Exploiting parallelism by data dependency elimination : a case study of circuit simulation algorithms

Wu, Wei; Gong, Fang; Krishnan, Rahul; Yu, Hao; He, Lei

2012

Wu, W., Gong, F., Krishnan, R., Yu, H., & He, L. (2012). Exploiting Parallelism by Data Dependency Elimination: A Case Study of Circuit Simulation Algorithms. IEEE Design & Test of Computers, 1-1.

https://hdl.handle.net/10356/95237

https://doi.org/10.1109/MDT.2012.2226201

Exploiting Parallelism by Data Dependency Elimination: A Case Study of Circuit Simulation Algorithms

Wei Wu
Electrical Engineering Department
University of California at Los Angeles
Los Angeles, CA
weiwu@ee.ucla.edu

Fang Gong
Electrical Engineering Department
University of California at Los Angeles
Los Angeles, CA
gongfang@ucla.edu

Rahul Krishnan,
Electrical Engineering Department
University of California at Los Angeles
Los Angeles, CA
r.krishnan390@UCLA.edu

Hao Yu
School of Electrical and Electronic Engineering
Nanyang Technological University
Singapore
haoyu@ntu.edu.sg

Lei He
Electrical Engineering Department
University of California at Los Angeles
Los Angeles, CA
lhe@ee.ucla.edu

Abstract— **Considering the increasing complexity of integrated circuit (IC) designs at Nano-Tera scale, multi-core CPUs and many-core GPUs have provided ideal hardware platforms for emerging parallel algorithm developments in electronic design automation (EDA). However, it has become extremely challenging to leverage parallel hardware platforms at extreme scale beyond 22nm and 60GHz where the EDA algorithms, such as circuit simulation, show strong data dependencies. This paper presents data dependency elimination approaches in circuit simulation algorithms such as parasitic extraction, transient simulation and periodic-steady-state (PSS) simulation, which paves the way towards unleashing the underlying power of parallel hardware platforms.**

Keywords- Parallelism and Concurrency, Data Dependencies, Parallel Algorithms, Circuit Simulation.

I. INTRODUCTION

Over the past few years, the 22nanometer (nm) design has become prevalent in digital circuits to increase the circuit density while the frequency of Radio-Frequency (RF) circuit has roared up to 60GHz, or even higher, to satisfy the increasing demand of mobile multimedia communication. Consequently, the complexities of post-layout level verification during parasitic extraction, transient and RF periodic-steady-state (PSS) simulations have increased significantly. The development of parallel algorithms tackles this issue by inventing new approaches towards parallel circuit simulation in electronic design automation (EDA).

Recently, multi-core CPUs and many-core GPUs have become widely adopted with largely reduced cost. Due to the increasing popularity of parallel hardware platforms, revolutionary development from sequential algorithms to their parallel counterparts is taking place in the software development community, including EDA.

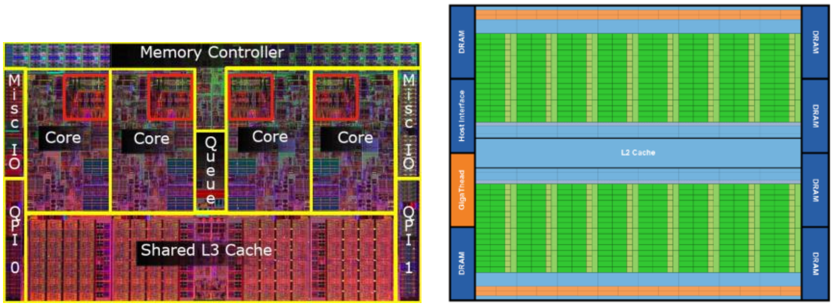
However, circuit simulation algorithms for designs at the extreme scale beyond 22nm and 60GHz are difficult for parallelization. Due to the nature of circuits, the circuit simulation algorithms usually deal with sparse matrices, as most components are sparsely interconnected with a few others components [1], [2]. Unlike dense algebra operations, algorithms for sparse data structure show irregular data dependence patterns [1]. At the same time, parasitics and EM coupling can result in strong correlation and hence also strong data dependency. As a result, the algorithms for circuit simulation cannot be effectively parallelized by simply unfolding “for” loops into parallel code.

Most EDA algorithms, especially circuit simulation algorithms, are relevant to graph algorithm or linear algebra [3]. To efficiently parallelize these algorithms on multi-core CPUs and many-core GPUs, a few recent innovations of parallelization have been proposed [4], [5], [6], [7] by reformulating the original irregular or coupled data into structured data with eliminated dependency. For example, board-block-diagonal (BBD) matrix formulation is deployed for the sparse MNA matrix with inverse-inductance [4]; fast-multiple-method (FMM) formulation is deployed for capacitance extraction in the presence of stochastic variation [5]; simplified elimination-tree scheduling is deployed for the sparse matrix factor during transient simulation [6]; and periodic-cyclic-structured matrix formulation is deployed for RF-PSS simulation by shooting-Newton method [7].

