e8	2.2e6	6.7e6	50	3.4e8	2.6e6	7.2e6	23	1.6e8	3.7	2.0
ibm16	2.5e5	1.6e6	9.1e5	4.4e6	170	7.5e8	2.5e6	7.6e6	62	4.7e8	2.9e6	7.9e6	21	1.7e8	4.5	2.9
ibm17	2.5e5	1.8e6	1.0e6	4.8e6	137	6.6e8	2.9e6	8.5e6	47	4.0e8	3.1e6	8.6e6	20	1.7e8	3.8	2.3
ibm18	2.7e5	1.7e6	9.8e5	4.8e6	191	9.1e8	2.6e6	8.0e6	65	5.2e8	3.1e6	8.5e6	23	2.0e8	4.7	2.7

Table 7.3: Physical runtimes of the hybrid solver, with 1e-6 error tolerance. $T1$ is preconditioning CPU time. $T2$ is solving CPU time. Unit is second.

Ckt	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10
$T1$	6.03	6.88	6.51	13.07	17.82	20.77	33.00	21.67	46.91	68.90
$T2$	1.16	1.28	1.95	4.26	6.82	6.22	11.90	9.73	17.07	26.09
Ckt	ibm01	ibm02	ibm03	ibm04	ibm05	ibm06	ibm07	ibm08	ibm09	
$T1$	0.66	1.54	1.49	1.89	1.96	2.70	3.83	5.03	5.20	
$T2$	0.14	0.29	0.28	0.34	0.37	0.43	0.74	1.16	1.05	
Ckt	ibm10	ibm11	ibm12	ibm13	ibm14	ibm15	ibm16	ibm17	ibm18	
$T1$	7.00	6.64	7.50	8.81	15.93	23.34	24.09	25.25	30.67	
$T2$	1.42	1.25	1.54	1.99	4.88	6.88	6.50	6.94	8.10	

Table 7.4: Runtime comparison inside the Waterloo Placer. $T3$ is the placer runtime with its original solver. $T4$ is the placer runtime with the hybrid solver. Unit is minute.

Ckt	ibm01	ibm02	ibm03	ibm04	ibm05	ibm06	ibm07	ibm08	ibm09
$T3$	12.6	18.7	17.6	16.4	57.0	23.1	27.6	34.1	34.4
$T4$	10.5	16.1	16.1	14.7	54.4	19.9	26.0	33.3	31.5
Ckt	ibm10	ibm11	ibm12	ibm13	ibm14	ibm15	ibm16	ibm17	
$T3$	53.4	42.1	57.1	54.1	106.7	117.0	123.9	181.4	
$T4$	48.7	39.0	51.0	51.8	101.4	110.1	117.0	170.6	

Chapter 8

Conclusion

The mathematical aspects of this thesis consist of two new ways of solving linear equations:

- Our stochastic linear equation solver performs computation by establishing the equivalence between linear equations and random walks, and has the desirable feature that it can evaluate a single entry in the solution vector without solving the entire system.
- Our hybrid linear equation solver is a random-walk preconditioned iterative solver, and is intended to compete with state-of-the-art direct or iterative solvers for applications that fall into the shaded region in Figure 1.1: the challenging problems where the large-scale left-hand-side matrix is relatively dense and multiple re-solves are needed. Its advantage over existing preconditioned iterative solvers is that each row of the preconditioner is independently calculated, and thereby avoids the error accumulation in traditional incomplete factorization. Consequently, a better accuracy can be achieved with the same preconditioner size, and less iterations are needed to solve $A\mathbf{x} = \mathbf{b}$ with the same error tolerance. The speedups are expected to be more prominent

for larger and denser matrices.

The engineering aspects of this thesis cover four problems in VLSI design automation:

- Power grid analysis. The locality property of the stochastic solver enables calculating a single node voltage in DC analysis, and a single node voltage at a single time point in transient analysis. Special variations of the stochastic solver are derived to improve simulation efficiency.
- Early-stage power grid analysis. A heuristic solution is proposed to find the worst VDD loss scenario, utilizing the localized computation of random walks to relate a single node voltage to the working modes of circuit blocks.
- Chip-level ESD simulation. A flexible network reduction algorithm based on random walks is applied to mitigate the high computational complexity, and to enable incremental simulation.
- Quadratic placement. The hybrid linear equation solver is tested on industrial quadratic placement benchmarks, and is integrated into an actual placement tool. Significant speedup over traditional ICCG is observed, and supports the earlier theoretical argument.

