Multi-Level Timing Simulation on GPUs

Schneider, Eric; Kochte, Michael A.; Wunderlich, Hans-Joachim

Proceedings of the 23rd Asia and South Pacific Design Automation Conference (ASP-DAC'18) Jeju Island, Korea, 22-25 January 2018

doi: https://doi.org/10.1109/ASPDAC.2018.8297368

Abstract: Timing-accurate simulation of circuits is an important task in design validation of modern nano-scale CMOS circuits. With shrinking technology nodes, detailed simulation models down to transistor level have to be considered. While conventional simulation at logic level lacks the ability to accurately model timing behavior for complex cells, more accurate simulation at lower abstraction levels becomes computationally expensive for larger designs.

This work presents the first parallel multi-level waveform-accurate timing simulation approach on graphics processing units (GPUs). The simulation uses logic and switch level abstraction concurrently, thus allowing to combine their advantages by trading off speed and accuracy. The abstraction can be lowered in arbitrary regions of interest to locally increase the accuracy. Waveform transformations allow for transparent switching between the abstraction levels. With the utilization of GPUs and thoughtful unification of algorithms and data structures, a fast and versatile high-throughput multi-level simulation is obtained that is scalable for millions of cells while achieving runtime savings of up to 89% compared to full simulation at switch level.

Preprint

Multi-Level Timing Simulation on GPUs

Eric Schneider, Michael A. Kochte and Hans-Joachim Wunderlich
University of Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany
{schneiec,kochte}@iti.uni-stuttgart.de, wu@informatik.uni-stuttgart.de

***Abstract*—Timing-accurate simulation of circuits is an important task in design validation of modern nano-scale CMOS circuits. With shrinking technology nodes, detailed simulation models down to transistor level have to be considered. While conventional simulation at logic level lacks the ability to accurately model timing behavior for complex cells, more accurate simulation at lower abstraction levels becomes computationally expensive for larger designs.**

This work presents the first parallel multi-level waveform-accurate timing simulation approach on graphics processing units (GPUs). The simulation uses logic and switch level abstraction concurrently, thus allowing to combine their advantages by trading off speed and accuracy. The abstraction can be lowered in arbitrary regions of interest to locally increase the accuracy. Waveform transformations allow for transparent switching between the abstraction levels. With the utilization of GPUs and thoughtful unification of algorithms and data structures, a fast and versatile high-throughput multi-level simulation is obtained that is scalable for millions of cells while achieving runtime savings of up to 89% compared to full simulation at switch level.

***Keywords*—timing simulation; switch level; multi-level; parallel simulation; GPUs**

I. Introduction

Throughout the design of nano-scaled circuits, the timing-accurate simulation of circuits plays an important role for design validation. Accurate simulation has become essential not only for the validation of the circuit timing itself, but also for power estimation and the evaluation of non-functional properties [1, 2] as well as test applications, such as fault simulation [3–5], which all need to be performed as early and as accurate as possible during the design phase. A lot of effort has been put into utilizing more accurate simulation models down to the layout [5–8]. Regarding complex cells, simulation at higher levels lacks the ability of accurately representing functional and timing behavior in order to allow low-level fault injection and reasonably accurate timing analyses [8]. The evaluation at lower levels, i.e., switch level and analog simulation in SPICE, is thus crucial for these applications. But the wider modeling capabilities and higher accuracy comes at the cost of drastic runtime complexity. Despite the parallelization of these simulations [9, 10], their application to larger designs remains still expensive.

In order to exploit the advantages of both high- and low-level simulation approaches, combined simulation across multiple abstraction levels has been proposed that trade-off speed and accuracy [2, 7, 11–14]. Multi-level approaches that are based on the use of low-level pre-characterization, aggregate and abstract electrical properties of cells [15], which are then utilized by higher-level simulators [2, 7]. The behavior of cells in different parameter corners as well as faults and patterns is thereby stored offline. These characterization steps have to be repeated for every new corner or cell that needs to be considered. Hierarchical multi-level solutions partition the circuit into regions for separate high- and low-level simulation [11–14], while the low-level simulation is limited to smaller parts in a design that are of particular interest. During the evaluation, simulation data is then exchanged at the boundaries between the different abstraction levels.

With the introduction of general purpose computing on graphics processing units (GPUs), the first circuit and fault simulation approaches have been proposed [16–21] that are able to vastly accelerate the simulation by parallelization through programs called *kernels*. These methods distribute the evaluation of independent circuit structures under different input stimuli to independent threads running on the many processing elements of the GPU. Although high simulation speed-ups of up to three orders of magnitude have been reported, the underlying programming paradigm involves certain restrictions that need to be tackled carefully, especially when moving to lower levels and mixing abstractions. Since memory transfers and synchronizations are costly compared to the execution of bare arithmetic instructions, accesses should be minimized or coalesced as much as possible and all threads should run isolated with uniform control flow and data structures in order to sustain high computing performance [16].

