Variation-Aware Fault Grading

A. Czutro*, M.E. Imhof†, J. Jiang‡, A. Mumtaz†, M. Sauer*, B. Becker*, I. Polian‡, H.-J. Wunderlich†

*University of Freiburg, Germany, {aczutro | sauerm | becker}@informatik.uni-freiburg.de
†University of Stuttgart, Germany, {imhof | mumtaz | wu}@iti.uni-stuttgart.de
‡University of Passau, Germany, {jie.jiang | ilia.polian}@uni-passau.de

***Abstract*—An iterative flow to generate test sets providing high fault coverage under extreme parameter variations is presented. The generation is guided by the novel metric of circuit coverage, calculated by massively parallel statistical fault simulation on GPGPUs. Experiments show that the statistical fault coverage of the generated test sets exceeds by far that achieved by standard approaches.**

***Keywords*-process variations, fault grading, Monte-Carlo, fault simulation, SAT-based, ATPG, GPGPU**

I. Introduction

End-of-the-roadmap CMOS transistors and emerging post-silicon devices are expected to be prone to extreme statistical process variations [1]. The variability in delay of a circuit element may well exceed $\sigma/\mu = 20\%$ [2]. For silicon CMOS, the sources of variation include sub-wavelength photolithography, random dopant distribution and line edge roughness, which affect the threshold voltage, oxide thickness and transistor geometry [3, 4]. While inter-die variations (lot-to-lot, wafer-to-wafer, within wafer) have been more widely considered in the past, in nano-scale technologies, intra-die variations affecting devices on the same die are equally important [5]. On top of this, dynamic variations in power-supply voltage and temperature, transistor aging, cross-coupling capacitance and multiple-input switching, adversely impact the clock frequency and may increase the variation by even 10% [6].

In the context of testing, the significance of variability was recognized early. The concept of *robust delay test* [7–9] aims at the detection of delay faults independently of delay variations that might occur elsewhere in the circuit. This concept imposes stringent requirements on the side-inputs of the path to be sensitized, which in turn may prevent detection in several cases. Recently, the detectability of small-delay faults (SDF) under process variations has regained increased interest, as such faults adequately abstract several relevant defect mechanisms in nano-scale technologies, including resistive opens [10, 11]. SDFs are traditionally detected by sensitizing the longest testable path through the fault location. However, the longest path is no longer unique under delay variations. Hence, algorithms to efficiently sensitize K longest paths (KLPG) have been devised [12]. Yet, this approach is guided solely by the propagation path's length and we experimentally found that it may not detect defects that manifest themselves as glitches.

Some researchers have explicitly considered process variations during test generation. For instance, Yilmaz et al. noted in [13] that simple heuristic approaches like n-*detect* [14] are able to increase the defect coverage significantly. But also more sophisticated methods have been employed. In this kind of approaches, a suitable defect-coverage metric, usually based on statistical timing analysis, is used to guide the ATPG process. Park et al. were the first to propose the concept of statistical delay-fault coverage [15]. Newer examples include [16] where delay defects are targeted with statistically distributed gate delays while considering specific noise effects. In [17], a metric called *parametric fault coverage* is introduced and utilized to guide an adaptive test flow. Ingelsson et al. [18] extend the resistive-bridge fault model by incorporating variation effects and also introduce a new evaluation metric called *process coverage*. In [19], gate-delay defect probabilities are calculated and assigned to the gates while considering process variations. The prediction of a parameter sub-space of a certain die was suggested in [20, 21] in order to reduce the test set's size. In [22], the testability of paths is evaluated statistically to guide the selection of paths for test generation.

In this work, we consider the problem of generating test patterns for SDFs under process variations. The core motivation is that a test pair may be valid for specific parameter values only. In general, multiple test pairs can be required to ensure the detection of a single fault in all possible circuit instances. This is illustrated in Figure 1 which shows two manufactured instances of the same circuit. The gate delays are described by Gaussian distributions; the red vertical lines indicate the actual gate delays. To detect an SDF at a, different test pairs are needed in the two circuit instances, because the longest path through a is not the same in both instances.

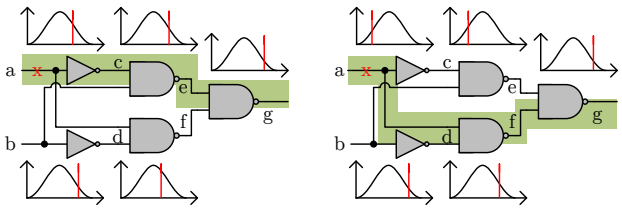

Fig. 1. Fault detection under parameter variations [23]. An SDF on a is detected by test pair 01/11 in the first circuit instance and by 00/10 in the second instance.

