A Combinatorial Multigrid Preconditioned Iterative Method for Large Scale Circuit Simulation on GPUs

Dimitrios Garyfallou, Nestor Evmorfopoulos and Georgios Stamoulis
Department of Electrical and Computer Engineering, University of Thessaly, Volos, Greece
Email: {digaryfa, nestevmo, georges}@e-ce.uth.gr

***Abstract*—Efficient large scale circuit simulation is among the most challenging problems facing the EDA industry today, since it is the only feasible way to verify a circuit's behaviour prior to manufacturing. In recent years, the emphasis has been placed on preconditioning methods which reduce the number of iterations for solving large Symmetric Diagonally Dominant systems resulting after the Modified Nodal Analysis. This paper presents a GPU-accelerated Preconditioned Conjugate Gradient (PCG) iterative method preconditioned by the Combinatorial Multigrid (CMG) for fast DC and transient analysis of large-scale linear circuits. Experimental results on IBM industrial power grids demonstrate speedups up to 4.69x and 4.50x for the PCG method and the CMG preconditioning algorithm, respectively, over the optimized CPU implementations.**

I. Introduction

Circuit simulation is indispensable for the design and verification of a broad range of large-scale integrated circuits (ICs) and electrical models such as power distribution networks, clock networks, or multiconductor buses. The most challenging problem is the simulation of the on-chip power delivery network as it is truly gigantic and constitutes a vital subsystem of modern nano-scale ICs which affects in a critical way the performance and correct operation of the device.

Static (DC) or transient analysis refers to the process of computing the response of an electrical circuit to a constant or time-varying stimulus. Since an electrical circuit can be generally modelled as a linear RLC network, this process requires the solution of very large and sparse linear systems of equations derived from Modified Nodal Analysis (MNA) [1]. Direct methods have been widely used in the past for solving the resulting linear systems, mainly because of their robustness in most types of problems. Unfortunately, these methods do not scale well with the dimension of the system, and become prohibitively expensive for circuits beyond a few thousand nodes, in both execution time and memory requirements.

On the other hand, iterative methods such as the Preconditioned Conjugate Gradient (PCG) involve only inner and matrix-vector products, and constitute a better alternative for huge linear systems, being more computationally and memory efficient. The main problem of iterative methods is their unpredictable convergence rate which depends on the properties of the system matrix. A preconditioning mechanism, which transforms the system into one with more favorable properties, is essential to guarantee fast and robust convergence.

An aspect of circuit simulation that has become very important recently is the acceleration of the solution method by harnessing massively parallel architectures, such as graphics processing units (GPUs) and multi-core processors. GPUs, in particular, provide an enormous computational power and as a result are preferred for computationally-intensive tasks. However, the direct solution methods that have been mostly employed in circuit simulation thus far are very difficult to parallelize, due to many sequential dependencies in the factorization process as well as in the forward and backward solves of the resulting triangular systems [2]. On the contrary, Krylov-subspace iterative methods offer ample possibilities for parallelism that have been explored sufficiently well.

Preconditioning of iterative methods for power grid analysis was originally proposed in [3], but the preconditioner used is the general-purpose incomplete Cholesky. A Generalized Minimal Residual (GMRES) solver preconditioned by the incomplete LU preconditioner is proposed by Liu et al. [4]. However, the GMRES solver does not take advantage of the symmetry and positive definiteness of the system, thus increasing the computational cost. The authors in [5] propose two parallel fast transform-based preconditioners for 2D and 3D power grids. The idea of preconditioning by Multigrid techniques is proposed in [6]. An alternative approach is preconditioning by hierarchical support graphs [7], which significantly accelerates the convergence rate of iterative methods for graph-based problems. Preconditioning has also been used for power systems and power flow problems. The authors in [8] propose preconditioners based on multifrontal direct methods, while Algebraic Multigrid preconditioners are proposed for Newton-Krylov power flow methods in [9].

