

Large power grid analysis using domain decomposition

Quming Zhou[†], Kai Sun[‡], Kartik Mohanram[†], Danny C. Sorensen[‡]

[†]Department of Electrical and Computer Engineering, Rice University, Houston, USA
{quming, kmram}@rice.edu

[‡]Department of Computational and Applied Mathematics, Rice University, Houston, USA
{kleinsun, sorensen}@rice.edu

Abstract: This paper presents a domain decomposition (DD) technique for efficient simulation of large-scale linear circuits such as power distribution networks. Simulation results show that by integrating the proposed DD framework, existing linear circuit simulators can be extended to handle otherwise intractable systems.

1. Introduction

With increasing power consumption and faster operating frequency of microprocessors, the design of effective power distribution networks has emerged as a critical design challenge. The parasitics associated with the power distribution network and time-varying circuit currents induce supply voltage variations across the integrated circuit. Such power supply variations may impact circuit performance and compromise noise immunity. Extensive simulations are usually performed to identify and correct such instances during design. The power distribution network typically has a grid structure and is commonly referred to as the power grid. The power grid for a modern integrated circuit may consist of several million electrical elements, which makes simulation computationally (time-resource) intensive.

The power grid is traditionally described as a large-scale linear system. Simulation of power grids usually involves both DC and transient analysis. DC analysis of the power grid is used to estimate the IR voltage drop by solving the system once, whereas transient analysis involves the simulation of the system at several time instants to estimate the Ldi/dt voltage drop. Note that an efficient DC analysis technique does not necessarily translate to efficient transient analysis.

Several simulation techniques have been developed for power grid simulation and analysis in literature. Since the computational complexity of direct methods to solve linear systems of size n is $O(n^3)$, sparsity and the grid structure in the power distribution network are usually exploited to reduce computational complexity [1], [6], [12], [15]. A pre-conditioned conjugate gradient iterative method, using incomplete Cholesky factorization as the pre-conditioner, was described in [1]. Although this pre-conditioner-based iterative method reduces the computational complexity of DC analysis of power grids from $O(n^3)$ to $O(n^2)$, it is not efficient for transient analysis since it is not possible to leverage previous simulation runs. A multi-grid approach that also exploits the

grid structure by mapping the original system to a coarsened grid, solving the coarsened grid, and remapping back to the original grid was described in [6]. The solution of the original system through remapping is obtained through an interpolation procedure. However, in the absence of error bounds, this method may not always be accurate. Moreover, the effort to keep track of the geometrical information of the power grid is expensive, further limiting its applications. Algebraic multi-grid methods were proposed in [12] and [15] to handle general network topologies. Algebraic multi-grid methods can be thought of as iterative solvers that use the multi-grid operator as a pre-conditioner. In such methods, the computational cost in each time step of the transient analysis is comparable to that for DC analysis, making it unsuitable for efficient transient analysis. Other approaches to power grid analysis include those based on random walks, model order reduction, and hierarchical analysis. Statistical techniques based on random walks [8], [10] are very fast but suffer from accuracy loss and convergence issues. Model order reduction methods are inefficient for power grid simulation due to (i) a large number of external terminals and (ii) the loss of sparsity in the reduced model [4]. Hierarchical techniques are applicable if the power grid is not flattened, and macro-models for local grids can be built to speed up simulation at the global level [7], [14].

In this paper, we present a domain decomposition [11] approach for power grid analysis. The proposed approach allows the solution of several sub-circuits sequentially or in parallel to obtain the solution of the original circuit. Existing circuit simulators can be used to solve the sub-problems obtained by domain decomposition (DD). This approach achieves improved performance in comparison to the original circuit simulator. Simulation results show that by integrating the proposed DD framework, existing linear circuit simulators can be extended to handle otherwise intractable systems. The proposed DD approach can also be parallelized to achieve performance gains over a serial implementation.

Section 2 presents a background on power grid analysis. Section 3 describes the proposed approach based on domain decomposition. Section 4 includes simulation results and a discussion. Section 5 is a conclusion.

