Fault Simulation with Parallel Exact Critical Path Tracing in Multiple Core Environment

Maksim Gorev
Department of Computer Engineering
Tallinn University of Technology
Tallinn, Estonia
Email: maksim.gorev@ttu.ee

Raimund Ubar
Department of Computer Engineering
Tallinn University of Technology
Tallinn, Estonia
Email: raimund.ubar@ati.ttu.ee

Sergei Devadze
Department of Computer Engineering
Tallinn University of Technology
Tallinn, Estonia
Email: serega@pld.ttu.ee

***Abstract*—A novel fault simulation method is proposed, based on exact critical path tracing beyond the Fan-out-Free Regions (FFR) throughout the full circuit. The method exploits two types of parallelism: bit-level parallelism for multiple pattern reasoning, and distribution the fault reasoning process between different cores in a multi-core processor environment. To increase the speed and accuracy of fault simulation, compared with previous methods, a mixed level fault reasoning approach is developed, were the fan-out re-convergence is handled on the higher FFR network level, and the fault simulation inside of FFRs relies on the gate-level information. To allow a uniform and seamless fault reasoning, Structurally Synthesized BDDs (SSBDD) are used for modeling on both levels. Experimental research demonstrated very promising results in increasing the speed and scalability of the method.**

I. Introduction

Fault simulation is one of the most important tasks in the digital circuit design and test flow. The efficiency of solving other tasks in this field like design for testability, test quality and dependability evaluation, test pattern generation, fault diagnosis relies heavily on the performance and speed of fault simulation. Such a dependence is growing especially in case of large circuits, and hence, the scalability of the fault simulation algorithms is decisive. Accelerating the fault simulation would consequently improve all the above-mentioned applications.

Parallel pattern single fault propagation (PPSFP) concept [1] has been widely used in combinational and full scan-path circuits for fault simulation. Many proposed fault simulation concepts incorporate PPSFP with other sophisticated techniques such as test detect [2], critical path tracing [3], [4], stem region [5] and dominator concept [4], [6]. These techniques have helped to reduce further simulation time. Another trend of fault simulation methods based on reasoning (deductive [7], concurrent [8] and differential simulation [9]) used to be very powerful since they allow to collect all detectable faults by a single run of the given test pattern. What they cannot do, is to produce reasoning for many test patterns in parallel.

The critical path tracing method [3], [4] eliminates explicit fault simulation for faults within Fan-out-Free Regions (FFR). A modified critical path tracing technique that excludes fault simulation for fan-out stems and includes a system of rules to check the exactness of critical path tracing beyond the FFRs, and which is linear in time, is proposed in [10]. However, the rule based strategy does not allow parallel analysis and rule check of many patterns simultaneously. This drawback was removed in [11] by introducing a novel concept of Parallel Pattern Exact Critical Path Tracing (PPECPT) which can be applied efficiently also beyond FFRs. In [12], the same method was extended from stuck-at faults (SAF) for a general class of X-faults. The main idea of the method was in compiling of a dedicated compact computing model through the circuit topology analysis, which allows exact critical path tracing throughout the full circuit and not only inside FFRs.

In this paper we propose a new PECPT method, where we implement two types of parallelism during fault simulation: (1) bit-level parallelism for multiple pattern reasoning, and (2) distributing the compiled computing model among a subset of different CPUs in a multi-core computing environment, so that each processor were responsible for parallel critical path tracing in a related particular sub-circuit area.

Another novelty of the paper is in developing of a mixed level fault reasoning approach, were the problems related to the fan-out re-convergence are handled on the higher FFR network level, using collapsed fault set, and the increased speed and accuracy in fault reasoning is achieved by fault reasoning inside FFRs using additional gate-level simulation data, without fault collapsing.

To speed-up simulation and improve the accuracy of fault reasoning compared with previous methods in [11], [12], we propose here a mixed level PPECPT method based on using of two types of Structurally Synthesized BDDs (SSBDD).

Since during a single run of parallel analysis of patterns throughout the circuit we process all the faults in the circuit, we can say that the approach we propose is exploiting concurrency in three dimensions: pattern dimension, fault dimension and computing model dimension, where the pattern and fault parallelism is utilized using each single CPU core, while computing model concurrency is achieved exploiting multiple CPUs. Compared to the traditional approaches which can use only pattern- and fault-parallelism in multi-CPU systems (at the bit and system level, respectively), such a new dimension addition gives further possibilities to speed up fault simulation in multi-processor systems.

The availability of parallel execution environments such as multiprocessor system on chips (MPSoCs), multicore processors and GPGPU devices provides a possibility for concurrent execution of the same algorithm for different data or for different parts of the same algorithms and the same data. This is done to utilize the available new hardware resources, as well as to speed up execution in comparison to uniprocessor system. In the landscape of fault simulation, where this paper is targeted, the growing size and complexity of digital circuits also requires speed up of available algorithms.