We demonstrate that traditional delay-test ATPG methods solely based on nominal gate delays are not effective under extreme variations. We present an iterative *fault grading* flow based on the concept of *circuit coverage* [24, 25]. In order to perform precise timing-aware fault simulation on an extremely large number of combinations of circuit instances, faults and

test patterns, a new highly parallel algorithm for *General-Purpose Graphics Processing Units* (GPGPU) is used.

We performed experiments on large industrial circuits. The obtained results show that our variation-aware method is able to achieve much better statistical fault coverage (SFC) than traditional non-variation-aware methods, even without increasing the total pattern pair count thanks to compaction techniques.

The remainder of the paper is organized as follows. The next section introduces some preliminary definitions and discusses appropriate metrics for evaluating fault and circuit coverage. Section III describes the simulation and ATPG algorithms' interaction within the circuit population. Section IV describes the Monte-Carlo delay fault simulation on GPGPUs where signals are represented as waveforms. Section V presents the SAT-based methods used for the generation of test patterns for instances not yet covered. Detailed experimental results are reported in Section VI, and Section VII closes the paper.

II. Preliminaries

In this work, every *circuit instance* is represented by a gate-level net list along with a fixed delay δ_g for every gate g. We consider *delay faults* (g, s) given by a *location* (a gate g) and a *fault size* s. If fault (g, s) is present in the circuit, then gate g's delay is increased to $\delta_g + s$.

Such faults are known as *gate-delay faults*. Recently, also the term *small-delay faults* has become widely used such as to mark contrast to transition faults where the defect-induced delay is assumed to be larger than the clock cycle, thus making them more easily detectable. Note that, due to the modeled delay variations, our simulation approach also covers the behavior of path-delay faults, where defect-induced delay is distributed along a path.

The value t_{obs} denotes the *observation time*. It is usually chosen such that all circuit outputs of a fault-free circuit have stabilized at their correct value after the application of an input pattern at time 0. In traditional circuit design, t_{obs} equals the maximum delay of the circuit in absence of variations, multiplied by a safety margin greater than 1. In recent technologies affected by high variability, t_{obs} is set more aggressively. It is accepted that the longest path of some manufactured circuit instances is slower than t_{obs}, even though the nominal instance has no path slower than t_{obs}. There is a yield-performance trade-off: increasing t_{obs} means a lower operation frequency (worse performance) for all circuits but a higher number of circuit instances that are timing-correct (better yield), and vice versa [26].

In the context of delay testing, the relevance of parameter variations lies in their effect on the delays of individual circuit elements, e.g. gate delays. Let N be the number of gate delays that can be affected by variation. A *parameter configuration* $p = (p_1, p_2, \ldots, p_N) \in P$ is a list of actual values of every modeled gate delay, and P is the space of all possible parameter configurations. A circuit instance corresponding to the parameter configuration p is denominated by C_p.

In this paper, we approximate the infinite set P of parameter configurations by a finite set I of circuit instances. Every circuit instance $i \in I$ has fixed gate delays that are chosen randomly according to a gate delay distribution.[1] Instance 0, called the *nominal circuit instance*, is defined to have no variations (each gate delay is the mean of the gate delay distribution). All the other instances ($i > 0$) are affected by variations.

A fault $f = (g, s)$ is defined to be *detected in* C_p by a given test pair set T, if at least one test pair leads to an incorrect response in C_p at time point t_{obs}, where the correct response is given by the nominal circuit instance's fault-free response at time t_{obs}. Furthermore, we define $det_{C_p}(f, T)$ to be 1 if T detects f in C_p, and 0 otherwise. Analogously, the *detectability* $det_i(f, T)$ is defined for an instance $i \in I$.

Let $\pi(p)$ be the probability that an actual manufactured circuit corresponds to parameter configuration p. For a fixed fault f, the *circuit coverage* of a test set T is determined by

$$CCov(f, T) = \int_{p \in P} det_{C_p}(f, T)\, \pi(p)\, dp. \tag{1}$$

This definition is adapted for the circuit instances $i \in I$ assuming their equiprobable distribution:

$$CCov(f, T) \approx \frac{\sum_{i \in I} det_i(f, T)}{|I|}. \tag{2}$$

This corresponds to the percentage of modeled circuit instances in which T detects f.

