Fast Transform-Based Preconditioners for Large-Scale Power Grid Analysis on Massively Parallel Architectures

Konstantis Daloukas, Nestor Evmorfopoulos, George Drasidis, Michalis Tsiampas,
Panagiota Tsompanopoulou and George I. Stamoulis

Department of Computer and Communications Engineering,
University of Thessaly, Volos
Greece
Email: {kodalouk, nestevmo, gedrasid, mitsiaba, yota, georges}@inf.uth.gr

***Abstract*—Efficient analysis of massive on-chip power delivery networks is among the most challenging problems facing the EDA industry today. In this paper, we present a new preconditioned iterative method for fast DC and transient simulation of large-scale power grids found in contemporary nanometer-scale ICs. The emphasis is placed on the preconditioner which reduces the number of iterations by a factor of $5X$ for a 2.6M-node industrial design and by $72.6X$ for a 6.2M-node synthetic benchmark, compared with incomplete factorization preconditioners. Moreover, owing to the preconditioner's special structure that allows utilizing a Fast Transform solver, the preconditioning system can be solved in a near-optimal number of operations, while it is extremely amenable to parallel computation on massively parallel architectures like graphics processing units (GPUs). Experimental results demonstrate that our method achieves a speed-up of $214.3X$ and $138.7X$ for a 2.6M-node industrial design, and a speed-up of $1610.5X$ and $438X$ for a 3.1M-node synthetic design, over state-of-the-art direct and iterative solvers respectively when GPUs are utilized. At the same time, its matrix-less formulation allows for reducing the memory footprint by up to 33% compared to the memory requirements of the best available iterative solver.**

I. Introduction

The relentless push for high-performance and low-power integrated circuits has been met by aggressive technology scaling, which enabled the integration of a vast number of devices on the same die but brought new problems and challenges to the surface. The on-chip power delivery network (power grid) consistutes a vital subsystem of modern nanometer-scale ICs, since it affects in a critical way the performance and correct operation of the devices. In order to determine the quality of the supply voltage delivered to the devices, the designer has to perform static and dynamic simulation of the electrical circuit modeling the power grid. This has become a very challenging problem for contemporary ICs, since power grids encountered in these circuits are extremely large (comprising several thousands or milions of nodes) and very difficult to simulate efficiently (especially over multiple time-steps).

Static (DC) or transient simulation refers to the process of computing the response of an electrical circuit to a constant or time-varying stimulus. Since a power delivery network can be generally modeled as a linear RLC circuit, the process of DC or transient simulation of large-scale power grids amounts to solving very large (and sparse) linear systems of equations. Direct methods (based on matrix factorization) have been widely used in the past for solving the resulting linear systems, mainly because of their robustness in most types of problems. They also have the property of reusability of factorization results in transient simulation with a fixed time-step. Unfortunately, these methods do not scale well with the dimension of the linear system, and become prohibitively expensive for circuits beyond a few thousand elements, in both execution time and memory requirements. In addition, a fixed time-step is almost never used in practice because it becomes very inefficient to constantly simulate during long intervals of low activity. All practical implementations of integration techniques for ordinary differential equations (ODEs) employ a variable or adaptive time-step mechanism [5]. In those cases, the reusability of matrix factorization in direct methods ceases to exist.

Iterative methods involve only inner products and matrix-vector products, and constitute a better alternative for large sparse linear systems in many respects, being more computationally- and memory-efficient. This holds even more so for modern nonstationary iterative methods which fall under the broad class of Krylov-subspace methods [20]. Iterative methods possess themselves a kind of reusability property for transient simulation, in that the solution at the last time-step provides an excellent initial guess for the next time-step, thus making a properly implemented iterative method converge in a fairly small number of iterations. In fact, this property also holds in the case of a variable time-step, since the quality of the last solution as initial guess for the next solution is not affected. The above features make iterative methods much more suitable for DC and variable time-step transient analysis of large-scale linear circuits such as power distribution networks.

The main problem of iterative methods is their unpredictable rate of convergence which depends greatly on the properties (specifically the condition number) of the system matrix. A preconditioning mechanism, which transforms the linear system into one with more favorable properties, is essential to guarantee fast and robust convergence. However, the ideal preconditioner (one that approximates the system matrix well and is inexpensive to construct and apply) differs according to each particular problem and each different type of system matrix. That is why iterative methods have not reached the maturity of direct methods and have not yet gained widespread acceptance in linear circuit simulation. Although general-purpose preconditioners (such as incomplete factorizations or sparse approximate inverses) have been developed, they are not tuned to any particular simulation problems and cannot improve convergence by as much as specially-tailored preconditioners.

Another aspect of circuit simulation that has become very important recently is to uncover hidden opportunities for parallelism in its intermediate steps. This is essential for harnessing the potential of contemporary parallel architectures, such as multi-core processors and graphics processing units (GPUs). GPUs, in particular, are massively parallel architectures whose computational power is about 1580 GFlops/s, greater by an order of magnitude than that of multi-core processors, and as a result they appear as a platform of choice for the efficient execution of computationally-intensive tasks. However, there has been little systematic research for the development of parallel simulation algorithms, and more specifically algorithms for power grid analysis that can be mapped onto massively parallel architectures like GPUs. This can be attributed in part to the difficulty in parallelization of direct linear solution methods that have been mostly employed thus far.

On the contrary, Krylov-subspace iterative methods offer ample

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA.

possibilities for parallelism that have been explored sufficiently well. However, the construction and application of the preconditioner is a very delicate part of parallelizing an iterative method because it is completely application-dependent (and traditional general-purpose preconditioners have very little room for parallelism). This paper presents the design and implementation of a parallel preconditioning mechanism that can be used in conjunction with a Krylov-subspace method, such as Preconditioned Conjugate Gradients (PCG), for efficient DC or transient analysis of power distribution networks. The proposed preconditioner is constructed in a way to approximate very closely the power grid under analysis, so that the total number of iterations of the iterative method is reduced to a great extent. At the same time, its specialized structure allows applying a Fast Transform-based solver that utilizes Fast Fourier Transform (FFT) for the solution of the necessary preconditioning step. The main characteristics of the application of a Fast Transform are the near-optimal operation complexity, as well as its inherent parallelism and low memory requirements, compared to a generic solver for linear systems. As a result, massively parallel architectures such as GPUs can be used to accelerate the simulation algorithm, while at the same time the application's memory demands render feasible the analysis of large power grids on such architectures. Experimental results show that our algorithm implemented on a GPU and applied on a 2.6M-node industrial power grid achieves a speedup of $214.3X$ over CHOLMOD [8] (a state-of-the-art direct linear solver) and $138.7X$ over a parallel version of PCG with incomplete Cholesky preconditioner implemented on a multi-core CPU. To the best of our knowledge, this is the first research effort that proposes a Fast Transform-based preconditioner and its parallel implementation on multi-core and GPU architectures for power grid analysis.

The rest of the paper is organized as follows. We present other research approaches for power grid analysis in Section II. Next, we briefly outline the modeling and time-domain analysis of on-chip power delivery networks as well as the theory behind iterative methods, preconditioning and Fast Transform solvers in Section III. Section IV presents the methodology behind the construction of efficient preconditioners for power grid analysis, as well as the resulting procedure and opportunities for parallel computation. We validate the performance of our approach using a series of power grid designs in Section V. Finally, Section VI concludes the paper.

II. Related Work

The growing need to simulate large power grids with small memory footprint and efficient parallel execution has led many researchers to deviate from the standard practice of direct factorization methods and present more suitable iterative methods.

Research works [7] and [22] have proposed iterative solvers for efficient simulation of power delivery networks. Power grid analysis was first formulated as a symmetric positive definite system to be solved by PCG in [7], but the preconditioner used was the general-purpose (and inefficient for specialized applications) incomplete Cholesky. A different pattern-based preconditioner was proposed in [22], but it is very simple and heuristic and does not appear to reduce the number of iterations significantly. The idea of multi-grid techniques for solving partial differential equations has been proposed for power grid analysis in [14] and [28].

Recently, parallel architectures have been utilized to accelerate power grid analysis. The authors in [23] have proposed domain decomposition as a parallel technique for analysis on multi-core architectures. GPUs are used in research approaches [21], [12], and [13] as parallel platforms that enable tackling power analysis of large-scale power networks. Authors in [13] propose multi-grid as a solution method for power grid analysis and they use multi-core and massively parallel single-instruction multiple-thread (SIMT) platforms to tackle power grid analysis, while authors in [21] formulate the traditional linear system as a special two-dimension Poisson equation and solve it using analytical expressions based on the FFT algorithm, with GPUs being used to further speed up the algorithm. However, both [21] and [13] only solve very regular grid structures with specialized techniques, which can limit their effectiveness for irregular power delivery networks that are found in late design stages. Instead, we propose to use such a regular structure as preconditioner in order to solve any practical (and possibly irregular) power delivery network.