The rest of the paper is organized as following. In Section 2 we present the theoretical basics of exact fault simulation by parallel critical path tracing beyond the fanout stems. In Section 3 we present the basics of fault simulation using SSBDDs, and in Section 4 we propose a new method of parallel critical path tracing based on mixed level fault simulation with two types of SSBDDs. Section 5 describes the method of distributing the task between multiple cores of the processor, Section 6 describes the results of experimental research

978-3-9815370-4-8/DATE15/©2015 EDAA

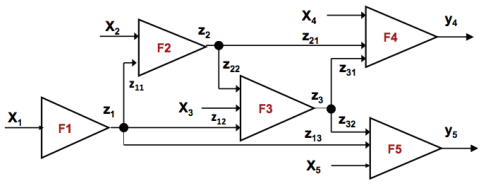


Fig. 1. Combinational circuit with 5 FFRs.

with related discussion, and Section 7 concludes the paper.

II. Parallel pattern critical path fault tracing

Consider a combinational circuit as a network of FFRs, where each of them is represented as a Boolean function

$$y = F(x_1, x_2, \ldots, x_n) = F(X) \tag{1}$$

where $X = x_1, x_2, \ldots, x_n$ is the input vector of the FFR. Such a network of 5 FFRs is represented in Fig.1. Let X_k denote the vector of input variables of the k-th FFR, z_k denote the internal fan-out stem variables (outputs of FFRs) with z_{kj} as fan-out branch variables for z_k (inputs of FFRs) and y denote the output variables of the circuit.

The fault simulation can be processed as calculation of Boolean derivatives: if $\partial y/\partial x = 1$ then the fault is propagated from x to y. This check can be performed in parallel for a set of test patterns. In order to extend the parallel critical path tracing beyond the fan-out free regions we use the concept of Boolean differentials [13].

Consider the full Boolean differential of the FFR $y = F(X)$ as

$$dy = y \oplus F((x_1 \oplus dx_1), \ldots, (x_n \oplus dx_n)) \tag{2}$$

Here, by ∂x we denote the change of the value of x because of the influence of a fault at x, and $\partial y = 1$ if some erroneous change of the values of arguments of the function (2) causes the change of the value of y, otherwise $\partial y = 0$.

In [11] we have shown that from the expression (2) the following relationship can be derived:

$$\frac{\partial y}{\partial x} = y \oplus F((x_1 \oplus \frac{\partial x_1}{\partial x}dx), \ldots, (x_n \oplus \frac{\partial x_n}{\partial x}dx)) \tag{3}$$

For example, the fault at z_2 is detected on y_4 if

$$\begin{aligned} \frac{\partial y_4}{\partial z_2} &= y \oplus F(X_4, z_{21} \oplus 1, z_{31} \oplus \frac{\partial z_3}{\partial z_2}dz_2) \\ &= y \oplus F(X_4, \overline{z_{21}}, z_{31} \oplus \frac{\partial z_3}{\partial z_2}dz_2) = 1 \end{aligned} \tag{4}$$

The formula 3 can be used for calculating the influence of the fault at the common fan-out stem x on the output y of the converging fan-out region by consecutive calculating of Boolean derivatives over related FFR chains starting from x up to y. For that purpose, for each converging fan-out stem, the corresponding formulas like (3) should be constructed for each converging FFRs. All these formulas will constitute partially ordered computation model for fault simulation. Since the formulas are Boolean, all computations can be carried out in parallel for a bunch of test patterns.

Introduce first the following notations for the formulas above which are used for calculating the Boolean derivatives:

- (x, y) - for $\partial y/\partial x$
- $\{X_k, y\}$ - for a subset of formulas $\{\partial y/\partial x \mid x \in X_k\}$
- $R_{xy}((x, x_1), (x, x_k))$ - for the general case (3)
- D_x - vector which shows if the fault at the node x is detected or not detected at any circuit output
- DX - a set of vectors D_x for the nodes $x \in X$

An example of a computational model of fault simulation for the circuit in Fig.1 is presented in Table I.

The formulas in Table I can be easily created and stored by the topological tracing of the circuit by algorithms developed in [11]. The algorithm has linear complexity. However, the complexity of the computational model and the related fault simulation speed depends on the structure of the circuit. As shown in the papers [14][12], the speed of fault simulation by the proposed parallel critical path tracing method outperforms the speed of the fault simulators of major CAD vendors.

III. Fault simulation with SSBDDs

The high speed of processing the formulas is achieved by using Structurally Synthesized BDDs (SSBDD) for modeling FFRs [15], [16]. Each FFR $y = F(X)$ is represented by an SSBDD G, and each signal path in the FFR represented by a variable $x \in X$ is modelled by a corresponding node in the G. All the faults on a signal path collapsed into the faults on the inputs of the FFR, are modelled by the faults at the nodes in G. Hence, the targets of the fault simulation are the faults at the SSBDD nodes.

Consider a circuit in Fig. 2, and its corresponding SSBDD. The circuit contains nine signal paths, and each of them is represented by a node in the graph. Note, only the branches of the fan-out inputs are represented in the SSBDD as the model of the FFR. Fault simulation is carried out by traversing the nodes in the graph according to the given test patterns as in the case of traditional BDDs [17].

For simplification the graphical representation of SSBDDs, we use here the following convention: from a node labelled by a variable x, the right-hand edge corresponds to the value $x = 1$, and the down-hand edge corresponds to the value $x = 0$. Correspondingly, the exit from the graph to the right means entering the terminal node with constant #1, and the exit from the graph downwards means entering the terminal node with constant #0.