Furthermore, we call the aggregated circuit coverage of all faults in a fault list F the *statistical fault coverage*:

$$SFC(F, T) = \frac{\sum_{f \in F} CCov(f, T)}{|F|} = \frac{\sum_{f \in F} \sum_{i \in I} det_i(f, T)}{|F| \cdot |I|}. \tag{3}$$

These metrics can be illustrated by the following example. Fig. 2 shows the detectability of 16 modeled faults (x-axis) in 4 modeled circuit instances (y-axis). Black points correspond to detections, white ones to non-detections. The approximated circuit coverage (Equation 2) of fault 0 is $3/4 = 75\%$, while the statistical fault coverage (Equation 3) is $36/64 = 56.25\%$.

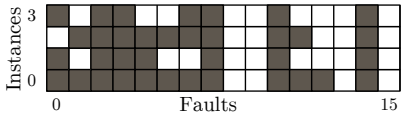

Fig. 2. Illustration of circuit coverage and statistical fault coverage

III. Fault Grading

The overall flow of the variation-aware fault-grading procedure is outlined in Figure 3. The procedure takes a set of circuit instances I and a fault list F as inputs. It constructs a test set to detect as many faults from F in as many instances from I

[1] We use a Gaussian distribution without correlations in the experiments, but the concept is applicable to any distribution, e.g. to distributions obtained statistically from actual measurements.

as possible, thus maximizing the circuit coverage for the faults and the overall statistical fault coverage SFC. Subsequently, the test set is compacted by eliminating test pairs without adverse impact on SFC.

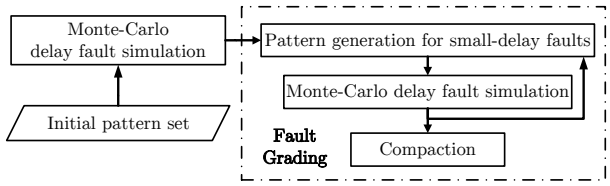

Fig. 3. Iterative procedure to maximize the circuit coverage

An initial set of test pairs is generated for the nominal circuit instance. Then, GPGPU-based delay-fault simulation is performed, explicitly considering all faults from F and all circuit instances from I. Details on the fault-simulation procedure are provided in Section IV. The simulation determines the fault-instance-pairs that are not covered by the initial test set and thus have a negative impact on SFC. To improve the coverage, new test pairs are generated using timing-aware automatic-test-pattern-generation algorithms (ATPG). As described in more detail in Section V, state-of-the-art delay ATPG algorithms make simplifying assumptions about the fault behavior. As a consequence, a generated test pair is likely, but not guaranteed to detect the fault for which it was generated. Therefore, the actual coverage achieved must be verified by accurate simulation.

The test-pair generation procedure outlined above succeeds in significantly improving SFC, yet at the cost of an increased number of test pairs. In order to obtain a more compact set of test pairs that are *effective*, i.e. test pairs that contribute to SFC, we developed the heuristic $T_{\min}$-algorithm. The experiments show that the algorithm's aggressive compaction preserves the SFC achieved by the original test set. Algorithm 1 describes the method.

For every fault $f \in F$, let $T^f = \{t_0^f, t_1^f, ..., t_e^f\}$ be the set of test pairs that detect f in at least one instance $i \in I$. Let $I(t_k^f) = \{i_0, i_1, ..., i_h\}$ be the set of instances in which test pair t_k^f detects f. The *effective pairs* T^F of T with respect to F are all test pairs that detect at least one $f \in F$ in at least one circuit instance.

For every fault f, the $T_{\min}$-algorithm sorts the effective test pair set T^f in descending order with respect to the number of instances in which the pair $t_k^f \in T^f$ detects f. The selected test pairs are kept in set $T_{\min}$ and the instances, in which f has been covered so far, are stored in I_{cov} (in the beginning, these sets are empty). The algorithm searches for a pair t_k that detects f in at least one yet-uncovered instance. If there are several such pairs, the one which detects the fault in most instances (largest $I(t_k^f) \setminus I_{cov}$) is selected. This pair is added to the solution set T_{min}, and all pairs from that set which are not essential, i.e. which do not detect a subset of faults covered by t_k^f, are removed. All the instances covered by the selected pattern are marked as covered by adding them to I_{cov}. The process is iterated over the complete fault list F.