Further investigations may be conducted in the following directions:

- Applications of the hybrid solver need to be identified. Any problem that requires the solution of a set linear equations with a large dimension is a potential application, especially those that fit within the shaded region in Figure 1.1.
- Applications of the stochastic solver need to be identified. Suitable problems are those that require incremental or localized analysis. For example, [52] is a recent followup to our work, and uses random walks to find sensitivities of the VDD drop at a critical node with respect to process variations.

- As discussed in Section 3.5, there are still unanswered questions in applying the hybrid solver on general matrices. Looking further, generalizing the methodology of the stochastic and hybrid solvers to nonlinear equations may lead to exciting opportunities.

In conclusion, this thesis represents a renewed look at stochastic techniques for the solution of linear equations, which, prior to this work, were consigned to the dust heap as theoretically interesting but impractical methods. A major contribution of this thesis is in developing these approaches to the point where they are credible competitors to, and often superior to, existing direct and iterative solvers.

It is hoped that this work will motivate new research thrusts in this domain, which may well result in techniques that surpass the solvers in this thesis.

Appendix A

Proof of Lemma 2

Suppose square matrix A has two (Doolittle) LU factorizations: $A = L_1U_1 = L_2U_2$. Then it must be true that ${L_2}^{-1}L_1 = U_2{U_1}^{-1}$. Because ${L_2}^{-1}$ and L_1 are both lower triangular matrices, their product must be lower triangular; because U_2 and ${U_1}^{-1}$ are both upper triangular, their product must be upper triangular. Therefore, the only possibility is that both ${L_2}^{-1}L_1$ and $U_2{U_1}^{-1}$ are diagonal matrices. Since L_1 and L_2 both have unit diagonal values, and it is easy to verify that ${L_2}^{-1}$ also has unit diagonal values, it must be true that ${L_2}^{-1}L_1 = I$. Therefore $L_1 = L_2$, and it follows that $U_1 = U_2$, and that the LU factorization is unique.

Appendix B

Proof of Convergence for Section 4.1.3

The modified nodal equation set for the circuit in the form of Figure 4.5(b) can be written as

$$(F_1 + F_2 + F_3)\,\mathbf{V} = \mathbf{b} \tag{B.1}$$

where matrix F_1 is the component that contains the contributions of resistors and the companion models of capacitors, matrix F_2 contains the contributions of the companion models of self-inductances, matrix F_3 contains the contributions of the companion models (voltage-controlled current sources) of mutual inductances, $\mathbf{V}$ is the vector of node voltages, and $\mathbf{b}$ is the vector of independent sources, which include original current/voltage sources, the voltage sources from the companion models of capacitors, and the current sources from the companion models of self-inductances [36]. Because the modified nodal equations are constructed for the circuit form of Figure 4.5(b), $\mathbf{V}$ includes both the end nodes of wire segments (nodes A's and B's in Figure 4.5), and the middle nodes (C's in Figure 4.5) that do not exist physically.

The iterative scheme described in Section 4.1.3 can be written as,

$$\begin{aligned}(F_1 + F_2)\,\mathbf{V^{k+1}} &= -F_3\mathbf{V^k} + \mathbf{b} \\ \mathbf{V^{k+1}} &= -(F_1 + F_2)^{-1} F_3\mathbf{V^k} + (F_1 + F_2)^{-1}\mathbf{b} \end{aligned} \tag{B.2}$$

where $\mathbf{V^k}$ is the solution vector from the previous iteration, and $\mathbf{V^{k+1}}$ is the updated solution vector. Note that the proposed algorithm does not perform the matrix computation of (B.2), and instead, it converts the circuit to the form of Figure 4.5(d), and uses random walks to carry out the computation. Nevertheless, the underlying iteration is equation (B.2).

Therefore, the necessary and sufficient condition for the iterative algorithm to converge is

$$\max_l \left|\lambda_l\left((F_1 + F_2)^{-1} F_3\right)\right| < 1 \tag{B.3}$$

where λ_l denotes the l^{th} eigenvalue of a matrix [83]. In order to prove condition (B.3), the following lemma is needed.

Lemma 6 *Matrices $(F_1 + F_2)$, $(F_1 + F_2 + F_3)$ and $(F_1 + F_2 - F_3)$ are positive definite.*

Matrix $(F_1 + F_2)$ is an irreducibly diagonally dominant matrix with positive diagonal entries, for any connected power grid, and therefore is positive definite [83].