This paper targets to summarize the aforementioned parallel algorithms for EDA circuit simulations. We first discuss the existing parallel hardware platforms and the methodologies of structuring the data access pattern and eliminating dependency. Then, three typical applications, ranging from circuit parameter extraction to transient simulation and RF-PSS simulation, are further discussed as case studies to illustrate the methodology.

II. Parallel Hardware Architectures

Increasing power and thermal densities on single-core processors has limited the growth of their operating frequency [8]. As such, the advancement of processor technology in the past decade was altered from increasing operating frequency on single core to integrating multiple cores into one single processor. Nowadays, the parallel computing hardware platforms, such as multi-core CPUs, many-core GPUs and FPGAs, are affordable and have become prevalent in consumer electronics.



(a) Intel's Nehalem Architecture (b) NVIDIA's Fermi Architecture

Fig. 1. Architecture of multi-core CPU and many-core GPU [9]

Current multi-core CPUs are usually integrated with 1 to 4 cores, or even 6 cores, on a single die. Beyond 6 cores, memory bandwidth becomes the bottleneck of further performance enhancement. Current X86 microprocessors, such as Intel® Xeon® processors with Nehalem architecture, whose layout is illustrated in Fig. 1(a), are examples of multi-core CPUs [10]. In addition, some coarse-grained parallelism programming environments (i.e., POSIX Threads, OpenMP and MPI) have been developed on multi-core systems as user-friendly solutions for parallelization.

For GPUs, NVIDIA®'s FERMI™ architecture integrates up to 512 CUDA cores, which demonstrates notable potential for scalability [11]. High level programming languages, CUDA and OpenCL, are developed to unleash the underlying power in the GPU. However, the CUDA cores, which are much smaller and simpler as illustrated in Fig. 1(b), are usually not general purpose and can only execute simple operations. Therefore, GPUs are typically applied to fine-grained parallelism where each operation is very simple to be implemented on one CUDA core or one thread.

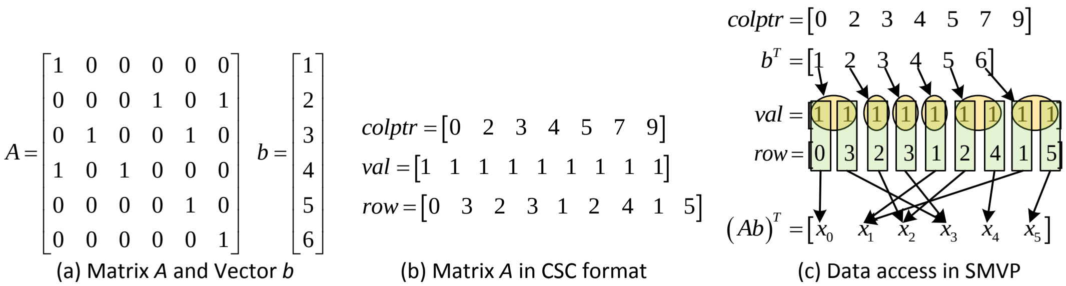
Another parallel hardware platform, FPGA, is featured with flexibility due to its reconfigurable architecture. It is usually deployed as a network processor [12] and accelerator for specific applications [13], [14]. Compared to multi-core CPUs and many-core GPUs, where friendly programming environments are developed, FPGAs are still programmed by low-level hardware description languages (HDL), such as Verilog HDL and VHDL. With the absence of efficient high-level programming languages, FPGAs are fading away from the major arena of parallel computing.

III. Data Access Pattern and Data Dependency

Parallelism efficiency is determined by the data structure in the algorithm. As an example, this section discusses the data access pattern and data dependency based on the data structures of dense matrices and sparse matrices, respectively.

In a dense matrix, each entry can be accessed directly by its row and column index. Thus the dense matrix-vector-products (MVP) can be easily parallelized on GPU or multi-core CPU by unfolding the multiplication operations on each core. Different from the dense matrix, the sparse matrix is usually stored in a compressed sparse column (CSC) format, as shown in Fig. 2(b), which consists of three vectors for row index, entry value and starting/ending boundary for each column. During the sparse MVP, the row index of each matrix entry needs to be accessed along with the multiplications. Since in each column of a sparse matrix, the nonzero entries are located in different location (with different row indices), it complicates the data access pattern, as illustrated in Fig. 2(c). Consequently, fine-grained parallelism cannot be achieved by straightforwardly mapping all the data and operations to multi-core/many-core platforms with balanced loads.