Consider a test pattern 1011101 (1234567) at the inputs of the FFR in Fig. 2. The pattern detects the fault at the input 3 by propagating the faulty signal from the input 3 to the output 8. On the SSBDD in Fig. 2 the edges activated by this pattern are highlighted in bold. The nodes traversed in the graph during simulation of the pattern are marked by gray color. The value on the output 8 of the circuit at this pattern is $y = 1$. Since the nodes 1, 2_2, 3, 4, 5_2 are traversed, all they are responsible for the value $y = 1$s, and hence, should be taken as fault candidates in case if the error will be noticed at the circuit output. All other nodes 2_1, 5_1, 6, and 7 have not contributed in fault

TABLE I. Levelized fault model equations.

L	Partially ordered formulas	Types of simulation tasks
7	$\forall x_{4,i} \in X_4 : D_{x_{4,i}} = \{x_{4,i}, y_4\}$, $Dz_{21} = (z_{21}, y_4), Dz_{31} = (z_{31}, y_4)$; $\forall x_{5,i} \in X_5 : D_{x_{5,i}} = \{x_{5,i}, y_5\}$, $Dz_{13} = (z_{13}, y_5), Dz_{32} = (z_{32}, y_5)$	Fault simulation inside the FFRs (F4 and F5)
6	$Dz_3 = Dz_{31} \vee Dz_{32}$	Fault simulation of fan-out stems (z3)
5	$\forall x_{3,i} \in X_3 : Dx_{3,i} = x_{3,i}, z_3 \wedge Dz_3$, $Dz_{22} = (z_{22}, z_3) \wedge Dz_3$, $Dz_{12} = (z_{12}, z_3) \wedge Dz_3$	Fault simulation inside the FFRs (F3)
4	$Dz_2 = Rz_2, y_4((z_2, z_{21}) \equiv 1, (z_2, z_{31})) \vee ((z_{22}, z_{32}) \wedge Dz_{32})$	Fault simulation of fan-out stems (z2)
3	$\forall x_{2,i} \in X_2 : Dx_{2,i} = x_2, i, z_2 \wedge Dz_2$, $Dz_{11} = z_{11}, z_2 \wedge Dz_2$	Fault simulation inside the FFRs (F2)
2	$Dz_1 = ((z_1, z_3) \wedge Dz_{31}) \vee Rz_1, y_5((z_1, z_3), (z_1, z_{13}) \equiv 1)$ where $(z_1, z_3) = Rz_1, z_3((z_1, z_{22}), (z_1, z_1 2) \equiv 1)$	Fault simulation of fan-out stems (z1)
1	$\forall x_{1,i} \in X_1 : Dx_{1,i} = x_1, i, z_1 \wedge Dz_1$	Fault simulation inside the FFRs (F1)

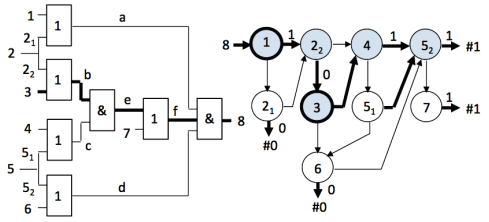


Fig. 2. An FFR of a combinational circuit and its SSBDD.

simulation, and hence, can be excluded from the fault candidates set. Next, by simulating the faults at candidate nodes we can easily notice that only the faults at the nodes 1 and 3 are detected by the given pattern, because at these faults on the graph the terminal node #0 will be reached which means y = 0.

In [11], the algorithms for parallel logic simulation and parallel fault simulation on SSBDDs were proposed. The algorithms are based on the ordering of nodes m by assigning them numerical labels, so that for each node m with label $n(m)$, all its predecessors mj must have labels $n(m_j)$ less than $n(m)$. Logic simulation is based on recursive calculating of the value of the formula

$$D(m) = (x(m) \wedge D(m^1)) \vee (\neg x(m) \wedge D(m^0)), \tag{5}$$

where $D(m)$ for the terminal nodes is equal to the respective constants #1 and #0. Here $x(m)$ denotes the node variable, m_1 and m_0 are the neighbors of m in directions of $x(m) = 1$, and $x(m) = 0$, respectively. Fault simulation is based on recursive calculating of values of the formulas

$$L(m^1) = L(m^1) \vee (L(m) \wedge x(m)), \tag{6}$$

$$L(m^0) = L(m^1) \vee (L(m) \wedge \neg x(m)), \tag{7}$$

$$S(x(m)) = \frac{\partial y}{\partial x(m)} = L(m) \wedge (D(m^0) \oplus D(m^1)) \tag{8}$$

where $S(x(m)) = 1$ means that the fault at x(m) is detected by the simulated test pattern, otherwise, if $S(x(m)) = 0$, the fault is not detected. Since all the presented formulas are Boolean, the algorithms can be applied by tracing the nodes of the SSBDDs can be applied in parallel for many test patterns, each of them represented by one bit of the computer word. The cost of simulation can be calculated by the number of operations needed for each node of SSBDD. For example, the cost of logic simulation is four operations per node, and the cost of fault simulation is seven operations per node. Hence, to fault simulate the SSBDD in Fig.2 which includes nine nodes, we need 9 * 7 = 63 operations. Example of using the algorithms can be found in [11]. Using SSBDDs instead of the gate-level circuit allows to increase both, the simulation speed for calculating the values of signals in the network of FFRs, and the fault reasoning, since only the collapsed fault set represented by nodes of SSBDDs is processed. This explains the efficiency of the method demonstrated in [11], [12].