This work presents the first multi-level timing simulation approach on GPUs that combines the evaluation of higher and lower abstraction levels transparently for efficient parallelization on many-core architectures. The abstraction level is switched during simulation in user-defined regions of interest to enable a more accurate modeling of the functional and timing behavior if required, while reducing the overall overhead of time-consuming low-level evaluations. By thoughtful unification of the models, similarities in data structures and algorithms can be exploited for an efficient parallelization of the execution being applicable even to larger designs.

The following section provides background on state-of-the-art GPU timing simulators. Section III introduces our novel multi-level simulation approach for GPUs. In Section IV, the handling of signals across the different abstraction levels is explained. Finally, experimental results regarding the scalability are discussed in Section V.

II. Background

Timing-accurate simulation at logic level is usually done in an event-driven manner based on the time wheel [22], that utilizes dynamic lists to schedule switching events at discrete points in time to compute the full switching histories (*waveforms*) of all cells. While general event-driven time simulation approaches allow to reduce simulation overhead, their parallelization is complex and requires thorough list management and synchronization at the cost of memory and performance [23]. In [24] a plain and oblivious time simulation on GPUs is proposed that computes signal waveforms under consideration of individual pin-to-pin delays for rising and falling transitions and also employs pulse filtering. The computational complexity is hidden by exploitation of massive high-throughput parallelization from structural and

data-parallelism that allows to utilize the high floating-point computing throughput of the GPU cores.

Typically, logic level timing simulators utilize timing data obtained from annotations in standard delay format (SDF), and establish look-up tables of gate delays to quickly retrieve a constant signal propagation delay for given input transitions as denoted in Fig. 1a). However, for complex cells the complexity of the timing descriptions quickly grows, due to conditional delays. Also, in case multiple transitions occur at the same time (e.g., rising transitions at all inputs of a NOR-gate), the simulation models typically pick the minimum delay over all inputs, completely ignoring low-level effects, such as simultaneous input switching found in CMOS cells [25].

In the switch level timing simulator of [21], the cell descriptions and evaluation algorithms are based on first-order CMOS parameters. Each transistor of a cell is viewed as a threshold-based binary switch that changes its internal resistance based on the applied input. Pull-up nets and pull-down nets are then viewed as input-controlled voltage dividers that charge the output load capacitance of a cell. Thus, the timing behavior is described in terms of RC-characteristics allowing to model relevant CMOS-related delay effects. Fig. 1b) depicts continuous signal slopes at input and output of an inverter cell with the output signal being a function of the *time constant* τ that has been derived from its RC-characteristics.

For the evaluation of a design (e.g., along the critical path), not all cells need to be simulated with lowest abstraction all the time, but rather only when high accuracy is actually required [11–14]. This leads to the development of multi-level approaches, by sharing inputs and outputs between fast high- and accurate low-level models in order to trade-off speed versus accuracy. For example, in [13] a simulator combining transaction level and logic level has been developed. By limiting the accurate and computationally more expensive evaluation to smaller regions, speed-ups of up to four orders of magnitude are achieved.

However, multi-level simulation in the GPU context often conflicts with the underlying many-core programming paradigm, since high- and low-level algorithms typically utilize different data structures and algorithms and can also involve working sets with large memory footprint. Excessive data transfers from GPU devices to host systems and synchronization routines cause performance penalties and should be avoided at all cost. Thus, in order to realize an efficient multi-level simulation, a thoughtful unification and organization of both algorithms and data structures is necessary.

III. Parallel Multi-Level Time Simulation

In this work, a multi-level simulation approach is presented that provides the waveform-accurate logic level time simulation with the ability to increase the accuracy to switch level wherever and whenever required. While the logic level abstraction provides a faster evaluation of the timing, the switch level abstraction allows to model and evaluate the behavior and effects of CMOS cells with transistor granularity, such as multiple-input switching and signal slopes. For the logic and switch level simulation, the simulation concepts of [24] and [21] are adopted. The logic descriptions of the cells use SDF-annotated timing. The first-order parameters of the switch level description are extracted from the descriptions of library cells and SPICE simulations of the transistor models. The transitioning between the abstractions is organized by distinguished *regions of interest* (ROI). Any node in the circuit

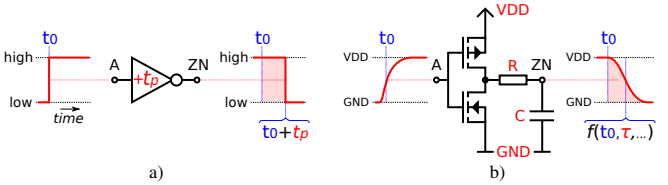

Fig. 1. Example of signals in a) logic level and b) switch level simulation of an inverter cell for comparison.

graph (input, output or cell) can be marked as *ROI*, causing its simulation to be performed with switch level accuracy. The set of active ROIs in a simulation instance will be referred to as a *ROI group*, which can represent a set of single isolated nodes, full paths, input or output cones as well as arbitrary sets of nodes.