Preconditioning has lately drawn attention as a method for efficiently tackling the analysis of large-scale and irregular power grid designs. Such a possibility is the topic of research works [12] and [25] for power grid analysis, and [18] in the context of IC thermal simulation. In [12], the preconditioning has been carried out by multigrid techniques. However, when used as preconditioner for iterative methods, multigrid is not very efficient because it is an iterative method by itself, and also solves a system approximately which can hinder the convergence of PCG. Moreover, some operations of multigrid are not always well-defined (e.g. mapping by interpolation from coarser to finer grids and back, and correction of solutions), and the construction of approximate matrices for all coarser grids is an expensive setup phase which has to be repeated every time the system is reconstructed in each time-step change during transient analysis. Our approach for preconditioning, based on the application of a Fast Transform, involves a much more straightforward and inexpensive implementation and reconstruction phase for transient simulation, while it also provides analytical solution of the preconditioned system (since it is actually a fast direct method).

The approach in [25] is the one closest to ours, in the sense that it uses a Fast Poisson-based preconditioner to accelerate the convergence rate of Conjugate Gradient. However, the proposed technique is based on the presumed existence of special areas in the grid with zero voltage drop as boundary condition, in order to formulate so-called "Poisson blocks" with Toeplitz matrix structure, while our approach does not necessitate such an assumption.

In the same context, authors in [27] present a support graph-based preconditioner that can provide a significant acceleration to the convergence rate of an iterative method. However, applying this preconditioner requires the solution of a triangular system which can hinder preconditioner's applicability on parallel architectures due to the limited parallelism of triangular solution algorithms. On the contrary, applying our Fast Transform preconditioner has far greater potential for parallelism than both multigrid and triangular solution algorithms, especially on GPUs [15] [3].

III. Theoretical Background

A. Power Grid Modeling and Transient Analysis

We place specific emphasis in this work on transient analysis with a variable time-step, although the results are perfectly applicable in DC analysis as well. The typical model of power grid for transient analysis is obtained by modeling each wire segment (between two contacts) as a resistance in series with an inductance, with capacitances to ground at both contact nodes. Let the electrical model of the power grid be composed of b composite R-L branches and N non-supply nodes.

By using the Modified Nodal Analysis (MNA) framework in such a linear circuit, we obtain the following system of differential equations [7]:

$$\tilde{\mathbf{G}}\mathbf{x}(t) + \tilde{\mathbf{C}}\dot{\mathbf{x}}(t) = \mathbf{e}(t) \tag{1}$$

where $\tilde{\mathbf{G}} = \begin{bmatrix} \mathbf{0} & \mathbf{A}_{rl} \\ -\mathbf{A}_{rl}^T & \mathbf{R}_b \end{bmatrix}$, $\tilde{\mathbf{C}} = \begin{bmatrix} \mathbf{C}_n & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_b \end{bmatrix}$, $\mathbf{x}(t) = \begin{bmatrix} \mathbf{v}_n(t) \\ \mathbf{i}_b(t) \end{bmatrix}$, $\mathbf{e}(t) = \begin{bmatrix} \mathbf{e}_n(t) \\ \mathbf{0} \end{bmatrix}$.

In the above system, $\mathbf{A}_{rl}$ is the $N \times b$ incidence matrix of the directed composite R-L branches (with elements $a_{ij} = \pm 1$ or $a_{ij} = 0$ depending on whether branch j leaves/enters or is not incident with node i), $\mathbf{v}_n(t)$, $\mathbf{i}_b(t)$ are the $N \times 1$ and $b \times 1$ vectors of node voltages and branch currents respectively, $\mathbf{e}_n(t)$ is a $N \times 1$ vector of excitations from independent sources at the nodes, $\mathbf{C}_n$ is a $N \times N$ diagonal matrix of the node capacitances, and $\mathbf{R}_b$, $\mathbf{L}_b$ are diagonal $b \times b$ matrices of the resistances and self-inductances of the composite R-L branches. Using the Backward-Euler approximation we obtain from (1) the following discretized

system of linear algebraic equations:

$$(\tilde{\mathbf{G}} + \frac{\tilde{\mathbf{C}}}{h_k})\mathbf{x}(h_k) = \mathbf{e}(h_k) + \frac{\tilde{\mathbf{C}}}{h_k}\mathbf{x}(h_{k-1}) \quad (2)$$

where $h_k,\ k = 1, \ldots$ is the chosen time-step that may in general vary during the analysis. By block-matrix operations on the above system we obtain the following system of coupled recursive equations [7]:

$$(\mathbf{A}_{rl}(\mathbf{R}_b + \frac{\mathbf{L}_b}{h_k})^{-1}\mathbf{A}_{rl}^T + \frac{\mathbf{C}_n}{h_k})\mathbf{v}_n(h_k) =$$
$$\frac{\mathbf{C}_n}{h_k}\mathbf{v}_n(h_{k-1}) - (\mathbf{A}_{rl}(\mathbf{R}_b + \frac{\mathbf{L}_b}{h_k})^{-1}\frac{\mathbf{L}_b}{h_k}\mathbf{i}_b(h_{k-1}) + \mathbf{e}(h_k) \quad (3a)$$

$$\mathbf{i}_b(h_k) = (\mathbf{R}_b + \frac{\mathbf{L}_b}{h_k})^{-1}\frac{\mathbf{L}_b}{h_k}\mathbf{i}_b(h_{k-1}) + (\mathbf{R}_b + \frac{\mathbf{L}_b}{h_k})^{-1}\mathbf{A}_{rl}^T\mathbf{v}_n(h_k) \quad (3b)$$

At each time-step h_k we have to solve the $N \times N$ linear system (3a) with system matrix $\mathbf{A} \equiv \mathbf{A}_{rl}(\mathbf{R}_b + \frac{\mathbf{L}_b}{h_k})^{-1}\mathbf{A}_{rl}^T + \frac{\mathbf{C}_n}{h_k}$ and then find branch currents from (3b). If we neglect inductances and model the grid as an *RC* circuit, the system (3) reduces to the following recursive system:

$$(\mathbf{A}_{rl}\mathbf{R}_b^{-1}\mathbf{A}_{rl}^T + \frac{\mathbf{C}_n}{h_k})\mathbf{v}_n(h_k) = \frac{\mathbf{C}_n}{h_k}\mathbf{v}_n(h_{k-1}) + \mathbf{e}(h_k) \quad (4)$$

In both the above cases, the system matrix can be shown to be Symmetric and Positive Definite (SPD), which means that efficient direct or iterative methods such as the Cholesky factorization or the method of Conjugate Gradients can be employed for its solution.

B. Iterative Linear Solution Methods

The method of Conjugate Gradients constitutes a very attractive method for solving large SPD linear systems from a computational and memory usage perspective. It involves only inner products and matrix-vector products, while it only needs to keep 4 vectors in memory during its execution (due to a property of short recurrences). It also scales well with the dimension of the system and is a very good candidate for mapping onto parallel architectures. However, the convergence rate of CG is not predictable in its plain (unpreconditioned) form, which limits its adoption in the simulation of industrial and large-scale designs.

Regarding the convergence rate of CG, it can be shown [4] that the required number of iterations (for a given initial guess and convergence tolerance) is bounded in terms of the spectral condition number $\kappa_2(\mathbf{A}) = \|\mathbf{A}\|_2\|\mathbf{A}^{-1}\|_2 \geq 1$ - specifically, it is $\mathcal{O}(\sqrt{\kappa_2(\mathbf{A})})$, which for SPD matrices becomes $\kappa_2(\mathbf{A}) = \frac{\lambda_{max}(\mathbf{A})}{\lambda_{min}(\mathbf{A})}$ where $\lambda_{max}(\mathbf{A})$, $\lambda_{min}(\mathbf{A})$ are the maximum and minimum eigenvalues of $\mathbf{A}$ respectively. This means that convergence of CG is fast when $\kappa_2(\mathbf{A}) \simeq 1$ and slow when $\kappa_2(\mathbf{A}) \gg 1$.

