# RAG: An Efficient Reliability Analysis of Logic Circuits on Graphics Processing Units

Min Li and Michael S. Hsiao
Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24061, USA
{minli, mhsiao}@vt.edu

***Abstract*—In this paper, we present RAG, an efficient Reliability Analysis tool based on Graphics processing units (GPU). RAG is a fault injection based parallel stochastic simulator implemented on a state-of-the-art GPU. A two-stage simulation framework is proposed to exploit the high computation efficiency of GPUs. Experimental results demonstrate the accuracy and performance of RAG. An average speedup of 412× and 198× is achieved compared to two state-of-the-art CPU-based approaches for reliability analysis.**

## I. INTRODUCTION

Reliability analysis, a process of evaluating the effects of errors due to both intrinsic noise and external transients will play an important role for both today's and tomorrow's nanometer-scale circuits. The probability of error due to manufacturing defects, process variation, aging and transient faults is believed to sharply increase due to rapidly diminishing feature sizes and complex fabrication processes [1].

Due to the exponential number of input combinations and the difficulty to model gate failures, the reliability analysis of logic circuits is computationally complex. Exact analysis methods, using probabilistic transfer matrices (PTMs) [2, 3] and probabilistic decision diagrams (PDDs) [4], can provide accurate reliability evaluation for small circuits. Several analysis heuristics, on the other hand, have been proposed to generate a highly accurate reliability estimation, such as probabilistic gate models (PGMs) [5] and Bayesian networks [6]. However, they all suffer from the problem of exponential computational complexity and are therefore practically infeasible for even mid-size circuits (with more than ten thousand gates). A simulation scheme based on stochastic computational models (SCMs) is proposed in [7]. Although the approach scales linearly, it requires long runtimes on large circuits to achieve an accurate estimation.

The graphics processing unit (GPU), a highly parallel, multi-threaded and many-core processor, has become a popular cost-effective parallel platform nowadays. Thousands of fine-grained threads on GPUs can be launched concurrently, with each thread executing the same kernel program but on a different data set. With the introduction of general-purpose computing on GPU (GPGPU) based on NVidia's compute unified device architecture (CUDA) [8], GPUs are now exploited to accelerate the time-consuming electronic design automation (EDA) algorithms[9–12].

In this paper, we present RAG, an efficient Reliability Analysis tool on GPUs. RAG is a fault-injection based parallel stochastic simulator for logic circuits. To the best of our knowledge, this is the first work that accelerates the reliability analysis of logic circuits on a GPU platform. We introduce a two-stage hierarchical simulation framework to efficiently utilize the computation power of the GPU without exceeding its memory limitation. By partitioning the circuit into fanout-free regions (FFRs), all the fanout stems in the circuit will be scheduled and processed in an order that minimizes storage requirements. Within each FFR, all the logic gates are also arranged and simulated in sequence to reduce space. Highly parallel execution is achieved by launching thousands of active threads which evaluate the same logic gate but with different vectors.

This work is supported in part by NSF grant 1016675.


## II. BACKGROUND

### A. Reliability Analysis and Fault Model

The errors that arise due to temporary deviation or malfunction of nano-devices can be characterized as probabilistic errors, each of which is an intrinsic property of each device. Such errors are hard to detect by regular testing methodologies because they may occur anywhere in the circuit and are not permanent defects. Thus, each device $i$ (logic gate or interconnect) will have a certain probability of error $\tau_i \in [0, 0.5]$, which will cause its output to flip symmetrically (from $0 \rightarrow 1$ or $1 \rightarrow 0$). A common fault model of a circuit with $n$ gates is defined as the failure probability $\vec{\tau} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where $\tau_k$ is the failure probabilities of the $k^{th}$ gate. In our experiments, we assign an universal error rate (failure probability) to all the gates for comparison with existing works. However, RAG is able to handle gates with different error rates.

The reliability of combinational logic circuits could be evaluated by several measures. Suppose we have a logic circuit with $m$ outputs. In [5], the probability of errors for each output $i$ is denoted as $\delta_i$. Among the $m$ outputs, the maximum probability of error, $max_{i=1}^{m}(\delta_i)$ is used in [6] and authors in [3, 7] utilized the average reliability value among outputs, $R_{avg}$, which is also the metric we used in our paper.