2. Background

The power grid can be described using the Modified Nodal Analysis (MNA) [9] as

$$Gx(t) + C\dot{x}(t) = u(t), \quad (1)$$

where x is a n dimensional real vector of node voltages and inductor currents, G is a $n \times n$ conductance matrix, C denotes the capacitance and inductance terms, and $u(t)$ includes the loads and voltage sources. n can be of the order of millions for reasonable size power grids. By applying the backward-Euler method to the system of equations in (1), we obtain

$$(G + C/h)x(t+h) = u(t+h) + Cx(t)/h, \quad (2)$$

where h is a fixed time step for the transient analysis. The system of equations (2) can be rewritten as

$$Ax(t+h) = B, \quad (3)$$

where $A = G + C/h$ and $B = u(t+h) + Cx(t)/h$.

Traditional approaches for the solution of (3) require a single initial factorization of the matrix A , and a substitution pass at each time step during iteration. Whereas factorization is significantly more expensive than the substitution pass, direct factorization of A by LU or Cholesky decomposition may be justified for use in transient analysis. This is because the factorization can be re-used in each time step, and the cost of factorization is negligible when amortized over the several time steps used in transient analysis. However, direct factorization of a large matrix A is computationally (CPU time and memory) intractable with increasing system size. Pre-conditioner-based iterative methods work around this by avoiding the initial factorization step. Instead, the system of equations in (3) is solved at each time step repeatedly with a time-dependent vector B . However, although iterative methods save on factorization costs, the cumulative costs over several time steps during transient analysis may exceed the cost of direct factorization. In this paper, we propose a divide-and-conquer approach based upon DD. It can complement traditional factorization-based as well as pre-conditioner-based iterative methods, and provide improvements in both runtime and problem tractability for both classes of approaches.

3. Domain decomposition (DD)

Domain decomposition methods, which are based on the general concepts of graph partitioning, refer to a technique of divide-and-conquer that have been primarily developed for solving partial differential equations [11].

Suppose the power grid described by (3) has been partitioned into m sub-domains Ω_i , $i = 1, 2, \dots, m$. We defer the discussion of partitioning to Sec. 3.4 in this paper. Based upon the partitioning, the nodes in the original system can be classified into (i) interior nodes of sub-domains and (ii) interface nodes. Note that interface nodes are not contained in any of the sub-domains. Based upon this, for a general partitioning of

the original system into m sub-domains, (3) has the following structure:

$$\begin{pmatrix} A_1 & & & E_1 \\ & A_2 & & E_2 \\ & & \ddots & \vdots \\ & & & A_m & E_m \\ F_1 & F_2 & \cdots & F_m & A_\Gamma \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \\ g \end{pmatrix} \quad (4)$$

In this system, the matrices A_1, A_2, \dots, A_m correspond to the m sub-domains, A_Γ corresponds to the interface nodes, and E_1, E_2, \dots, E_m and F_1, F_2, \dots, F_m are matrices that capture connectivity information between the interface and the corresponding sub-domain. Each x_i represents the sub-vector of state variables that are interior to sub-domain Ω_i , and y represents the sub-vector of all interface variables. The matrices f_1, f_2, \dots, f_m correspond to the loads and voltage sources contained within the corresponding sub-domain Ω_i , and the matrix g corresponds to the loads and voltage sources at the interface nodes.

Theorem 3.1: If there is no direct coupling capacitance or direct mutual inductance between sub-domains, the systems described by equations (3) and (4) are equivalent.

Proof: When DD is used to partition the original system, the nodes are classified into either interior nodes or interface nodes. Cross terms between x_i and x_j ($i \neq j$) are encountered during MNA only if coupling capacitances or mutual inductances exist in the system. Since such effects can be negligible at the interface nodes (see note below), the cross terms are zero and the two systems are equivalent. ■

Note that the assumption of Theorem 3.1 is generally true because mutual inductance effects in the power grid are still negligible [13]. Furthermore, flip-chip technology uses C4 bumps that provide a more direct power delivery path in modern designs, resulting in power grid shells [2]. This property can be exploited to select interface nodes that naturally isolate the original system into sub-domains.