In this work, we present a GPU-accelerated PCG solver preconditioned by the Combinatorial Multigrid (CMG) [10], an efficient algorithm for solving Symmetric Diagonally Dominant (SDD) systems, for DC and transient analysis of large scale circuits. The rest of the paper is organized as follows. The next section provides some background material on the MNA modeling and the PCG iterative method, while Section III presents the CMG algorithm. In Section IV, we describe our hybrid CPU/GPU circuit simulation method. We demonstrate the performance gains of our approach on the IBM power grid designs in Section V. Finally, Section VI concludes the paper.

II. Theoretical Background

A. MNA Modelling and DC Analysis

Every large scale linear circuit can be modelled as an RLC network which may contain several million nodes, and up to hundreds of thousands of input current sources. Let the electrical model of the circuit be composed of N nodes and n inductive branches. Given the current source vector as the excitation of the circuit, the node voltages can be obtained by

solving the following ordinary differential equation, obtained using the Modified Nodal Analysis (MNA):

$$\widetilde{\mathbf{G}}\mathbf{x}(t) + \widetilde{\mathbf{C}}\frac{\mathbf{x}(t)}{dt} = \mathbf{b}(t) \tag{1}$$

where $\widetilde{\mathbf{G}} = \begin{bmatrix} \mathbf{G} & \mathbf{A}_L \\ -\mathbf{A}_L^T & \mathbf{0} \end{bmatrix}, \widetilde{\mathbf{C}} = \begin{bmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{0} & \mathbf{L} \end{bmatrix}, \mathbf{x}(t) = \begin{bmatrix} \mathbf{v}(t) \\ \mathbf{i}(t) \end{bmatrix}, \mathbf{b}(t) = \begin{bmatrix} \mathbf{e}(t) \\ \mathbf{0} \end{bmatrix}$.

In the above system, $\mathbf{G}$ and $\mathbf{C}$ are the $N \times N$ node conductance and node capacitance matrices, $\mathbf{L}$ is the $n \times n$ dense inductance matrix, and $\mathbf{A}_L$ is the corresponding $N \times n$ node-to-branch incidence matrix. Also, $\mathbf{v}(t)$ and $\mathbf{i}(t)$ are the $N \times 1$ and $n \times 1$ vectors of node voltages and inductive branch currents, while $\mathbf{e}(t)$ is the $N \times 1$ vector of excitations from independent sources at the nodes.

For the transient analysis, we can use the Backward-Euler numerical integration method to obtain the system of linear algebraic equations to be solved. In both DC and transient analysis, the system matrix can be shown to be SDD and efficient methods such as the Conjugate Gradient (CG) can be employed for its solution as describen in [3]-[7]. Although our work can be easily applied to transient analysis, in this paper we demonstrate our implementation for DC analysis. On the DC operation point, the system of differential equations (1) is reduced to the following algebraic linear system:

$$\widetilde{\mathbf{G}}\mathbf{x} = \mathbf{b} \tag{2}$$

B. Iterative Solution Methods and Preconditioning

Iterative solution methods are very attractive for solving sparse Symmetric Positive Definite (SPD) systems. Especially, CG is an extremely effective algorithm since it only requires two inner products, three saxpy ($\mathbf{y} = a\mathbf{x} + \mathbf{y}$) operations, one Sparse Matrix-Vector (SpMV) multiplication and a limited amount of storage ($\mathbf{A} + 6n$, where $\mathbf{A}$ is the system matrix and n denotes the order of the matrix). It also scales well with the dimension of the system and offers ample possibilities for parallelizeation on GPUs.