Data dependency is another critical issue for parallelism. Although the algorithm of sparse MVP is hard to be parallelized with fine-grained patterns, it can be considered as several vector-vector-products, which are independent and can be executed simultaneously. However, for more complicated sparse algebra algorithm, such as sparse matrix LU factorization [6], if we consider the algorithm as several tasks, strong dependency exists between the tasks. The data dependency is usually illustrated by the directed acyclic graph (DAG), where each node represents a task and the edge illustrates the dependencies between tasks. A DAG of sparse matrix LU factorization is shown in Fig. 5(b) during the case study. If these tasks are directly assigned to parallel hardware, they cannot be efficiently parallelized due to high overhead of synchronization that caused by data dependencies.



$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ $b = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$
(a) Matrix A and Vector b
$colptr = [0 \ 2 \ 3 \ 4 \ 5 \ 7 \ 9]$
$val = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]$
$row = [0 \ 3 \ 2 \ 3 \ 1 \ 2 \ 4 \ 1 \ 5]$
(b) Matrix A in CSC format
$colptr = [0 \ 2 \ 3 \ 4 \ 5 \ 7 \ 9]$
$b^T = [1 \ 2 \ 3 \ 4 \ 5 \ 6]$
$val = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]$
$row = [0 \ 3 \ 2 \ 3 \ 1 \ 2 \ 4 \ 1 \ 5]$
$(Ab)^T = [x_0 \ x_1 \ x_2 \ x_3 \ x_4 \ x_5]$
(c) Data access in SMVP

Fig. 2. Data access in sparse matrix operations

To parallelize these applications, one solution is to build customized architecture on FPGAs to deal with the task synchronization, such as the GraphStep [13], [15]. However, the on-chip resources and long development cycle limit the application of FPGAs on EDA algorithms. Based on existing architecture, such as multi-core CPUs, an effective solution is to study the algorithm itself and reformulate it to cater to the architecture of parallel hardware. Building structured algorithms [4], [7] and eliminating the data dependency [5], [6] are typical approaches to achieve this goal.

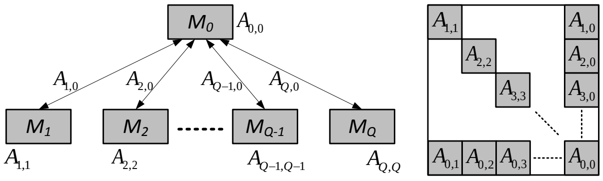
IV. Case study: Applications in Circuit Simulation

In this section, three typical algorithms, ranging from parameter extraction to transient simulation and RF-PSS simulation, are illustrated to show how parallelism can be achieved by the structured reformulation to eliminate the data dependency.

A. Inductive Interconnection Analysis and Capacitance Extraction

The extraction and analysis of inductance and capacitance is important during layout simulation. With transistor size scaling down to 22nm and RF operating frequency scaling up to 60GHz, the strong inductive coupling and the stochastic capacitive coupling are difficult to model and analyze. In this subsection, we describe parallelization algorithms by BBD formulation of the sparse MNA with inverse-inductance and by FMM formulation of capacitance extraction with stochastic variation, respectively.

1) Build BBD structure for Inductive Interconnections: The post-layout circuits are analyzed using the modified nodal analysis (MNA) algorithm [16], where the circuit is represented by a large sparse circuit matrix. For the RC network, an efficient solution is to formulate the BBD structured circuit matrix by network decomposition [17]. As shown in Fig. 3(a), the RC network can be partitioned into a few independent blocks and each block at the leaf level may only have coupling with a top-level super block (i.e., M0). Then, the circuit matrix can be formulated into a BBD fashion as shown in Fig. 3(b), where the diagonal blocks represent the connections inside each block, and the border blocks indicate the connections between the top-level block and other blocks.



(a) Block-wise partition of RC network (b) BBD matrix

Fig. 3. BBD Matrix Formulated by Block-wise Partition

In the traditional MNA algorithm, the state variables are branch currents and node voltages. In the inductive interconnections, there are a large number of nonzero fill-ins between different blocks in BBD structure to represent the long-range mutual inductance. Therefore, it is difficult to directly formulate a BBD structure for RLC network when there exists strong inductive coupling from L matrix, which is important for 60GHz RF designs. As L matrix is not diagonal dominant, simply pruning mutual inductance results in the loss of passivity.