The following propositions follow directly from the above theorem:

Proposition 3.1: If A is symmetric and positive definite, A_i is also symmetric and positive definite with $F_i = E_i^T$.

Proposition 3.2: For an original system with n nodes, the size of x_i is $O(n)$ and the size of y is $O(\sqrt{n})$.

3.1. Schur complement

To simplify notation, (4) can be rewritten as

$$A \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \text{ where } A = \begin{pmatrix} A_D & E \\ F & A_\Gamma \end{pmatrix}. \quad (5)$$

Consider the linear system written in (5), in which A_D is assumed to be non-singular. From the first equation, the sub-domain variables x can be expressed as

$$x = A_D^{-1}(f - Ey). \quad (6)$$

Substitution of this form into the second equation yields the following system of equations:

$$(A_\Gamma - FA_D^{-1}E)y = g - FA_D^{-1}f. \quad (7)$$

The matrix

$$A_\Gamma - FA_D^{-1}E \quad (8)$$

is called the *Schur complement* matrix associated with the interface variables y . Solving equation (6) constitutes the computational core of the DD technique. A consolidated three-step procedure to solve the original system is as follows. First form the Schur complement matrix $A_\Gamma - FA_D^{-1}E$ and the right hand side of (7). Next, solve equation (7) for the interface variables y . Finally, by substituting y in equation (6), obtain a solution for the sub-domain variables x .

A high-level DD algorithm DOMAIN-DECOMPOSITION-SOLVER that takes as inputs A_D , A_Γ , E , F , f and returns the solution for the interface variables y and the sub-domain variables x is as follows.

- 1) Solve $A_D P = E$ for P , i.e., obtain $P = A_D^{-1}E$
- 2) Form the Schur complement matrix $S = A_\Gamma - FP$
- 3) Solve $A_D q = f$ for q , i.e., obtain $q = A_D^{-1}f$
- 4) Calculate $g' = g - Fq$
- 5) Solve the equation $Sy = g'$ for interface variables y
- 6) Solve $x = q - Py$ for the sub-domain variables x

There are three opportunities for parallelization in the above algorithm, all of which exploit the block diagonal structure of matrix A_D . They are in (i) step 1 where the equation $A_D P = E$ is solved, (ii) step 3 where the equation $A_D q = f$ is solved, and (iii) step 6 where the equation $x = q - Py$ is solved.

3.2. An example of two sub-domains

An example based on two sub-domains to illustrate the above algorithm is as follows. With two sub-domains, equation (3) has the following form:

$$\begin{pmatrix} A_1 & 0 & E_1 \\ 0 & A_2 & E_2 \\ F_1 & F_2 & A_\Gamma \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ y \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ g \end{pmatrix} \quad (9)$$

where x_1 , x_2 , and y are the vectors of node voltage and inductor current in sub-domain 1, sub-domain 2, and the interface respectively. For this system (9), the Schur complement matrix is

$$S = A_\Gamma - F_1 A_1^{-1} E_1 - F_2 A_2^{-1} E_2. \quad (10)$$

The interface variable y can be obtained by solving the equation

$$Sy = g - F_1 A_1^{-1} b_1 - F_2 A_2^{-1} b_2. \quad (11)$$

Finally, x_1 and x_2 can be obtained by

$$x_1 = A_1^{-1} b_1 - A_1^{-1} E_1 y, \quad (12)$$

$$x_2 = A_2^{-1} b_2 - A_2^{-1} E_2 y. \quad (13)$$

Suppose that the inverse operator A_i^{-1} is implemented by LU factorization. For the example above, the LU factorization is performed for A_1 and A_2 . If A_i is half the size of the