IV. Mixed level fault simulation with SSBDDs

Recently Shared SSBDDs (S^3BDD) as a new type of BDDs were proposed to speed-up logic simulation in digital circuits [18], [19]. In the following we propose a two level implementation of the proposed method of critical path tracing, where as the objectives of higher level, the fan-out nodes of the network of FFRs are considered, and as the objectives of lower level, the fan-out branches and fan-out free primary inputs of the network of FFRs are considered. The processing of formulas (3) for calculation of detectability of faults at fan-out nodes is carried out on the higher level using SSBDDs as in Fig. 2, and for computing the detectability of faults at the inputs of FFRs, we will use the data calculated by gate-level logic simulation. To speed up computing of detectability of faults at the inputs of FFRs, we propose to use S^3BDDs which can be processed in a similar way as SSBDDs. In Fig. 3, an S^3BDD is presented for calculation of the detectability of the faults at the inputs of FFRs. Each entry x in S^3BDD corresponds to a node variable $x(m)$ in the SSBDD in Fig.2, and the path from the particular entry to the terminal node represents an AND-function of conditions needed for detectability of the input variable x of the given FFR. For example, the path in Fig.3 from the entry 3 through the nodes $\neg 2_2$, c, $\neg 7$, a and d to the terminal node #1 corresponds to the detectability condition of detecting the faults at the $input3$ of the FFR in Fig.2.

The set of these detectability AND-functions for all of the input variables of the given FFR can be easily created from the gate-level structure of the FFR. To combine them in a form of S^3BDD like in Fig.3 we can use the algorithm of optimized S^3BDD synthesis developed in [19].

The cost of fault simulation using S^3BDDs can be calculated in terms of the number of operations needed, and is equal to $C = N - N_T$ where N is the number of all nodes in the S^3BDD model, and N_T is the number of end nodes of the model. For the S^3BDD model in Fig.3 we have $C = 17 - 2 = 15$, which is four times less than 63 operations needed for simulation of the SSBDD in Fig. 2. Consider, as an example, the mixed level work share in the computing processes of the level 2 in Table I between SSBDD and S^3BDD models. These processes handle the critical path tracing over the nested configuration of three fan-out re-convergence areas. In the process

$$(z_1, z_3) = R_{z_1}, z_3((z_1, z_{22}), (z_1, z_{12})1), \tag{9}$$

(z_1, z_{22}) is computed at the low-level on the S^3BDD for the FFR with output z_2, whereas R_{z_1},z_3 is computed at the higher level using the SSBDD of z_3 after the following updates of the node variable values: $z_2 2 = z_2 2 \oplus (z_1, z_2 2)$, and $z_1 2 = \neg z_1 2$. On the other hand, in the process

$$D_{z_1} = ((z_1, z_3) \wedge D_{z_{31}}) \vee R_{z_1}, y5((z_1, z_3), (z_1, z_{13})1), \tag{10}$$

$D_{z_{31}}$ is computed at the low-level on the S^3BDD for the FFR with output y_4, whereas R_{z_1},y_5 is computed at the higher level using the SSBDD of y_5 after the following updates: $z_{32} = z_{32} \oplus (z_1, z_3)$, and $z_{13} = \neg z_1$.

Additional side-effect of the mixed-level fault reasoning is the increase of the accuracy in reporting the detected faults. Using the information about the gate-level structure of FFRs, allows to specify the detected faults inside the FFRs. For example, the entries a' and d' in the S^3BDD in Fig.3 are introduced to mark the sub-graphs for calculating the detectability of internal gate-level faults at the nodes a and d, respectively, inside the FFR, presented in Fig.2. Similar entries

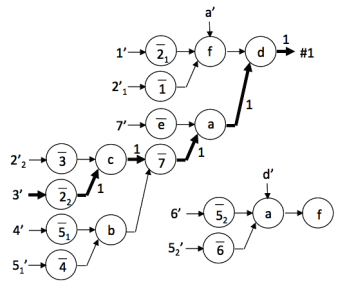


Fig. 3. Direct fault simulation using S^3BDDs.

can be added in Fig.3 for other internal nodes b, c, e, and f in the same FFR.

The speed-up in mixed-level fault reasoning and the increasing accuracy of detected fault reporting is accompanied with additional time cost needed for logic simulation of FFRs at the gate-level. However, when comparing the total times for logic simulation and fault simulation this payload increase will be negligible.