To achieve a sparse yet passive structured organization of RLC data in BBD formulation, vector-potential nodal analysis (VNA) based RLC representation has been introduced [4], [18]. Instead of using branch currents as state variables for inductance L, magnetic flux is utilized as the state variable. During the formulation of the circuit matrix by VNA, one can stamp L^{-1} matrix instead of L. Moreover, since L^{-1} is diagonal dominant, its coupling entries can be pruned without the loss of passivity [4], [18]. Based on the VNA state matrix, one can build the sparse yet passive BBD formulation, which further facilitates the parallel simulation on multi-core CPUs [19].

The proposed method is evaluated in a model order reduction framework. The BBD-VNA based reduction method (BVOR) is compared with the nodal analysis (NA) based reduction method (SAPOR) and MNA based reduction method (PACT). In the experiments, 3 types of RLC circuits (14 circuits in total), including buses, clock trees and mesh networks, are deployed. While comparing the simulation runtime on the reduced circuits, we demonstrate that BVOR achieves 2.8-33.2x speedup over SAPOR (11.7x in average), and 2.4-28.7x speedup over PACT (9.1x in average) [4].

2) Dependency Elimination for Capacitance Extraction: The parallel capacitance extraction under process variation is also difficult in the presence of stochastic variation for digital designs at 22nm. In particular, the work in [5] models the process variation by stochastic orthogonal polynomials (SOP) and further incorporates the variation into a modified fast-multipole method (FMM) to evaluate the potential interactions between conductor surface panels in parallel. In particular, the potential interaction evaluation needs to calculate a matrix-vector product (MVP) and the modified FMM algorithm tries to reduce the complexity of MVP calculation from $O(N^2)$ to nearly $O(N)$ where N is the number of variables.

In general, the parallel FMM algorithm assigns surface panels into small cubes and builds a hierarchical oct-tree of cubes such that the potential interactions between well-separated cubes at different levels can be evaluated on different processors in parallel. Clearly, there exists strong data dependency between different processors which can significantly degrade the performance. To this end, a dependency list as shown in Fig. 4 is used in [5] to pre-fetch the needed data for each processor before its computation.

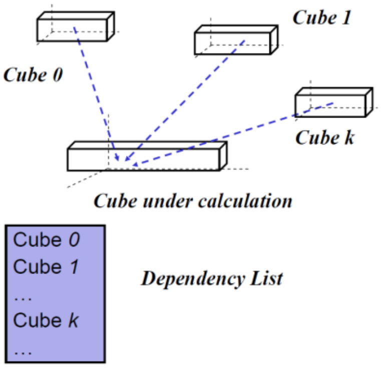


Fig. 4. Pre-fetch operation with dependency list

The dependency list of one cube under study records other cubes that requires its computation results (shown in the shaded area) so as to distribute its generated data ahead of time. In other words, the processors handling those dependent cubes can pre-fetch needed data and proceed without any latency, thereby eliminating the dependency between different processors. [5] has studied the proposed algorithm on examples with a different number of variables as shown in Table(I), where the parallel algorithm shows good scalability for speedup with respect to the number of processors.

Table I: Runtime (seconds) comparison with different number of variables

# variable	12360	10320	11040	12480
1 processor	0.737515/1.0	0.541515/1.0	0.605635/1.0	0.968310/1.0
2 processors	0.440821/1.7X	0.426389/1.4X	0.352113/1.7X	0.572964/1.7X
3 processors	0.367040/2.0X	0.274881/2.0X	0.301311/2.0X	0.489045/2.0X
4 processors	0.273408/2.7X	0.190120/2.9X	0.204606/3.0X	0.340954/2.8X

B. Sparse Direct Solver for Transient Simulation

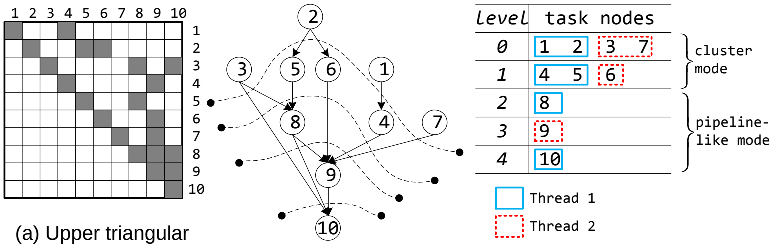
After circuit parameter extraction, the block circuit matrix in the BBD partition is usually sparse and solving the sparse circuit matrix is identified as the bottleneck during the general transient simulation in SPICE. The transient simulation is critical to verify high-precision designs at 22nm such as transient noise. According to Synopsys®'s white paper, the sparse direct solver can consume more than half the simulation time for large post-layout circuits and it is difficult to be parallelized [20].

In general, the LU factorization algorithm is deployed to solve a sparse matrix, which includes two steps: 1) symbolic analysis to determine the position of non-zeros in matrix L and U; and 2) numerical factorization to calculate the values of each non-zero. For circuit simulations, while the symbolic analysis needs to be performed only once to calculate the sparse pattern, the numerical factorization is repeatedly executed as sparse entries are updated.