Preconditioning is a technique that is used to transform the original linear system into one with more favorable spectral properties, and is essential to guarantee fast and robust convergence of CG. Algorithm 1 describes the *Preconditioned* Conjugate Gradient method for the solution of an SPD linear system $\mathbf{Ax} = \mathbf{b}$. The preconditioner solve step $\mathbf{Mz} = \mathbf{r}$ in every iteration (line 6) effectively modifies the CG algorithm to solve the system $\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}$, which has the same solution as the original one $\mathbf{Ax} = \mathbf{b}$ [4]. If $\mathbf{M}$ approximates $\mathbf{A}$ in some way, then $\mathbf{M}^{-1} \simeq \mathbf{A}^{-1}$ and $\kappa_2(\mathbf{M^{-1}A}) \simeq \kappa_2(\mathbf{I}) = 1$, which makes the CG converge quickly. So the motivation behind preconditioning is to find a matrix $\mathbf{M}$ with the following properties: 1) the convergence rate of the preconditioned system $\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}$ is fast, and 2) a linear system involving $\mathbf{M}$ (i.e. $\mathbf{Mz} = \mathbf{r}$) can be solved much more efficiently than the original system involving $\mathbf{A}$ (i.e. $\mathbf{Ax} = \mathbf{b}$).

In general, the condition number $\kappa_2(\mathbf{A})$ and the number of iterations grows as a function of the matrix dimension N. It can be very beneficial if the system matrix $\mathbf{A}$ can be approximated well enough by a preconditioner $\mathbf{M}$, so that $\kappa_2(\mathbf{M}^{-1}\mathbf{A})$ and the number of iterations become independent of the dimension (i.e. they are bounded by a constant, $\mathcal{O}(1)$), and at the same time the solution of the preconditioned systems $\mathbf{Mz} = \mathbf{r}$ (which then effectively receives the whole burden of the algorithm) can be performed in $\mathcal{O}(N)$ or slightly higher number of operations. In that case the performance of PCG will be optimal or very close to optimal. The effect of good preconditioning is even more pronounced if the operations for the solution of $\mathbf{Mz} = \mathbf{r}$ can be performed in parallel. Fortunately, as we will describe in Section IV, matrices arising from power grids can be well-approximated by preconditioners with special structure such that the number of iterations becomes bounded (or very slowly rising), while the systems $\mathbf{Mz} = \mathbf{r}$ can be solved by applying a Fast Transform in a near-optimal number of operations in a sequential implementation, and even less operations in a parallel environment (owing to the large parallel potential of Fast Transforms as well as other parallelization opportunities). The next section describes the special form of the preconditioner matrices, and the solution of the corresponding linear systems $\mathbf{Mz} = \mathbf{r}$ by a Fast Transform.

C. Fast Transform Solvers for Systems with Special Structure

Let $\mathbf{M}$ be a $N \times N$ block-tridiagonal matrix with m blocks of size $n \times n$ each (overall $N = mn$), which has the following form:

$$\mathbf{M} = \begin{bmatrix} \mathbf{T}_1 & \gamma_1\mathbf{I} & & & \\ \gamma_1\mathbf{I} & \mathbf{T}_2 & \gamma_2\mathbf{I} & & \\ & \ddots & \ddots & \ddots & \\ & & \gamma_{m-2}\mathbf{I} & \mathbf{T}_{m-1} & \gamma_{m-1}\mathbf{I} \\ & & & \gamma_{m-1}\mathbf{I} & \mathbf{T}_m \end{bmatrix} \quad (5)$$

where $\mathbf{I}$ is the $n \times n$ identity matrix and $\mathbf{T}_i$, $i = 1, \ldots, m$, are $n \times n$ tridiagonal matrices of the form:

$$\mathbf{T}_i = \begin{bmatrix} \alpha_i + \beta_i & -\alpha_i & & & \\ -\alpha_i & 2\alpha_i + \beta_i & -\alpha_i & & \\ & \ddots & \ddots & \ddots & \\ & & -\alpha_i & 2\alpha_i + \beta_i & -\alpha_i \\ & & & -\alpha_i & \alpha_i + \beta_i \end{bmatrix}$$
$$= \alpha_i \begin{bmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{bmatrix} + \beta_i\mathbf{I} \quad (6)$$

We will describe an algorithm for the solution of a linear system $\mathbf{Mz} = \mathbf{r}$ with matrix $\mathbf{M}$ of the form (5), by the use of a Fast Transform solver in $\mathcal{O}(mn \log n) = \mathcal{O}(N \log n)$ operations. Such a solution is based on the fact that the eigen-decomposition of the tridiagonal matrices $\mathbf{T}_i$ is known beforehand, and that the matrices of eigenvectors that diagonalize $\mathbf{T}_i$ are matrices that correspond to a Fast Transform. More specifically, it can be shown [9] that each $\mathbf{T}_i$ has n distinct eigenvalues $\lambda_{i,j}$, $j = 1, \ldots, n$, which are given by:

$$\lambda_{i,j} = \beta_i + 4\alpha_i \sin^2 \frac{(j-1)\pi}{2n} = \beta_i + \alpha_i(2\cos\frac{(j-1)\pi}{n} - 2) \quad (7)$$

as well as a set of n orthonormal eigenvectors $\mathbf{q}_j$, $j = 1, \ldots, n$, with elements:

$$q_{j,k} = \begin{cases} \sqrt{\frac{1}{n}}\cos\frac{(2k-1)(j-1)\pi}{2n} & j = 1, \quad k = 1, \ldots, n \\ \sqrt{\frac{2}{n}}\cos\frac{(2k-1)(j-1)\pi}{2n} & j = 2, \ldots, n, \quad k = 1, \ldots, n \end{cases} \quad (8)$$

Algorithm 1 Preconditioned Conjugate Gradients

1: $\mathbf{x}$ = initial guess $\mathbf{x}^{(0)}$
2: $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$
3: $iter = 0$
4: **repeat**
5: $\quad iter = iter + 1$
6: $\quad$ **Solve** $\mathbf{Mz} = \mathbf{r}$ $\quad$ (**Preconditioner Solve Step**)
7: $\quad \rho = \mathbf{r} \cdot \mathbf{z}$
8: $\quad$ **if** $iter == 1$ **then**
9: $\quad\quad \mathbf{p} = \mathbf{z}$
10: $\quad$ **else**
11: $\quad\quad \beta = \rho/\rho 1$
12: $\quad\quad \mathbf{p} = \mathbf{z} + \beta\mathbf{p}$
13: $\quad$ **end if**
14: $\quad \rho 1 = \rho$
15: $\quad \mathbf{q} = \mathbf{Ap}$
16: $\quad \alpha = \rho/(\mathbf{p} \cdot \mathbf{q})$
17: $\quad \mathbf{x} = \mathbf{x} + \alpha\mathbf{p}$
18: $\quad \mathbf{r} = \mathbf{r} - \alpha\mathbf{q}$
19: **until** convergence

Note that the $\mathbf{q}_j$ do not depend on the values of α_i and β_i, and are the same for every matrix $\mathbf{T}_i$. If $\mathbf{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_n]$ denotes the matrix whose columns are the eigenvectors $\mathbf{q}_j$, then due to the eigen-decomposition of $\mathbf{T}_i$ we have $\mathbf{Q}^T\mathbf{T}_i\mathbf{Q} = \mathbf{\Lambda}_i = diag(\lambda_{i,1}, \ldots, \lambda_{i,n})$. By exploiting this diagonalization of the matrices $\mathbf{T}_i$, the system $\mathbf{Mz} = \mathbf{r}$ with $\mathbf{M}$ of the form (5) is equivalent to the following system (due to $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$):

$$\begin{bmatrix} \mathbf{Q}^T & & \\ & \ddots & \\ & & \mathbf{Q}^T \end{bmatrix} \mathbf{M} \begin{bmatrix} \mathbf{Q} & & \\ & \ddots & \\ & & \mathbf{Q} \end{bmatrix} \begin{bmatrix} \mathbf{Q}^T & & \\ & \ddots & \\ & & \mathbf{Q}^T \end{bmatrix} \mathbf{z} = \begin{bmatrix} \mathbf{Q}^T & & \\ & \ddots & \\ & & \mathbf{Q}^T \end{bmatrix} \mathbf{r} \Leftrightarrow$$

$$\begin{bmatrix} \mathbf{\Lambda}_1 & \gamma_1\mathbf{I} & & & & \\ \gamma_1\mathbf{I} & \mathbf{\Lambda}_2 & \gamma_2\mathbf{I} & & & \\ & \ddots & \ddots & \ddots & & \\ & & \gamma_{m-2}\mathbf{I} & \mathbf{\Lambda}_{m-1} & \gamma_{m-1}\mathbf{I} & \\ & & & \gamma_{m-1}\mathbf{I} & \mathbf{\Lambda}_m & \end{bmatrix} \tilde{\mathbf{z}} = \tilde{\mathbf{r}} \tag{9}$$

where

$$\tilde{\mathbf{z}} = \begin{bmatrix} \mathbf{Q}^T & & \\ & \ddots & \\ & & \mathbf{Q}^T \end{bmatrix} \mathbf{z}, \quad \tilde{\mathbf{r}} = \begin{bmatrix} \mathbf{Q}^T & & \\ & \ddots & \\ & & \mathbf{Q}^T \end{bmatrix} \mathbf{r}$$

