GPU-Accelerated Parallel Monte Carlo Analysis of Analog Circuits by Hierarchical Graph-based Solver

Yan Zhu and Sheldon X.-D. Tan
Department of Electrical Engineering, University of California, Riverside, CA 92521, USA

***Abstract*—In this article, we propose a new parallel matrix solver, which is very amenable for Graphic Process Unit (GPU) based fine-grain massively-threaded parallel computing. The new method is based on the graph-based symbolic analysis technique to generate the computing sequence of determinants in terms of determinant decision diagrams (DDDs). DDD represents very simple data dependence and data parallelism, which can be explored much easier by GPU massively-threaded parallel computing than existing LU-based methods. The new method is based on the hierarchical determinant decision diagrams (HDDDs). Inspired by the inherent data parallelism and simple data dependence in the evaluation process of HDDD, we design GPU-amenable continuous data structures to enable fast memory access and evaluation of massive parallel threads. In addition to parallelism in DDD graph, the new algorithm can naturally explore data independence existing in Monte Carlo and frequency domain analysis. The resulting algorithm is a general-purpose matrix solver suitable for fine-grain massive GPU-based computing for any circuit matrices. Experimental results show that the new evaluation algorithm can achieve about two orders of magnitude speedup over the serial CPU based evaluation and more than 4× speedup over numerical SPICE-based simulation method on some large analog circuits.**

1. Introduction

As the VLSI technology marches into nanometer scale, the real circuit parameters will differ from what they are designed to be due to process variations. This situation becomes worse as technology continues to scale to 90 nm and below owing to the increasing process-induced variability [1], [2]. For example, due to an inverse-square-root-law dependence with the transistor area, the mismatch of CMOS devices nearly doubles for each process generation less than 90 nm [3], [4]. To consider the impacts of process variations on circuit performance, Monte-Carlo based statistical approach is the most reliable solutions to this problem. But the prohibitive computational costs of Monte Carlo method prevents it from solving large analog circuits.

Modern computer architecture has shifted towards designs that employ multiple processor cores on a chip, so called multi-core processor [5], [6]. The graphic processing unit (GPU) is one of the most powerful many-core computing systems in mass-market use. For instance, the state-of-the-art NVIDIA Kepler K40 GPU with 2880 cores has a peak performance of over 4 TFLOPS versus about 80–100 GFLOPS of Intel i7 series Quad-core CPUs [7], [8]. In addition to the primary use of GPUs in accelerating graphics rendering operations, there has been considerable interest in exploiting GPUs for general purpose computation (GPGPU) [9]. Meanwhile, the introduction of new parallel programming interfaces for general purpose computations, such as Compute Unified Device Architecture (CUDA) [10], Stream SDK, and emerging OpenCL [10], [11], [12], and especially emerging OpenACC [13] compilers, have made GPUs powerful and attractive choice for solving large engineering problems.

This research was supported in part by NSF grants under No. CCF-1116882, No. CCF-1017090, No. OISE-1130402.

Parallelization on GPU platforms is an emerging strategy to improve the efficiency of general analysis techniques. Within the past several years, a number of GPU-based parallel circuit simulation techniques have been proposed. They include on-chip power grid analysis methods [14], [15], [16], [17], [18], fast thermal analysis [19], logic simulation [20] and general SPICE simulation [21], [22], where only the device model evaluation has been parallelized on GPUs. Traditional SPICE-like numerical simulators based on sparse LU decomposition turn out difficult to be parallelized on GPUs due to inherent data dependence and irregular memory access. A GPU-based sparse LU solver based on the G/P left-looking algorithm [23] has been proposed, which shows the limited benefits on GPU platforms for sparse LU factorization. Recently a GPU-based statistical analysis method based on the DDD structure has been proposed with promising results [24]. But this method cannot handle large analog circuits as it uses the flat-DDD structure.

In this article, we propose a new parallel matrix solver, which is very amenable for Graphic Process Unit (GPU) based fine-grain parallel computing. The new method is based on the graph-based symbolic analysis technique to determine the computing sequence of determinants in terms of determinant decision diagrams (DDDs). DDD represents very simple data dependence and data parallelism, which can be explored much easier by GPU massively-threaded parallel computing than existing LU-based methods. The new method is based on the hierarchical determinant decision diagrams (HDDDs), which essentially allows the analysis of arbitrarily large circuit. We design novel data structures to represent the HDDD graphs in the GPUs to enable fast memory access of massive parallel threads for computing the numerical values of DDD graphs. In addition to parallelism in DDD graph, the new algorithm can naturally explore parallelism existing in Monte Carlo and frequency domain analysis. The resulting algorithm is a general-purpose matrix solver suitable for fine-grain massively-threaded parallel computing for any circuit matrices. Experimental results show that the new evaluation algorithm can achieve about one to two orders of magnitude speedup over the serial CPU based evaluations and more than 4× speedup over numerical SPICE-based simulation method on some large analog circuits with some examples giving 25× speedup.