A typical numerical factorization algorithm for $N \times N$ matrix, A, is the left-looking Gilbert/Peierls algorithm [21], as shown in Algorithm 1. The basic idea of parallelizing Algorithm 1 is to unfold the N tasks (iterations) in the outer *for* loop. However, strong dependency can be identified among these tasks. It is easy to generate a DAG to represent the dependency of all tasks from the symbolic structure of *U*, as illustrated in Fig. 5 [6]. Here we define the task *p* as the parent of task *p* if there is an edge pointing from *p* to *i*. It is obvious that a task is dependent on its parent task(s).

Algorithm 1 Left-looking G/P numerical factorization

```
$L = I;$
for $k = 1:N$ do
    $x = A(:,k);$
    for $j = 1:k-1, where\, U(j,k) != 0$ do
        $x(j+1:n) -= L(j+1:n, j) * x(j);$
    end for
    $U(i:k,k) = x(1:k);$
    $L(k:N,k) = x(k:N)/U(k,k);$
end for
```



level	task nodes
0	1 2 3 7
1	4 5 6
2	8
3	9
4	10

(a) Upper triangular matrix *U*

(b) *EGraph*

(c) *EScheduler*

Fig. 5. Upper matrix U, DAG and the graph partition result [6]

One idea is to process these *N* tasks in a *pipeline-like* mode. Assuming there are multiple parent tasks, p_1, p_2, ... , p_k, for task *i*, fortunately, the task *i* does not have to wait for all its parent tasks to finish. Part of task *i* can be processed with the data from those finished parent tasks. Therefore, task *i* can even be overlapped with some of its parent nodes, which results in a *pipeline-like* structure.

However, the overhead of synchronizing these tasks is a drawback of the *pipeline-like* mode. To improve the efficiency, *Cluster Mode* is defined by analyzing the DAG and categorizing the tasks without dependency into a group. Then the tasks in each group can be parallelized without synchronization. Given the DAG, we group the task(s) without parent task(s) each time and eliminate them from the original DAG. It is obvious that tasks in the same group are independent of each other. In Fig. 5(b), the DAG is processed iteratively and tasks are categorized into five groups as in Fig. 5(c).

To achieve higher parallel efficiency, one can combine the *Cluster Mode* with *Pipeline Mode*. For example, in Fig. 5(c), we process groups 0 and 1 in cluster mode to reduce the overhead on thread synchronization, while groups 3-5 are factorized in pipeline mode so as to fully utilize the computation capability of 2 threads. To evaluate the proposed *Hybrid Mode* [6], we compared it with *All Pipeline-like Mode* and *All Cluster Mode* using 26 circuit matrices from the University of Florida sparse matrix collection [22]. Using 4 threads, the acceleration rates of these three parallel modes over the sequential algorithm is illustrated in Fig. 6. It is obvious that the proposed *Hybrid Mode* outperforms other parallel modes because it takes full advantage of all threads while maintaining the smallest overhead on thread synchronization. The proposed parallel solver is also compared to other solvers and achieves better performance [6], such as 1.18-4.45x (with 1-8 threads) faster than KLU (optimized for circuit simulation problems) [2] and even higher speedup over SuperLU_MT (a general-purpose parallel sparse matrix solver) [23], [24].