V. Reordering the Computing Model using Levels

As was mentioned earlier circuit partitioning technique into levels for concurrent execution have been already used before [20][22][23]. The level i gate is defined in [20] as one having primary inputs of the circuit and outputs of level k gates as its inputs, such that $k < i$. However in [22], which cites the previous paper the definition is slightly different, stating that level of a gate represents its distance in gates from primary inputs (PI's) of the circuit. This definition is more strict in the sense that one of the inputs of the level i gate, must originate from the level $i-1$, if $i \neq 0$. This difference however is crutial for parallelisation, because the use of the first definition could potentially result in bigger number of levels with fewer gates in them. As levels should be evaluated sequentially - this could decrease the amount of parallelism dramatically. In our case, as we deal with FFRs, we would stick to the second definition and rephrase it for our purpose.

Both, the logic simulation model and the computational model for fault back-tracing described in Section II are presented as networks of partially ordered formulas linked to each other by variables and computed using SSBDDs. Here and throughout the paper we would use the word *variable* to indicate these elements of the circuit. The level of variable is its distance in variables from PI's or, in other words, the level i variable should have at least one of its inputs originating from level i-1 variable, if $i \neq 0$.

In the computational model, the variables are numbered in serial fashion starting at primary inputs and finishing at primary outputs. Variables are serialized such that each input of variable i is the output of variable k, where $k < i$. This is very similar to the first definition of levels from [20].We are using OpenCL framework for parallel execution[21]. Therefore it is necessary to define regions of variables, belonging to the same level, as sub-array. Only variables of particular level must be included into sub-array. If variable x belongs to level i, then level i should be represented as a continuous sequence of variables starting from variable x to variable y, such that every variable z ($x \leq z < y$) belongs to level i and variable y belongs to level $i+1$. This is why it is necessary, to reorder the variables according to our definition of levels. Note that this operation is only required once and does not belong to fault simulation process. The reordered computational model can be saved as a file and used later for simulation, without a need to repeat this step.

Fault model represents segments of critical path to be simulated. Each segment starts at the output of particular variable and ends at primary output of the circuit. Therefore there is one-to-one correspondence between critical path segments to be fault simulated and particular variable. This fact makes it possible to use levelized structure of computational model for Fault model as well. It is important because using levels we could analyze critical path segments starting at the same level in parallel, thus speeding up the fault simulation.

OpenCL framework requires single program for all the parallel devices, which would manipulate on different data. Such program is called kernel. It is executed on all available devices in parallel for all variables inside a single level. The best way to provide the data for kernel is an array. During preparation of the computational model the variable indexes are placed into an array according to their levels. The kernel only requires to know the offset of the level inside the array of variable indexes and the size of this level. Host CPU schedules the kernel executions level by level into the OpenCL execution queue. The execution in the queue is strictly ordered, so that OpenCL driver handles the synchronisation between consecutive kernel executions. This ensures that all variables of the current level have been computed, before moving to the next level.

VI. Results

The experiments were carried out on IBM System x3500 M3 7380 Server (2x 6-core Xeon E5690 running at 3,47Ghz with hyperthreading) using 64-bit Novell SuSe Linux Enterprise Server 11 x86_64. This system has 12 physical CPU cores, 12 virtual hyperthreading cores and 96Gb of RAM. Simulation times were calulated for the sets of 10000 random test patterns. The circuits from three benchmark suites ISCAS'85, ISCAS'89, ITC'99 were simulated. The same circuits as in [12] were chosen in order to compare the results.

Concurrent execution time T_p of PECPT fault simulation can be divided into two parts: $T_p = T_o + T_c$. The first part is the time T_o, which we would call *concurrency overhead*. This is required to make a transition from "single thread"- to "multiple thread"-execution and back again. This time slot involves creation of multiple threads, allocating additional memory, synchronisation at the end of computation and transition back to single thread. The second part is time T_c, which is pure *computation time* required by all threads to deliver a result. This time can be seen in Table II and can be treated as a lower possible bound for concurrent computation. The concurrency overhead T_o depends on the amount of parallel hardware used and increases with number of CPUs. The computation time T_c depends on the amount of computation required. Parallel simulation time $T'_p = T_p + T_{fm} + T_{ff}$, where T_{fm} is the time required to compile the fault model and T_{ff} is the time of fault-free logic simulation.

Table II shows the results of the PECPT execution time T'_p in comparison to PPECPT T_{PPECPT} [14]. As we see, the new method outperforms considerably the previous method, and the gain increases with the size of the circuit (up to the order of magnitude in case of the circuit b19 containing 450 thousands gates). Amount of calculation for small circuits is small, which makes overall execution time T_p large in comparison to computation time T_c. This can be expressed by overhead ratio $R = T_p/T_c$ and is clearly seen from results in Table II. Overhead ratio R for the case of maximum acceleration is also brought in the table to see the concurrency overhead for different circuits. It can be seen from the Table II that overhead ratio is getting closer to one, with growth of the circuit size. In case of circuit b19 the speedup gets almost identical to ideal, because T_p and T_c become almost equal.