Algorithm 1 Compaction algorithm $T_{\min}$

Require: T^F
  $T_{\min} := \emptyset$
  for all faults f in F **do**
    $I_{cov} := \emptyset$ ▷ instances in which f detected
    Sort pairs t_k in T^f by descending $|I(t_k^f)|$
    Find $t_k^f \in T^f$ with largest $I(t_k^f) \setminus I_{cov}$
    if (such t_k^f exists) **then**
      $T_{\min} := T_{\min} \cup \{t_k^f\}$ ▷ add to the solution
      Remove from $T_{\min}$ test pairs that became non-essential
      Add to I_{cov} instances in which f is detected by t_k^f
    end if
  end for
  return $T_{\min}$

IV. Monte-Carlo Delay Fault Simulation on GPGPUs

Given a set I of circuit instances, a set G of fault locations, a set S of possible fault sizes and a test set T, we are interested in circuit coverage and statistical fault coverage as defined in Section II and therefore in the detection status of each fault in each instance of the circuit. Calculating this information results in $|I| \cdot |T|$ fault-free simulations and $|I| \cdot |G| \cdot |S| \cdot |T|$ fault simulations. To cope with this complexity, our simulation algorithm is designed for massive parallelism using General-Purpose Graphics Processing Units (GPGPU). Each GPGPU core is assigned its own combination of fault and circuit instance, and all the cores perform their simulations in parallel. Since GPGPUs have restrictions on the kinds of operations their cores can perform in parallel, event-based simulation is inefficient on such an architecture. Instead, techniques similar to pattern-parallel logic simulation are employed.

The GPGPU delay fault simulation is based on the test-power simulator described in [27]. However, only a brief overview can be provided in this paper due to the limited available space.

A. Outline of the simulation algorithm

This section outlines the general fault-simulation algorithm for a set I of circuit instances, a test pair (v_1, v_2) and a fault f. The simulator supports different rising and falling delays, arbitrary probability distributions and pulse filtering. The basic data structure employed by the simulator to represent signals is the *waveform*. The waveform of signal line l contains the complete history of rising and falling transitions on l, from the stabilization time of v_1 until the stabilization time of v_2. The simulator processes the gates in topological order starting at the primary inputs and computes, for each gate, the waveform on its output based on the waveforms on its inputs using the algorithm outlined in Section IV-B. The sets of waveforms computed for the primary outputs are compared with the fault-free reference waveforms to determine whether the fault has been detected.

This process is parallelized by assigning different circuit instances $i \in I$ to different GPGPU cores. For each line l, different GPGPU cores simultaneously compute waveforms associated to l in different circuit instances.

B. Waveform representation and manipulation

The simulator represents waveforms by a simple data structure that facilitates parallelization on GPGPUs. For a waveform w with m transitions, we denote the time of the first transition by t_0, the time of the second transition by t_1, and so on. Furthermore, we require that the waveform's initial logical value is always 0. (To represent a waveform with initial value 1, we define a special time value $-\infty$ and set $t_0 := -\infty$.)

This allows the full specification of a waveform by a list of transition times $w = (t_0, t_1, t_2, ..., t_{m-1})$ with $t_0 \leq t_1 \leq \cdots \leq t_{m-1}$. The logical value of a waveform w at time t is denominated by $w(t)$ and is derived from the parity of the smallest index j for which $t_j > t$. Figure 4 shows some example waveforms and their representations.

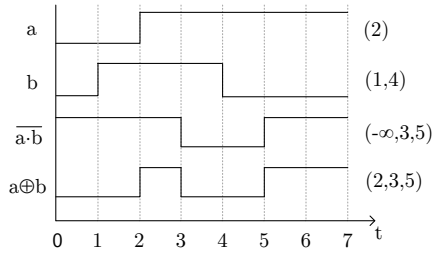

Fig. 4. Waveforms and their representations (Rising, falling delays of 1)

Let a, b and z be two inputs and one output of a logic gate that implements the Boolean function $\circ$, such as AND, NAND, OR, NOR or XOR. The computation of the waveform $z = a \circ b$ from a and b is based on the fact that, for $t > 0$, $a(t) = \text{parity}(j)$ for some index j, $b(t) = \text{parity}(k)$ for some index k, and therefore $z(t) = \text{parity}(j) \circ \text{parity}(k)$. Algorithm 2 performs the evaluation for a cell function with an arbitrary number of input waveforms $w^1, \ldots, w^n$ by iterative merge-sort. In each step of the algorithm, the first unprocessed transition of one of the input waveforms is determined (time t), the logic function of the gate is calculated, and if the resulting value is different from the current value parity(j), a new transition at time $t +$ delay is added to z. This procedure can be easily extended to varying gate delays for different logic conditions, multi-output cells and pulse filtering.

V. Test Pattern Generation for Delay Faults

To pin-pointedly generate tests for delay faults in a specific circuit instance, we employ timing-aware automatic test pattern generation (ATPG). The used ATPG algorithms are based on the sensitization of a path through the fault location g. A structural path through g is a sequence of gates $g_1, \ldots, g_k$ ($g \in \{g_1, \ldots, g_k\}$) such that g_1 is an input of the circuit, g_k is an output of the circuit, and such that the output of g_{j-1} drives an input of g_j for all $1 < j \leq k$. This input of g_j is called the *on-path input* of g_j, all other inputs are called *off-path inputs*. The delay of the path is the sum of the delays of its gates. The difference between t_{obs} and this delay is called *slack* of the path. It represents the maximum amount of time by which the path may be delayed without leading to a fault effect.