2. Review of hierarchical DDD (HDDD)-based symbolic analysis

The new parallel direct solver is based on a hierarchical DDD graph-based technique [25]. The DDD technique employs directed binary decision diagram to represent a determinant where the paths in the graph represent the product terms from the determinant. By exploiting the sparsity of circuit matrices and the sharing within symbolic expressions in a systematic manner, DDD enables symbolic analysis of

978-1-4799-7792-5/15/$31.00 ©2015 IEEE

much larger analog circuits than previous symbolic methods. However, the capability of DDD was limited, since the number of nodes in the DDD data structure grows exponentially with the size of a circuit. This bottleneck was resolved by the introduction of HDDD technique [26], which highly reduced the size of DDD by dividing a circuit in different levels.

Here we give a brief review of the HDDD technique. The system equation of a linearized time-invariant analog circuit can be represented by the modified nodal analysis approach in the following general form:

$$Ax = b \tag{1}$$

where A represents the circuit matrix, x is the vector of circuit unknown variables—node voltage and branch current, and b is the vector of external sources. Now we divide the circuit into two parts—a subcircuit and the rest of the circuit.

Then (1) can be rewritten in the following form:

$$\begin{bmatrix} A^{II} & A^{IB} & \\ A^{BI} & A^{BB} & A^{BR} \\ & A^{RB} & A^{RR} \end{bmatrix} \begin{bmatrix} X^I \\ X^B \\ X^R \end{bmatrix} = \begin{bmatrix} B^I \\ B^B \\ B^R \end{bmatrix} \tag{2}$$

Here, the circuit unknown variable x is divided into three disjoint sets—X^I, X^B, and X^R, where the sup-scripts I, B, R represent, respectively, internal variables, boundary variables and the rest variables. The circuit matrix A and the external source vector b are divided accordingly. The internal variables are the node voltages and branch currents of the subcircuit, the boundary variables are those shared by the subcircuit and the rest of the circuit, and the rest variables are only related to the rest of the circuit. Using Schur decomposition to eliminate internal variables X^I, we can transform (2) to:

$$\begin{bmatrix} A^{BB*} & A^{BR} \\ A^{RB} & A^{RR} \end{bmatrix} \begin{bmatrix} X^B \\ X^R \end{bmatrix} = \begin{bmatrix} B^{B*} \\ B^R \end{bmatrix}, \tag{3}$$

where

$$A^{BB*} = A^{BB} - A^{BI}(A^{II})^{-1}A^{IB}, \tag{4}$$

and

$$B^{B*} = B^B - A^{BI}(A^{II})^{-1}B^I. \tag{5}$$

Suppose the length of X^B is m and the length of X^R is l, then elements in A^{BB*} can be expanded as:

$$a^{BB*}_{u,v} = a^{BB}_{u,v} - \frac{1}{det(A^{II})} \sum_{k_1,k_2=1}^{m} a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} a^{IB}_{k_2,v} \tag{6}$$

where $u, v = 1, ..., l$ and $\Delta^{II}_{k_2,k_1} = (-1)^{k_2+k_1} det(A_{k_2,k_1})$.

We call $a^{BB*}_{u,v}$ a *compound* element, which can be obtained by the reduction of an original part $a^{BB}_{u,v}$ and a composite part $\frac{1}{det(A^{II})} \sum_{k_1,k_2=1}^{m} a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} a^{IB}_{k_2,v}$. The determinant $\Delta^{II}_{k_2,k_1}$ within a nonzero term $a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} a^{IB}_{k_2,v}$ is called a *valid cofactor* of the subcircuit. $det(A^{II})$ is called the *system determinant* of the subcircuit.

Similarly, elements in B^{B*} can be expanded as

$$b^{B*}_u = b^B_u - \frac{1}{det(A^{II})} \sum_{k_1,k_2=1}^{m} a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} b^I_{k_2}, \tag{7}$$

where $u = 1, ..., l$. The foregoing process is called subcircuit suppression. Given a practical circuit and a good partition of it, A^{BI} and A^{IB} are usually very sparse and l is usually much smaller than m. This implies that only a few nonzero terms of $a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} a^{IB}_{k_2,v}$ and $a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} b^I_{k_2}$ will be generated, thus the efficiency of HDDD technique is guaranteed. (6) and (7) show that to decide the value of an element in the suppressed MNA, we need to first evaluate the system determinant $det(A^{II})$ and the valid cofactors $\Delta^{II}_{k_2,k_1}$. According to [25], every determinant can be represented by a DDD. Since all these DDDs are related to the same subcircuit, there are a lot of internal sharing among the DDDs. As a result, we can represent all the determinants of the subcircuit in a compact data structure called multi-root DDD.