A. Overview

The multi-level simulation is outlined in Fig. 2. The simulation is composed of two phases: a pre-processing (step 1–3) and the actual simulation phase (step 4–6). During pre-processing, the combinational netlist is extracted from the synthesized design and the timing is annotated (step 1). The cells in the netlist are topologically sorted (step 2) and the resulting levelized netlist format allows the simulator to process the design in a single pass. Once initialized, the user can specify ROI groups to be simulated with switch level accuracy (step 3).

The defined ROI groups are then provided as input to the simulator where each group is processed in an individual simulation loop (step 4–6). Starting with each loop, the simulator marks all cells in the current group (step 4) to be simulated at lower level during simulation and the node descriptions are updated accordingly during the process. Then, input stimuli are applied to the circuit inputs and the cells in the levelized netlist are simulated level by level in topological order from inputs to outputs (step 5). The switching between the different abstraction levels is completely transparent, as waveform representations are transformed during the evaluation of a cell. Eventually, the signal waveforms of the circuit outputs are computed and evaluated given a user-defined sample time. After simulation, the marked cells are restored for simulation of the next group of ROIs.

B. Data Structures

The implemented multi-level approach processes logic and switch level abstractions interchangeably throughout the simulation. To enable an efficient simulation on the GPU, both the logic and switch level kernels access the same memory for input and output data. Hence, waveforms of both types as well as cell descriptions of different abstraction levels are present in the GPU device memory.

The signal switching histories of the logic level timing simulation are represented by binary waveforms, each of which is modeled as a sequence $(t_0, t_1, ...)$ of temporally ordered time points $t_i \in \mathbb{R}^+$ that indicate *toggles* of the respective signal value. Similarly, in the switch level simulation the waveforms are modeled as sequences $(p_0, p_1, ...)$ of so-called *pivots* [21]. A pivot p_i comprises parameters that describe the signal change of a cell output as a function over a continuous time interval $[t_i, t_{i+1}]$. Exponential curve segments are utilized as pivot function to closely resemble the (dis-)charging processes of RC-subcircuits caused by input signal changes. A voltage waveform of a signal is then formed

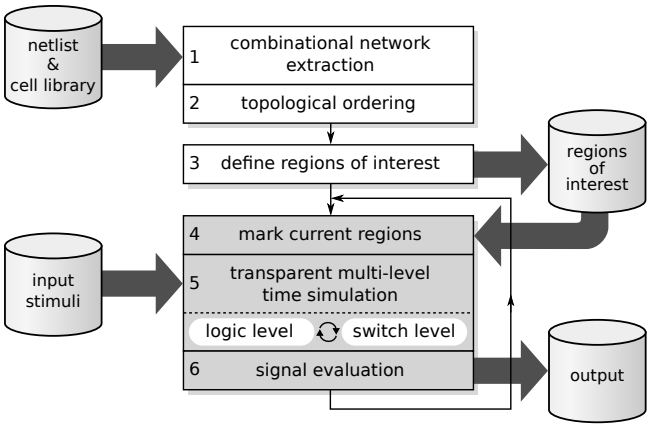

Fig. 2. Flow-chart of the overall multi-level simulation algorithm.

by cascading pivot segments into a piece-wise function that approximates the changes in the voltage over time. Each pivot in the switch level model is represented by a compact tuple $p_i = (t_i, \overline{v}_i, \tau_i)$ composed of a time point t_i denoting the start of the i-th curve segment, as well as a stationary voltage $\overline{v}_i$ the exponential curve segment is heading to for the time after t_i. Finally, a time constant τ_i expresses the steepness of the curve. This constant depends on the transistor states and is calculated during execution of the switch level algorithm from the parametric capacitances and resistances inside of the cell, as well as the capacitive load connected to the output terminal. The signal voltage is then described by the corresponding curve segment in the interval $t \in [t_i, t_{i+1}]$ given [21]:

$$w(t) := (w(t_i) - \overline{v}_i) \cdot e^{-\frac{t-t_i}{\tau_i}} + \overline{v}_i, \quad t_i \leq t \leq t_{i+1}. \qquad (1)$$

This way, voltage waveforms are modeled continuously in time and value as shown in Fig. 3, allowing to consider effects like multiple-input switching and varying signal slopes with high accuracy compared to SPICE.

Since during simulation the level of abstraction can change, the type of input and output waveforms must be identified and handled accordingly. If the type of an input waveform does not match the abstraction of the current cell, the switching events in the respective waveform are transformed to the required abstraction level during execution on the GPU without interaction of the host process (cf. Section IV).

C. Evaluation Kernel

Algorithm 1 outlines the implemented multi-level evaluation of a circuit. The inputs are a levelized netlist description G, where each cell is described at either logic or switch level, as well as stimuli for all circuit primary and pseudo-primary inputs stored in a memory W. The cells of the circuit netlist are then simulated level by level from inputs to outputs, with all nodes on a level being evaluated by different threads in parallel similar to [21, 24]. The evaluation of each cell is performed by a mergesort algorithm that sorts all local switching events at the inputs in temporal order, such that all events are processed in a single pass.