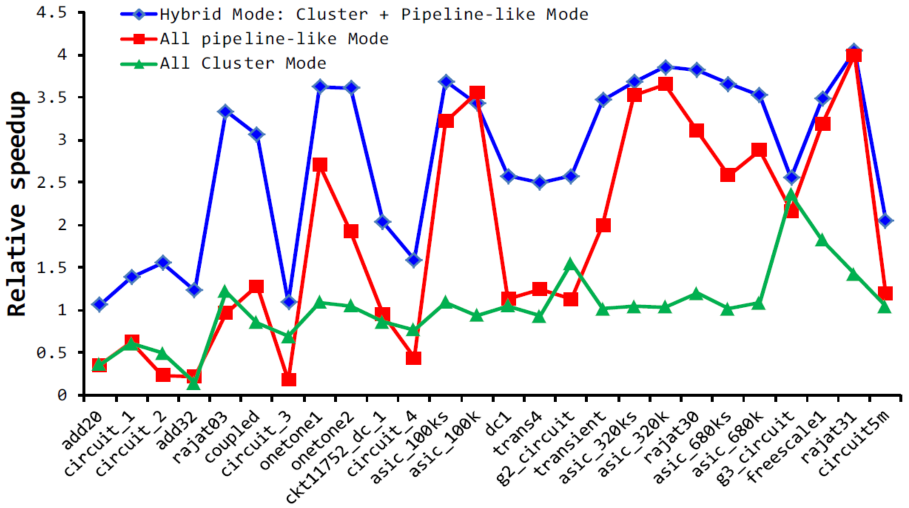


Fig. 6. Performance comparison of three parallel modes

C. *Periodic Arnoldi Shooting for RF-PSS Simulation*

The analysis of RF circuits is notoriously difficult to speed up as accuracy cannot be compromised for precise design at the scale of 60GHz. The shooting-Newton method is usually chosen to find the PSS solution due to its strong convergence properties [7]. However, the resulting Jacobian (sensitivity matrix) during the shooting-Newton method can be a large-scale dense matrix. The iterative GMRES with the use of a standard Krylov-subspace and an implicit matrix formulation (matrix-free GMRES) [25] may alleviate part of the cost but still has limited performance for large-scale RF/MM-ICs designed at 60GHz or beyond.

To accelerate the matrix-free GMRES algorithm, a straightforward parallelization of MVP on multi-core CPUs or many-core GPUs can be beneficial. In [7], [26], we show that studying the structure of the shooting Jacobian is an effective way to further explore the parallelism not only from MVP. Since most RF circuits usually have periodic inputs and can be characterized as a PSS problem, the state matrix in terms of the shooting Jacobian generally becomes

$$J_{\phi T} = A_P \times A_{P-1} \times \cdots \times A_1 \tag{1}$$

where $A_j = \left[G_j + \frac{C_j}{h_j}\right]^{-1} \frac{C_{j-1}}{h_j}$ is the state matrix for the j^{th} time step. G_j and C_j are the linearized admittance and capacitance matrices in the j^{th} time step, while h_j is the j^{th} time step [26].

Note that $j = 1, \ldots, p$ represent for p steps in one period. In [26], we show that the above multiplied product $J_{\phi T}$, the shooting Jacobian matrix, has an identical invariant subspace as follows:

$$J = \begin{bmatrix} 0 & & & \mathbf{A}_p \\ \mathbf{A}_1 & \ddots & & \\ & \ddots & & \\ & & \mathbf{A}_{p-1} & 0 \end{bmatrix}, \tag{2}$$

It has a *periodic-cyclic-block* structured Krylov-subspace, which can be determined through a parallel periodic Arnoldi method and can be parallelized [26]. As shown in Algorithm 2, a periodic Arnoldi method [26] is employed to generate the periodic Krylov-subspace, i.e., the block matrices $\mathbf{V}_j^m$ and $\mathbf{H}_j^m$, $j = 1, \ldots, p$. Here, the subscript j denotes the index of the periodic blocks ($j = 1, \ldots, p$), and the superscript i denotes the index of the order of the Krylov-subspace ($i = 1, \ldots, m$). Note that the periodic structure of the generated Krylov-subspace is preserved in Algorithm 2 because each orthonormalized base $\mathbf{v}_j^i$ is constructed separately for each $\mathbf{A}_j$. As such, one can explore the parallelism during the GMRES iteration. Because of the independent calculation of the new basis vector inside each subspace for different time-steps j ($j = 1, \ldots, p$) in Algorithm 2, the structure-preserved Arnoldi iteration can be highly parallelized on GPU. We call the approach PAS-GMRES.

Algorithm 2 A matrix-free periodic Arnoldi method

```
Input: $\mathbf{A}_j$ by pre-factorized matrices ( $j = 1, \ldots, p$ )
Initialize: $\mathbf{V}_1^0$ by $\mathbf{v}^0$ and $\mathbf{V}_0^i$ by 0
while $\mathbf{h}^{i+1} > tol$ & $i < maxIter$ do
    for $j = 1 : p$ do
        Set $\mathbf{h}_j^i = \mathbf{V}_{j+1}^{i-1}\mathbf{A}_j\mathbf{v}_i^j$
        Set $\mathbf{w} = \mathbf{A}_j\mathbf{v}_i^j - \mathbf{V}_{j+1}^{i-1}\mathbf{h}_j^i$
        Set $\mathbf{g}_j^i = \| \mathbf{w} \|_2, \mathbf{H}_j^i = \begin{bmatrix} \mathbf{H}_j^{i-1} & \mathbf{h}_j^i \\ 0 & \mathbf{g}_j^i \end{bmatrix}$
        Set $\mathbf{v}_{j+1}^i = \mathbf{w} / \mathbf{g}_j^i,\ \mathbf{V}_{j+1}^i = [\mathbf{V}_{j+1}^{i-1}, \mathbf{v}_{j+1}^i]$
    end for
end while
Output: Block matrices $\mathbf{V}_j^m$ and $\mathbf{H}_j^m$ ( $j = 1, \ldots, p$ )
```