original matrix A , the total factorization cost for A_1 and A_2 is less than the factorization cost of A . Note that $A_1^{-1} E_1$ and $A_2^{-1} E_2$ are computed by solving the equations $A_i P_i = E_i$ for P_i , and that the E_i are sparse matrices. Once the Schur complement matrix S given by

$$S = A_\Gamma - F_1 P_1 - F_2 P_2 \quad (14)$$

is obtained, the equations $A_i q_i = b_i$ are solved for q_i . The q_i are easy to compute, since the LU factorizations for the A_i can be reused. The interface variables can be computed by solving $Sy = g - F_1 q_1 - F_2 q_2$. Here, S is a small matrix in comparison to A . Finally, the sub-domain variables x_i are obtained by solving the equation

$$x_i = q_i - P_i y. \quad (15)$$

3.3. Complexity analysis

Without loss of generality, assume that a circuit with n nodes is partitioned into m equal sub-circuits. Then, each sub-domain approximately has n/m nodes. Suppose that solving a linear system by direct or iterative methods has a complexity of $O(n^p)$, where n is the size of the linear system and $p \geq 1$. Using the proposed DD technique, the computational cost of solving a single sub-system is $O((n/m)^p)$. Multiplying by m , the complexity of the solution for the entire system is $O((n^p)/(m^{p-1}))$. Note that this only occurs in the theoretically ideal case. For most practical cases, it is essential to factor in auxiliary computations in DD algorithms, such as addition, multiplication, the formation of the Schur complement matrix (which is much smaller than the sub-domain matrices), and the cost for solving the interface. Based on this, it is reasonable to expect that a parallel version of the DD algorithm will achieve speed-up over the serial case for large n .

3.4. Circuit partitioning

It is well known that general graph partitioning is a \mathcal{NP} -hard problem. The proposed DD technique uses an efficient but sub-optimal algorithm to partition the circuit into nearly balanced sub-domains.

Consider a circuit that needs to be partitioned into m sub-domains. Begin by selecting a subset of m nodes in the circuit such that the pairwise distance between them is nearly equal. The distance between two nodes is defined in a graph-theoretic sense, and is measured in terms of the length of the shortest path needed to reach one node from the other. Such nodes are termed the cores of the sub-domains. The expansion starts from the core of a sub-domain. At every step, each core incorporates the nodes that are closest to it in terms of distance. The sub-domains thus grow larger and larger during the expansion. A node can be incorporated exactly once into exactly one sub-domain. The expansion stops when all the nodes are incorporated into a sub-domain. Those nodes whose neighbors belong to other domains are marked as the interface nodes and are removed from their domains. In this manner, the entire circuit is partitioned into m sub-domains and the interface.

The following algorithm DOMAIN-PARTITION takes as input a circuit with n nodes and m core nodes v_1, v_2, \dots, v_m , and returns m sub-domains as the output.

- 1) Assign default color 0 to all the nodes,
 - for** ($i = 1: 1: m$) **do** $\text{color}(v_i) = i$
- 2) **define** node-set = $\{v_i : 1 \leq i \leq m\}$
interface = \emptyset , nNodes = m
- 3) **while** (nNodes $\leq n$) **do**
 - **for** $v_i \in \text{node-set}$ **do**
 - **for** every neighbor of v_i , say v_j , **do**
 - * **if** $\text{color}(v_j) == 0$
 - $\text{color}(v_j) = \text{color}(v_i)$
 - nNodes = nNodes + 1
 - node-set = node-set $\cup \{v_j\}$
 - * **else**
 - interface = interface $\cup \{v_j\}$
 - $\text{color}(v_i) = m+1$
 - * **end**
 - **end**
 - **end**

This algorithm is a simple solution to partition a circuit into sub-circuits. There are no common nodes between different domains. Fig. 1 shows an example, in which the matrix A was partitioned into four small domains and the interface. The sparse structure of A was preserved in the sub-domains. The interface nodes account for a small fraction of all nodes and this proportion decreases as the problem size increases.