If the $N \times 1$ vectors $\mathbf{r}$, $\mathbf{z}$, $\tilde{\mathbf{r}}$, $\tilde{\mathbf{z}}$ are also partitioned into m blocks of size $n \times 1$ each, i.e.

$$\mathbf{r} = \begin{bmatrix} \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_m \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_m \end{bmatrix}, \quad \tilde{\mathbf{r}} = \begin{bmatrix} \tilde{\mathbf{r}}_1 \\ \vdots \\ \tilde{\mathbf{r}}_m \end{bmatrix}, \quad \tilde{\mathbf{z}} = \begin{bmatrix} \tilde{\mathbf{z}}_1 \\ \vdots \\ \tilde{\mathbf{z}}_m \end{bmatrix}$$

then we have: $\tilde{\mathbf{r}}_i = \mathbf{Q}^T\mathbf{r}_i$ and $\tilde{\mathbf{z}}_i = \mathbf{Q}^T\mathbf{z}_i \Leftrightarrow \mathbf{z}_i = \mathbf{Q}\tilde{\mathbf{z}}_i$, $i = 1, \ldots, m$.

However, it can be shown [24] that each product $\mathbf{Q}^T\mathbf{r}_i = \tilde{\mathbf{r}}_i$ corresponds to a Discrete Cosine Transform of type-II (DCT-II) on $\mathbf{r}_i$, and each product $\mathbf{Q}\tilde{\mathbf{z}}_i = \mathbf{z}_i$ corresponds to an Inverse Discrete Cosine Transform of type-II (IDCT-II) on $\tilde{\mathbf{z}}_i$. This means that the computation of the whole vector $\tilde{\mathbf{r}}$ from $\mathbf{r}$ amounts to m independent DCT-II transforms of size n, and the computation of the whole vector $\mathbf{z}$ from $\tilde{\mathbf{z}}$ amounts to m independent IDCT-II transforms of size n. A modification of Fast Fourier Transform (FFT) can be employed for each of the m independent DCT-II/IDCT-II transforms [24], giving a total operation count of $\mathcal{O}(mn \log n) = \mathcal{O}(N \log n)$.

If now $\mathbf{P}$ is a permutation matrix that reorders the elements of a vector or the rows of a matrix as $1, n+1, 2n+1, \ldots, (m-1)n+1,\ 2, n+2, 2n+2, \ldots, (m-1)n+2,\ \ldots,\ n, n+n, 2n+n, \ldots, (m-1)n+n$, and $\mathbf{P}^T$ is the inverse permutation matrix, then the system (9) is further equivalent to:

$$\mathbf{P} \begin{bmatrix} \mathbf{\Lambda}_1 & \gamma_1\mathbf{I} & & & & \\ \gamma_1\mathbf{I} & \mathbf{\Lambda}_2 & \gamma_2\mathbf{I} & & & \\ & \ddots & \ddots & \ddots & & \\ & & \gamma_{m-2}\mathbf{I} & \mathbf{\Lambda}_{m-1} & \gamma_{m-1}\mathbf{I} & \\ & & & \gamma_{m-1}\mathbf{I} & \mathbf{\Lambda}_m & \end{bmatrix} \mathbf{P}^\mathbf{T}\mathbf{P}\tilde{\mathbf{z}} = \mathbf{P}\tilde{\mathbf{r}} \Leftrightarrow$$

$$\begin{bmatrix} \tilde{\mathbf{T}}_1 & & & \\ & \tilde{\mathbf{T}}_2 & & \\ & & \ddots & \\ & & & \tilde{\mathbf{T}}_n \end{bmatrix} \tilde{\mathbf{z}}^P = \tilde{\mathbf{r}}^P \tag{10}$$

where

$$\tilde{\mathbf{T}}_j = \begin{bmatrix} \lambda_{1,j} & \gamma_1 & & & \\ \gamma_1 & \lambda_{2,j} & \gamma_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \gamma_{m-2} & \lambda_{m-1,j} & \gamma_{m-1} \\ & & & \gamma_{m-1} & \lambda_{m,j} \end{bmatrix} \tag{11}$$

and $\tilde{\mathbf{z}}^P = \mathbf{P}\tilde{\mathbf{z}}$, $\tilde{\mathbf{r}}^P = \mathbf{P}\tilde{\mathbf{r}}$. If the $N \times 1$ vectors $\tilde{\mathbf{z}}^P$, $\tilde{\mathbf{r}}^P$ are partitioned into n blocks $\tilde{\mathbf{z}}_j^P, \tilde{\mathbf{r}}_j^P$ of size $m \times 1$ each, then the system (10) effectively represents n independent tridiagonal systems $\mathbf{T}_j\tilde{\mathbf{z}}_j^P = \tilde{\mathbf{r}}_j^P$ of size m which can be solved w.r.t. the blocks $\tilde{\mathbf{z}}_j^P$, $j = 1, \ldots, n$ (to produce the whole vector $\tilde{\mathbf{z}}^P$) in a total of $\mathcal{O}(mn) = \mathcal{O}(N)$ operations. For each such system the coefficient matrix (11) is known beforehand and is determined exclusively by the eigenvalues (7) and the values γ_i of matrix $\mathbf{M}$, while the right-hand side (RHS) vector $\tilde{\mathbf{r}}_j^P$ is composed of specific components of (DCT-II)-transformed blocks of vector $\mathbf{r}$. The equivalence of the system $\mathbf{Mz} = \mathbf{r}$, with $\mathbf{M}$ as in (5), to the system (10) gives a procedure for fast solution of $\mathbf{Mz} = \mathbf{r}$ which is described in Algorithm 2. Note that apart from the resultant near-optimal complexity of $\mathcal{O}(N \log n)$ operations, the m DCT-II/IDCT-II transforms and the n tridiagonal systems are completely independent to each other and can be solved in parallel. Furthermore, both the FFT and the tridiagonal solution algorithm are highly-parallel algorithms by themselves, allowing for further acceleration of the individual transforms/solutions when executed on parallel platforms. These issues are further examined in Section IV-B, in the context of our proposed approach for power grid analysis.

IV. Proposed Methodology For Power Grid Analysis

A. Preconditioner Construction and Storage

As mentioned in Section III-B, the intuition behind preconditioner's formulation is to create a matrix $\mathbf{M}$ that will approximate the system matrix $\mathbf{A}$ as faithfully as possible, while at the same time enable the utilization of efficient algorithms for the solution of systems $\mathbf{Mz} = \mathbf{r}$. We have developed such an algorithm based on a Fast Transform solver in the previous section for a class of matrices with special structure. This section will describe the construction of a preconditioner with such structure from a given power grid by exploiting its spatial geometry.

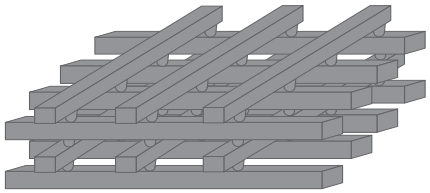
Practical power grids are created as orthogonal wire meshes with very regular spatial geometries, with possibly some irregularities imposed by design constraints (e.g. some missing connections between adjacent nodes), and arranged in a few - typically 2 to 6 - metal layers of alternating routing directions (horizontal and vertical). Due to the presence of vias between successive metal layers, the actual grid has the structure of a 3D mesh, with very few planes along the third dimension. However, as it was observed in [13], electrical resistances of vias are usually much smaller than wire resistances, leading to voltage drops much less than 1mV. Also, data in [22] show that almost all circuit elements (mainly resistances) in each metal layer have the same values (with few differences due to grid irregularities).