Table II: Runtime time comparison of different GMRES methods

Ckt	#Eq	Time (s)		
		CPU-GMRES	GPU-GMRES	GPU-PAS-GMRES
dc-converter	481	35.8	1.13	0.10
BJT-mixer	504	51.1	2.69	0.56
switch-cap	654	52.5	0.30	0.04
freq-mult	649	139.0	5.42	0.81
LNA	1055	1238.0	27.2	1.04

In Table II, we demonstrate the speedup of the GPU parallelized PAS-GMRES. When the GPU parallelization is applied to the PAS-GMRES solver, there is a further speedup of parallel GPU-PAS-GMRES over the GPU-GMRES (matrix-free) at up to 27x for these examples.

V. DISCUSSION AND CONCLUSION

The EDA community is undergoing an overhaul of parallelization to keep pace with the increasing complexity of VLSI circuits at extreme scales. To unleash the underlying power of parallel hardware for EDA applications, the algorithm itself has to be studied in depth to eliminate the data dependency. In this paper, three circuit simulation algorithms are studied to illustrate the methodology of dependency elimination by means of building structured algorithms. In the example of parasitic extraction, VNA and matrix stretching are proposed to formulate a BBD matrix for inductive interconnect, and s stochastic FMM is developed for capacitance extraction with variation. By analyzing and partitioning the dependency with DAG and combining the cluster and pipeline-like mode, a high acceleration rate is achieved for sparse circuit matrix solver, which is currently viewed as the bottleneck of parallelism for transient simulation. In addition, the parallelism of periodic Arnoldi shooting is also presented for RF-PSS analysis, which takes advantage of the cyclic matrix structure. As a methodology of parallelism, algorithm-structure study to eliminate data-dependency is expected to be meaningful as well for other applications in or beyond the EDA community.

REFERENCE

[1] Y. Deng, B. Wang, and S. Mu, “Taming irregular EDA applications on GPUs,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, nov. 2009, pp. 539 –546.

[2] T. A. Davis and E. Palamadai Natarajan, “Algorithm 907: KLU, A direct sparse solver for circuit simulation problems,” *ACM Trans. Math. Softw.*, vol. 37, pp. 36:1–36:17, September 2010.

[3] B. Catanzaro, K. Keutzer, and B.-Y. Su, “Parallelizing CAD: A timely research agenda for EDA,” in *45th ACM/IEEEDesign Automation Conference (DAC)*, june 2008, pp. 12 –17.

[4] H. Yu, C. Chu, Y. Shi, D. Smart, L. He, and S.-D. Tan, “Fast analysis of a large-scale inductive interconnect by block-structure-preserved macromodeling,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 10, pp. 1399 –1411, oct. 2010.

[5] F. Gong, H. Yu, L. Wang, and L. He, “A parallel and incremental extraction of variational capacitance with stochastic geometric moments,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. PP, no. 99, pp. 1 –9, 2011.

[6] X. Chen, W. Wu, Y. Wang, H. Yu, and H. Yang, “An EScheduler-based data dependence analysis and task scheduling for parallel circuit simulation,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 58, no. 10, pp. 702 –706, oct. 2011.

[7] X.-X. Liu, H. Yu, J. Relles, and S.-D. Tan, “A structured parallel periodic arnoldi shooting algorithm for RF-PSS analysis based on GPU platforms,” in *16th Asia and South Pacific Design Automation Conference (ASP-DAC)*, jan. 2011, pp. 13 –18.

[8] P. Gepner and M. Kowalik, “Multi-core processors: New way to achieve high system performance,” in *International Symposium on Parallel Computing in Electrical Engineering*, sept. 2006, pp. 9 –13

[9] P. N. Glaskowsky. (2009) Nvidias fermi: The first complete gpu computing architecture. White Paper. NVIDIA.

[10] (2008) First the tick, now the tock: Next generation intel® microarchitecture (nehalem). White Paper. Intel.

[11] (2009) NVIDIAs next generation cuda® compute architecture: Fermi®. White Paper. NVIDIA.

[12] J. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, “Netfpga–an open platform for gigabit-rate network switching and routing,” in *IEEE International Conference on Microelectronic Systems Education*. IEEE, 2007, pp.160–161.

[13] N. Kapre and A. DeHon, “Parallelizing sparse matrix solve for SPICE circuit simulation using FPGAs,” in *International Conference on Field-Programmable Technology*, dec. 2009, pp. 190 –198.