Algorithm 2 $z = \text{evaluate}(w^1, \ldots, w^n)$

Require: gate$(v^1, \ldots, v^n)$: Gate function to perform
Require: delay : The delay of the gate
  $i_1, \ldots, i_n := 0; j := 0;$
  if gate$(0, \ldots, 0) = 1$ **then**
    $z_0 := -\infty; j := 1;$ ▷ output starts with 1 for inverting gates
  end if
  while $\min\{w^1_{i_1}, \ldots, w^n_{i_n}\} < \infty$ **do**
    $t := \min\{w^1_{i_1}, \ldots, w^n_{i_n}\};$ ▷ time of earliest transition
    $k \in \{k \mid w^k_{i_k} = t\};$ ▷ choose one input waveform
    $i_k := i_k + 1;$ ▷ consume earliest transition
    if parity$(j) \neq$ gate(parity$(i_1), \ldots,$ parity$(i_n))$ **then**
      $z_j := t +$ delay$; j := j + 1;$ ▷ add transition to output waveform
    end if
  end while
  $z_j := \infty;$ **return** z

A. Path-oriented delay ATPG

Intuitively, a path is *sensitized* by a test pair (v_1, v_2) if a transition at its input propagates to its output, thus exposing delays along the path. The path-oriented ATPGs employed in this work [28, 29] generate test pairs for fault (g, s) that sensitize a number of longest paths through g. If one of these paths has slack less than or equal to s, the corresponding test pair is likely to detect the fault. However, there are cases in which detection is invalidated by effects not taken into account by the ATPG procedure.

A path $g_1, \ldots, g_k$ is formally defined to be sensitized by a test pair (v_1, v_2) if it launches a transition at g_1 and justifies certain *sensitization conditions* on the off-path inputs of all g_j. The sensitization conditions considered in this work are shown in Table I for AND/NAND gates and explained in detail in [30, 31]. S1 denotes a signal that is at stable logic 1 over two clock cycles (application of v_1 and v_2); H1 stands for a signal that stabilizes to logic 1 in both cycles, but may or may not have glitches in between. U1 stands for a signal that eventually stabilizes at logic 1 in the second cycle (but no conditions are imposed on the first cycle). Finally, XX stands for a signal on which no conditions are imposed at all.

TABLE I
Sensitization conditions for AND/NAND gates

Type	If on-path transition is	Off-path inputs are set to	Additional conditions
hazard-free robust	$0 \to 1$	S1	–
	$1 \to 0$	S1	–
robust	$0 \to 1$	U1	–
	$1 \to 0$	S1	–
strong non-robust	$0 \to 1$	U1	–
	$1 \to 0$	H1	–
weak non-robust	$0 \to 1$	U1	–
	$1 \to 0$	U1	–
restricted functional	$0 \to 1$	U1	gate output must undergo transition
	$1 \to 0$	XX	
functional	$0 \to 1$	U1	–
	$1 \to 0$	XX	–

Robust and hazard-free robust sensitization are particularly important, as they detect faults of sufficient size independently of delays elsewhere in the circuit. The other sensitization conditions used are weaker as the timing in other paths of the circuit may invalidate fault detection, yet often they allow sensitization of longer paths than (hazard-free) robust tests. Therefore, no sensitization condition is universally optimal, and it is even possible that tests generated under all sensitization conditions miss a detectable delay fault. Our experiments showed that a combination of several conditions is helpful for the construction of test sets during the fault grading.

The test pattern generation makes use of two methods that employ different internal representations of the circuit and therefore have different expressive power. Specific sensitization conditions necessitate the choice of a particular method.

B. SAT-based ATPG

The purely SAT-based approach from [28] benefits from the efficient search optimization techniques incorporated into modern SAT-solvers. It represents delays by integer numbers and employs sophisticated rounding strategies to compensate for resulting small inaccuracies. For a given target gate g, the ATPG measures the length L_g of the longest sensitizable path through g. This is done by means of iterative SAT-solving using a binary search over the space of possible path lengths. Several learning strategies are employed to narrow down the search space. Then, a SAT-instance $S_g[= L_g]$ is constructed, such that it is satisfiable iff there is a path of length L_g that passes through g, and if the path is sensitizable according to the chosen sensitization conditions. All resulting paths of that length and the test pattern pairs that sensitize them are extracted from the Boolean solutions of $S_g[= L_g]$.