### B. Previous Work

The authors in [2] employ probability transfer matrices (PTMs) to capture non-deterministic behavior in logic circuits. However, PTMs are not applicable for large benchmarks due to the massive matrix storage and manipulation overhead. Although PTMs are extended to a reliability estimation using input vector sampling in [3], our results show that RAG is more efficient and can handle even bigger circuits with the same accuracy. Three scalable algorithms are proposed in [5], in which the authors made a constraint on the maximum number of gates to be erroneous simultaneously. Also, those algorithms do not scale well for mid-size benchmarks. In [7],

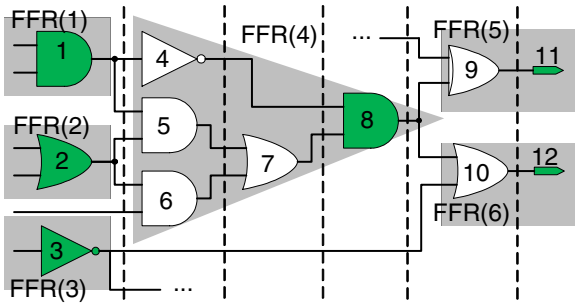


Fig. 1. Fanout Stems and Fanout Free Regions.

a traditional approach to reliability analysis is proposed that employs fault injection and simulation in a Monte Carlo framework. *Only* 1000 samples are simulated which will be shown insufficient in our experiments. Also, substantial simulation overhead is introduced by adding exclusive-or gates for every logic device in their formulation.

Although parallel fault simulation on GPUs have been extensively studied in [10–12], only the single fault model has been explored. In reliability analysis, faults may be excited at multiple gates at the same time, which requires more intensive simulation and much more memory consumption for parallel computing. The GPU-based logic simulation proposed in [9] can be extended using our gate failure model for reliability analysis. However, repeating simulation of duplicated gates between macro-gate introduces extra computations. Also, as a sequential simulator, pattern-parallelism is not employed.

## III. Proposed Method

### A. Framework

The high-level flow of RAG is shown in Figure 2, where the white boxes denote the CPU workload, and the shaded boxes are the GPU workload. At the beginning, the CPU reads in the circuit netlist and generates the list of stems with the corresponding fanout free regions. A post-order tree traversal algorithm is employed to arrange the simulation sequence of gates (non-stem gates) within each FFR. Because we only need to store the stems' values, the memory for the FFRs can be shared, thus saving much space. After that, we formulate another scheduling for the simulation of stems as an optimization problem whose objective is to minimize the memory storage of logic values needed. In other words, we need not allocate memory to store the values for all the stems. Rather, storage of a stem's value can be reused whenever *all* its successors (child stems) have been evaluated. A greedy approach is employed in this scheduling algorithm to help RAG further reduce the memory usage so as to achieve a high bandwidth and full occupancy during simulation. Then, the scheduled circuit netlist is transferred to the GPU. A GPU-based stochastic simulator $k_{sim}$ is executed as Kernel 1 on the GPU. Thousands of threads could be launched with each simulating a total of $p$ random vectors. Since each thread maintains its own error vectors, a parallel reduction kernel $k_{red}$ is launched to compute the total number of errors for each outputs and transfer the summed up $\vec{\epsilon}$ back to the CPU. Finally, RAG will compute the average error probability for each output, $\delta$ and the $R_{avg}$ to give a comprehensive reliability analysis of the logic circuits.

*1) Fault Injection based Stochastic Simulation ($k_{sim}$):* In kernel $k_{sim}$, a total of $t$ threads are launched on the GPU. Each thread will be assigned $p$ vectors and thus runs $p$ iterations to simulate all of them. For each iteration, the scheduled stems and gates within the corresponding FFRs will be processed in sequence. We call the function $eval$ to evaluate both the faulty and fault-free model of the circuit and recorded the value at location $g$ in vectors $\vec{v}$ and $\vec{v^e}$, respectively. Although branches exist for different gate types, no branch divergence is introduced because all the threads within one warp are simulating the same gate simultaneously. The faults are generated by the parallel random number generator according to the error rate vector ($\vec{\tau}$). Faults are injected in by flipping the logic values. At the end of simulation of each vector, the number of errors on each output will be recorded in vector $\vec{\epsilon}$ on the global memory.

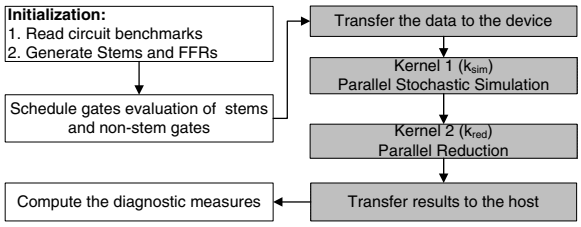


Fig. 2. The framework of RAG.