Based on these observations, we create a preconditioner matrix that approximates the system matrix of the power grid by a process of regularization of the 3D power grid to a regular 2D grid, consisting of the following steps:

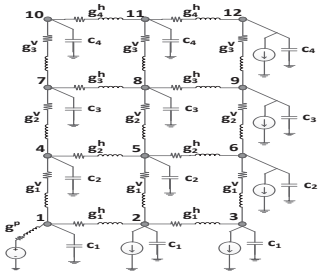
1) Determine the distinct x- and y-coordinates of all nodes in the different layers of the 3D grid, and take their Cartesian product to specify the location of the nodes in the regular 2D grid.
2) By disregarding via resistances between layers, collapse the 3D grid onto the regular 2D grid by adding together all horizontal branch conductances $g^h \equiv \frac{1}{r^h + \frac{l^h}{h_k}}$ connected in parallel between adjacent nodes in the x-direction of the 2D grid, and all vertical branch conductances $g^v \equiv \frac{1}{r^v + \frac{l^v}{h_k}}$ connected in parallel between adjacent nodes in the y-direction of the

Algorithm 2 Fast Transform algorithm for the preconditioner solve step $\mathbf{Mz} = \mathbf{r}$

1: Partition the RHS vector $\mathbf{r}$ into m blocks $\mathbf{r}_i$ of size n, and perform DCT-II transform ($\mathbf{Q}^T\mathbf{r}_i$) on each block to obtain transformed vector $\tilde{\mathbf{r}}$
2: Permute vector $\tilde{\mathbf{r}}$ by permutation $\mathbf{P}$, which orders elements as $1, n+1, \ldots, (m-1)n+1,\ 2, n+2, \ldots, (m-1)n+2,\ \ldots,\ n, n+n, \ldots, (m-1)n+n$, in order to obtain vector $\tilde{\mathbf{r}}^P$
3: Solve the n tridiagonal systems (10) with known coefficient matrices (11), in order to obtain vector $\tilde{\mathbf{z}}^P$.
4: Apply inverse permutation $\mathbf{P}^T$ on vector $\tilde{\mathbf{z}}^P$ so as to obtain vector $\tilde{\mathbf{z}}$.
5: Partition vector $\tilde{\mathbf{z}}$ into m blocks $\tilde{\mathbf{z}}_i$ of size n, and perform IDCT-II transform ($\mathbf{Q}\tilde{\mathbf{z}}_i$) on each block to obtain final solution vector $\mathbf{z}$



(a) Geometry structure of a 3D power grid with 4 layers. Vias represent connections between adjacent metal layers.



(b) Regular 2D grid obtained after the regularization process.

Fig. 1. Example of a power delivery network with 4 horizontal and 3 vertical rails, along with the regular 2D grid used for preconditioning. The figure depicts only the VDD rails.

2D grid (where r^h, l^h denote the resistance and inductance of horizontal branches, and r^v, l^v denote the resistance and inductance of vertical branches - the inductances might not be present in the model). If a conductance of the 3D grid occupies multiple nodes of the regular 2D grid, it is decomposed into a corresponding number of pieces. The node capacitances corresponding to the same regular grid nodes are also added together during the collapsing.

3) In the regular 2D grid, substitute horizontal branch conductances by their average value in each horizontal rail, and vertical branch conductances by their average value in each horizontal slice (enclosed between two adjacent horizontal rails). Substitute node capacitances in each horizontal rail by their average value as well.

Fig. 1(a) depicts a 3D 4-layer power delivery network with $m = 4$ horizontal rails and $n = 3$ vertical rails in likewise-routed layers. Fig 1(b) shows the 2D regular grid that results from the previous regularization process used to construct the preconditioner matrix. If we use the depicted natural node numbering (proceeding horizontally, since this is always the routing direction of the lowest-level metal layer), the matrix $\mathbf{A}_{rl}(\mathbf{R}_b + \frac{\mathbf{L}_b}{h_k})^{-1}\mathbf{A}_{rl}^T + \frac{\mathbf{C}_n}{h_k}$ that corresponds to the regular 2D grid will be the following block-tridiagonal matrix:

$$\begin{bmatrix} \mathbf{T}_1 & -g_1^v\mathbf{I} & & \\ -g_1^v\mathbf{I} & \mathbf{T}_2 & -g_2^v\mathbf{I} & \\ & -g_2^v\mathbf{I} & \mathbf{T}_3 & -g_3^v\mathbf{I} \\ & & -g_3^v\mathbf{I} & \mathbf{T}_4 \end{bmatrix}$$

where $\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3, \mathbf{T}_4$ are 3×3 tridiagonal matrices (each one corresponding to a horizontal rail of the 2D grid) which have the form:

$$\mathbf{T}_1 = \begin{bmatrix} g_1^h+g_1^v+g^p+\frac{c_1}{h_k} & -g_1^h & \\ -g_1^h & 2g_1^h+g_1^v+\frac{c_1}{h_k} & -g_1^h \\ & -g_1^h & g_1^h+g_1^v+\frac{c_1}{h_k} \end{bmatrix}$$

$$\mathbf{T}_2 = \begin{bmatrix} g_2^h+g_1^v+g_2^v+\frac{c_2}{h_k} & -g_2^h & \\ -g_2^h & 2g_2^h+g_1^v+g_2^v+\frac{c_2}{h_k} & -g_2^h \\ & -g_2^h & g_2^h+g_1^v+g_2^v+\frac{c_2}{h_k} \end{bmatrix}$$

$$\mathbf{T}_3 = \begin{bmatrix} g_3^h+g_2^v+g_3^v+\frac{c_3}{h_k} & -g_3^h & \\ -g_3^h & 2g_3^h+g_2^v+g_3^v+\frac{c_3}{h_k} & -g_3^h \\ & -g_3^h & g_3^h+g_2^v+g_3^v+\frac{c_3}{h_k} \end{bmatrix}$$

$$\mathbf{T}_4 = \begin{bmatrix} g_4^h+g_3^v+\frac{c_4}{h_k} & -g_4^h & \\ -g_4^h & 2g_4^h+g_3^v+\frac{c_4}{h_k} & -g_4^h \\ & -g_4^h & g_4^h+g_3^v+\frac{c_4}{h_k} \end{bmatrix}$$

In the above, g_i^h is the average horizontal conductance in the i-th horizontal rail, g_i^v is the average vertical conductance in the i-th horizontal slice, and c_i is the average node capacitance in the i-th horizontal rail. Also h_k is the current analysis time-step (possibly variable), and $g^p \equiv \frac{1}{r^p+\frac{l^p}{h_k}}$ is the parasitic conductance of the supply pads (r^p and l^p denote the resistance and inductance of the supply pads respectively).

We observe that the form of the above matrix is almost identical to (5), with the exception of the pad parasitic conductance g^p in few places along the diagonal (considering that the number of voltage pads is much smaller than the number of nodes N). In order to obtain a preconditioner $\mathbf{M}$ with an exact form that can be efficiently solved by the application of a Fast Transform, we can just omit entirely those pad parasitics. However, we have found that in practice it is usually better to amortize the total sum of pad conductances of a specific horizontal rail (in the regular 2D grid) to all nodes of this rail, i.e. assume that all nodes of the i-th horizontal rail have pad conductance $\bar{g}_i^p = \frac{(\sum g^p)_i}{n}$, where $(\sum g^p)_i$ is the sum of the actual pad conductances attached to nodes of the i-th horizontal rail. This also has the beneficial effect of making the preconditioner $\mathbf{M}$ non-singular in the case of DC analysis (where capacitances and inductances are absent). In the above example, the block $\mathbf{T}_1$ would become:

$$\mathbf{T}_1 = \begin{bmatrix} g_1^h+g_1^v+\bar{g}_1^p+\frac{c_1}{h_k} & -g_1^h & \\ -g_1^h & 2g_1^h+g_1^v+\bar{g}_1^p+\frac{c_1}{h_k} & -g_1^h \\ & -g_1^h & g_1^h+g_1^v+\bar{g}_1^p+\frac{c_1}{h_k} \end{bmatrix}$$

where $\bar{g}_1^p = \frac{g^p}{3}$. It is not difficult to generalize the procedure to an arbitrary $m \times n$ power grid. In that case, the preconditioner will comprise m blocks of size $n \times n$ and have the form (5), where $\alpha_i = g_i^h,\ \beta_i = g_i^v + g_{i-1}^v + \bar{g}_i^p + \frac{c_i}{h_k},\ \gamma_i = -g_i^v,\quad i = 1, \ldots, m$ (with $g_0^v = g_m^v = 0$).