However, the convergence rate of CG is not predictable up front. It is well known that the method suffers from slow convergence rate when the condition number of $\mathbf{A}$ is large, which is defined as $\kappa(\mathbf{A}) = \frac{\lambda_{max}(\mathbf{A})}{\lambda_{min}(\mathbf{A})}$, where $\lambda_{max}(\mathbf{A})$ and $\lambda_{min}(\mathbf{A})$ are the maximum and minimum eigenvalues of $\mathbf{A}$, respectively. To improve the convergence speed, it is necessary to apply a preconditioning mechanism which transforms the initial linear system into an equivalent one with more favourable spectral condition number. Algorithm 1 describes the PCG method for the solution of an SPD linear system $\mathbf{Ax} = \mathbf{b}$. The preconditioner solve step $\mathbf{Mz} = \mathbf{r}$ effectively modifies the CG algorithm to solve the system $\mathbf{M^{-1}Ax} = \mathbf{M^{-1}b}$, which has the same solution as the original system and exhibits condition number $\kappa(\mathbf{M^{-1}A}) \simeq \kappa(\mathbf{I}) = 1$, when the preconditioner $\mathbf{M}$ approximates $\mathbf{A}$ in some way.

Algorithm 1 Preconditioned Conjugate Gradient

$\mathbf{x} =$ initial guess $\mathbf{x}^{(0)}$
$\mathbf{r} = \mathbf{b} - \mathbf{Ax}$
$iter = 0$
while ***not_converged*** **do**
  $iter = iter + 1$
  Solve $\mathbf{Mz} = \mathbf{r}$ **(Preconditioner Solve Step)**
  $\rho = \mathbf{r} \cdot \mathbf{z}$
  if $iter == 1$ **then**
    $\mathbf{p} = \mathbf{z}$
  else
    $\beta = \rho / \rho 1$
    $\mathbf{p} = \mathbf{z} + \beta \mathbf{p}$
  end if
  $\rho 1 = \rho$
  $\mathbf{q} = \mathbf{Ap}$
  $\alpha = \rho / (\mathbf{p} \cdot \mathbf{q})$
  $\mathbf{x} = \mathbf{x} + \alpha \mathbf{p}$
  $\mathbf{r} = \mathbf{r} - \alpha \mathbf{q}$
end while

III. The Combinatorial Multigrid Algorithm

One of the fastest solution methods for linear systems is Multigrid (MG) [11]. The idea of MG is to transform the initial system into a bigger hierarchical system consisting of different resolutions of the same problem. In particular, based on the system matrix $\mathbf{A}$, it generates a hierarchy of grids $\mathbf{M} = \{\mathbf{M}_0, \ldots, \mathbf{M}_d\}$ that look similar but in a different level of detail. The main classes of MG are the Algebraic Multigrid (AMG) and the Geometric Multigrid (GMG). The first one constructs the "coarse" grids/matrices based on their algebraic informations, while the latter one uses a discretization process.

Recent research led to a fast algorithm ($\mathcal{O}(m \log n)$ in theory with an $\mathcal{O}(n)$ implementation, where n is the matrix dimension and m the non-zero elements) for solving SDD linear systems [10]. CMG is an hybrid MG algorithm, based on principles borrowed from both GMG and AMG, which can be used as a solver and/or a preconditioner. The construction of this preconditioner is driven by graph-theory and the well known correspondence between graphs and Laplacians. The Laplacian $\mathbf{A}$ of a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{w})$ with positive weights is defined by:

$$\mathbf{A}_{ij} = \begin{cases} \mathbf{deg}(\mathbf{v}(i)) & \text{, if } i = j \\ -\mathbf{w}_{ij} & \text{, if } i \neq j \end{cases} \tag{3}$$

where $\mathbf{V}$ denotes the vertices, $\mathbf{E}$ the edges, $\mathbf{w}$ the weights and $\mathbf{deg}(\mathbf{v}(i))$ is the degree of $\mathbf{v}(i)$. CMG constructs a hierarchy of graphs/matrices and recursively solves the coarser problem. It can be described by the two following steps:

1) **[Construction of the Steiner preconditioner hierarchy]** CMG uses a graph sparsification algorithm in order to decompose the Laplacian of system matrix $\mathbf{A}$ into disjoint clusters and form the MG preconditioner. For each cluster $\mathbf{V}_i$, it adds an extra node $\mathbf{p}_i$ and creates the corresponding Steiner preconditioner $\mathbf{S}_i$, as a star graph rooted at $\mathbf{p}_i$ with leaves corresponding to the vertices in $\mathbf{V}_i$. A quotient matrix/graph $\mathbf{Q}$, which represents the connections of the roots $\mathbf{p}_i$, is formed for every Steiner preconditioner. The algorithm is applied iteratively on the Laplacian of $\mathbf{Q}$ to construct the hierarchy of preconditioning grids.
2) **[Combinatorial Multigrid Solution Algorithm]** The main idea of CMG is to add a correction $\mathbf{R}_i^T \mathbf{Q}_{i+1}^{-1} \mathbf{R}_i \mathbf{r}_i$ to the iterate $\mathbf{x}_i$ of the MG level i, where $\mathbf{r}_i = \mathbf{b}_i - \mathbf{A}_i \mathbf{x}_i$ is the smooth residual error, $\mathbf{Q}_{i+1}$ is the smaller quotient graph and $\mathbf{R}_i \in \mathbb{R}^{dim(A_i) \times dim(A_{i+1})}$ is a restriction operator. The intuition is that for smooth residuals, the low-rank matrix $\mathbf{R}_i^T \mathbf{Q}_{i+1}^{-1} \mathbf{R}_i \mathbf{r}_i$ is a good approximation of $\mathbf{A}_i^{-1}$. The multigrid algorithm is presented in Algorithm 2.

Algorithm 2 Combinatorial Multigrid Algorithm

function $\mathbf{x}_i = \text{CMG}(\mathbf{A}_i, \mathbf{b}_i)$
  $\mathbf{D} = diagonal(\mathbf{A_i})$
  $\mathbf{x}_i = \mathbf{D}^{-1}\mathbf{b}_i$
  $\mathbf{r}_i = \mathbf{b}_i - \mathbf{A}_i\mathbf{x}_i$
  $\mathbf{b}_{i+1} = \mathbf{R}_i\mathbf{r}_i$
  $\mathbf{z} = \text{CMG}(\mathbf{A}_{i+1}, \mathbf{b}_{i+1})$
  for $i = 1$ to $\mathbf{t}_i - 1$ **do**
    $\mathbf{r}_{i+1} = \mathbf{b}_{i+1} - \mathbf{A}_{i+1}\mathbf{z}$
    $\mathbf{z} = \mathbf{z} + \text{CMG}(\mathbf{A}_{i+1}, \mathbf{r}_{i+1})$
  end for
  $\mathbf{x}_i = \mathbf{x}_i + \mathbf{R}_i^T\mathbf{z}$
  $\mathbf{x}_i = \mathbf{r}_i - \mathbf{D}^{-1}(\mathbf{A}_i\mathbf{x}_i - \mathbf{b}_i)$
end function

IV. Proposed Hybrid CPU-GPU Circuit Simulator

A. CMG in Circuit Simulation

There is a fairly well known analogy between RLC networks and graph Laplacians. As mentioned in Section II, the resulting system is linear and SDD, thus the CMG algorithm can be employed as a solver or preconditioner for the solution of Eq. (2). In this work, we use the PCG iterative method described in Algorithm 1 for the solution of the DC system. For the preconditioner solve step ($\mathbf{Mz} = \mathbf{r}$), we harness the CMG algorithm to improve the convergence rate. Our DC circuit simulator can be described by the following steps:

- *Parsing of SPICE netlist and MNA system creation.*
- *Construction of the CMG multilevel preconditioner.*
- *Solving using the CMG-preconditioned PCG method.*