Along with execution time there are two speedup values we compute for every benchmark. These are S_p and S_c. Both include single CPU (non-parallel) computation time of fault model T_{tpl} and fault-free simulation T_{ffs} of the circuit. Along with these S_p uses parallel execution time T_p for its computation and S_c uses pure parallel computation time T_c. The equations for speedup values S_p and S_c are as following:

$$S_p = \frac{T_{PPECPT}}{T_{tpl} + T_{ffs} + T_p} = \frac{T_{PPECPT}}{T'_p}$$

$$S_c = \frac{T_{PPECPT}}{T_{tpl} + T_{ffs} + T_c} = \frac{T_{PPECPT}}{T'_c}$$

S_c can be though as topmost ideal case of speedup by PECPT algorithm. It can be seen from the results that smaller circuits achieve

TABLE II. EXECUTION TIMES OF PPECPT AND PECPT.

Circuit	T_{ppecpt}, s	Concurrency overhead (PECPT)				Pure computation (PECPT)		
		T'_p, s	S_p	S_p #cpu	R	T'_c s	S_c	S_c #cpu
c1908	0,0568	0,0846	0,67	6	2,86	0,0330	1,72	5
c2670	0,0405	0,0873	0,46	4	6,52	0,0334	1,21	6
c3540	0,1830	0,1315	1,39	8	1,81	0,0754	2,43	7
c5315	0,0849	0,0922	0,92	4	3,05	0,0487	1,74	5
c6288	1,4610	0,6211	2,35	6	1,61	0,3883	3,76	8
c7552	0,1545	0,1187	1,30	6	1,94	0,0718	2,15	6
s13207	0,1798	0,1332	1,35	5	5,05	0,0857	2,10	10
s15850	0,4714	0,2107	2,24	8	2,34	0,1370	3,44	7
s35932	0,2554	0,1739	1,47	10	1,95	0,1381	1,85	12
s38417	0,7453	0,2427	3,07	12	1,95	0,1869	3,99	12
s38584	0,5945	0,2492	2,39	9	2,43	0,1791	3,32	12
b14	2,7742	0,8752	3,17	8	1,29	0,7300	3,80	9
b15	5,0420	1,1771	4,28	10	1,49	0,9258	5,45	10
b17	14,8550	2,4053	6,18	20	1,29	2,1121	7,03	12
b18	67,3279	7,1499	9,42	24	1,09	6,7738	9,94	24
b19	147,6501	14,4685,	10,20	24	1,03	14,0707	10,49	24

small or negative speedup. On the other hand bigger circuits take advantage of higher number of processors. Such poor result for smaller circuits can be explained by low amount of parallelism accompanied by high overhead ratio R. Both of the factors change positively when circuit size becomes bigger. One of the challenges of this method is that different number of processors is required to achieve maximum speedup for different circuits. The number of processors used to achieve maximum speedup is brought under #cpu columns. It can be seen that this number grows along with the circuit size.

Speedup S_p dependence on the number of processors is shown in Fig.4a (ISCAS'85), Fig.4b (ISCAS'89), Fig.4c (ITC'99). The fluctuation in speedup of some circuits can be explained by the fact, that it is up to OpenCL runtime to decide which processors to use for execution. Because our testsystem has virtual hyperthreading cores they can also be arbitrarily chosen for execution, which could influence the speed of execution in situations where less physical cores are used for computation, although the overall number of cores is bigger. For all the benchmarks we can see that after the limit of physical cores is reached the speedup is starting to decline or stays the same. On the ITC'99 benchmarks b18 and b19 it is slightly increasing, when more than 12 cores are used. This shows that the larger the circuits, the more number of cores can be exploited to achieve the maximum speed-up of simulation.

We compared PECPT to single processor simulators, such as FSIM, PPECPT and commercial simulators C1 and C2. We normalised execution time of all the simulators using previous results from [12] and execution time of PPECPT from Table II, because PECPT was executed on different hardware. The comparison is shown in Table III.

PECPT proves to be in average 3.8 times quicker than FSIM and around two times - than PPECPT for relatively smaller ISCAS benchmarks. The speedup over commercially available simulators is more than eight times over C1 and two orders of magnitude over C2. When ITC'99 benchmark circuits are also taken into consideration the average speedup over PPECPT grows to 4.0 and over C1 even 8.7 times in average, which suggests that simulation of bigger circuits benefits more from our method.

We have also compared PECPT speedup results to GPU based parallel fault simulator and fault table generater GFTABLE [24]. GFTABLE is pattern parallel simulator, which uses bit- and thread-level PP to boost the performance of uniprocessor simulator FSIM. We have used the results from Table 4 in [12] to normalize PECPT speedup. Normalization is required because PECPT speedup is computed in relation to PPECPT, while GFTABLE speedup is computed in relation to FSIM. As there is no FSIM execution time provided for ITC'99 benchmarks, we have taken the average ratio of 1.7 reported in [12] to normalize PECPT results for those circuits.

The Fig. 5 shows speedup in comparison to uniprocessor version of FSIM for both algorithms. We have arranged circuits in sequence where their corresponding number of gates is growing. This way we could clearly see the speedup dependency on the size of the circuit. It can be seen that PECPT proves to be more beneficial on the circuit sizes comparable to ITC'99 benchmarks. For example for circuit c5315 from ISCAS'85 the speedup is 8.03 for GFTABLE and 1.50 - PECPT, while for the ITC'99 circuit b15 the speedup is 2.57 for GFTABLE and 7.28 - PECPT. It is interesting to note that results of the GFTABLE decrease when circuit size is growing, while PECPT in opposite gives less gain in speed-up, while circuit size is smaller.