### B. Scheduling of Stems

*Definition 1:* For a circuit with $n$ stems, represented as $\vec{s} = \{s_1, s_2, \ldots, s_n\}$, we define their corresponding simulation slots as $\vec{a} = \{a_1, a_2, \ldots, a_n\}$, and their storage location on $\vec{v}$ and $\vec{v_e}$ as $\vec{g} = \{g_1, g_2, \ldots, g_n\}$.

First, we show that the ordering of stems can result in different memory footprints. For example, in Figure 1, if all six stems $\{s_1, \ldots, s_6\}$ are simulated in a levelized manner, we have $\vec{a}_1 = \{1, 2, 3, 4, 5, 6\}$. As noted before, the storage of a stem will be released once all its child stems are processed. Therefore, we have $\vec{g}_1 = \{1, 2, 3, 1, 2, 1\}$, $V = 3$. When simulating $s_4$, all its fanin stems $f_1$ and $f_2$ will be free. Therefore, the storage locations 1 and 2 could be released and reused for the following stems. However, if we change the processing sequence to $\vec{a}_2 = \{1, 2, 4, 3, 6, 5\}$, we have the storage locations of the stems as $\vec{g}_2 = \{1, 2, 1, 2, 2, 1\}$, $V = 2$, in which we only need 2 storage spaces for simulating these six stems instead of 3. Therefore, a different ordering might result in a different storage requirement.

We propose a greedy algorithm to schedule the stems and reduce the storage requirement. A sorted list is maintained which includes all the stems that are ready to be processed. The list is kept sorted in terms of the key, $free$. For each stem $s$, $free[s]$ represents the number of storage spaces that could be released after $s$ being added to the simulation queue. As it is greedy, each iteration will pick the stem with the highest $free$ value from $ready$ and add it to the simulation queue. By applying the algorithm, Table I ($Stem$ columns) show that the space requirement for stems is reduced by 3× to 6× against the original one without scheduling.

### C. Scheduling of Non-stem Gates

Since the gates in each FFR are simulated independently, the low-latency on-chip memory called shared memory can be re-used for each FFR which makes it ideal for storing the values of these gates. However, because only 16 KB shared memory is available on each multiprocessor, we need to reduce the maximum memory usage among the FFRs.

TABLE I
CIRCUIT CHARACTERISTICS

| Circuit | # Gates | #in | #out | $S_{FFR}$ | | $S_{stem}$ | | $Mem_s$ (KB) | | $Mem_t$ (MB) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Orig. | Opt. | Orig. | Opt. | Orig. | Opt. | Comp. | Orig. | Opt. |
| s35932 | 19876 | 1763 | 2048 | 15 | 4 | 7375 | 1538 | 15 | 4 | 1285 | 488 | 152 |
| s38584 | 22447 | 1464 | 1730 | 106 | 9 | 5761 | 2391 | 109 | 9 | 1433 | 385 | 191 |
| s38417 | 25585 | 1664 | 1742 | 48 | 8 | 7350 | 2001 | 49 | 8 | 1626 | 477 | 169 |
| b15_C | 9371 | 485 | 519 | 34 | 9 | 2872 | 981 | 35 | 9 | 592 | 182 | 73 |
| b17_C | 33741 | 1452 | 1512 | 120 | 12 | 9695 | 3444 | 123 | 12 | 2120 | 605 | 245 |
| b18_C* | 117963 | 3357 | 3365 | 188 | 12 | 34431 | 5605 | 193 | 12 | 7354 | 2088 | 428 |
| b19_C* | 237962 | 6666 | 6672 | 168 | 12 | 69768 | 11398 | 172 | 12 | 14845 | 4495 | 910 |

We also take FFR(4) in Figure 1 as an example with the same notation used in Section III-B. The original order $\vec{a}_1 = \{4, 5, 6, 7, 8\}$ results in the assignment, $\vec{g}_1 = \{1, 2, 3, 2, 1\}$, $V = 3$. However, if the gates are simulated by another order $\vec{a}_2 = \{5, 6, 7, 4, 8\}$, the storage would be $\vec{g}_2 = \{1, 2, 1, 2, 1\}$, $V = 2$, which is one space less than the original one. For each stem $s$, we employ an approach based on the post-order tree traversal to determine the gate simulation order within the FFR. Effectively, we consider the gates in FFR(s) as a tree. At the pre-processing stage, for each node, its child nodes are sorted in terms of their number of child nodes. The node with more child nodes are preferable to be simulated earlier so that more memory spaces could be free. By applying the algorithm, our results show that the space requirement for simulating the FFRs can be reduced by more than $10\times$, as shown under the $Mem_s$ columns in Table I.