[14] W. Wu, Y. Shan, X. Chen, Y. Wang, and H. Yang, “FPGA accelerated parallel sparse matrix factorization for circuit simulations,” in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science, A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, and T. El-Ghazawi, Eds. Springer Berlin / Heidelberg, 2011, vol. 6578, pp. 302–315.

[15] M. deLorimier, N. Kapre, N. Mehta, D. Rizzo, I. Eslick, R. Rubin, T. Uribe, T. Knight, and A. DeHon, “Graphstep: A system architecture for sparse-graph algorithms,” in *14th IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006.

[16] C.-W. Ho, A. Ruehli, and P. Brennan, “The modified nodal approach to network analysis,” *IEEE Trans. Circuits Syst.*, vol. 22, no. 6, pp. 504 – 509, jun 1975.

[17] F. Wu, “Solution of large-scale networks by tearing,” *IEEE Trans. Circuits Syst.*, vol. 23, no. 12, pp. 706 – 713, dec 1976.

[18] H. Yu, Y. Shi, L. He, and D. Smart, “A fast block structure preserving model order reduction for inverse inductance circuits,” in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 2006, pp. 7–12.

[19] C. Bomhof and H. van der Vorst, “A parallel linear system solver for circuit simulation problems,” *Numerical Linear Algebra with Applications*, 2000.

[20] (2010) Accelerating analog simulation with hspice precision parallel technology. White Paper. Synopsys.

[21] J. R. Gilbert and T. Peierls, “Sparse partial pivoting in time proportional to arithmetic operations,” *SIAM J. Sci. Statist. Comput.*, vol. 9, pp. 862 – 874, 1988.

[22] T. A. Davis and Y. Hu, “University of florida sparse matrix collection,” *ACM Transactions on Mathematical Software (to appear)*.

[23] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, “A supernodal approach to sparse partial pivoting,” *SIAM J. Matrix Analysis and Applications*, vol. 20, no. 3, pp. 720–755, 1999.

[24] X. S. Li, “An overview of SuperLU: Algorithms, implementation, and user interface,” vol. 31, no. 3, pp. 302–325, September 2005.

[25] R. Telichevesky, K. Kundert, and J. White, “Efficient steady-state analysis based on matrix-free Krylov-subspace methods,” 1995.

[26] X.-X. Liu, H. Yu, and S.-D. Tan, “A robust periodic arnoldi shooting algorithm for efficient analysis of large-scale rf/mm ics,” in *47th ACM/IEEE Design Automation Conference (DAC)*, june 2010, pp. 573 –578.

Wei Wu (S’10) obtained his B.S. degree and M.S. degree of electrical engineering from Beihang Uniersity (Beijing, China) in 2007 and 2010 respectively. He is currently a PhD graduate student in Electrical Engineering department, University of California, Los Angeles (UCLA). His research interests include parallel/reconfigurable computing, and their applications in computer aided design (CAD) algorithms.

Fang Gong (S’08) received his B.S. degree from Computer Science Department at Beihang Uniersity in 2005. Also, he graduated from Computer Science Department at Tsinghua University with M.S. degree in 2008. After that, he continue to work toward his Ph.D. degree in the Electrical Engineering Department at University of California, Los Angeles. His research interests mainly focus on numerical computing and stochastic techniques for CAD, including fast circuit simulation, yield estimation and optimization. He also works on numerics parallel and distributed computing.

Rahul Krishnan (S’12) obtained his B.S. degree in Electrical Engineering from the University of California, Los Angeles (UCLA) in 2011. He is currently a Master's graduate student in the Electrical Engineering department at UCLA. His research interests include stochastic estimation for CAD, Smart Grid, and battery modeling for electric vehicles (EV).

Hao Yu (S’02–M’06) obtained his B.S. degree from Fudan University (Shanghai China) in 1999. and obtained M.S./Ph. D degrees both from electrical engineering department at UCLA in 2007, with major of the integrated circuit and embedded computing. He was a senior research staff at Berkeley Design Automation (BDA) since 2006, one of top-100 start-ups selected by Red-herrings at Silicon Valley. Since October 2009, he is an assistant professor at circuits and systems division of electrical and electronic engineering school, Nanyang Technological University (NTU), Singapore. His research interests include 3D IC system design, analog/RF circuit design, RF circuit simulation algorithms.

Lei He (M’99–SM’08) is a professor at electrical engineering department, University of California, Los Angeles (UCLA) and was a faculty member at University of Wisconsin, Madison between 1999 and 2002. He also held visiting or consulting positions with Cadence, Empyrean Soft, Hewlett-Package, Intel, and Synopsys, and was technical advisory board member for Apache Design Solutions and Rio Design Automation. Dr. He obtained Ph.D. degree in computer science from UCLA in 1999. His research interests include modeling and simulation, VLSI circuits and systems, and cyber physical systems.