B. Single GPU Implementation

In the single GPU-accelerated implementation, we port all the PCG and CMG operations to the GPU. This eliminates the need for additional memory transfers between the host and the device reducing the communication overhead, provided that the GPU has sufficient memory to accommodate the working set (especially the system matrix $\mathbf{A}$ and preconditioner $\mathbf{M}$). On the CPU side, after the SPICE netlist parsing and MNA system construction, the CMG preconditioner hierarchy $\mathbf{M}$ is created. Once the preconditioner is created, the host side is responsible to allocate and copy the system matrix $\mathbf{A}$, the right-hand side vector $\mathbf{b}$, the initial guess $\mathbf{x}^{(0)}$, each preconditioner matrix $\mathbf{M}_i$, the inverse of its main diagonal $\mathbf{invD}(\mathbf{M}_i)$, the restriction operator $\mathbf{R}_i$ and some helper vectors to the GPU. When the result is satisfactory, the solution $\mathbf{x}$ of the linear system $\mathbf{Ax} = \mathbf{b}$ is copied back to the CPU side. Note that all the PCG/CMG control flow resides in the CPU. Also, the smallest (level d) CMG grid is solved using Cholesky factorization on the CPU, due to the small size of the system. Figure 1 depicts the proposed single GPU-accelerated method. The working set resides in the red box, PCG algorithm is inside the blue box and CMG preconditioner solve step is in the green box.

C. A Dual-GPU Approach for Solving Larger Systems

In comparison with the CPU system memory, the GPU device memory has considerably smaller capacity and the working set for huge systems exceeds its memory. An approach to overcome this limitation is to re-design our implementation for a dual-GPU configuration. The idea is to assign all the PCG operations to the first GPU, while the CMG-preconditioned solve step is implemented on the second GPU.

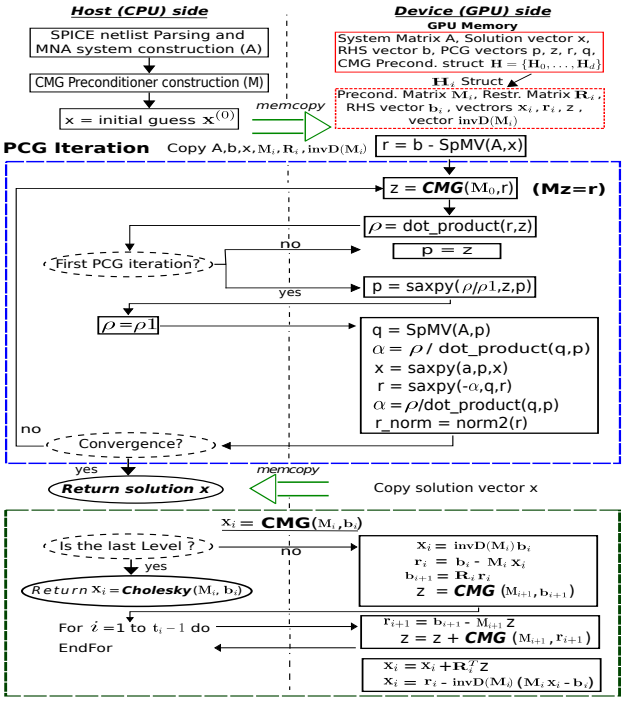

Fig. 1: Single GPU-accelerated circuit simulation method.

This optimization splits the working set almost equally, since the PCG kernel operates only on the system matrix $\mathbf{A}$, whereas the CMG only on the preconditioner $\mathbf{M}$. Figure 2 describes, in a compact way, the proposed dual GPU-accelerated method. The working set resides in the red boxes, the CPU part of the PCG/CMG algorithms is inside the blue box and the corresponding GPU kernels are in the green boxes. The initial memory copies to the GPUs can be performed asynchronously. The only need for communication between the GPUs is the exchange of the vectors $\mathbf{r}$, $\mathbf{z}$, where the GPUDirect technology can be used to provide fast Peer-to-Peer data transfers/accesses without CPU involvement. Furthermore, the latest NVIDIA® architectures (Pascal, Volta) provide the NVLink interconnect which can offer significantly higher bandwidth.