C. Structural KLPG

The second method is an extended version [29] of the K-Longest-Path-Generation (KLPG) algorithm from [12]. Unlike the SAT-based ATPG from the previous section, it works directly on the circuit structure. This approach is used when working with weak non-robust and functional sensitization, which are not supported by the SAT-based ATPG.

In order to find the K longest paths through the target gate g that are sensitizable according to the chosen sensitization conditions, OptKLPG successively collects *partial paths* that start at one of the primary inputs in the transitive fan-in of g. For each generated partial path, a quick sensitization check based on local implications is performed. Partial paths that fail the check are discarded, while the remaining partial paths are extended by adding gates from the transitive fan-in or fan-out of g to the partial path. All partial paths under consideration are kept in a data structure called *path store*. The optimality of the found K longest paths is guaranteed by a second data structure that contains relevant partial paths which do not fit into the path store [29]. Once a complete path has been generated, a vector pair that sensitizes the path is generated by mapping the required sensitization conditions to a complex fault model that can be processed by a SAT-ATPG-engine, in our case TIGUAN [32, 33].

VI. Experimental Results

Experimental results are shown for the combinational cores of industrial NXP circuits synthesized using the Nangate 45nm OpenCellLibrary [34]. To obtain delays appropriate for state-of-the-art 22nm technology, t_{obs} and the nominal rising and falling delays of every gate were scaled by 0.75 [1]. To generate the set I of circuit instances, the delay of each gate was modeled by a Gaussian distribution with the mean equal to the scaled nominal delay and the variance of 20%. For every instance from I, a fixed delay was randomly derived from this distribution. In total, we considered one nominal circuit and 100 circuit instances affected by variations; we refer to this set of 101 circuit instances by I_{100}. We generated a fault list consisting of 100 randomly chosen fault locations and 9 fixed fault sizes $(0.1 \cdot t_{obs}, \ldots, 0.9 \cdot t_{obs})$. In total, $(1 + 100) \cdot 100 \cdot 9 = 90900$ simulation runs are required for a test pair.

The initial test pair set T_1 was generated by a commercial ATPG tool targeting transition faults at the chosen fault locations in the nominal circuit instance. We also generated a 5-detect test set T_5 using the same tool.

While the generation of T_1 and T_5 does not explicitly take process variations into account, t_{obs} is calculated by the synthesis tool under pessimistic safety margins implicitly accounting for process variations. The numbers of test pairs and the corresponding statistical fault coverages (SFC) of T_1 and T_5 are listed in Table II. The SFC achieved by these variation-unaware test sets is rather low. Furthermore, T_5 has only marginally better SFC than T_1 for all circuits except p35k which has very low SFC numbers anyway. This clearly shows that traditional non-variation-aware ATPG methods such as N-detect are not sufficient for nano-scale technologies affected by massive variations.

The final three columns of the table report the performance of T_1 augmented by the test set T_{ATPG} generated by our variation-aware test flow for faults not covered by T_1. The SFC is improved considerably for all circuits. In comparison to N-detect, the SFC is almost doubled. To verify these results, we applied the generated test pairs to a new collection of 200 random circuit instances I_{200}. The SFC measured for this new set of circuit instances, shown in the last column of Table II, has only minimal difference to the results for I_{100}, and therefore

TABLE II
STATISTICAL FAULT COVERAGE (%) FOR 101 CIRCUIT INSTANCES, 100 FAULT LOCATIONS AND 9 FAULT SIZES

		Transition-fault test sets				$T_1 \cup T_{\text{ATPG}}$		
		T_1		T_5		I_{100}		I_{200}
Circuit	Gates	Tests	SFC	Tests	SFC	Tests	SFC	SFC
p35k	23267	216	0.50	838	5.17	16997	29.23	29.01
p45k	25679	53	17.12	195	18.84	4305	29.16	28.86
p78k	70479	31	34.06	69	35.50	6333	54.37	54.16
p89k	58638	44	14.69	178	16.80	5650	27.65	27.35
p100k	61006	38	16.22	133	17.41	8080	29.02	28.85

TABLE III
COMPACTION RESULTS

	Number of tests (I_{100})			SFC (I_{200})	
Circuit	All	Eff	$T_{\min}$	All	$T_{\min}$
p35k	16997	14275	286	29.01	28.99
p45k	4305	3676	161	28.86	28.84
p78k	6333	6171	420	54.16	54.13
p89k	5650	5244	129	27.35	27.31
p100k	8080	6734	213	28.85	28.83

clearly underscores the validity of our results. Note that the absolute numbers of SFC are rather low. A large portion of the undetected faults are likely to be undetectable, however there is no redundancy proof engine that matches the simulation model exactly. We performed experiments with approximate redundancy definitions based on path sensitization, but found little correspondence to the detection of faults as reported by the accurate simulation.