Let $\mathbf{y}$ be any real-valued nonzero vector, and we have

$$\mathbf{y^T}(F_1 + F_2 + F_3)\,\mathbf{y} = \mathbf{y^T}F_1\mathbf{y} + \mathbf{y^T}F_2\mathbf{y} + \mathbf{y^T}F_3\mathbf{y} \tag{B.4}$$

Because F_1 is a diagonally dominant matrix with positive diagonal entries, (maybe reducible, i.e., representing an unconnected resistor network), and hence must be nonnegative definite, we have

$$\mathbf{y^T}F_1\mathbf{y} \geq 0 \tag{B.5}$$

Let Γ be the number of inductors, and they are labeled $1, 2, \cdots, \Gamma$. Let $\zeta_{i,1}$ and $\zeta_{i,2}$ be the node indices at the two ends of inductor i, in other words, they are the nodes B and C in Figure 4.5; let them be defined with consistent direction, in other words, for parallel wire segments, $\zeta_{i,1}$'s always point to the same direction. In the K matrix, $K_{i,i}$ represents the self-inductance of inductor i, and $K_{i,j}$, $i \neq j$, represents the mutual inductance between inductor i and inductor j. From [16] [39], we know that $K_{i,j} = K_{j,i}$, and that

$$K_{i,i} > \sum_{j \in \{1,\cdots,\Gamma\}, j \neq i} |K_{i,j}| \tag{B.6}$$

The contribution of inductor i to matrix F_2 is shown below, with the row and column indices marked outside the matrix [36].

$$\begin{array}{c|cc|} & \zeta_{i,1} & \zeta_{i,2} \\ \hline \zeta_{i,1} & hK_{i,i} & -hK_{i,i} \\ \zeta_{i,2} & -hK_{i,i} & hK_{i,i} \\ \hline \end{array}$$

Hence, the contribution of inductor i to $\mathbf{y^T} F_2 \mathbf{y}$ is

$$hK_{i,i} y_{\zeta_{i,1}}^2 - hK_{i,i} y_{\zeta_{i,1}} y_{\zeta_{i,2}} - hK_{i,i} y_{\zeta_{i,1}} y_{\zeta_{i,2}} + hK_{i,i} y_{\zeta_{i,2}}^2 = hK_{i,i}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2$$

Therefore

$$\mathbf{y^T} F_2 \mathbf{y} = \sum_{i=1}^{\Gamma} hK_{i,i}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2 \tag{B.7}$$

The contribution of the mutual inductances between inductor i and inductor j to matrix F_3 is shown below, with the row and column indices marked outside the matrix, and these entries correspond to the voltage-controlled current sources in

Figure 4.4 and Figure 4.5(b) [36].

$$\begin{array}{c|cccc|}
 & \zeta_{i,1} & \zeta_{i,2} & \zeta_{j,1} & \zeta_{j,2} \\
\zeta_{i,1} & & & hK_{i,j} & -hK_{i,j} \\
\zeta_{i,2} & & & -hK_{i,j} & hK_{i,j} \\
\zeta_{j,1} & hK_{j,i} & -hK_{j,i} & & \\
\zeta_{j,2} & -hK_{j,i} & hK_{j,i} & &
\end{array}$$

Hence, the contribution of mutual inductances $K_{i,j}$ and $K_{j,i}$ to $\mathbf{y^T}F_3\mathbf{y}$ is

$$\begin{aligned}
& hK_{i,j}(y_{\zeta_{i,1}}y_{\zeta_{j,1}} - y_{\zeta_{i,1}}y_{\zeta_{j,2}} - y_{\zeta_{i,2}}y_{\zeta_{j,1}} + y_{\zeta_{i,2}}y_{\zeta_{j,2}}) \\
& +hK_{j,i}(y_{\zeta_{i,1}}y_{\zeta_{j,1}} - y_{\zeta_{i,1}}y_{\zeta_{j,2}} - y_{\zeta_{i,2}}y_{\zeta_{j,1}} + y_{\zeta_{i,2}}y_{\zeta_{j,2}}) \\
= \; & hK_{i,j}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})(y_{\zeta_{j,1}} - y_{\zeta_{j,2}}) + hK_{j,i}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})(y_{\zeta_{j,1}} - y_{\zeta_{j,2}}) \\
= \; & 2hK_{i,j}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})(y_{\zeta_{j,1}} - y_{\zeta_{j,2}}) \qquad (\text{since} \quad K_{i,j} = K_{j,i})
\end{aligned}$$

Therefore

$$\mathbf{y^T}F_3\mathbf{y} = \sum_{i,j\in\{1,\cdots,\Gamma\},i\neq j} 2hK_{i,j}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})(y_{\zeta_{j,1}} - y_{\zeta_{j,2}}) \tag{B.8}$$