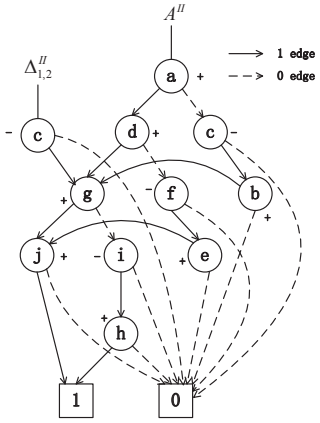

Fig. 1. Multi-Root DDD representation for A^{II} and $\Delta^{II}_{1,2}$

For example, assume that $A^{II} = \begin{bmatrix} a & b & 0 & 0 \\ c & d & e & 0 \\ 0 & f & g & h \\ 0 & 0 & i & j \end{bmatrix}$, $A^{IB} = \begin{bmatrix} k \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $A^{BI} = [\ 0 \quad n \quad 0 \quad 0\ ]$, $A^{BB} = [p]$, $A^{BR} = [q]$, $A^{RB} = [r]$, and $A^{RR} = [w]$. Then the system determinant of the subcircuit is

$$det(A^{II}) = \begin{vmatrix} a & b & 0 & 0 \\ c & d & e & 0 \\ 0 & f & g & h \\ 0 & 0 & i & j \end{vmatrix} \tag{8}$$

and there is only one valid cofactor, which is

$$\Delta^{II}_{1,2} = (-1) det(A^{II}_{1,2}) = (-1) \begin{vmatrix} c & e & 0 \\ 0 & g & h \\ 0 & i & j \end{vmatrix}. \tag{9}$$

Now a multi-root DDD for $det(A^{II})$ and $\Delta^{II}_{1,2}$ shown in Fig. 1 is constructed. The value of *1-terminal* is 1, and that of *0-terminal* is 0. Each node a_i in the figure represents a symbolic expression $D(a_i)$ defined recursively as:

$$D(a_i) = a_i \cdot sign(a_i) \cdot D_{a_i} + D_{\overline{a}_i}, \tag{10}$$

where D_{a_i} and $D_{\overline{a}_i}$ represent, respectively, the symbolic expressions of the nodes pointed by the *1-edge* and *0-edge* of a_i. According to this recursive relationship, we can get the value of each root by traversing all the nodes of the multi-root DDD in a depth-first manner.

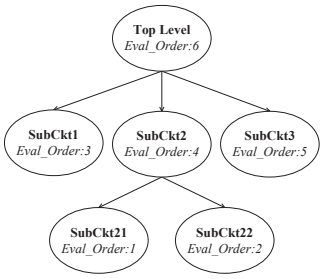

Fig. 2. Example of a circuit hierarchy

For a large circuit hierarchy with more than one subcircuit, e.g. Fig. 2, the circuit suppression process is performed recursively by visiting the circuit hierarchy in a bottom-up fashion. Note that if any subcircuit has two or more children overlapping in the boundry, Eq.(6) need to be modified to following general form:

$$a_{u,v}^{BB*} = a_{u,v}^{BB} - \sum_{p=1}^{n}(\frac{1}{det(A^{II_p})} \sum_{k_1,k_2=1}^{m_p} a_{u,k_1}^{BI_p} \Delta_{k_2,k_1}^{II_p} a_{k_2,v}^{IB_p}) \quad (11)$$

where n is the number of children subcircuits that contribute to the composite part of $a_{u,v}^{BB*}$. Values with subscript "p" correspond to the pth child. Eq.(7) can be updated analogously. The rest of the paper bases on this general form.

3. The proposed HDDD-based parallel Monte Carlo analysis method

In this section, we first provide an overview of how the previous hierarchy DDD evaluation process can be parallelized for Monte Carlo analysis of analog circuits, and how to generate data structures amenable for GPU. A detailed implementation on GPU will be shown in the next section.