## IV. EXPERIMENTAL RESULTS

We evaluated RAG's performance on a set of large full-scan ISCAS89 and ITC99 benchmark circuits which are available from [13]. Our fault simulation platform consists of a workstation with Intel Quad-core i7 3.33 GHz CPU, 4 GB memory and one NVIDIA GeForce GTX 285 graphics card which has 30 SMs and 240 processing cores with the clock speed of 1.476 GHz. The operating system on the host machine is a 32-bit Ubuntu GNU/Linux distribution. To achieve an accurate reliability evaluation, RAG simulates 2.4 million samples.

### A. Scheduling Evaluation and Memory Performance

The characteristics of the circuits and the storage requirements for stems and FFRs are reported in Table I. We note that all the listed benchmarks are unable to be simulated without the two-stage hierarchical simulation framework because of the 950 MB device memory limitation. Also, after applying the proposed greedy scheduling algorithms, the circuits marked with $*$ could be handled by RAG. For each circuit listed in the first column, the number of logic gates, inputs and outputs are listed in Columns 2, 3 and 4, respectively. Columns 5 and 6 report the original and optimized storage usage for the gates within FFRs in terms of the number of elements. The original and optimized storage requirements for the stems are reported in Columns 7 and 8, respectively. The actual shared memory usage for storing these FFRs (Columns 5 and 6) ($Mem_s$) per SMs are listed in Columns 9 and 10. The original storage usage is the maximum size of FFRs in the circuit before applying our proposed algorithm. The total global memory usage ($Mem_t$) for the GPU are listed in Columns 11 through 13 including the storage of circuit netlist, error counters for each output and the logic values of stems. Column 11 (Comp.) listed the memory usage without the proposed two-stage simulation where all the signal values are stored. For example, consider circuit b19_C with more than 200K gates. Without our scheduling algorithms, we could not simulate the circuit due to its large size. However, after applying the proposed scheduling algorithm, the size for storing the value of stems reduced to 11398 (701 MB), which is less than one-sixth of the original value, 69768 (4286 MB). Also, the storage for FFRs reduced from 172 (172 KB) to 12 (12 KB) which makes it fit into the shared memory.

### B. Accuracy and Performance Evaluation

First, we compared RAG with two state-of-the-art reliability analysis tools with the results reported in Table II. An exact approach with sampling based on PTMs [3] and the non-exact observability-based method [5] were used in our comparison. The failure rate used for each gate was 0.1. Note that the benchmarks used in the previous work only targeted ISCAS85 circuits [13], which are notably smaller than the additional ones we use in this paper. Columns 2 and 5 list the average reliability and runtimes for PTMs which are referred from Table III in [3]. As an comparison, both the reliability and run time for the proposed RAG are listed in Columns 3 and 7, respectively. Since the authors in [5] did not provide the results for reliability evaluation, we only list its runtimes under maximum-3 gate failure model with 50k samples in Column 6. Column 4 shows that the reliability analysis using RAG is highly accurate compared with the PTMs approach. Only an average percentage error of 0.29% is introduced. Moreover, RAG could achieve an average speedup of $412.11\times$ and $198\times$ comparing with these two tools. We note that our experimental platform is more advanced than the one used in [3] and [5]. However, assuming a four-fold difference of platform performance, RAG could still get more than $20\times$ speedup.

To further examine the performance of RAG on large circuits, we implemented two stochastic simulators based on fault injection with 32 bit-parallelism. One is event driven based simulator ($Sim_{event}$) and the other is a compiled random logic simulator ($Sim_{comp.}$) which is a sequential version of RAG (all the gates are evaluated). We note that the sequential simulators are faster than PTMs [3] and [5] because the tools are bit-parallelized and implemented on a more advanced platform. We evaluated 9 large circuits with three different failure rate, 0.005, 0.05, and 0.10. We note that error rates less than 0.005 were not included, since we assume a high error probability in nano circuits, as previous authors had [2–7]. Also, RAG is able to handle gates with different error rates by simply maintaining an error rate vector ($\vec{\tau}$). The average reliability among outputs are listed in Column 2, 3 and 4 in terms of different error rates. Since RAG and $Sim_{comp.}$ evaluate all the gates for each simulation run, the runtimes will be the same under any error rate. Therefore, we only list their runtimes for an error rate of 0.05 in Column 8 and 9. However, since $Sim_{event}$ only processes the gates whose logic