Therefore

$$
\begin{aligned}
\left|\mathbf{y}^{\mathbf{T}} F_3 \mathbf{y}\right| &= \left| \sum_{i,j\in\{1,\cdots,\Gamma\}, i\neq j} 2hK_{i,j}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})(y_{\zeta_{j,1}} - y_{\zeta_{j,2}}) \right| \\
&\leq \left| \sum_{i,j\in\{1,\cdots,\Gamma\}, i\neq j} hK_{i,j}((y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2 + (y_{\zeta_{j,1}} - y_{\zeta_{j,2}})^2) \right| \\
&\leq \left| \sum_{i,j\in\{1,\cdots,\Gamma\}, i\neq j} hK_{i,j}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2 \right| \\
&\quad + \left| \sum_{i,j\in\{1,\cdots,\Gamma\}, i\neq j} hK_{i,j}(y_{\zeta_{j,1}} - y_{\zeta_{j,2}})^2 \right| \\
&= \left| \frac{1}{2} \sum_{i=1}^{\Gamma} \sum_{j\in\{1,\cdots,\Gamma\}, j\neq i} hK_{i,j}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2 \right| \\
&\quad + \left| \frac{1}{2} \sum_{j=1}^{\Gamma} \sum_{i\in\{1,\cdots,\Gamma\}, i\neq j} hK_{i,j}(y_{\zeta_{j,1}} - y_{\zeta_{j,2}})^2 \right| \\
&= \frac{1}{2} \sum_{i=1}^{\Gamma} (y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2 \left| \sum_{j\in\{1,\cdots,\Gamma\}, j\neq i} hK_{i,j} \right| \\
&\quad + \frac{1}{2} \sum_{j=1}^{\Gamma} (y_{\zeta_{j,1}} - y_{\zeta_{j,2}})^2 \left| \sum_{i\in\{1,\cdots,\Gamma\}, i\neq j} hK_{i,j} \right| \\
&\leq \frac{1}{2} \sum_{i=1}^{\Gamma} (y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2 \sum_{j\in\{1,\cdots,\Gamma\}, j\neq i} |hK_{i,j}| \\
&\quad + \frac{1}{2} \sum_{j=1}^{\Gamma} (y_{\zeta_{j,1}} - y_{\zeta_{j,2}})^2 \sum_{i\in\{1,\cdots,\Gamma\}, i\neq j} |hK_{i,j}| \qquad \text{(B.9)}
\end{aligned}
$$

Then applying equation (B.6), we have

$$\begin{aligned}
\left|\mathbf{y}^{\mathbf{T}} F_3 \mathbf{y}\right| &\leq \frac{1}{2}\sum_{i=1}^{\Gamma}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2 hK_{i,i} + \frac{1}{2}\sum_{j=1}^{\Gamma}(y_{\zeta_{j,1}} - y_{\zeta_{j,2}})^2 hK_{j,j} \\
&= \frac{1}{2}\sum_{i=1}^{\Gamma}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2 hK_{i,i} + \frac{1}{2}\sum_{i=1}^{\Gamma}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2 hK_{i,i} \\
&= \sum_{i=1}^{\Gamma}(y_{\zeta_{i,1}} - y_{\zeta_{i,2}})^2 hK_{i,i} \\
&= \mathbf{y}^{\mathbf{T}} F_2 \mathbf{y} \qquad \text{(applying equation (B.7))} \qquad \text{(B.10)}
\end{aligned}$$

Therefore

$$\mathbf{y}^{\mathbf{T}} F_2 \mathbf{y} \pm \mathbf{y}^{\mathbf{T}} F_3 \mathbf{y} \geq 0 \qquad \text{(B.11)}$$

Substituting (B.5) and (B.11) into equation (B.4), we get

$$\mathbf{y}^{\mathbf{T}}(F_1 + F_2 + F_3)\mathbf{y} \geq 0 \qquad \text{(B.12)}$$

Now we need to show that (B.5) and (B.11) cannot both be equalities. Note that, in order for (B.10) to be an equality after applying inequality (B.6), vector $\mathbf{y}$ must satisfy the condition

$$y_{\zeta_{i,1}} = y_{\zeta_{i,2}} \qquad \text{for} \quad \text{i} \in \{1, \cdots, \Gamma\}$$

For such a vector $\mathbf{y}$, we can merge $\zeta_{i,1}$ and $\zeta_{i,2}$ into one node, and obtain a shortened vector $\mathbf{y}'$. In other words, nodes B and C in Figure 4.5 are merged into one node. Correspondingly, the rows for $\zeta_{i,1}$ and $\zeta_{i,2}$ in matrix F_1 are merged into one row by adding entries, and columns for $\zeta_{i,1}$ and $\zeta_{i,2}$ in matrix F_1 are merged into one column by adding entries. Thus we obtain a new matrix ${F_1}'$, which is the same as the modified nodal left-hand-side matrix if all inductors are ignored. Because ${F_1}'$ is an irreducibly diagonally dominant matrix with positive diagonal entries, for any connected power grid, we have

$$\mathbf{y}^{\mathbf{T}} F_1 \mathbf{y} = \mathbf{y}'^{\mathbf{T}} {F_1}' \mathbf{y}' > 0,$$

It follows that (B.5) and (B.11) cannot both be equalities.