3.5. Sub-domain solvers

The different implementations of DD are characterized by their sub-domain solvers, which has a profound impact on the performance in terms of CPU time and memory requirements. In general, there are two kinds of solvers, direct solvers and iterative solvers.

Direct solvers perform a factorization such as Gaussian elimination on the matrix A to get lower and upper triangular matrices L and U . The system is then solved using forward and backward substitution on the vector B . After factorization, direct solvers are usually much faster than iterative solvers. The primary drawbacks of direct methods are memory requirement and fill-in.

Iterative methods [11] use simple matrix operations to avoid the costs of direct factorization. Two of the most common computational kernels are matrix-vector multiplication or transpose matrix-vector multiplication. The pre-conditioned iterative solvers can be very fast when solving a system a few times. However, as the number of simulation steps increase, the iterative solver is rendered inefficient due to the similar workload at each time step.

A good pre-conditioner is necessary for most iterative solvers. Pre-conditioning is a kind of modification of the original problem that accelerates the iterative methods. For example, in solving a linear system $Ax = B$, an explicit or

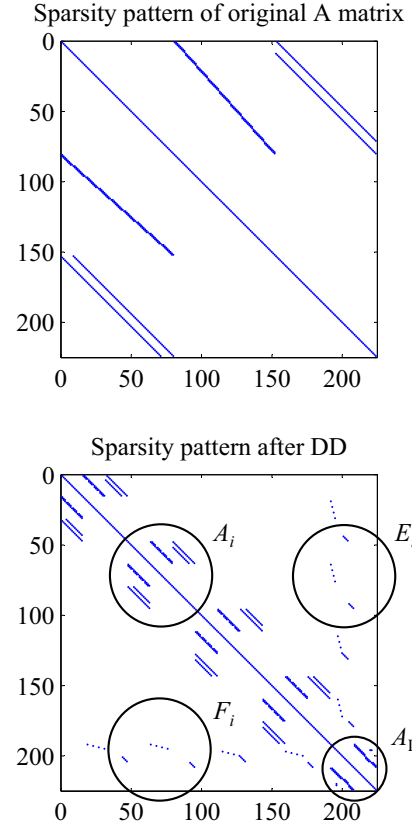


Fig. 1. This figures illustrates circuit partitioning with four domains and a single interface. The sparse structure of the original matrix A (top-half of the figure) was preserved in each of the sub-domains A_i (bottom-half of the figure). The matrices E_i , F_i , and A_Γ are also circled for illustration in the bottom-half of the figure.

implicit operator M^{-1} such that $M \approx A$ is sought to solve $M^{-1}Ax = M^{-1}B$ efficiently. The pre-conditioners used in this paper are based on incomplete factorization and algebraic multi-grid methods. For a sparse matrix A , the incomplete factorization computes a sparse lower triangular matrix L and a sparse upper triangular matrix U . The purpose is to make the residual $R = A - LU$ satisfy some constraints. The basic idea of multi-grid pre-conditioning is to approximate the original system by interpolations of a smaller system on coarse grids.

4. Results

The proposed domain decomposition method has been implemented and integrated into a linear simulator written in C++. Mesh networks were used to model the upper-two global power grids, which consist of RLC wires, voltage sources, current sources, and decoupling capacitors. The interface nodes are approximately $(m/2)\sqrt{n}$, where m is the number of domains and n is the total number of nodes in the network. Fig. 2 shows a part of a grid model used in our experiments.

4.1. Simulation environment

- CPU: AMD AthlonXP 2600+ (1.9GHz)
- Memory: 2.0 GB
- OS: Red Hat Enterprise Linux

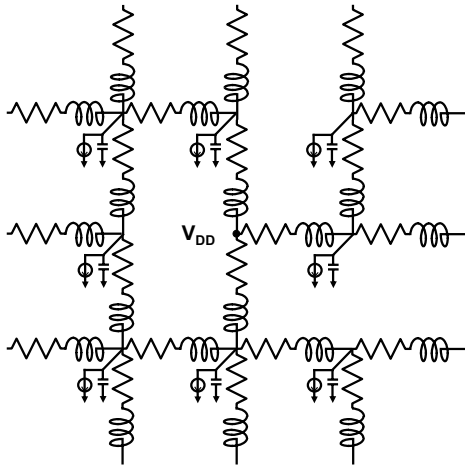


Fig. 2. This figure illustrates a small portion of the power grids used for the simulations. Since the matrix A for such systems has a sparse symmetric positive definite structure [1], the conjugate gradient algorithm can be used for simulation.