In HDDD-based analysis, the circuit is represented in hierarchy. To evaluate the whole circuit, we go through the circuit hierarchy in a bottom-up fashion, as indicated by *Eval_Order* in Fig.2. Each subcircuit in the hierarchy is represented by a multi-root DDD. To evaluate it, first we need to calculate the numerical value of the corresponding MNA elements. The original part of an MNA element is based on device value of the current subcircuit. The composite part can be obtained by subsituting the evaluation result of the children subcircuits' multi-root DDDs to (11). Then, the MNA element value is mapped to the multi-root DDD of the current subcircuit and a depth-first traversal is performed.

Now consider the evaluation of a multi-root DDD at a single frequency in a single Monte Carlo run. The data dependency within a multi-root DDD is very simple: a node can be evaluated when the value of all its children is obtained. If several nodes satisfy this condition at the same time, we call that they are at the same *level*. The level of a DDD node a_i can be determined as:

$$L(a_i) = max(L_{a_i}, L_{\overline{a}_i}) + 1. \quad (12)$$

where L_{a_i} and $L_{\overline{a}_i}$ represent, respectively, the level of the nodes pointed by the *1-edge* and *0-edge* of a_i. Specifically, we define the level of both *1-terminal* and *0-terminal* as -1. The parallel scheme here is simple: nodes at the same level can be evaluated at the same time. And the evaluation process goes from the lowest level to the highest level.

MNA element index	0	1	2	3	4	5	6	7	8	9
MNA element value	a	b	c	d	e	f	g	h	i	j

Node index	0	1	2	3	4	5	6	7	8	9	10
MNA element	2(c)	6(g)	9(j)	8(i)	7(h)	0(a)	3(d)	5(f)	4(e)	2(c)	1(b)
Left child index	1	2	-1	4	-1	6	1	8	2	10	1
Right child index	-2	3	-2	-2	-2	9	7	-2	-2	-2	-2
Sign	-1	1	1	-1	1	1	1	-1	1	-1	1
Layer	3	2	0	1	0	5	3	2	1	4	3

Fig. 3. Linear storage of multi-root DDD information

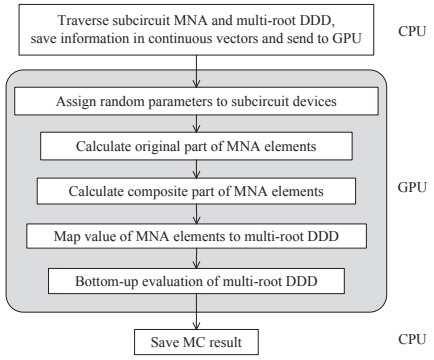

Fig. 4. The flow of GPU-based Monte Carlo analysis for a subcircuit

To make the evaluation process amenable to GPU, first we need to store information of both subcircuit MNA and the graph-based multi-root DDD into one-dimensional linearized vectors. Fig. 3 shows the linearized vectors for the example in Fig. 1. First a specific index is assigned to each MNA element, and the value is stored in a vector. Then the multi-root DDD is traversed, and each node in it is also assigned a unique index. Specifically, we define the index of *1-terminal* as -1, and that of *0-terminal* as -2. The left child index and the right child index are, respectively, the indexes of the nodes pointed by the its *1-edge* and *0-edge*. Besides, the sign and level of each node are also stored in linear vectors.

After the linearized vectors are transferred from CPU to GPU, the parallel statistical evaluation process of the corresponding subcircuit starts, as Fig. 4 shows.

When the process finishes, the results are copied from GPU to host memory. These results are later used in (11) by upper-level subcircuits.

4. Implementation of HDDD-based parallel Monte Carlo analysis on GPU

In this section, we explain in detail the parallel Monte Carlo evaluation process within each subcircuit, as indicated by GPU part in Fig. 4. The largest grid size in the latest NVIDIA Tesla K40 is $2G \times 64K \times 64K$ blocks, and each block can have as many as 1024 threads. The following implementation conforms to these constraints.

A. Random number assignment on MNA elements

The first key process for HDDD-based Monte Carlo analysis is to generate random device values according to the user-specified distribution in SPICE netlist. On GPU, we use the CURAND library to generate small deviations to nominal device values, so that we can obtain 1600 different Monte Carlo samples to simulate the effects of process variation. The