Therefore, (B.12) can never be equality, and can be replaced by

$$\mathbf{y^T}(F_1 + F_2 + F_3)\mathbf{y} > 0 \tag{B.13}$$

This is true for any real-valued nonzero vector $\mathbf{y}$. Therefore, matrix $(F_1 + F_2 + F_3)$ is positive definite.

Similarly, by equations (B.5) and (B.11), and the fact that they cannot both be equalities,

$$\mathbf{y^T}(F_1 + F_2 - F_3)\mathbf{y} = \mathbf{y^T}F_1\mathbf{y} + \mathbf{y^T}F_2\mathbf{y} - \mathbf{y^T}F_3\mathbf{y} > 0 \tag{B.14}$$

This is true for any real-valued nonzero vector $\mathbf{y}$. Therefore, matrix $(F_1 + F_2 - F_3)$ is positive definite.

By now we have proven Lemma 6, it can be used to prove condition (B.3), which is replicated as follows.

Lemma 7 $\max_l \left|\lambda_l\left((F_1 + F_2)^{-1} F_3\right)\right| < 1$

Let λ be any eigenvalue of matrix $(F_1 + F_2)^{-1} F_3$, and let $\mathbf{y}$ be the corresponding eigenvector. By Lemma 6, $(F_1 + F_2 + F_3)$ is positive definite, and we have

$$\begin{aligned}
\mathbf{y^T}(F_1 + F_2 + F_3)\mathbf{y} &> 0 \\
\mathbf{y^T}(F_1 + F_2) \cdot \left(I + (F_1 + F_2)^{-1} F_3\right)\mathbf{y} &> 0 \\
\mathbf{y^T}(F_1 + F_2)(\mathbf{y} + \lambda\mathbf{y}) &> 0 \\
\mathbf{y^T}(F_1 + F_2)(1 + \lambda)\mathbf{y} &> 0
\end{aligned} \tag{B.15}$$

Since $(1 + \lambda)$ is a scalar, and can be taken out, we get

$$(1 + \lambda)(\mathbf{y^T}(F_1 + F_2)\mathbf{y}) > 0 \tag{B.16}$$

By Lemma 6, $\mathbf{y}^{\mathbf{T}}(F_1 + F_2)\mathbf{y}$ must be a positive scalar, and therefore,

$$\begin{aligned} 1 + \lambda &> 0 \\ \lambda &> -1 \end{aligned} \tag{B.17}$$

Similarly, because $(F_1 + F_2 - F_3)$ is positive definite, we have

$$\begin{aligned} \mathbf{y}^{\mathbf{T}}(F_1 + F_2 - F_3)\mathbf{y} &> 0 \\ \mathbf{y}^{\mathbf{T}}(F_1 + F_2) \cdot \left(I - (F_1 + F_2)^{-1} F_3\right) \mathbf{y} &> 0 \\ \mathbf{y}^{\mathbf{T}}(F_1 + F_2)(1 - \lambda)\mathbf{y} &> 0 \\ (1 - \lambda)(\mathbf{y}^{\mathbf{T}}(F_1 + F_2)\mathbf{y}) &> 0 \\ 1 - \lambda &> 0 \\ \lambda &< 1 \end{aligned} \tag{B.18}$$

Therefore

$$|\lambda| < 1 \tag{B.19}$$

This is true for any eigenvalue of matrix $(F_1 + F_2)^{-1} F_3$. Therefore Lemma 7 is true, and our iterative algorithm in Section 4.1.3 is guaranteed to converge.

Bibliography

[1] S. Adya, I. Markov, ISPD02 Benchmarks. Available at `http://vlsicad.eecs.umich.edu/BK/ISPD02bench`

[2] P. R. Amestoy, T. A. Davis and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886-905, 1996.

[3] R. Barrett, M. Berry, T. F. Chan, J. W. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. A. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994.

[4] S. G. Beebe, "Simulation of complete CMOS I/O circuit response to CDM stress," *Proceedings of Electrical Overstress/Electrostatic Discharge Symposium*, pp. 259-270, 1998.

[5] R. M. Bevensee, "Probabilistic potential theory applied to electrical engineering problems," *Proceedings of the IEEE*, vol. 61, no. 4, pp. 423-437, 1973.

[6] S. Bodapati and F. N. Najm, "High-level current macro-model for power grid analysis," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 385-390, 2002.