The preconditioner construction requires only one parsing of the netlist of the electrical circuit representing the power grid, and is of complexity $\mathcal{O}(N)$ (considering that the number of electrical elements is of the same order as the number of nodes N), which is very inexpensive since it represents a one-time cost, roughly comparable to one iteration (and amortized over multiple iterations) of the CG method. During transient analysis with variable time-step (which is almost always used in practical simulation scenarios - and completely rules out direct methods), the construction has to be repeated at every change of time-step in the same $\mathcal{O}(N)$ operations (this is also necessary for all other known preconditioners, and is in fact very expensive for some of them e.g. multigrid preconditioners). However, a considerable simplification is possible in the very common case of resistive or RC-only electrical models (i.e. when inductances are absent from the model) since the change of time-step does not affect any actions in the construction procedure, and thus there is no need for a full reconstruction (but only, actually, an update in the eigenvalues (7) of the preconditioner matrix).

Apart from the near-optimal complexity of solving the systems $\mathbf{Mz} = \mathbf{r}$, one other salient feature of the proposed preconditioner is that there is no need for explicit storage of the preconditioner matrix $\mathbf{M}$. As it is easily observed, only the eigenvalues (7) and the values $\gamma_i = -g_i^v$ of $\mathbf{M}$ are necessary in the execution of Algorithm 2 (for formulation of the tridiagonal systems (10) with coefficient matrices (11)), and thus only storage for those $mn + (m-1)$ values needs to be allocated. A small memory footprint is very important for mapping the algorithm onto architectures with limited available memory space such as GPUs.

B. Procedure Implementation and Opportunities for Parallelism

After the preconditioner construction and storage, the whole procedure involves execution of Algorithm 1 with Algorithm 2 in place of

the preconditioner solve step $\mathbf{Mz} = \mathbf{r}$. Every part of this procedure offers ample multi-grain parallelism, both data- and task-level, thus enabling highly parallel computing efficiency. This comes in contrast with most standard preconditioning methods, such as incomplete factorizations, which have limited parallelism, either data-level or task-level. The details for each major part of the procedure, as well as the opportunities for introducing parallelism are presented below.

1) Main CG Algorithm: As seen in Algorithm 1, apart from the preconditioner solve step, the PCG method involves 2 inner products and 1 sparse matrix-vector product per iteration, which can be implemented efficiently by available BLAS-1 and BLAS-2 (Basic Linear Algebra Subroutines) kernels. The algorithm also has 3 scalar-vector products with vector updates per iteration which can be fully parallelized.

2) DCT-II & IDCT-II Transforms: As already mentioned in Section III-C, in order to apply the m independent DCT-II and IDCT-II transforms of size n (steps 1 and 5 of Algorithm 2) we can use a modification of the one-dimensional Fast Fourier Transform (FFT) algorithm [24], which gives a near-optimal sequential complexity of $\mathcal{O}(n \log n)$ operations for each of these $2m$ transforms. FFT is also a highly parallel algorithm and an ideal candidate for mapping onto a multi-core processor or a GPU, with a parallel complexity of $\mathcal{O}((n \log n)/p)$, where p is the number of available processors [11]. This parallelization gain is especially evident on GPUs which offer a large amount of processing cores and can greatly reduce the cost of applying the one-dimensional FFT.

3) Solution of Tridiagonal Systems: The solution of tridiagonal systems (step 3 in Algorithm 2) offers abundant data-level parallelism as well, and various algorithms have been proposed in the literature for its implementation on parallel architectures. These can be classified to algorithms that target coarse-grain parallelism (and are appropriate for multi-core processors) such as two-way Gaussian elimination or Bondelli's algorithm [19], and to algorithms that exploit fine-grain parallelism (and are appropriate for GPUs) such as Parallel Cyclic Reduction [26]. The latter can offer the greatest speed-up, as was naturally expected.

4) Task-Level Parallelism: Algorithm 2 entails m DCT-II transforms (of size n), n tridiagonal systems (of size m), and m IDCT-II transforms (of size n), all of which are totally independent and thus each one can be solved separately from the others. This translates to additional task-level parallelism, which can lead to further acceleration of the whole preconditioner solve step in *multi-GPU* systems, where all independent transforms and tridiagonal solvers can be executed in parallel without requiring any data communication between different GPUs.

V. Experimental Results

A. Experimental Setup

We evaluated the performance of our methodology for power grid analysis by comparing four methods for solving the linear system (4): the PCG method with zero-fill Incomplete Cholesky preconditioner (ICCG), the proposed method of using PCG with the Fast Transform preconditioner (FTCG) both implemented on a multi-core architecture (*FTCG-CPU*) and a GPU (*FTCG-GPU*), and CHOLMOD [8] which is a state-of-the-art CPU-based direct solver for sparse SPD linear systems. For the GPU implementation, we have ported the entire CG iterative method (Algorithm 1) and the preconditioner solve step (Algorithm 2) on the GPU. This eliminates the need for additional memory transfers between the host and the GPU and reduces the communication overhead, provided that the GPU has sufficient memory to accommodate the algorithm's working set (especially the system matrix $\mathbf{A}$ in sparse form). The only part of our algorithm that is implemented on the CPU for the GPU implementation is the preconditioner's initial construction and reconstruction procedure (during a time-step change). This procedure effectively results in the computation of the eigenvalues (7) and the values $\gamma_i = -g_i^v$ of the preconditioner matrix $\mathbf{M}$, which is the only information needed to store the preconditioner. Afterwards, the CPU is responsible for transferring these values to the main memory of the GPU (to update the elements of the preconditioner).

We have used Intel Math Kernel Library (MKL) [1] for implementing the CPU versions of the ICCG and FTCG-CPU algorithms, and CUDA library [2] (version 4.1, along with CUBLAS, CUSPARSE and CUFFT libraries) for mapping the FTCG-GPU algorithm on the GPU. The implementations of FTCG-CPU and FTCG-GPU involve the whole PCG method described in Algorithm 1, with Algorithm 2 in place of the preconditioner solve step of line 6. We note that both MKL and CUDA libraries contain implementations of BLAS-1 and BLAS-2 kernels, FFT routines and tridiagonal solvers, all especially optimized for execution on multi-core and GPU architectures respectively. We executed all experiments on a Linux workstation, comprising an Intel Core 2 Quad Q9300 processor running at 2.5GHz, with 8GB main memory and an NVIDIA GeForce GTX 560 Ti GPU with 1GB of main memory. We used Intel and NVIDIA compilers for compiling our source code for the CPU and the GPU respectively, with the optimizations flags that resulted to the lowest execution time.

We have employed a set of industrial power grid designs [17] and a set of synthetic benchmarks, ranging from simple to more complicated designs, for the experimental validation of the proposed approach. Table I presents the details of each benchmark. The industrial (IBM) benchmarks are typical representatives of quite irregular designs from custom microprocessors, while the synthetic benchmarks are representatives of regular power delivery networks that are produced from most industrial automated routing tools. The IBM benchmarks were resistive-only (an extensive set of full-RLC benchmarks was unavailable at the time of writing) and were simulated in transient mode over multiple time-steps, while the synthetic benchmarks were full-RLC and had typical layer parameters (e.g. average resistance) taken from the industrial benchmarks and real designs.

B. Transient Analysis Results

To compare the four different methods for power grid analysis, we have conducted transient simulation with variable time-step over a total of 100 time-steps. The simulation results for the power grid benchmarks are presented in Table II. Execution time ($Time$) refers to the average time required for solution at each time-step, including any overhead for matrix re-factorization (in CHOLMOD) and preconditioner reconstruction (in iterative methods) whenever the time-step changes. Due to limited memory on the GPU, benchmarks *ckt5* and *ckt6* were executed using single-precision arithmetic, while double-precision arithmetic was used for the rest of the benchmarks. The iterative solvers were terminated when the solution residual was below 10^{-6}. This threshold is typically sufficient for ensuring a maximum error less than $1mV$ (and effectively results in perfect accuracy that is indistinguishable from direct methods).

TABLE I
Circuit details for the set of power grid benchmarks. N is the total number of nodes, N_r is the number of resistors, N_i is the number of current sources, N_v is the number of voltage sources (without any short-circuits), and N_l is the number of metal layers.