For each cell, first the waveforms at its inputs are fetched (line 3). The type of each input waveform is then determined and the respective data structures for the input processing are initialized (line 4–8). During the process, the initial signal value of each waveform is determined in order to initialize the state of the cell and the output waveform (line 9). The events of all input waveforms are then processed in

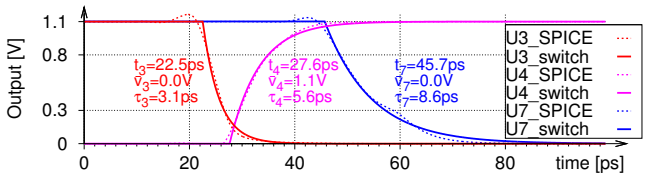

Fig. 3. Signal transitions from SPICE transient analysis (dotted) of a small example circuit represented as pivot curve primitives (bold) at switch level.

the main cell simulation loop (line 10–23) in temporal order from earliest to latest by using a cell-local schedule E, that keeps the immediate next event to be processed for each input waveform. The earliest next input event is *consumed*, indicating a change in an input signal value. All implications of the value change on the cell state are transformed to the targeted abstraction level of the current cell. If the current cell to be evaluated is marked as a region of interest (ROI), the low-level *switch level* simulation kernel is called in order to process the state change (line 14), otherwise the logic level kernel is used (line 17). If an input event causes a change in the output signal value, a new switching event is appended to the output waveform (line 20). After processing the event, the next event in the input waveform is determined and scheduled for evaluation in the loop (line 22). When all events have been processed, the main simulation loop terminates and the output waveform of the cell is stored (line 24).

Algorithm 1: Transparent multi-level circuit simulation.

```
   Input: netlist G, primary input waveforms (stored in W)
1  foreach level L in the netlist G do
2      foreach node n on level L (in parallel) do
3          Load input waveforms I ⊆ W for node n.
4          foreach waveform w_i in I do
5              Look-up abstraction level of w_i.
6              Set-up data structures and initial state.
7              Get first event e of w_i and put into schedule E.
8          end
9          Initialize output waveform w_n.
10         while Events to process in schedule E do
11             Remove earliest event e from E.
12             if node n is ROI then
13                 Transform e to switch level event.
14                 Compute new switch level state of n.
15             else
16                 Transform e to logic level event.
17                 Compute new logic level state of n.
18             end
19             if new state of n causes output change then
20                 Compute output event and add to w_n.
21             end
22             Get next event e of w_i ∈ I and put into E.
23         end
24         Store w_n to waveforms W := W ∪ {w_n}.
25     end
26 end
```

To simplify the memory management on the GPU, all waveforms have a specified capacity for storing events. Similar to [26, 27], *overflow checks* are performed throughout simulation ensuring that all switches are contained. In case overflows did occur on a level, additional memory is allocated for the culprit waveforms and the simulation of the level is repeated. This causes some overhead due to memory management, that quickly diminishes after processing a few stimuli.

D. Parallelization

A multi-dimensional parallelization scheme is adopted to speed up the simulation [26]. The simulation kernels exploit structural parallelism from cells on the same level, that can be

processed concurrently due to input and output independence, as well as waveform parallelism from the different pattern pairs (waveform stimuli) to be evaluated.

For the parallel execution on a GPU, all simulation kernels create *grids of threads* on the devices as depicted in Fig. 4. The grids are two-dimensional arrays of threads, with each thread computing the respective output waveform of one cell for a particular waveform input. In the vertical dimension, each thread computes the function of a different, but data-independent cell, whereas in the horizontal dimension, the cell is concurrently evaluated for different input stimuli. If a cell is marked as ROI, the respective threads will execute the switch level algorithm (denoted by '$*$').

On the GPU, threads are scheduled for parallel execution in so-called *batches*. The threads of each batch are then processed in a *single instruction multiple data* (SIMD) fashion on the GPU multi-processing elements. All threads within a batch have been aligned to evaluate the same cell, but for different input stimuli, which allows to sustain the memory coalescing properties for memory accesses [24]. Even though the abstraction levels can be mixed arbitrarily throughout the circuit, no further control-flow divergence is caused, since the abstraction of the cell, and hence the used functions for the waveform transformation and evaluation, remains the same for all the threads of a batch.

The indices of threads within the grid structure are utilized to navigate and coalesce memory accesses for efficient access to circuit data and waveform storage. Furthermore, by identifying topological dependencies between different ROI groups, the simulation parallelism can be further enhanced by simultaneous processing of multiple ROI groups. ROI groups that do not share common output logic can be activated and evaluated concurrently in one and the same simulation loop.

IV. Waveform Transformation and Evaluation

The transformation between the different waveform formats is done bidirectionally from high abstraction to lower abstraction and vice versa during cell evaluation. We utilize two mappings to transform between high-level waveforms of discrete logic values and low-level waveforms with continuous voltage levels. The mappings are applied to the input waveforms events during the evaluation of a cell for calculating the implications and state changes on its respective abstraction level.