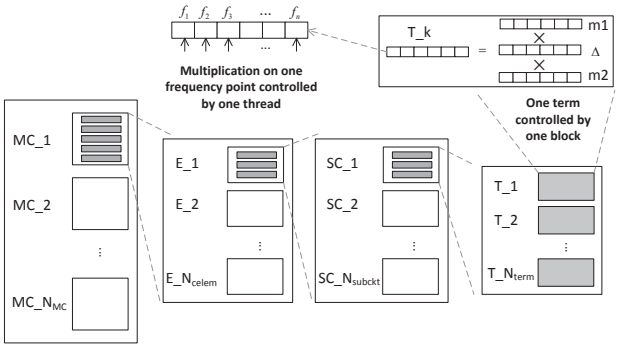

Fig. 5. First step of composite part calculation

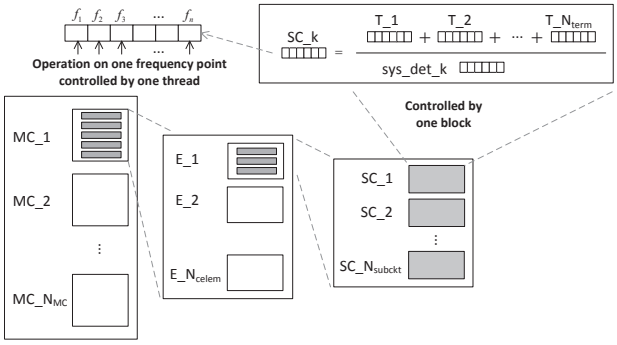

Fig. 6. Second step of composite part calculation

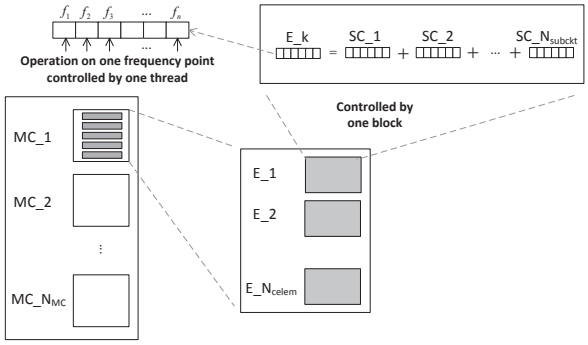

Fig. 7. Third step of composite part calculation

stamping pattern from device value to MNA element is stored in texture memory, so that fast memory access is enabled.

B. MNA element parallel computing in HDDD

The second key process in HDDD-based analysis is to compute the values of MNA elements. We calculate original part and the composite part of each element separately.

After the previous random number assignment, the complex value of all the MNA elements's original part at different frequencies and different MC runs can be evaluated. This part is relatively easy, and the details can be referred to [24]. Then the GPU starts calculating the composite part. Note that not all the elements are compound elements. Only those with a non-zero composite part need to be calculated.

According to Eq.(11), the composite part can be calculated in three steps. First, each valid term is obtained by multiplying the three values in $a_{u,k_1}^{BI_p}\Delta_{k_2,k_1}^{II_p}a_{k_2,v}^{IB_p}$. Second, the contribution of the pth child is obtained by adding up all the valid terms related to it and dividing the sum by its system matrix. Third, the contribution from all n children are summed up.

The first step has five levels of parallelism in total, as shown in Fig. 5. N_{MC} is the number of MC runs, N_{celem} is the number of compound elements that need to be calculated, N_{subckt} is the maximum number of children subcircuits contributing to a composite part, N_{term} is the maximum number of valid terms generated by a child subcircuit, and n is the number of frequency points in analysis. $m1$, Δ and $m2$ represent, respectively, $a_{u,k_1}^{BI_p}$, $\Delta_{k_2,k_1}^{II_p}$ and $a_{k_2,v}^{IB_p}$ in Eq.(11). They are evaluation results of the children subcircuits. On GPU, we launch a two-dimensional grid with a total number of $N_{MC} \times N_{celem} \times N_{subckt} \times N_{term}$ blocks. This number would not be very large given a good circuit partition with a relatively small number of boundary nodes. For each block within the grid, we launch 64 threads to evaluate different frequency points in parallel, which practically achieves the best performance.

The second step includes four levels of parallelism. Fig. 6 indicates that we can parallelize the evaluation of different MC samples, different elements, different children's contribution in the composite part and different frequency points. "sys_det" is system determinant of the corresponding child subcircuit. On GPU, we launch $N_{MC} \times N_{celem} \times N_{subckt}$ blocks and 64 threads. The threads add up all the valid terms of the corresponding subcircuit, then divide the sum by the system determinant at different frequency points.

In the third step, parallelism is reflected in three levels. As shown in Fig. 7, we launch $N_{MC} \times N_{celem}$ blocks and 64 threads within each block. Each thread sums up the contribution of all the subcircuits to the corresponding element at one frequency point.

After the three steps, we configure GPU as a $N_{MC} \times N_{elem}$ grid. N_{elem} here is the total number of none-zero MNA elements. Each block in this grid contains $N_{threads}$ threads. If the MNA element is compound, the threads reduce its original part by its composite part.