TABLE III. EXECUTION TIME COMPARISON.

circuit	#fanouts	#branches		Simulation time,s				
		max	avg	fsim	c1	c2	ppecpt	pecpt
c2670	290	28	3,7	0,081	0,223	2,430	0,041	0,087
c3540	356	22	4,5	0,407	1,505	8,745	0,183	0,132
c5315	510	31	5	0,149	0,594	6,047	0,085	0,092
c6288	1456	16	2,6	2,389	5,489	56,072	1,461	0,621
c7552	812	72	4,1	0,348	1,043	11,332	0,155	0,119
s13207	1224	37	3,7	0,225	0,503	6,291	0,180	0,133
s15850	1518	34	3,6	0,943	2,112	19,379	0,471	0,211
s35932	5295	1449	3,4	0,412	1,058	17,477	0,255	0,174
s38417	4569	49	3,2	1,725	3,343	33,007	0,745	0,243
s38584	3946	88	4,5	1,124	2,155	29,727	0,595	0,249
Average speedup				*3,786*	*8,748*	*92,460*	*2,024*	*1,000*
b14	2409	82	4,8	n/a	9,413	n/a	2,774	0,875
b15	2353	95	4,8	n/a	7,411	n/a	5,042	1,177
b17	8145	149	4,8	n/a	22,340	n/a	14,855	2,405
Average speedup				*n/a*	*8,774*	*n/a*	*4,118*	*1,000*

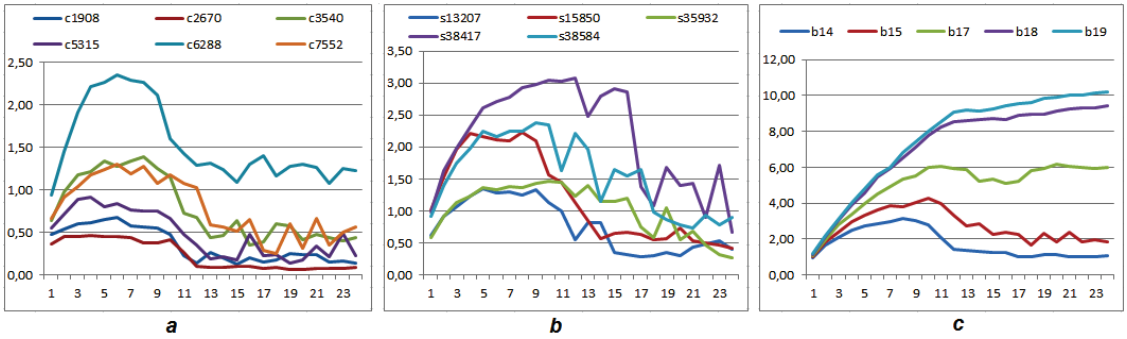
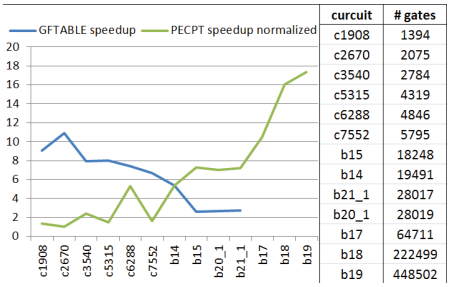


Fig. 4. Speedup vs #CPU for PECPT. a). ISCAS'85 benchmarks, b). ISCAS'89 benchmarks, c). ITC'99.



curcuit	# gates
c1908	1394
c2670	2075
c3540	2784
c5315	4319
c6288	4846
c7552	5795
b15	18248
b14	19491
b21_1	28017
b20_1	28019
b17	64711
b18	222499
b19	448502

Fig. 5. Comparison of GFTABLE and PECPT.

It is stated in [24], that performance of GFTABLE for bigger circuits is influenced by amount of global memory available on GPU. This highlights the scalability bottelneck of the GFTABLE. The results of our approach also depend on the amount of system memory available, but CPU systems in general are more flexible in increasing memory size than GPUs. Even for the circuits, which could fit into GPU memory we can see slight decrease in performance of GFTABLE. Contrary the results of our approach in average become better while circuit size increases.

A. Future work

In order to make the method more practical it is needed to define the number of CPUs involved to provide the best speedup for particular circuit. We believe this can be achieved by further research because the optimum number of CPUs and speedup depend on circuit parameters.

VII. Conclusion

We have proposed a new method for concurrent pattern parallel exact critical backtracing based fault simulation by exploiting circuit processing concurrency. The first time, the parallelization in fault simulation is carried out simultaneously in three dimensions: pattern parallelism, fault parallelism and computing model parallelism, where the pattern- and fault-parallelism are utilized using each single CPU core, while computing model parallelism is achieved using multiple CPUs.

A novel mixed level technique for fault reasoning was proposed to speed up and to increase the accuracy of fault simulation, compared with previous methods.