A. Logic to Switch Level Transformation

At logic level, the signal transitions are considered to be rectangular and binary in value (*high* or *low*). These transitions are modeled by an infinitely small time constant $\tau_\varepsilon > 0$ at switch level, which allows to approximate instantaneous transitions with negligible error. Given the VDD and GND voltage levels of the targeted technology, the initial pivot is set in the output waveform to specify the initial voltage level of the cell. All transitions at times $(t_0, t_1, ...)$ in the original binary waveform are then translated by substitution of the signal toggles t_i one after another with pivots. Based on the targeted logic value of a transition, the stationary voltage of its corresponding pivot p_i is selected as follows:

$$t_i \mapsto p_i := \begin{cases} (t_i, \mathrm{VDD}, \tau_\varepsilon) & \text{if } (t_i \textit{ rising}), \\ (t_i, \mathrm{GND}, \tau_\varepsilon) & \text{elif } (t_i \textit{ falling}), \\ (t_i, \frac{\mathrm{VDD}+\mathrm{GND}}{2}, \tau_\varepsilon) & \text{else.} \end{cases} \quad (2)$$

However, such instantaneous transitions never occur in CMOS circuits. To provide more appropriate input waveforms

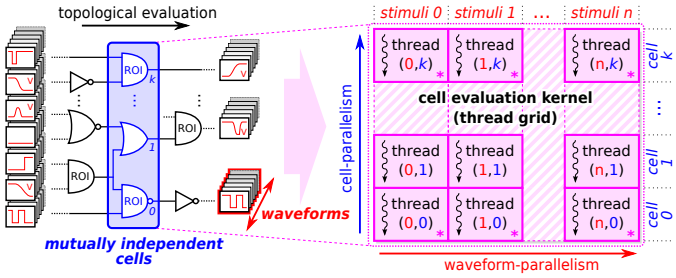

Fig. 4. Two-dimensional parallel evaluation of multiple data-independent cells and stimuli of varying abstraction in a topologically ordered netlist.

for the cell under investigations, the RC-properties of the driving cells are extracted from cell library and layout. These are utilized to adjust the time point of the switching as well as input slope. For this, we assume the standard definition of the propagation delay of the driving cell. Thus, for binary output toggles, the original signal, i.e., at electrical level, is assumed to have passed the $\frac{\mathrm{VDD}+\mathrm{GND}}{2}$ voltage level. Suppose the signal value of the waveform w at time t_i is $w(t_i) = 0$ and a new curve segment $p_i = (t_i, \overline{v}_i, \tau_i)$ starts charging the signal line to voltage $\overline{v}_i$ with time constant τ_i. Given a particular threshold voltage level V_{th}, the time x when the curve segment meets the threshold is determined by [21]:

$$x := t_i - \tau_i \cdot \log\left(\frac{V_{th} - \overline{v}_i}{w(t_i) - \overline{v}_i}\right). \quad (3)$$

Suppose at time x a binary switch occurs in logic simulation. The above equation is transformed in order to fit a time parameter t_i of the targeted curve segment p_i according to the RC-characteristics of the driving cell given the parameter τ_i to match the transition time x. While $w(x)$ represents the initial signal value at the time of the transition, hence, $w(x) = \mathrm{GND}$ (VDD) for rising (or falling, respectively), the threshold value V_{th} is set to the 50 percent voltage level given by $\frac{\mathrm{VDD}+\mathrm{GND}}{2}$. The fitted parameters are then obtained by transforming and solving Eq. (3) for t_i:

$$t_i := x + \tau_i \cdot \log(0.5). \quad (4)$$

The solution is then utilized as the respective starting point of the pivot segment for either rising $(t_i, \mathrm{VDD}, \tau_i)$ or falling transitions $(t_i, \mathrm{GND}, \tau_i)$. The curve of this pivot reflects the transient response of an RC-element, thus allowing for a more realistic input representation for the targeted cell. The resulting waveforms of the transformations with infinitely small time constant ("τ_ε *trans.*") as well as with consideration of RC-characteristics to match the 50 percent voltage level ("*RC 50% VDD*") are shown in Fig. 5. Note that the waveform transformation using τ_ε overlaps the logic level representation ("*source*") and closely matches with negligible error ($< \tau_\varepsilon$) according to Eq. (4).

B. Switch to Logic Level Transformation

Once a region of interest has been simulated, the obtained low-level signal information has to be transformed again prior to the continuation of the higher-level simulation. During simulation, the output signals of cells might show intermediate voltage levels, which do not correspond to well-defined *high* or *low* logic values. Since these intermediate values might be interpreted differently by the subsequent stages, an *unknown* (X) value will be assumed. Thus, we use a threshold interval $(V_{thL}, V_{thH}) \subset [\mathrm{GND}, \mathrm{VDD}]$, which is bounded by a *low* (V_{thL}) and *high* (V_{thH}) signal threshold in order to

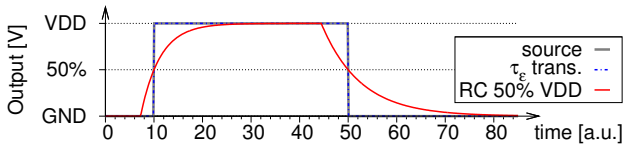

Fig. 5. Transformation of a source waveform from logic to switch level description with varying RC-characteristics for rising and falling transitions.