Benchmark	N	N_r	N_i	N_v	N_l
ibmpg2	127K	208K	37K	80	5
ibmpg3	851K	1.40M	201K	494	5
ibmpg4	953K	1.56M	276.9K	312	6
ibmpg5	1.07M	1.07M	540.8K	100	3
ibmpg6	1.67M	1.64M	761.4K	132	3
ibmpgnew1	1.46M	1.42M	357.9K	494	N/A
ibmpgnew2	1.46M	2.35M	357.9K	494	N/A
ibmX400	2.62M	2.62M	106.6K	200	N/A
ckt1	525K	1.04M	131K	160	2
ckt2	1.04M	2.09M	349K	180	2
ckt3	2.09M	4.19M	568K	190	3
ckt4	3.14M	6.28M	834K	200	4
ckt5	4.19M	8.38M	978K	215	4
ckt6	6.29M	12.57M	1.2M	250	5

TABLE II

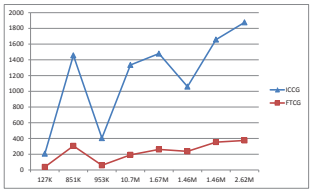
RUNTIME RESULTS FOR THE FOUR SOLVERS. $Iter.$ IS THE AVERAGE NUMBER OF ITERATIONS REQUIRED FOR CONVERGENCE OF EACH ITERATIVE METHOD OVER ALL SIMULATION TIME-STEPS. $Time$ DENOTES THE AVERAGE TIME REQUIRED FOR THE SOLUTION AT EACH TIME-STEP. Spd_{CHOL} AND Spd_{ICCG} DENOTE THE SPEEDUP OF FTCG-CPU AND FTCG-GPU OVER CHOLMOD AND ICCG RESPECTIVELY. THE CONVERGENCE TOLERANCE FOR ITERATIVE SOLVERS WAS 10^{-6} AND CONVERGENCE WAS ACHIEVED IN ALL CASES.

*BENCHMARKS CKT5 AND CKT6 WERE SIMULATED IN SINGLE PRECISION DUE TO MEMORY LIMITATIONS FOR STORING THE MAIN SYSTEM MATRIX ON THE GPU.

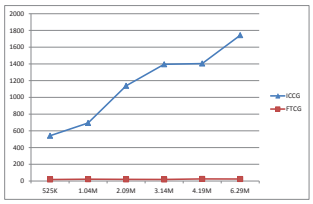
Benchmark	CHOLMOD	ICCG		FTCG-CPU				FTCG-GPU			
	$Time\ (s)$	$Iter.$	$Time\ (s)$	$Iter.$	$Time\ (s)$	Spd_{CHOL}	Spd_{ICCG}	$Iter.$	$Time\ (s)$	Spd_{CHOL}	Spd_{ICCG}
ibmpg2	4.9	207	1.735	38	0.33	14.9X	5.32X	38	0.07	69.8X	24.8X
ibmpg3	130.2	1457	83.578	306	6.92	18.8X	12.1X	306	1.15	113.1X	72.7X
ibmpg4	221.9	405	21.357	59	2.85	77.8X	7.5X	59	0.45	493.1X	47.7X
ibmpg5	110.9	1334	82.358	192	8.67	12.7X	9.5X	192	1.17	94.6X	70.4X
ibmpg6	235.1	1479	143.557	261	11.96	19.6X	12.0X	261	1.70	137.5X	84.1X
ibmpgnew1	353.0	1059	82.435	237	7.89	44.7X	10.4X	237	1.07	330.7X	77.4X
ibmpgnew2	339.9	1657	172.38	354	15.39	22.0X	11.2X	354	2.06	164.8X	83.7X
ibmX400	553.3	1876	357.82	375	20.65	26.7X	17.3X	375	2.58	214.3X	138.7X
ckt1	67.7	541	26.7	18	1.80	37.6X	14.8X	18	0.16	423.1X	166.8X
ckt2	147.6	694	66.8	22	4.51	32.7X	14.8X	22	0.33	440.5X	199.4X
ckt3	696.4	1138	221.8	20	8.54	81.5X	25.9X	20	0.63	1105.1X	352.1X
ckt4	1530	1395	416.1	19	12.83	119.2X	32.4X	19	0.95	1610.5X	438.0X
ckt5*	1131.2	1404	324	25	14.1	80.2X	23.0X	25	0.90	1256.8X	360.0X
ckt6*	N/A	1743	607	24	22.1	N/A	27.4X	24	1.40	N/A	433.2X

As we can observe, CHOLMOD (the direct solver) suffers from a super-linear increase in execution time as the size of the power grid increases. In fact, the analysis of the largest design was infeasible due to excessive memory requirements. On the other hand, both ICCG and PCG using our Fast Transform-based preconditioner (FTCG) achieved lower execution times when implemented on a multi-core CPU (where CHOLMOD was also executed). In particular, the multi-core implementation of the FTCG method (FTCG-CPU) showed a speed-up ranging from $14.9X$ to $119.2X$ in comparison to CHOLMOD.

Restricting now the comparison to the iterative methods, we observe a significant acceleration of the convergence rate (or reduction in the number of iterations) of PCG when our Fast Transform preconditioner is applied, as is apparent from Fig. 2. In particular, the proposed preconditioner was able to reduce the number of iterations by a factor ranging from $4.4X$ to $6.8X$ compared to the incomplete Cholesky preconditioner for the industrial benchmarks. This is a testament to the capability of the proposed preconditioner to provide an extremely good approximation of matrices of actual power grids. In addition, the number of iterations appears fairly constant with the increase of the problem size, while it exhibits a linear increase for the incomplete Cholesky preconditioner (which is a well-known theoretical fact [4]). In fact, for very regular problems (like the synthetic benchmarks) the constant number of iterations of Fast Transform preconditioners can be proven theoretically [6] [10]. The reduced iteration count is the major factor for the speed-up of the multi-core implementation of FTCG w.r.t. ICCG, which ranges from $5.32X$ to $17.3X$ for the IBM benchmarks (work [27] reports a similar number of ICCG iterations for the IBM benchmarks) and from $14.8X$ to $32.4X$ for the synthetic benchmarks. Additional acceleration is gained when GPUs are utilized. GPUs can take advantage of the inherent parallelism of FFT and the tridiagonal solution algorithm by employing their vast amount of computational resources. Our algorithm achieves an average speedup of $74.9X$ and $324X$ over ICCG for the industrial and synthetic benchmarks respectively. The acceleration becomes even more pronounced as the size of the power grid increases. The maximum speedup over ICCG is equal to $138.7X$ for a 2.6M-node industrial power grid, and $433.2X$ for a 6.2M-node synthetic power grid. The runtime performance of our algorithm compared to the ICCG solver is illustrated in Fig. 3. Although ICCG has not been implemented on a GPU, the maximum achievable speed-up is about $4X$ [16], which still renders our algorithm considerably more efficient. The GPU implementation of FTCG is even more efficient compared to the direct solver. Referring back to Table II, we can observe that FTCG-GPU is able to achieve orders of magnitude acceleration in the simulation time, with a speed-up ranging from $69.8X$ to $1610.5X$ over CHOLMOD. Further speedup is achievable



(a) Convergence rate for the industrial benchmarks.



(b) Convergence rate for the synthetic benchmarks.

Fig. 2. Convergence rate of PCG using Incomplete Cholesky preconditioner (ICCG) and the proposed Fast Transform preconditioner (FTCG).

by exploiting task-level parallelism on multi-GPU systems (as remarked in Section IV-B) even though it was not implemented in this work.

It is noted that the GPU execution time includes the communication overhead for transferring the updated eigenvalues (7) and the values γ_i of the preconditioner matrix after every reconstruction from the host to the GPU. However, the bandwidth of the PCI-Express bus was able to effectively hide this additional overhead. As a result, the time required for these data transfers was smaller than 1% of the execution time for each time-step. Moreover, the reconstruction step can be executed asynchronously with the execution of the algorithm on the GPU and practically eliminate this communication overhead.

As pointed in Section IV-A, the proposed Fast Transform preconditioner also eliminates the need for explicit storage of the preconditioner matrix $\mathbf{M}$, since only the eigenvalues (7) and the values γ_i need to be kept in memory. This matrix-less formulation of the preconditioner contributes significantly to the reduction of the memory requirements of our proposed approach. Table III presents the cumulative memory amount required by each solver for the analysis of three of the largest benchmarks. We can observe that there is a dramatic increase in memory usage for the CHOLMOD algorithm with the increase of the grid size. For the largest design, CHOLMOD required more than $10GB$ of main memory, thus making its execution impractical on our target platform. On the other hand, our algorithm

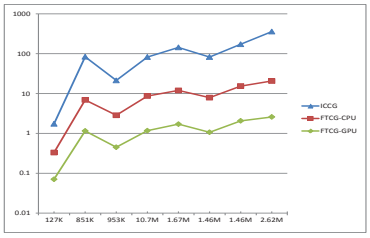
TABLE III
MEMORY REQUIREMENTS (MB) FOR THE LARGEST BENCHMARKS (IN SINGLE-PRECISION ARITHMETIC) FOR THE METHODS UNDER COMPARISON. $M_{FTCG-GPU}$ IS THE CUMULATIVE MEMORY ON THE GPU AND THE CPU REQUIRED BY THE FTCG-GPU IMPLEMENTATION.