[7] A. Brambilla and P. Maffezzoni, "Statistical method for the analysis of interconnects delay in submicron layouts," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 8, pp. 957-966, 2001.

[8] C. Brennan, J. Kozhaya, R. Proctor, J. Sloan, S. Chang, J. Sundquist and T. Lowe, "ESD design automation for a 90nm ASIC design system," *Proceedings of Electrical Overstress/Electrostatic Discharge Symposium*, 2004.

[9] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can recursive bisection alone produce routable placements?," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 477-482, 2000.

[10] H. Chen, C. Cheng, N. Chou, A. B. Kahng, J. F. MacDonald, P. Suaris, B. Yao and Z. Zhu, "An algebraic multigrid solver for analytical placement with layout based clustering," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 794-799, 2003.

[11] H. H. Chen and D. D. Ling, "Power supply noise analysis methodology for deep-submicron VLSI chip design," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 638-643, 1997.

[12] T. Chen and C. C. Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 559-562, 2001.

[13] E. G. Jr. Coffman, M. R. Garey and D. S. Johnson, "Approximation algorithms for bin packing: a survey," *Approximation Algorithms for NP-Hard Problems*, pp. 46-93, PWS Publishing, Boston, MA, 1997.

[14] J. H. Curtiss, “Sampling methods applied to differential and difference equations,” *Proceedings of IBM Seminar on Scientific Computation*, pp. 87-109, 1949.

[15] E. Cuthill and J. McKee, “Reducing the bandwidth of sparse symmetric matrices,” *Proceedings of the ACM National Conference*, pp. 157-172, 1969.

[16] A. Devgan, H. Ji and W. Dai, “How to efficiently capture on-chip inductance effects: Introducing a new circuit element K,” *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 150-155, 2000.

[17] A. Dharchoudhury, R. Panda, D. Blaauw, R. Vaidyanathan, B. Tutuianu and D. Bearden, “Design and analysis of power distribution networks in PowerPC™ microprocessors,” *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 738-743, 1998.

[18] P. G. Doyle and J. L. Snell, *Random Walks and Electric Networks*, Mathematical Association of America, Washington, DC, 1984.

[19] I. S. Duff, A. M. Erisman and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, New York, NY, 1986.

[20] J. P. Eckhardt and K. A. Jenkins, “PLL phase error and power supply noise,” *IEEE 7th Topical Meeting on Electrical Performance of Electronic Packaging*, pp. 73-76, 1998.

[21] H. Eisenmann and F. M. Johnannes, “Generic global placement and floorplanning,” *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 269-274, 1998.

[22] G. E. Forsythe and R. A. Leibler, “Matrix inversion by a Monte Carlo method,” *Mathematical Tables and Other Aids to Computation*, vol. 4, no. 31, pp. 127-129, 1950.

[23] *Fundamentals of ESD*, ESD Association, Rome, NY, 2001.

[24] A. George and J. W. H. Liu, “The evolution of the minimum degree ordering algorithm,” *SIAM Review*, vol. 31, no. 1, pp. 1-19, 1989.

[25] A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[26] B. Goplen and S. S. Sapatnekar, “Efficient thermal placement of standard cells in 3D ICs using a force directed approach,” *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 86-89, 2003.

[27] F. W. Grover, *Inductance Calculations*, Dover Publications, New York, NY, 1954.

[28] GSRC Floorplan Benchmarks. Available at `http://www.cse.ucsc.edu/research/surf/GSRC/progress.html`

[29] W. Hackbusch, *Multi-Grid Methods and Applications*, Springer Verlag, New York, NY, 1985.

[30] J. H. Halton, “Sequential Monte Carlo,” *Proceedings of the Cambridge Philosophical Society*, vol. 58, pp. 57-78, 1962.

[31] J. H. Halton, “A retrospective and prospective survey of the Monte Carlo method,” *SIAM Review*, vol. 12, no. 1, pp. 1-63, 1970.

[32] J. H. Halton, "Sequential Monte Carlo techniques for the solution of linear systems," *Journal of Scientific Computing*, vol. 9, pp. 213-257, 1994.

[33] J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods*, Methuen & Co. Ltd., London, UK, 1964.

[34] P. Heggernes, S. C. Eisenstat, G. Kumfert and A. Pothen, "The computational complexity of the Minimum Degree algorithm," *Proceedings of 14th Norwegian Computer Science Conference*, pp. 98-109, 2001.

[35] R. Hersh and R. J. Griego, "Brownian motion and potential theory," *Scientific American*, vol. 220, pp. 67-74, 1969.