C. Evaluation of multi-root DDDs

The third key process for HDDD-based analysis is to evaluate the multi-root DDD of the current subcircuit. Fig. 8 shows the evaluation process of the multi-root DDD in Fig. 1. The GPU block number is $N_{MC} \times N_{nodes}$, i.e., number of MC runs times number of nodes in multi-root DDD, and there are 64 threads per block. The layer-by-layer iteration process is

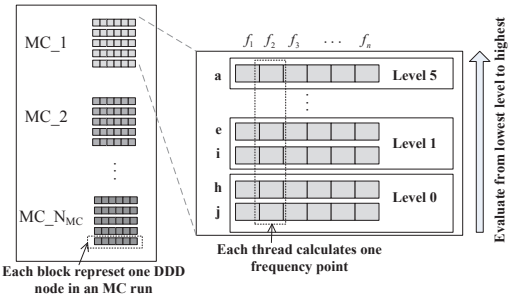

Fig. 8. Level-wise evaluation of the Multi-Root DDD in Fig. 1

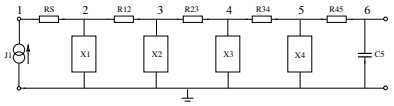

Fig. 9. A low pass filter

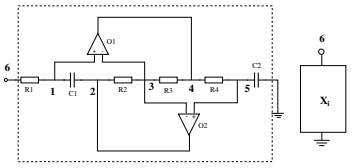

Fig. 10. The schematic of each block in the low pass filter

controlled by CPU due to data dependency. At each iteration, one layer of nodes are evaluated. An active thread first reads value of a node's corresponding MNA element, then evaluates the node according to (10) and information in Fig. 3.

If the number of Monte-Carlo runs or the number of frequency points exceeds the limits of the GPU, the process is repeated until all MC samples and frequency points are evaluated. During the whole calculation process, coalesced memory access is enabled, so the threads in a warp always execute the same code path. As a consequence, the GPU kernel launching always exhibits a highly data intensive pattern, and the global memory traffic is efficiently reduced.

5. Experimental results and discussions

In this section, the performance of the proposed GPU solver for Monte Carlo Analysis is discussed. We test the program on several benchmark circuits, and compare its runtime with two versions of solvers—the CPU version of HDDD and HSPICE. All of our programs are implemented in C++, and the GPU computation part is implemented with NVIDIA CUDA. The programs run on a Linux server with 2.60 GHz Intel Xeon E2670 CPU and 64 GB memory. The GPU card used is Tesla K40 with 2880 CUDA cores and 12 GB global memory.

Our benchmarks include two sets of filter circuits and a μA741 amplifier. The first set of filter circuit is shown in Fig. 9 and subcircuit of each block is shown in Fig. 10. The second set is shown in Fig. 11. The subcircuit of each block is shown in Fig. 12.

For the two sets of filters, the opamps at bottom level can be configured as either the simple linear opamp in Fig. 13 or the Miller opamp in Fig. 14.

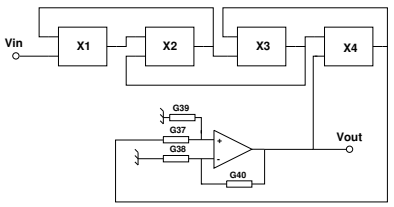

Fig. 11. A band pass filter

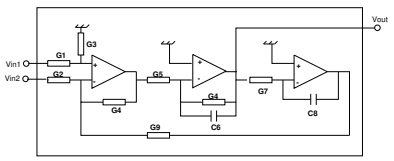

Fig. 12. The subcircuit of the band pass filter

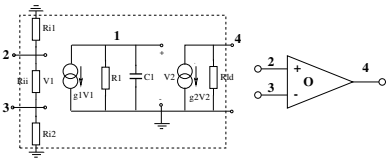

Fig. 13. A linear opamp

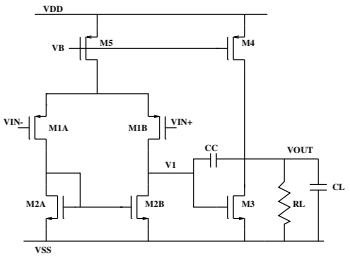

Fig. 14. A Miller Opamp

Besides the two sets of filters, the two-way partitioned μA741 circuit in Fig. 15 is also used as benchmark. The small-signal model of each bipolar is shown in Fig. 16.