- Compilers: GNU g++
- Packages: Trilinos 5.0 [5], UMFPACK 4.4 [3]

4.2. Simulation results

The simulation results for four circuits, A with 10 K nodes, B with 250 K nodes, C with 1 M nodes, and D with 4 M nodes are presented here. Circuits A, B, and C were partitioned into four domains. Circuit D was partitioned into sixteen domains.

Tables 1 and 2 report the average CPU time in seconds for DC and 20-step transient simulations of the four circuits respectively. Note that “—” indicates that the algorithm either ran out of memory or that it could not finish in a reasonable time allowed for completion. The different power grid simulation frameworks that were implemented (or used) are summarized below.

- SPICE: the general circuit simulator
- LU: computing Cholesky factorization, solving the linear system k times by forward and backward solvers;
- IF: generating incomplete Cholesky factorization pre-conditioner (IF) with drop tolerance 0.001, solving the linear system k times by conjugate gradient method with IF pre-conditioner;
- MG: generating one-level multi-grid pre-conditioner (MG) with direct solver on coarse grid and Gauss-Seidel prior/post smoothers, solving the linear system k times by conjugate gradient method with MG pre-conditioner;
- DD+LU: DD method solving the linear system k times with direct solver (LU) for sub-domain problems;
- DD+IF: DD method solving the linear system k times with iterative solver (IF pre-conditioner) for sub-domain problems;
- DD+MG: DD method solving the linear system k times with iterative solver (MG pre-conditioner) for sub-domain problems.

From these simulation results, it is possible to draw the following conclusions:

Table 1
Runtimes for DC simulation (secs)

Circuit	A	B	C	D
No. nodes	10 K	250 K	1 M	4 M
No. sub-domains	4	4	4	16
SPICE	64	—	—	—
LU	0.25	46	450	—
IF	0.14	22	51	—
MG	0.60	20	88	—
DD+LU	0.70	38	291	—
DD+IF	0.76	64	352	1644
DD+MG	0.94	45	320	1477

Table 2

Runtimes for transient simulation with 20 time steps (secs)

Circuit	A	B	C	D
No. nodes	10 K	250 K	1 M	4 M
No. sub-domains	4	4	4	16
SPICE	133	—	—	—
LU	0.80	66	540	—
IF	2.26	429	953	—
MG	5.78	192	763	—
DD+LU	1.09	54	365	—
DD+IF	2.23	590	1597	7271
DD+MG	5.81	204	948	3928

- DC simulations: If sufficient memory to generate the incomplete factorization pre-conditioner is available, the iterative solver with IF pre-conditioner is the fastest method. Otherwise, DD with iterative solvers (with IF or MG pre-conditioners) on sub-domain problems can handle extra large size problems;
- Transient simulations: If sufficient memory for direct solvers on sub-domains is available, DD+LU is the fastest method. Otherwise, DD+MG is the best method, especially for extra large size problems.

In Table 1, in the DC simulation of Circuit C with 1 M nodes, the IF method is faster than the DD+IF method. This is because the computational cost of solving the interface nodes through a dense Schur complement matrix S defined by equation 8 may be higher than the gain from solving sub-domains. Our simulations use very tightly connected power grids. As a result, the interface nodes constitute a large fraction of the total number of nodes and this leads to a large dense Schur complement matrix. This observation is true of other runs on circuits A, B, and C as well. In some designs, the number of interface nodes may be much smaller than the square root of the total number of nodes. For instance, a large memory array with three local metal layers may have several millions of nodes but only a few hundred interface nodes connected to the upper global grid [14]. In such cases, the Schur complement matrix is small and does not affect performance even if it is dense.