[36] C. Ho, A. E. Ruehli and P. Brennan, "The modified nodal approach to network analysis," *IEEE Transactions on Circuits and Systems*, vol. 22, no. 6, pp. 504-509, 1975.

[37] A. P. Hurst, P. Chong and A. Kuehlmann, "Physical placement driven by sequential timing analysis," *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 379-386, 2004.

[38] *JEDEC Standard No. 22-C101-A: Field-Induced Charged-Device Model Test Method for Electrostatic-Discharge-Withstand Thresholds of Microelectronic Components*, JEDEC Solid State Technology Association, Arlington, VA, 2000.

[39] H. Ji, A. Devgan and W. Dai, "KSim: a stable and efficient RKC simulator for capturing on-chip inductance effect," *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 379-384, 2001.

[40] R. Jiang, T. Chen and C. C. Chen, "PODEA: power delivery efficient analysis with realizable model reduction," *Proceedings of the International Symposium on Circuits and Systems*, vol. 4, pp. 608-611, 2003.

[41] M. D. Ker and H. Chang, "Whole-chip ESD protection strategy for CMOS IC's with multiple mixed-voltage power pins," *Proceedings of International Symposium on VLSI Technology, Systems and Applications*, pp. 298-301, 1999.

[42] D. S. Kershaw, "The incomplete cholesky-conjugate gradient method for the iterative solution of systems of linear equations," *Journal of Computational Physics*, vol. 26, pp. 43-65, 1978.

[43] C. N. Klahr, "A Monte Carlo method for the solution of elliptic partial differential equations," in *Mathematical Methods for Digital Computers*, chap. 14, John Wiley and Sons, New York, NY, 1962.

[44] A. W. Knapp, "Connection between Brownian motion and potential theory," *Journal of Mathematical Analysis and Application*, vol. 12, pp. 328-349, 1965.

[45] D. Kouroussis and F. N. Najm, "A static pattern-independent technique for power grid voltage integrity verification," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 99-104, 2003.

[46] J. A. Jr. Kowaleski, T. Truex, D. Dever, D. Ament, W. Anderson, L. Bair, S. Bakke, D. Bertucci, R. Castelino, D. Clay, J. Clouser, A. DiPace, V. Germini, R. Hokinson, C. Houghton, H. Kolk, B. Miller, G. Moyer, R. O. Mueller, N. O'Neill, D. A. Ramey, Y. Seok, J. Sun, G. Zelic and V. Zlatkovic, "Implementation of an Alpha microprocessor in SOI," *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pp. 248-249, 2003.

[47] J. Kozhaya, S. R. Nassif and F. N. Najm, “A multigrid-like technique for power grid analysis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 10, pp. 1148-1160, 2002.

[48] K. S. Kundert and A. Sangiovanni-Vincentelli, SPARSE 1.3: A Sparse Linear Equation Solver. Available at
`http://www.netlib.org/sparse`

[49] Y. L. Le Coz and R. B. Iverson, “A stochastic algorithm for high speed capacitance extraction in integrated circuits,” *Solid-State Electronics*, vol. 35, no. 7, pp. 1005-1012, 1992.

[50] J. Lee, K. Kim, Y. Huh, P. Bendix and S. Kang, “Chip-level charged-device modeling and simulation in CMOS integrated circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 1, pp. 67-81, 2003.

[51] Q. Li, Y. Huh, J. Chen, P. Bendix and S. Kang, “Full chip ESD design rule checking,” *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 503-506, 2001.

[52] P. Li, “Variational analysis of large power grids by exploring statistical sampling sharing and spatial locality,” *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 645-651, 2005.

[53] T. Li, S. Ramaswamy, E. Rosenbaum and S. Kang, “Circuit-level simulation and layout optimization for deep submicron EOS/ESD output protection device,” *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 159-162, 1997.

[54] T. Li, C. Tsai, E. Rosenbaum and S. Kang, "Modeling, extraction and simulation of CMOS I/O circuits under ESD stress," *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, vol. 6, pp. 389-392, 1998.

[55] A. W. Marshall, "The use of multi-stage sampling schemes in Monte Carlo," *Symposium of Monte Carlo Methods*, pp. 123-140, John Wiley & Sons, New York, NY, 1956.

[56] O. J. McAteer, *Electrostatic Discharge Control*, McGraw-Hill, New York, NY, 1990.

[57] MCNC Floorplan Benchmark Suite. Available at `http://www.cse.ucsc.edu/research/surf/GSRC/MCNC`

[58] M. E. Muller, "Some continuous Monte Carlo methods for the Dirichlet problem," *Annals of Mathematical Statistics*, vol. 27, pp. 569-589, 1956.