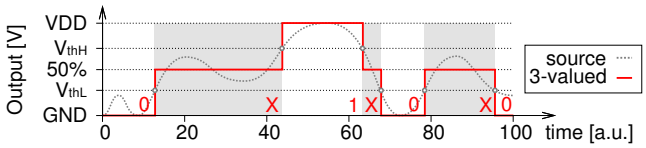

Fig. 6. Threshold-based transformation ($V_{thL} < 50\% < V_{thH}$) of an arbitrary continuous signal to a ternary logic waveform with *unknowns*.

map voltages to logic values. The mapping of arbitrary voltage values w of a continuous waveform to a logic symbol is as follows [28]:

$$val : \mathbb{R} \to \{0, 1, X\}, val(w) := \begin{cases} 0 & \text{if } w \leq V_{thL}, \\ 1 & \text{elif } w \geq V_{thH}, \\ X & \text{else.} \end{cases} \quad (5)$$

During waveform transformation, the pivot elements of the source signal are processed from earliest to latest. Within each curve interval, the times of all possible intersection points with the V_{thL} and V_{thH} thresholds are identified. For each point, the targeted logic level waveform is assumed to switch its value according to Eq. (5) at the given times.

An example of a low-level signal transformation from a continuous signal waveform to logic level using a threshold interval is illustrated in Fig. 6. After any intersection point of a signal with a threshold level has been determined, the logic value is classified from its current value and applied to the waveform. In addition, pulse filtering can be used to remove glitches that are unreasonable or physically impossible, even in case of X-pulses.

The logic level timing model supports pin-to-pin delays for rising and falling transitions [24]. In order to cope with *unknown* signal values, the waveform modeling was extended by a three-valued logic $\mathbf{E}_3 = \{0, 1, X\}$ [29]. The propagation of unknowns (X) during logic level simulation is done pessimistically. If a cell enters an undefined state, due to an input change, the minimum propagation delay of the arriving pin is applied. On the other hand, if the cell output transitions to a defined state, the maximum propagation delay of the pin is used. Unknown values that occur on an input of a cell are masked by defined controlling off-path signal values.

V. Experimental Results

The developed multi-level simulation approach has been evaluated on designs from the ISCAS'89 and ITC'99 benchmarks, as well as designs provided by NXP. All designs have been synthesized using a 45nm standard-cell library. Full-scan design is assumed, hence only the combinational structures of the designs are considered during simulation. In order to provide reasonable input stimuli, n-detect transition-fault test pattern pairs have been generated using a commercial ATPG tool ($n = 10$). Each circuit has been simulated with different numbers of selected regions of interest (ROIs) that have been randomly distributed over the design. All experiments were conducted on a host system (eight Intel® Xeon® processors clocked at 3.0GHz and 128GB of RAM) equipped with NVIDIA® Tesla® K80 GPU-accelerator cards, with each GPU device having 2496 cores clocked at 875MHz with 12GB of global device memory. However, only one Xeon® processor core and one GPU device were used at a time during each simulation. Regarding the circuit data, the maximum memory occupied on the GPU for storing the description of circuit p3881k with 3.7 million nodes was 360MB, which is roughly 3% of the available global device memory.

Table I summarizes the runtime impact of the ROI activation during simulation. For each circuit, the size of the design in nodes (cells and input/output ports) and the number of applied input stimuli pairs are given in columns 2 and 3. Column 4 shows the runtime of a serial commercial event-driven logic simulation simulating the provided stimuli set. As for the presented GPU-accelerated multi-level simulation, absolute runtimes for different ROI counts are given in columns 5–14 along with the relative savings compared to the full switch level simulation on GPU. A plain logic simulation on the GPU without any active ROIs is shown in column 5. The number of ROIs then consecutively increases throughout columns 6–14. In columns 6–8, an absolute number of random ROIs (*"#Nodes"*) has been activated, whereas for columns 9–13, a relative amount of the total nodes was chosen. Column 14 represents the full switch level simulation. All times have been extracted and averaged from three independent runs with different random ROI distributions each. Finally, a simulation scenario (*"Longest Paths"*) was investigated where ROIs have been activated along circuit paths with less than 20% slack with respect to the nominal circuit delay in order to process signals along longest paths with lower abstraction. The number of ROIs in percentage of all nodes as well as the respective runtime of the simulation are reported in the last two columns. All runtimes are compared to the full-switch level simulation on the GPU (Col. 14) to show the relative runtime savings.