Benchmark	M_{CHOL}	M_{ICCG}	$M_{FTCG-CPU}$	$M_{FTCG-GPU}$
ibmX400	3350	311	239	239
ckt5	7700	615	471	471
ckt6	N/A	919	685	685

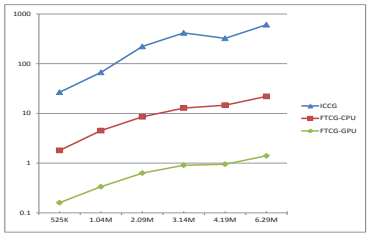
provides scalable memory performance. The total amount of required memory increased linearly with the increase in the number of power grid nodes, while the algorithm consumed less memory by an average factor of 10.7X and 1.33X than CHOLMOD and ICCG respectively. The lack of additional storage for the preconditioner matrix is the reason for the reduced memory w.r.t. ICCG. Even for our largest benchmark, the algorithm consumed less than $690MB$ of memory, which renders feasible the analysis of large-scale power grids on GPUs (which are characterized by the limited amount of main memory). In addition, limited memory consumption can have beneficial effects on the actual execution of our algorithm when it is mapped onto a GPU. In that case, the available memory can accommodate the algorithm's working set, thus eliminating the need for additional data transfers between the GPU and the host.

VI. CONCLUSIONS

We have presented a new parallel, preconditioned iterative method for analysis of large-scale power delivery networks found in modern ICs, with computational and memory requirements that scale linearly with the size of the power grid. The construction of the preconditioner takes into account the various topology characteristics of the power grid, thus accelerating convergence rate. Moreover, its structure, based on the application of a Fast Transform, allows for harnessing the potential of GPUs for parallel processing. Evaluating our methodology on a set of benchmarks showed that it can achieve a speedup ranging from $69.8X$ to $1610.5X$ over CHOLMOD and from $24.8X$ up to $438X$ over a preconditioned iterative method with incomplete factorization preconditioner when GPUs are utilized. At the same time, the proposed preconditioner reduces memory requirements by a factor of up to 10.7X and 1.33X over the aforementioned analysis methods.



(a) Execution time for the industrial benchmarks.



(b) Execution time for the synthetic benchmarks.

Fig. 3. Execution time of PCG using Incomplete Cholesky (ICCG) and the proposed Fast Transform preconditioner implemented on multi-core (FTCG-CPU) and GPU architectures (FTCG-GPU). The vertical axis is in logarithmic scale.

REFERENCES

[1] *Intel Math Kernel Library*. [Online]. Available: http://software.intel.com/en-us/articles/intel-mkl/

[2] *NVIDIA CUDA Programming Guide, CUSPARSE, CUBLAS, and CUFFT Library User Guides*. [Online]. Available: http://developer.nvidia.com/nvidia-gpu-computing-documentation

[3] M. Ament, G. Knittel, D. Weiskopf, and W. Strasser, "A Parallel Preconditioned Conjugate Gradient Solver for the Poisson Problem on a Multi-GPU Platform," in *Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010.

[4] O. Axelsson and A. Barker, *Finite Element Solution of Boundary Value Problems. Theory and Computation*. Academic Press, 1984.

[5] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*. Wiley, 2008.

[6] R. H. Chan and C. K. Wong, "Sine Transform Based Preconditioners for Elliptic Problems," *Numerical Linear Algebra with Applications*, vol. 4, no. 5, pp. 351–368, 1997.

[7] T.-H. Chen and C. C.-P. Chen, "Efficient Large-Scale Power Grid Analysis Based on Preconditioned Krylov-Subspace Iterative Methods," in *ACM/IEEE Design Automation Conf.*, 2001.

[8] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, "Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate," *ACM Trans. Math. Softw.*, vol. 35, no. 3, pp. 22:1–22:14, 2008.

[9] C. C. Christara, "Quadratic Spline Collocation Methods for Elliptic Partial Differential Equations," *BIT Numerical Mathematics*, vol. 34, no. 1, pp. 33–61, 1994.

[10] C. C. Christara and K. S. Ng, "Fast Fourier Transform Solvers and Preconditioners for Quadratic Spline Collocation," *BIT Numerical Mathematics*, vol. 42, no. 4, pp. 702–739, 2002.

[11] Z. Cui-xiang, H. Guo-qiang, and H. Ming-he, "Some New Parallel Fast Fourier Transform Algorithms," in *Int. Conf. on Parallel and Distributed Computing, Applications and Technologies*, 2005.

[12] Z. Feng and Z. Zeng, "Parallel Multigrid Preconditioning on Graphics Processing Units (GPUs) for Robust Power Grid Analysis," in *ACM/IEEE Design Automation Conf.*, 2010.

[13] Z. Feng, Z. Zeng, and P. Li, "Parallel On-Chip Power Distribution Network Analysis on Multi-Core-Multi-GPU Platforms," *IEEE Trans. VLSI Syst.*, vol. 19, no. 10, pp. 1823–1836, 2011.

[14] J. Kozhaya, S. Nassif, and F. Najm, "A Multigrid-Like Technique for Power Grid Analysis," *IEEE Trans. Computer-Aided Design*, vol. 21, no. 10, pp. 1148–1160, 2002.

[15] M. Krotkiewski and M. Dabrowski, *Efficient Solution of Poisson's Equation on Modern GPUs Using Structured Grids*. [Online]. Available: http://www.notur.no/notur2011/material/krotkiewski-notur2011.pdf

[16] R. Li and Y. Saad, "GPU-Accelerated Preconditioned Iterative Linear Solvers," Minnesota Supercomputer Institute, University of Minnesota, Tech. Rep., 2010.

[17] S. Nassif, "Power Grid Analysis Benchmarks," in *Asia and South Pacific Design Automation Conf.*, 2008.

[18] H. Qian and S. Sapatnekar, "Fast Poisson Solvers for Thermal Analysis," in *IEEE/ACM Int. Conf. Computer-Aided Design*, 2010.

[19] P. Quesada-Barriuso, J. Lamas-Rodríguez, D. B. Heras, M. Bóo1, and F. Argüello, "Selecting the Best Tridiagonal System Solver Projected on Multi-Core CPU and GPU Platforms," in *Int. Conf. on Parallel and Distributed Processing Techniques and Applications (as part of WorldComp2011 Conference)*, 2011.

[20] Y. Saad, *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[21] J. Shi, Y. Cai, W. Hou, L. Ma, S. X.-D. Tan, P.-H. Ho, and X. Wang, "GPU friendly Fast Poisson Solver for Structured Power Grid Network Analysis," in *ACM/IEEE Design Automation Conf.*, 2009.

[22] J. Shi, Y. Cai, S. X.-D. Tan, J. Fan, and X. Hong, "Pattern-Based Iterative Method for Extreme Large Power/Ground Analysis," *IEEE Trans. Computer-Aided Design*, vol. 26, no. 4, pp. 680–692, 2007.

[23] K. Sun, Q. Zhou, K. Mohanram, and D. C. Sorensen, "Parallel Domain Decomposition for Simulation of Large-Scale Power Grids," in *ACM/IEEE Design Automation Conf.*, 2007.

[24] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*. SIAM, 1992.

[25] J. Yang, Y. Cai, Q. Zhou, and J. Shi, "Fast Poisson Solver Preconditioned Method for Robust Power Grid Analysis," in *IEEE/ACM Int. Conf. on Computer-Aided Design*, 2011.

[26] Y. Zhang, J. Cohen, and J. D. Owens, "Fast Tridiagonal Solvers on the GPU," in *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2010.

[27] X. Zhao, J. Wang, Z. Feng, and S. Hu, "Power Grid Analysis with Hierarchical Support Graphs," in *IEEE/ACM Int. Conf. on Computer-Aided Design*, 2011.

[28] C. Zhuo, J. Hu, M. Zhao, and K. Chen, "Power Grid Analysis and Optimization Using Algebraic Multigrid," *IEEE Trans. Computer-Aided Design*, vol. 27, no. 4, pp. 738–751, 2008.