TABLE II
COMPARISON WITH OTHER TOOLS

| Circuit | Avg. Reliability | | | Runtimes (in seconds) | | | Speedup | |
|---|---|---|---|---|---|---|---|---|
| | PTM [3] | RAG | err(%) | PTM [3] | [5] | RAG | PTM [3] | [5] |
| c499 | 0.779 | 0.7787 | 0.04 | 18.70 | 59.64 | 0.10 | 192.52 | 614.01 |
| c880 | 0.680 | 0.6802 | 0.03 | 30.00 | NA | 0.15 | 204.10 | NA |
| c1908 | 0.627 | 0.6308 | 0.60 | 220.70 | 12.91 | 0.28 | 793.96 | 46.44 |
| c2670 | 0.872 | 0.8727 | 0.08 | 72.70 | 13.13 | 0.28 | 258.53 | 46.69 |
| c3540 | 0.564 | 0.5661 | 0.37 | 330.94 | 41.59 | 0.49 | 675.16 | 84.85 |
| c5315 | 0.697 | 0.7033 | 0.91 | 233.62 | NA | 0.67 | 348.41 | NA |
| **avg** | | | **0.29** | | | | **412.11** | **198** |

TABLE III
RUNTIME IN STEPS

| Bench | Runtimes (in seconds) | | | |
|---|---|---|---|---|
| | $K_{sim}$ | $k_{red}$ | Init. | Misc. |
| c880 | 0.105 | 2.8e-5 | 1.5e-4 | 0.042 |
| s38584 | 5.233 | 4.73e-4 | 0.024 | 0.083 |
| s38417 | 5.989 | 4.72e-4 | 0.025 | 0.087 |
| b17_C | 8.370 | 4.23e-4 | 0.028 | 0.082 |
| b18_C | 30.150 | 8.86e-4 | 0.183 | 0.116 |
| b19_C | 63.244 | 17.09e-4 | 0.697 | 0.194 |

TABLE IV
COMPARISON WITH TOOLS ON THE SAME PLATFORM WITH LARGE CIRCUITS

| Circuit | Avg.Reliability | | | Runtimes (in seconds) | | | | | Speedup | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Event-driven Sim. | | | Compiled Sim. | RAG | Event | Compiled |
| | $\tau$=0.005 | $\tau$=0.05 | $\tau$=0.10 | $\tau$=0.005 | $\tau$=0.05 | $\tau$=0.10 | | | $\tau$=0.005 | |
| c880 | 0.959 | 0.770 | 0.680 | 2.68 | 10.92 | 11.31 | 7.84 | 0.15 | 18.20 | 53.37 |
| c7552 | 0.932 | 0.762 | 0.713 | 15.93 | 83.78 | 87.32 | 72.50 | 1.01 | 15.79 | 71.88 |
| s35932 | 0.985 | 0.909 | 0.858 | 80.38 | 393.64 | 425.30 | 324.31 | 4.50 | 17.87 | 72.11 |
| s38584 | 0.959 | 0.752 | 0.660 | 80.42 | 450.14 | 508.95 | 384.95 | 5.34 | 15.06 | 72.10 |
| s38417 | 0.949 | 0.716 | 0.627 | 90.14 | 513.55 | 568.32 | 439.39 | 6.10 | 14.78 | 72.02 |
| b15_C | 0.969 | 0.790 | 0.691 | 51.33 | 205.84 | 211.56 | 168.63 | 2.35 | 21.89 | 71.90 |
| b17_C | 0.966 | 0.768 | 0.658 | 152.32 | 830.31 | 790.86 | 619.54 | 8.48 | 17.96 | 73.05 |
| b18_C | 0.963 | 0.758 | 0.644 | 557.11 | 2975.57 | 3028.27 | 2258.53 | 30.45 | 18.30 | 74.17 |
| b19_C | 0.963 | 0.758 | 0.645 | 1224.97 | 6050.29 | 6091.47 | 5545.65 | 64.14 | 19.10 | 78.67 |
| **avg** | | | | | | | | | **17.66** | **71.77** |