As shown in the table, the ratio of the runtime between logic level (Col. 5) and switch level (Col. 14) ranged up to 10.5× (for b18). Besides minor random fluctuations, the runtimes of the mixed abstraction scaled linearly with the amount of active ROIs from lowest to highest between the logic and full switch level simulation runtimes. Some systematic runtime abnormalities have been observed in the pure switch level simulation, which sometimes showed lower runtime compared to the simulation with a smaller number of ROIs (e.g., 50–75% of the total nodes). Here, the signal slopes in the waveforms at switch level caused a more aggressive pulse-filtering which in turn reduced the simulation time. Yet, by activation of fewer ROIs, the multi-level approach allows to avoid full switch-level simulation providing runtime savings of over 80% (up to 89% for single ROI activation). Even for multi-million cell circuits, the maximum average simulation time per pattern pair was 90ms for p3726k, and usually in the range of few milliseconds for the other circuits. The peak simulation throughput measured during the experiments was 444 million node evaluations per second (MEPS).

VI. Conclusion

This paper presents the first high-throughput multi-level time simulation for efficient parallel execution on graphics processing units (GPUs). It utilizes mixing of different abstractions by combining fast waveform-accurate high-level simulation at logic level with low-level switch level modeling. The presented approach exploits similarities in data structures and execution patterns of the simulation models and transitions between abstractions in user-defined regions of interests to

TABLE I
IMPACT OF ACTIVE REGIONS OF INTEREST (ROI) ON RUNTIME AND RUNTIME SAVINGS OF THE GPU-ACCELERATED MULTI-LEVEL SIMULATION.

Circuit[1]	Nodes[2]	Pattern-Pairs[3]	Comm. Event-Driven[4]		Full Logic-Level[5]	Mixed Abstraction								Full Switch-Level[14]	Longest Paths	
						Active ROIs (#Nodes)			Active ROIs (% of total Nodes)						ROIs[15]	Time[16]
						1[6]	10[7]	100[8]	1%[9]	10%[10]	25%[11]	50%[12]	75%[13]			
s38417	19.0k	348	5.49s	*runtime:*	36ms	52ms	53ms	58ms	55ms	76ms	110ms	126ms	152ms	173ms	15.63%	85ms
				saving:	78.8%	70.0%	69.0%	66.5%	68.3%	55.8%	36.2%	26.9%	12.1%	–		51.0%
s38584	23.1k	563	14.10s	*runtime:*	51ms	72ms	73ms	78ms	85ms	126ms	148ms	185ms	216ms	266ms	10.87%	98ms
				saving:	80.9%	73.0%	72.3%	70.5%	68.1%	52.6%	44.4%	30.4%	18.8%	–		63.2%
b17	42.8k	2135	2:17m	*runtime:*	227ms	250ms	257ms	265ms	322ms	537ms	674ms	1.08s	1.52s	1.80s	7.36%	419ms
				saving:	87.4%	86.1%	85.7%	85.3%	82.1%	70.1%	62.5%	39.8%	15.3%	–		76.7%
b18	125.3k	3174	0:13h	*runtime:*	922ms	1.03s	1.12s	1.02s	1.32s	2.16s	3.23s	4.91s	7.07s	9.69s	11.81%	3.68s
				saving:	90.5%	89.3%	88.5%	89.5%	86.4%	77.7%	66.6%	49.3%	27.0%	–		62.0%
b19	250.2k	4651	0:41h	*runtime:*	2.62s	2.89s	3.12s	3.46s	3.95s	7.22s	9.00s	14.26s	20.62s	27.36s	12.63%	10.39s
				saving:	90.4%	89.4%	88.6%	87.3%	85.6%	73.6%	67.1%	47.9%	24.6%	–		62.0%
p951k	1.09M	7063	4:18h	*runtime:*	24.20s	26.62s	27.81s	28.37s	27.93s	38.87s	51.18s	1:13m	1:37m	2:06m	1.81%	32.76s
				saving:	80.9%	78.9%	78.0%	77.6%	77.9%	69.3%	59.5%	41.9%	23.0%	–		74.1%
p1522k	1.09M	17980	12:34h	*runtime:*	50.60s	53.09s	47.46s	54.62s	1:04m	1:56m	2:23m	3:23m	4:41m	6:03m	0.41%	49.47s
				saving:	86.1%	85.4%	87.0%	85.0%	82.3%	67.9%	60.5%	44.0%	22.5%	–		86.4%
p2927k	1.67M	22107	28:54h	*runtime:*	1:41m	1:44m	1:56m	1:42m	2:14m	3:50m	4:37m	6:30m	8:44m	0:11h	0.70%	1:51m
				saving:	85.0%	84.6%	82.9%	84.9%	80.2%	66.1%	59.3%	42.6%	22.9%	–		83.5%
p3188k	2.85M	26502	48:28h	*runtime:*	3:00m	3:18m	3:18m	3:22m	3:53m	7:00m	9:38m	0:15h	0:20h	0:22h	0.13%	3:27m
				saving:	86.6%	85.2%	85.2%	84.9%	82.7%	68.7%	57.0%	31.4%	6.6%	–		84.5%
p3726k	3.56M	15512	48:14h	*runtime:*	2:23m	2:39m	2:33m	2:40m	4:46m	7:39m	9:45m	0:16h	0:23h	0:18h	0.14%	3:57m
				saving:	87.1%	85.6%	86.2%	85.6%	74.3%	58.8%	47.4%	11.6%	-25.7%	–		78.6%
p3847k	2.96M	31653	48:47h	*runtime:*	4:48m	4:51m	4:47m	5:01m	5:34m	9:11m	0:11h	0:16h	0:23h	0:31h	0.58%	4:42m
				saving:	84.5%	84.4%	84.6%	83.8%	82.0%	70.3%	62.9%	46.3%	25.7%	–		84.8%
p3881k	3.69M	12092	27:31h	*runtime:*	2:18m	2:13m	2:20m	2:14m	2:35m	4:08m	5:35m	7:32m	0:10h	0:13h	0.08%	2:05m
				saving:	83.2%	83.8%	82.9%	83.7%	81.1%	69.8%	59.3%	45.0%	23.9%	–		84.8%