The performance of the compaction algorithm $T_{\min}$ (see Section III) is reported in Table III. The total number of generated pattern pairs, the number of effective pairs, and the number of pairs left after the application of the $T_{\min}$ algorithm are quoted. Large differences in the pattern count corroborate the importance of the compaction strategy. It is interesting that the size of the smallest test sets produced for I_{100} by the $T_{\min}$-algorithm is comparable to T_5 (cf. Table II), and in 3 cases even lower. This shows that using a variation-aware test-generation flow can significantly improve SFC (by more than 50% for all circuits considered) with pattern counts similar to N-detect test sets. The final three columns show the statistical fault coverage calculated with respect to the instance set I_{200}. Although the compacted test set $T_{\min}$ is constructed taking only I_{100} into account, its detection capability essentially matches that of the original uncompacted test set for all circuits.

VII. CONCLUSIONS

State-of-the-art delay-fault ATPG methods reach their limits when applied to circuits affected by extreme process variations. We presented a variation-aware fault-grading flow that iteratively improves the test pair set by optimizing its circuit coverage and statistical fault coverage. The core of the method is the massively parallel fault simulation algorithm optimized to run on GPGPUs, combined with two path-oriented ATPG approaches, one SAT-based and one structural. The flow is complemented by a heuristic compaction algorithm which drastically reduces the number of test pairs without sacrificing coverage. Experimental evidence shows that the flow succeeds in generating test pairs that substantially outperform test sets obtained by conventional tools.

VIII. ACKNOWLEDGMENT

The authors would like to thank Stefan Holst and Eric Schneider for their support regarding the GPGPU simulator.

Parts of this work were supported by the German Research Foundation (DFG) under grants BE 1176/14-2, PO 1220/2-2, GRK 1103, WU 245/5-1 and WU 245/5-2.