D. Exploiting Multi-GPU Systems for Further Acceleration

Even greater acceleration (and solution of larger systems) can be obtained by exploiting multi-GPU systems. The main concept of such implementations is the partition of the matrices and vectors to several blocks and the assignment of each block to a GPU which is responsible to calculate the corresponding part of the result. The communication between the multiple hosts and GPUs can be achieved using the Message Passing Interface (MPI). Several multi-GPU works have been proposed [12], offering great scalability as the number of available GPUs increases.

V. Experimental Results

We developed an optimized C/C++ CPU-only version of the proposed method which we use as a reference for our hybrid CPU-GPU implementation. We take advantage of OpenMP [13] and Intel® Math Kernel Library (MKL) [14] for the CPU

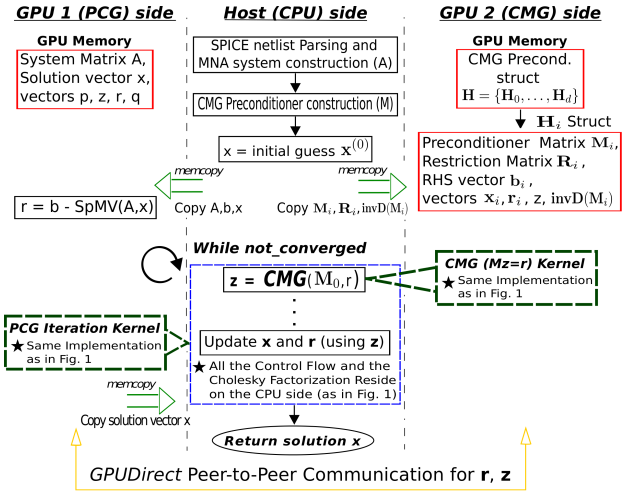

Fig. 2: Dual GPU-accelerated circuit simulation method.

TABLE I: Runtime Results.

Benchmark	CPU time (s)		GPU time (s)		Speedup	
	PCG	CMG	PCG	CMG	PCG	CMG
ibmpg1	2.84	2.25	1.96	1.34	1.45X	1.67X
ibmpg2	0.40	0.30	0.21	0.16	1.92X	1.88X
ibmpg3	3.21	2.16	0.90	0.68	3.58X	3.18X
ibmpg4	2.15	1.47	0.52	0.38	4.13X	3.83X
ibmpg5	501.74	317.87	106.87	70.70	4.69X	4.50X
ibmpgnew1	576.08	368.99	128.31	88.73	4.47X	4.16X
ibmpgnew2	4.74	3.17	1.34	1.05	3.53X	3.02X

TABLE II: IBM Power Grid Benchmarks.

Benchmark	$\#i$	$\#n$	$\#r$	$\#s$	$\#v$	$\#l$
ibmpg1	10774	30638	30027	14208	14308	2
ibmpg2	37926	127238	208325	1298	330	5
ibmpg3	201054	851584	1401572	461	955	5
ibmpg4	276976	953583	1560645	11682	962	6
ibmpg5	540800	1079310	1076848	606587	539087	3
ibmpgnew1	357930	1461036	2352355	461	955	NA
ibmpgnew2	357930	1461039	1422830	929722	930216	NA

where i stands for current sources, n for nodes, r for resistors, s for shorts, v for voltage sources and l for metal layers.

version, while for the GPU mapping we exploit the CUDA *cuSPARSE* and *cuBLAS* libraries [15]. Both MKL and CUDA libraries contain optimized BLAS kernels for handling vectors and sparse matrices.

It is worth mentioning that although we can store only the upper part of $\mathbf{A}$, as it is symmetric, the *cuSPARSE* SpMV method for this case has to use atomic operations to resolve the existing race conditions, resulting in poor performance. To this end, we store the full matrix for our GPU implementation, whereas for the CPU version we store only the upper part of $\mathbf{A}$. For both sparse matrix $\mathbf{A}$ and preconditioner $\mathbf{M}$, the Compressed Sparse Row (CSR) format is used.