[59] R. Panda, D. Blaauw, R. Chaudhury, V. Zolotov, B. Young and R. Ramaraju, "Model and analysis for combined package and on-chip power grid simulation," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 179-184, 2000.

[60] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in C: the Art of Scientific Computing*, second edition, Cambridge University Press, New York, NY, 1994.

[61] H. Qian, J. N. Kozhaya, S. R. Nassif and S. S. Sapatnekar, "A chip-level electrostatic discharge simulation strategy," *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 315-318, 2004.

[62] H. Qian, S. R. Nassif and S. S. Sapatnekar, “Random walks in a supply network,” *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 93-98, 2003.

[63] H. Qian, S. R. Nassif and S. S. Sapatnekar, “Power grid analysis using random walks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 8, pp. 1204-1224, 2005.

[64] H. Qian, S. R. Nassif and S. S. Sapatnekar, “Early-stage power grid analysis for uncertain working modes,” *Proceedings of International Symposium on Physical Design*, pp. 132-137, 2004.

[65] H. Qian, S. R. Nassif and S. S. Sapatnekar, “Early-stage power grid analysis for uncertain working modes,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 5, pp. 676-682, 2005.

[66] H. Qian and S. S. Sapatnekar, “Hierarchical random-walk algorithms for power grid analysis,” *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 499-504, 2004.

[67] H. Qian and S. S. Sapatnekar, “A hybrid linear equation solver and its application in quadratic placement,” *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 905-909, 2005.

[68] H. Qian and S. S. Sapatnekar, The Hybrid Linear Equation Solver Binary Release. Available at
`http://mountains.ece.umn.edu/~sachin/hybridsolver`

[69] G. M. Royer, “A Monte Carlo procedure for potential theory problems,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 19, no. 10, pp. 813-818, 1971.

[70] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, 2003.

[71] S. S. Sapatnekar and H. Su, "Analysis and optimization of power grids," *IEEE Design and Test of Computers*, vol. 20, pp. 7-15, 2003.

[72] N. Sclater, *Electrostatic Discharge Protection for Electronics*, TAB Books, Blue Ridge Summit, PA, 1990.

[73] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 2, pp. 510-522, 1985.

[74] J. C. Shah, A. A. Younis, S. S. Sapatnekar and M. M. Hassoun, "An algorithm for simulating power/ground networks using Padé approximations and its symbolic implementation," *IEEE Transaction on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, pp. 1372-1382, 1998.

[75] S. Sinha, H. Swaminathan, G. Kadamati and C. Duvvury, "An automated tool for detecting ESD design errors," *Proceedings of Electrical Overstress/Electrostatic Discharge Symposium*, pp. 208-217, 1998.

[76] T. Skalicky, LASPack. Available at `http://www.mgnet.org/mgnet/Codes/laspack`

[77] SPEC CPU2000 Results. Available at `http://www.specbench.org/cpu2000/results/cpu2000.html`

[78] A. Srinivasan and V. Aggarwal, "Stochastic linear solvers," *Proceedings of the SIAM Conference on Applied Linear Algebra*, 2003.

[79] G. Steele, D. Overhauser, S. Rochel and S. Z. Hussain, "Full-chip verification methods for DSM power distribution systems," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 744-749, 1998.

[80] J. Stinson and S. Rusu, "A 1.5 GHz third generation Itanium processor," *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pp. 252-253, 2003.

[81] H. Su, K. H. Gala and S. S. Sapatnekar, "Fast analysis and optimization of power/ground networks," *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 477-480, 2000.

[82] C. J. K. Tan and M. F. Dixon, "Antithetic Monte Carlo linear solver," *Proceedings of International Conference on Computational Science*, pp. 383-392, 2002.

[83] R. S. Varga, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.

[84] N. Viswanathan and C. C. Chu, "FastPlace: efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," *Proceedings of International Symposium on Physical Design*, pp. 26-33, 2004.

[85] K. P. Vorwerk, A. Kennings and A. Vannelli, "Engineering details of a stable force-directed placer," *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 573-580, 2004.

[86] K. P. Vorwerk and A. Kennings, "An improved multi-level framework for force-directed placement," *Proceedings of ACM/IEEE Design Automation and Test in Europe*, pp. 902-907, 2005.

[87] W. Wasow, “A note on the inversion of matrices by random walks,” *Mathematical Tables and Other Aids to Computation*, vol. 6, no. 38, pp. 78-81, 1952.

[88] R. D. Yates and D. J. Goodman, *Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers*, John Wiley and Sons, New York, NY, 1999.

[89] M. Zhao, R. V. Panda, S. S. Sapatnekar and D. Blaauw, “Hierarchical analysis of power distribution networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 2, pp. 159-168, 2002.