Note that the Trilinos [5] package we used for computation has already been well optimized for large scale sparse computing. The direct solver that was used, UMFPACK [3], for

LU or Cholesky factorization is also one of the best available packages. Our DD-based approach has shown that it is possible to improve the performance of existing algorithms for power grid analysis. The improvement will predictably be greater for a parallel implementation since the decoupled structures obtained by DD can be simulated independently of each other with cheap synchronization steps.

5. Conclusion

This paper described an efficient domain decomposition framework for large-scale power grid simulation. Simulation results indicate that this approach can be used to extend existing linear circuit simulators to handle otherwise intractable systems. Further, although the simulation runs were performed serially on a single machine for fair comparison, the special structure that arises from the use of domain decomposition is amenable to parallelization and further performance gains.

References

- [1] T. Chen and C. C. Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods," *Proc. Design Automation Conference*, pp. 559–562, 2001.
- [2] E. Chiprout, "Fast flip-chip power grid analysis via locality and grid shells," *Proc. Intl. Conference on Computer-aided Design*, pp. 485–488, 2004.
- [3] T. A. Davis, "UMFPACK version 4.4 user guide", <http://www.cise.ufl.edu/research/sparse/umfpack/>.
- [4] P. Feldmann and F. Liu, "Sparse and efficient reduced order modeling of linear subcircuits with large number of terminals," *Proc. Intl. Conference on Computer-aided Design*, pp. 88–92, 2004.
- [5] M. A. Heroux and J. M. Willenbring, "Trilinos users guide", Sandia National Laboratories, SAND2003-2952, 2003, <http://software.sandia.gov/trilinos/>.
- [6] J. N. Kozhaya, S. R. Nassif, and F. N. Najm, "A multigrid-like technique for power grid analysis," *IEEE Trans. Computer-aided Design*, vol. 21, pp. 1148–1160, Oct. 2002.
- [7] Y.-M. Lee, Y. Cao, T. H. Chen, J. Meiling, and C.C. Chen, "HiPRIME: Hierarchical and passivity preserved interconnect macromodeling engine for RLKC power delivery," *IEEE Trans. Computer-aided Design*, vol. 24, pp. 797–806, Jun. 2005.
- [8] P. Li, "Power grid simulation via efficient sampling-based sensitivity analysis and hierarchical symbolic relaxation," *Proc. Design Automation Conference*, pp. 664–669, 2005.
- [9] L. T. Pillage, R. A. Rohrer, and C. Visweswariah, *Electronic and system simulation methods*, McGraw Hill, New York, 1994.
- [10] H. Qian, S. R. Nassif, and S. S. Sapatnekar, "Power grid analysis using random walks," *IEEE Trans. Computer-aided Design*, vol. 24, pp. 1204–1224, Aug. 2005.
- [11] Yousef Saad, *Iterative methods for sparse linear systems*, SIAM, 2003.
- [12] H. Su, E. Acar, and S. R. Nassif, "Power grid reduction based on algebraic multigrid principles," *Proc. of Design Automation Conference*, pp. 109–112, 2003.
- [13] H. Su, K. H. Gala, and S. S. Sapatnekar, "Analysis and optimization of structured power/ground networks," *IEEE Trans. Computer-aided Design*, vol. 22, pp. 1533–1544, Nov. 2003.
- [14] M. Zhao, R. V. Panda, S. S. Sapatnekar, and D. Blaauw, "Hierarchical analysis of power distribution networks," *IEEE Trans. Computer-aided Design*, vol. 21, pp. 159–168, Feb. 2002.
- [15] Z. Zhu, B. Yao, and C.-K. Cheng, "Power network analysis using an adaptive algebraic multigrid approach", *Proc. Design Automation Conference*, pp. 105–108, 2003.