The runtime information of all the benchmarks for 1600 Monte Carlo runs is listed in Table I. "lp_lin" refers to the low pass filter configured with the linear opamp in Fig. 13. "lp_pino" refers to the low pass filter configured with the Miller opamp in Fig. 14. "bp_lin" and "bp_pino" refer to the band pass filter configured with linear opamps and Miller opamps respectively. "μA741" refers to the circuit in Fig. 15. Although the test circuits are complex, we can see from the second column that their HDDDs are not very large, which is the major advantage of HDDD. The last three columns represent, respectively, the runtime of the proposed GPU solver, the CPU version of the solver and HSPICE. It is clear that parallel solver outperforms its CPU counterpart by two orders of magnitude, and the speedup over commercial HSPICE is more than 4×. In several examples, the speedup over HSPICE even exceeds 25×. We remark that HSPICE is an optimally implemented product-quality simulator, while

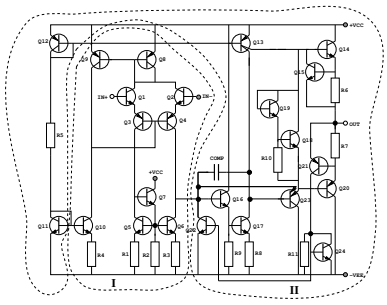

Fig. 15. The circuit schematic of μA741

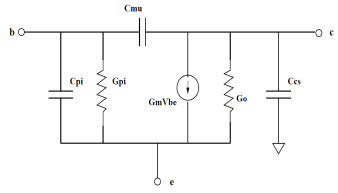

Fig. 16. μA741 bipolar small signal model

TABLE I
PERFORMANCE COMPARISON OF GPU, CPU AND HSPICE ON MONTE CARLO ANALYSIS

circuit name	# freq points	# DDD nodes	GPU time(s)	CPU time(s)	HSPICE(s)
lp_lin	512	182	0.97	96.2	69.5
lp_pino	512	454	4.6	256.4	108.1
bp_lin	832	173	6.17	272.1	28.4
bp_pino	832	251	9.49	336.8	62.7
μA741	640	832	3.81	448.1	14.6

TABLE II
PERFORMANCE VARIATIONS OF THE GPU SOLVER GIVEN DIFFERENT CIRCUIT PARTITIONS

partition scheme	# HDDD nodes	MNA evaluation time(s)	HDDD evaluation time(s)	total GPU time(s)
partition 1	832	2.71	1.12	3.81
partition 2	950	0.81	1.40	2.21
partition 3	1001	0.91	1.39	2.30
partition 4	1720	0.05	3.06	3.11

our solver is less optimized and has more potentials for further improvement.

In our current implementation, the circuits are partitioned beforehand. However, the GPU time in Table I can be impacted by different circuit partition schemes. Table II shows a performance comparison given four different partitions of the μA741 circuit in Fig. 15. Partition 1 is a two-level circuit hierarchy and treats block I and II as children of the top level circuit. The last line of Table I is based on this partition. Partition 2 treats block II as the top level circuit and block I as its child. Partition 3 treats block II as child of block I. Partition 4 is a flat circuit, combining block I and II in one level. We can see that partition 2 is the fastest among the four and can achieve about $2\times$ speedup over partition 1. From column 3 through column 5 of Table II, we can see the trade-off between MNA evaluation time and HDDD evaluation time and how that impact the total performance. As circuit hierarchy gets simpler, evaluation time of HDDD increases since there are more nodes in HDDD. On the other hand, the MNA element evaluation time reduces since less compound elements are generated. So to achieve a better performance, we need to strike a good balance between the two. Table II indicates that balancing the size of each subcircuit in the circuit hierarchy would be beneficial. It is also helpful to reduce the number of boundary nodes. Given that the not-so-good partition 1 already achieves more than 4X speedup, there are good reasons to believe that the proposed method will show even better performance if more intelligent partitioning schemes are developed in the future.

6. Conclusion

A new parallel matrix solver amenable for Graphic Process Unit (GPU) based fine-grain massively-threaded parallel computing is proposed. The new method is based on the hierarchical determinant decision diagram, which is capable of representing very large scale analog integrated circuits symbolically with low complexity. Inspired by the inherent data parallelism and simple data dependence in the evaluation process of HDDD, a set of GPU-amenable continuous data structures are designed to enable fast memory access and evaluation of massive parallel threads. Experimental results show that the new evaluation algorithm can achieve about two orders of magnitude speedup over the serial CPU based evaluations and more than $4\times$ speedup over numerical SPICE-based simulation method on some large analog circuits.

References

[1] R. Rutenbar, "Next-generation design and EDA challenges," in *Proc. Asia South Pacific Design Automation Conf. (ASPDAC)*, January 2007. Keynote speech.