For the evaluation we used a Linux workstation with an Intel® 14-core Xeon® processor at 2.30GHz and 128GB memory, along with an NVIDIA® Tesla™ K80 accelerator of 4992 CUDA cores and 24GB total memory running at 0.87GHz (2 GPUs with 2496 cores and 12GB of global memory each). We have employed double-precision arithmetic, while the convergence tolerance of PCG was set to 10^{-6}.

Table I presents the runtimes and speedups achieved by our dual-GPU implementation of Fig. 2, for the IBM power grids [16]. Those benchmarks, explained in Table II, are resistive-only and vary over a reasonable range of size. We can observe that the GPU-accelerated CMG preconditioner solve step is up to 4.50x faster than the corresponding CPU implementation. The runtime speedup of the entire PCG method, including the CMG preconditioner solve step, is up to 4.69x. Moreover, it can be observed that as the dimension of the system increases, the method achieves better speedups. Note that the proposed implementation is far more suggested for transient simulation, where the system has to be solved at multiple time-steps. Finally, the exploitation of multi-GPU systems can result to more significant speedups, as described in Section IV-D.

VI. Conclusion

In this paper, we have presented an hybrid CMG-preconditioned iterative method for large-scale circuit simulation. Specifically, we focused on the acceleration of the PCG algorithm on single and multiple GPUs, exploiting CMG as a preconditioning technique. Experimental results on IBM power grids using an NVIDIA® K80 dual GPU, showed speedups up to 4.69x and 4.50x for the PCG and the CMG preconditioning algorithm, respectively.

References

[1] C.-W. Ho, A. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Transactions on Circuits and Systems*, 1975.

[2] X. S. Li, M. Shao, I. Yamazaki, and E. G. Ng, "Factorization-based sparse solvers and preconditioners," *Journal of Physics: Conference Series*, 2009.

[3] T.-H. Chen and C. C.-P. Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods," in *Proceedings of the 38th Design Automation Conference*, 2001.

[4] X. X. Liu, H. Wang, and S. X. D. Tan, "Parallel power grid analysis using preconditioned GMRES solver on CPU-GPU platforms," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013.

[5] K. Daloukas, N. Evmorfopoulos, P. Tsompanopoulou, and G. Stamoulis, "Parallel Fast Transform-Based Preconditioners for Large-Scale Power Grid Analysis on Graphics Processing Units (GPUs)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[6] Z. Feng, Z. Zeng, and P. Li, "Parallel On-Chip Power Distribution Network Analysis on Multi-Core-Multi-GPU Platforms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2011.

[7] X. Zhao, J. Wang, Z. Feng, and S. Hu, "Power grid analysis with hierarchical support graphs," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011.

[8] S. K. Khaitan and J. D. McCalley, "A Class of New Preconditioners for Linear Solvers Used in Power System Time-Domain Simulation," *IEEE Transactions on Power Systems*, 2010.

[9] R. Idema, G. Papaefthymiou, D. Lahaye, C. Vuik, and L. van der Sluis, "Towards Faster Solution of Large Power Flow Problems," *IEEE Transactions on Power Systems*, 2013.

[10] I. Koutis, G. L. Miller, and D. Tolliver, "Combinatorial Preconditioners and Multilevel Solvers for Problems in Computer Vision and Image Processing," *Comput. Vis. Image Underst.*, 2011.

[11] U. Trottenberg, C. Oosterlee, and A. Schuller, *Multigrid*, 2000.

[12] G. Oyarzun, R. Borrell, A. Gorobets, and A. Oliva, "MPI-CUDA sparse matrixvector multiplication for the conjugate gradient method with an approximate inverse preconditioner," *Computers Fluids*, 2014.

[13] *The OpenMP API specification for parallel programming*. [Online]. Available: http://www.openmp.org/

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

[15] *CUDA CUSPARSE and CUBLAS Library User Guides*. [Online]. Available: http://developer.nvidia.com/nvidia-gpu-computing-documentation/

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