REFERENCES

[1] "Int'l. Technology Roadmap for Semiconductors (ITRS)," 2011.
[2] Y. Ye, S. Gummalla, C. Wang, C. Chakrabarti, and Y. Cao, "Random Variability Modeling and its Impact on Scaled CMOS Circuits," *J. of Comp. Electr.*, vol. 9, no. 3, pp. 108–113, 2010.
[3] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical Analysis and Optimization for VLSI: Timing and Power*. Springer, 2005.
[4] S. Nassif, "Modeling and Analysis of Manufacturing Variations," in *IEEE Conf. on Cust. Integ. Circ.*, 2001, pp. 223–228.
[5] K. Bowman and J. Meindl, "Impact of Within-Die Parameter Fluctuations on Future Maximum Clock Frequency Distributions," in *IEEE Conf. on Cust. Integ. Circ.*, 2001, pp. 229–232.
[6] K. Bernstein, D. Frank, A. Gattiker, W. Haensch, B. Ji, S. Nassif, E. Nowak, D. Pearson, and N. Rohrer, "High-Performance CMOS Variability in the 65-nm Regime and Beyond," *IBM J. of Res. & Dev.*, vol. 50, no. 4.5, pp. 433–449, 2006.
[7] S. M. Reddy, M. Reddy, and V. D. Agrawal, "Robust Tests for Stuck-Open Faults in CMOS Combinational Logic Circuits," in *Int'l Symp. on Fault-Tolerant Comp.*, 1984, pp. 44–49.
[8] G. Smith, "Model for Delay Faults Based Upon Paths," in *IEEE Int'l. Test Conf.*, 1985, pp. 342–351.
[9] C. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Trans. on CAD*, vol. 6, no. 5, pp. 694–703, 1987.
[10] R. Rodríguez-Montañés, J. Pineda de Gyvez, and P. Volf, "Resistance Characterization for Weak Open Defects," *IEEE Design & Test of Computers*, vol. 19, no. 5, pp. 18–26, 2002.
[11] A. Czutro, N. Houarche, P. Engelke, I. Polian, M. Comte, M. Renovell, and B. Becker, "A Simulator of Small-Delay Faults Caused by Resistive-Open Defects," in *IEEE Europ. Test Symp.*, 2008, pp. 113–118.
[12] W. Qiu and D. Walker, "An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit," in *IEEE Int'l. Test Conf.*, 2003, pp. 592–601.
[13] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Test-Pattern Grading and Pattern Selection for Small-Delay Defects," in *IEEE VLSI Test Symp.*, 2008, pp. 233–239.
[14] S. Ma, P. Franco, and E. McCluskey, "An Experimental Chip to Evaluate Test Techniques Experiment Results," in *IEEE Int'l. Test Conf.*, 1995, pp. 663–672.
[15] E. Park, M. Mercer, and T. Williams, "Statistical Delay Fault Coverage and Defect Level for Delay Faults," in *IEEE Int'l. Test Conf.*, 1988, pp. 492 –499.
[16] J. Liou, A. Krstic, Y. Jiang, and K. Cheng, "Modeling, Testing, and Analysis for Delay Defects and Noise Effects in Deep Submicron Devices," *IEEE Trans. on CAD*, vol. 22, no. 6, pp. 756–769, 2003.
[17] M. Shintani, T. Uezono, T. Takahashi, H. Ueyama, T. Sato, K. Hatayama, T. Aikyo, and K. Masu, "An Adaptive Test for Parametric Faults Based on Statistical Timing Information," in *IEEE Asian Test Symp.*, 2009, pp. 151 –156.
[18] U. Ingelsson, B. Al-Hashimi, S. Khursheed, S. Reddy, and P. Harrod, "Process Variation-Aware Test for Resistive Bridges," *IEEE Trans. on CAD*, vol. 28, no. 8, pp. 1269–1274, 2009.
[19] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Interconnect-Aware and Layout-Oriented Test-Pattern Selection for Small-Delay Defects," in *IEEE Int'l. Test Conf.*, 2008, pp. 1–10.
[20] W. Daasch and R. Madge, "Variance Reduction and Outliers: Statistical Analysis of Semiconductor Test Data," in *IEEE Int'l. Test Conf.*, 2005.
[21] I. Polian, B. Becker, S. Hellebrand, H.-J. Wunderlich, and P. Maxwell, "Towards Variation-Aware Test Methods," in *IEEE Europ. Test Symp.*, 2011, pp. 219–225.
[22] J. Chung, J. Xiong, V. Zolotov, and J. Abraham, "Testability-Driven Statistical Path Selection," *IEEE Trans. on CAD*, vol. 31, no. 8, pp. 1275 –1287, 2012.
[23] F. Hopsch, B. Becker, S. Hellebrand, I. Polian, B. Straube, W. Vermeiren, and H.-J. Wunderlich, "Variation-Aware Fault Modeling," *Sci.China Inf.Sci.*, vol. 54, no. 9, pp. 1813–1826, 2011.
[24] B. Becker, S. Hellebrand, I. Polian, B. Straube, W. Vermeiren, and H.-J. Wunderlich, "Massive Statistical Process Variations: A Grand Challenge for Testing Nanoelectronic Circuits," in *IEEE Int'l. Conf. on Dep. Sys. and Netw. Workshops*, 2010, pp. 95–100.
[25] F. Hopsch, H.-J. Wunderlich, I. Polian, B. Becker, and S. Hellebrand, "Testing Nanoelectronic Systems Under Massive Statistical Process Variations," in *IEEE Asian Test Symp.*, 2010, p. xviii.
[26] M. Sauer, A. Czutro, B. Becker, and I. Polian, "On the Quality of Test Vectors for Post-Silicon Characterization," in *IEEE Europ. Test Symp.*, 2012.
[27] S. Holst, E. Schneider, and H.-J. Wunderlich, "Scan Test Power Simulation on GPGPUs," in *IEEE Asian Test Symp.*, 2012.
[28] M. Sauer, J. Jiang, A. Czutro, I. Polian, and B. Becker, "Efficient SAT-Based Search for Longest Sensitisable Paths," in *IEEE Asian Test Symp.*, 2011.
[29] J. Jiang, M. Sauer, A. Czutro, B. Becker, and I. Polian, "On the Optimality of K Longest Path Generation Algorithm Under Memory Constraints," in *Des., Autom. and Test in Europe*, 2012.
[30] N. K. Jha and S. K. Gupta, *Testing of Digital Systems*. Cambridge U. Press, 2003.
[31] S. Reddy, *Models in Hardware Testing*. Springer, 2010, ch. 3.
[32] A. Czutro, I. Polian, M. Lewis, P. Engelke, S. M. Reddy, and B. Becker, "Thread-Parallel Integrated Test Pattern Generator Utilizing Satisfiability Analysis," *Int'l. J. of Par. Prog.*, vol. 38, no. 3-4, pp. 185–202, 2010.
[33] A. Czutro, M. Sauer, T. Schubert, I. Polian, and B. Becker, "SAT-ATPG Using Preferences for Improved Detection of Complex Defect Mechanisms," in *IEEE VLSI Test Symp.*, 2012.
[34] "Nangate Open Cell Library v1.3 v2009/07."