Experiments showed that the average speed-up compared to the best uniprocessor based simulators is around 3-4 times in average, and up to order of magnitude compared to the available state-of-the-art commercial uniprocessor based simulators. The method is well scaling, the speed up of the method grows with the size of the circuit, opposite to the pattern-parallel simulation method. The reason lies in the memory bottleneck of shared-memory systems, which increases more rapidly for pattern-parallel systems with the growth of the circuit size.

Acknowledgment

The work was supported in part by EU FP7 STREP project BASTION, Estonian ICT project FUSETEST, by EU through the European Structural and Regional Development Funds, by the Estonian Doctoral School in Information and Communication Technology and by the IT Academy Program of Information Technology Foundation for Education.

References

[1] J. A. Waicukauski and et al.,*Fault simulation for structured VLSI*, VLSI Systems Design, pp.20-32, Dec. 1985

[2] B. Underwood, J. Ferguson. *The Parallel Test Detect Fault Simulation Algorithm*. ITC, pp.712-717, 1989

[3] M. Abramovici, P. R. Menon, D. T. Miller. *Critical Path Tracing An Alternative to Fault Simulation*. Proc. 20th Design Automation Conference, pp. 2-5, 1987.

[4] K. J. Antreich, M. H. Schulz. *Accelerated Fault Simulation and Fault Grading in Combinational Circuits*. IEEE Trans. On Computer-Aided Design, Vol. 6, No. 5, pp.704-712, 1987.

[5] F. Maamari, J. Rajski. *A Method of Fault Simulation Based on Stem Region*. IEEE Trans. on CAD, Vol.9, No.2, pp. 212-220, 1990.

[6] D. Harel, R. Sheng, J. Udell. *Efficient Single Fault Propagation in Combinational Circuits*. Int. Conf. on CAD, pp.2-5, 1987.

[7] D. B. Armstrong. *A deductive method for simulating faults in logic circuits*. IEEE Trans. Comp., C-21(5), 464-471, 1972.

[8] E. G. Ulrich, T. Baker. *Concurrent simulator of nearly identical digital networks*. IEEE Trans.on Comp.,7(4), pp.39-44, 1974.

[9] W. T. Cheng, M. L. Yu. *Differential fault simulation: a fast method using minimal memory*. DAC, pp.424-428, 1989.

[10] L. Wu,D. M.H.Walker, *A Fast Algorithm for Critical Path Tracing in VLSI Digital Circuits*, 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems(DFT'05), 3-5 October, 2005,pp.178-186.

[11] R. Ubar, S. Devadze, J. Raik, A. Jutman. *Ultra Fast Parallel Fault Analysis on Structural BDDs*. 12th IEEE European Test Symposium ETS 2007, Freiburg, Germany, May 20-24, 2007.

[12] R. Ubar, S. Devadze, J. Raik, A. Jutman. *Parallel X-Fault Simulation with Critical Path Tracing Technique*. IEEE Conf. Design, Automation & Test in Europe - DATE-2010, Dresden, Germany, March 8-12, 2010, pp. 1-6.

[13] Thayse, *Boolean Calculus of Differences*, Springer Verlag, 1981.

[14] R. Ubar, S. Devadze, J. Raik, A. Jutman. *Parallel Fault Backtracing for Calculation of fault Coverage*. 13th Asia and South Pacific Design Automation Conference - ASP-DAC 2008, Seoul, Korea, Jan. 21-24, 2008, pp. 667-672.

[15] R. Ubar. *Test Synthesis with Alternative Graphs*. IEEE Design and Test of Computers. Spring, 1996, pp.48-59.

[16] R. Ubar. *Combining Functional and Structural Approaches in Test Generation for Digital Systems*. Journal of Microelectronics and Reliability, Elsevier Science Ltd. Vol. 38:3, pp.317-329, 1998.

[17] Minato, S. *BDDs and Applications for VLSI CAD*. Kluwer Academic Publishers. 1996.

[18] D. Mironov, R. Ubar, J. Raik. *Logic Simulation and Fault Collapsing with Shared Structurally Synthesized BDDs*. IEEE European Test Symposium, Paderborn, Germany, May 26-30, 2014.

[19] D. Mironov, R. Ubar. *Lower Bounds of the Size of Shared Structurally Synthesized BDDs*. IEEE 17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS). Warsaw, April, 23-25, 2014, pp. 77-82.

[20] Amin, M.B. ; Vinnakota, B., *Data parallel fault simulation*. International Conference on Computer Design: VLSI in Computers and Processors (ICCD '95), 1995, pp. 610-615.

[21] *OpenCL standard for parallel programming of heterogenious systems*. https://www.khronos.org/opencl/

[22] Varshney, A.K., Vinnakota, B., Skuldt, E., Keller, B., *High Performance Parallel Fault Simulation*. International Conference on Computer Design 2001 (ICCD 2001), 2001, pp. 308-313.

[23] Gulati, K., Khatri, S.P., *Towards Acceleration of Fault Simulation using Graphics Processing Units*. 45th ACM/IEEE Design Automation Conference 2008 (DAC 2008), 2008, pp. 822-827.

[24] Gulati, K. ; Khatri, S.P., *Fault Table Generation using Graphics Processing Units*. IEEE International High Level Design Validation and Test Workshop 2009 (HLDVT 2009), 2009, pp. 60-67.