[2] S. Nassif, "Model to hardware correlation for nm-scale technologies," in *Proc. IEEE International Workshop on Behavioral Modeling and Simulation (BMAS)*, Sept 2007. Keynote speech.

[3] H. Masuda, S. Ohkawa, A. Kurokawa, and M. Aoki, "Challenge: variability characterization and modeling for 65- to 90-nm processes," in *Custom Integrated Circuits Conference, 2005. Proceedings of the IEEE 2005*, pp. 593–599, Sept 2005.

[4] J. Kim, K. Jones, and M. Horowitz, "Fast, non-monte-carlo estimation of transient performance variation due to device mismatch," in *Proc. IEEE/ACM Design Automation Conference (DAC)*, 2007.

[5] Intel Corporation, "Intel multi-core processors, making the move to quad-core and beyond (White Paper)," 2006. http://www.intel.com/multi-core.

[6] AMD Inc., "Multi-core processors—the next evolution in computing (White Paper)," 2006. http://multicore.amd.com.

[7] D. B. Kirk and W.-M. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach, 2ed*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2013.

[8] "NVIDIA Tesla's Servers and Workstations." http://www.nvidia.com/object/tesla-servers.html.

[9] D. Göddeke, "General-purpose computation using graphics harware." http://www.gpgpu.org/, 2011.

[10] NVIDIA Corporation, "CUDA (Compute Unified Device Architecture)," 2011. http://www.nvidia.com/object/cuda_home.html.

[11] AMD Inc., "AMD Steam SDK." http://developer.amd.com/gpu/ATIStreamSDK, 2011.

[12] Khronos Group, "Open Computing Language (OpenCL)." http://www.khronos.org/opencl, 2011.

[13] "Openacc directives for accelerators." http://openacc-standard.org.

[14] 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 *Proc. Design Automation Conf. (DAC)*, pp. 178–183, July 2009.

[15] Z. Feng and Z. Zeng, "Parallel multigrid preconditioning on graphics processing units (GPUs) for robust power grid analysis," in *Proc. Design Automation Conf. (DAC)*, pp. 661–666, 2010.

[16] Z. Feng, Z. Zeng, and P. Li, "Parallel On-Chip Power Distribution Network Analysis on Multi-Core-Multi-GPU Platforms," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 10, pp. 1823–1836, 2011.

[17] X. Liu, S. X.-D. Tan, Z. Liu, H. Wang, and T. Xu, "Transient analysis of large linear dynamic networks on hybrid gpu-multicore platforms," in *10th IEEE International NEWCAS Conference*, pp. 173–176, June 2012.

[18] X. Liu, H. Wang, and S. X.-D. Tan, "Parallel power grid analysis using preconditioned gmres solvers on cpu-gpu platforms," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 561–568, Nov. 2013.

[19] X. Liu, Z. Liu, S. X.-D. Tan, and J. Gordon, "Full-chip thermal analysis of 3D ICs with liquid cooling by GPU-accelerated gmres method," in *Proc. Int. Symposium. on Quality Electronic Design (ISQED)*, pp. 352–357, March 2012.

[20] B. Wang, Y. Zhu, and Y. Deng, "Distributed time, conservative parallel logic simulation on GPUs," in *Proc. Design Automation Conf. (DAC)*, pp. 761–766, 2010.

[21] K. Gulati and S. P. Khatri, *Hardware Acceleration of EDA Algorithms*. Springer, 2010.

[22] K. Gulati, J. F. Croix, S. P. Khatr, and R. Shastry, "Fast circuit simulation on graphics processing units," in *Proc. Asia South Pacific Design Automation Conf. (ASPDAC)*, pp. 403–408, 2009.

[23] L. Ren, X. Chen, Y. Wang, C. Zhang, and H. Yang, "Sparse LU factorization for parallel circuit simulation on GPU," in *Proc. Design Automation Conf. (DAC)*, pp. 1125–1130, 2012.

[24] X.-X. Liu, S. X.-D. Tan, and H. Wang, "Parallel statistical analysis of analog circuits by gpu-accelerated graph-based approach," in *Proc. Design, Automation and Test In Europe. (DATE)*, pp. 851–857, March 2012.

[25] C.-J. Shi and X.-D. Tan, "Canonical symbolic analysis of large analog circuits with determinant decision diagrams," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 1–18, Jan. 2000.

[26] X.-D. Tan and C.-J. Shi, "Hierarchical symbolic analysis of large analog circuits via determinant decision diagrams," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 401–412, April 2000.