values are changed, different error rate will result in different runtimes. Columns 5, 6 and 7 show the runtime for different error rate, $\tau$. It is noted that $\text{Sim}_{event}$ is much faster than $\text{Sim}_{comp.}$ when $\tau = 0.005$. However, when $\tau$ increases to 0.05 or larger, $\text{Sim}_{comp.}$ performs better since the overhead of event scheduling begin to take a toll on the even-driven simulator. Therefore, In Columns 10 and 11, we list the speedup of RAG against $\text{Sim}_{event}$ for $\tau = 0.005$ and $\text{Sim}_{comp.}$, respectively. An average speedup of nearly 18× and 72× were obtained.

In Table III, the break down of runtimes for various steps in the simulation is reported in seconds. The stochastic simulation kernel is listed in Column 2 ($K_{sim}$) which dominates the total runtime in Table IV. This shows the efficiency of RAG because most of the computations are parallelized and accelerated on the GPU. Column 3 lists another reduction kernel ($K_{red}$) which is very lightweight. The cost for scheduling of stems and FFRs on the CPU is shown in Column 4 as an initialization of RAG (a one-time cost). Since both scheduling algorithms are linear to the size of the circuits, the overhead is negligible compared with the kernel runtimes. The last column (Misc.) includes the data initialization on the GPU and the transmission overhead between the CPU and the GPU.

## V. CONCLUSIONS

In this paper, we proposed a novel tool, RAG, which is the first GPU-based reliability analysis of logic circuits. RAG achieves both high accuracy and efficiency by exploiting the power of GPUs with a novel two-stage simulation framework. The proposed algorithms for stems scheduling and post-order tree traversal for non-stem gates arrangement allow for an efficient utilization of the available computational resources on the GPU. Experimental results showed that RAG could achieve an average of 412× and 198× speedup against two state-of-the-art reliability analysis tools (one is an exact approach with sampling and the other is heuristic-based) without compromising accuracy. Moreover, for large benchmarks which are not able to be handled efficiently by previous tools, RAG is also 17× and 71× faster than another two stochastic simulation based reliability analysis tools implemented on the conventional processor architecture.

## REFERENCES


[1] G. Bourianoff, "The future of nanocomputing," *Computer*, pp. 44–53, 2003.

[2] S. Krishnaswamy, G. Viamontes, I. Markov, and J. Hayes, "Accurate reliability evaluation and enhancement via probabilistic transfer matrices," in *Proc. Design Automation & Test Europe Conf.*, 2005, pp. 282–287.

[3] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Probabilistic transfer matrices in symbolic reliability analysis of logic circuits," *ACM Trans. Design Automation Electronic Systems*, vol. 13, no. 1, pp. 1–35, 2008.

[4] A. Abdollahi, "Probabilistic decision diagrams for exact probabilistic analysis," in *Proc. Int. Conf. Computer-Aided Design*, 2007, pp. 266–272.

[5] M. R. Choudhury and K. Mohanram, "Reliability analysis of logic circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 3, pp. 392–405, 2009.

[6] T. Rejimon and S. Bhanja, "Scalable probabilistic computing models using Bayesian networks," in *Proc. Int. Symp. Circuits & Systems*, 2005, pp. 712–715.

[7] H. Chen and J. Han, "Stochastic computational models for accurate reliability evaluation of logic circuits," in *Proc. Great Lake Symp. VLSI*, 2010, pp. 61–66.

[8] "NVIDIA CUDA homepage," http://www.nvidia.com/object/cuda_home.html.

[9] D. Chatterjee, A. DeOrio, and V. Bertacco, "Event-driven gate-level simulation with gp-gpus," in *Proc. Design Automation Conf.*, 2009, pp. 557–562.

[10] K. Gulati and S. P. Khatri, "Fault Table Computation on GPUs," *J. Electronic Testing: Theory and Applicat.*, vol. 26, no. 2, pp. 195–209, 2010.

[11] M. A. Kochte, M. Schaal, H. J. Wunderlich, and C. G. Zoellin, "Efficient fault simulation on many-core processors," in *Proc. Design Automation Conf.*, 2010, pp. 380–385.

[12] M. Li and M. S. Hsiao, "FsimGP$^2$: An efficient fault simulator with GPGPU," in *Proc. Asian Test Symp.*, 2010, pp. 15–20.

[13] "IWLS 2005 benchmarks," http://www.iwls.org/iwls2005/benchmarks.html.