selectively trade simulation speed for accuracy during execution. The transformation of waveform representations between abstraction levels allows for fast and transparent evaluation that enables efficient and timing-accurate multi-level timing simulation of circuits. It is applicable to designs with millions of cells and achieves runtime savings of up to 89% compared to full switch level simulation on GPU.

ACKNOWLEDGMENT

This work has been funded by the German Research Foundation (DFG) under the projects WU 245/16-1 and WU 245/17-1.

REFERENCES

[1] W. Huang, S. Ghosh, S. Velusamy *et al.*, "HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design," *IEEE Trans. on Very Large Scale Integration Systems (TVLSI)*, vol. 14, no. 5, pp. 501–513, May 2006.

[2] J. Jiang, M. Aparicio, M. Comte *et al.*, "MIRID: Mixed-Mode IR-Drop Induced Delay Simulator," in *Proc. 22nd Asian Test Symp. (ATS)*, Nov. 2013, pp. 177–182.

[3] M. Tehranipoor, K. Peng, and K. Chakrabarty, *Test and Diagnosis for Small-Delay Defects*. Springer New York, 2011.

[4] A. Czutro, N. Houarche, P. Engelke *et al.*, "A Simulator of Small-Delay Faults Caused by Resistive-Open Defects," in *Proc. 13th European Test Symp. (ETS)*, May 2008, pp. 113–118.

[5] A. D. Singh, "Cell Aware and Stuck-Open Tests," in *Proc. IEEE 21st European Test Symp. (ETS)*, May 2016, pp. 1–6, Paper 15.1.

[6] C. Sebeke, J. P. Teixeira, and M. J. Ohletz, "Automatic Fault Extraction and Simulation of Layout Realistic Faults for Integrated Analogue Circuits," in *Proc. European Conf. on Design and Test (EDTC)*, Mar. 1995, pp. 464–468.

[7] F. Hapke, W. Redemund, A. Glowatz *et al.*, "Cell-Aware Test," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 33, no. 9, pp. 1396–1409, Sep. 2014.

[8] H. H. Chen, S. Y.-H. Chen, P.-Y. Chuang, and C.-W. Wu, "Efficient Cell-Aware Fault Modeling by Switch-Level Test Generation," in *Proc. IEEE 25th Asian Test Symp. (ATS)*, Nov. 2016, pp. 197–202.

[9] K. Gulati, J. F. Croix, S. P. Khatri, and R. Shastry, "Fast Circuit Simulation on Graphics Processing Units," in *Proc. 14th Asia and South Pacific Design Automation Conf. (ASP-DAC)*, Jan. 2009, pp. 403–408.

[10] L. Han, X. Zhao, and Z. Feng, "TinySPICE: A Parallel SPICE Simulator on GPU for Massively Repeated Small Circuit Simulations," in *Proc. ACM/EDAC/IEEE 50th Design Automation Conf. (DAC)*, May 2013, pp. 1–8, Article 89.

[11] S. Gai, P. L. Montessoro, and F. Somenzi, "MOZART: A Concurrent Multilevel Simulator," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 7, no. 9, pp. 1005–1016, Sep. 1988.

[12] W. Meyer and R. Camposano, "Active Timing Multilevel Fault-Simulation with Switch-Level Accuracy," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 14, no. 10, pp. 1241–1256, Oct. 1995.

[13] M. A. Kochte, C. G. Zöllin, R. Baranowski *et al.*, "Efficient Simulation of Structural Faults for the Reliability Evaluation at System-Level," in *Proc. IEEE 19th Asian Test Symp. (ATS)*, Dec. 2010, pp. 3–8.

[14] R. S. Khaligh and M. Radetzki, "Modeling Constructs and Kernel for Parallel Simulation of Accuracy Adaptive TLMs," in *Proc. Conf. on Design, Automation Test in Europe (DATE)*, Mar. 2010, pp. 1183–1188.

[15] F. J. Ferguson and J. P. Shen, "A CMOS Fault Extractor for Inductive Fault Analysis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 7, no. 11, pp. 1181–1194, Nov. 1988.

[16] K. Gulati and S. P. Khatri, "Towards Acceleration of Fault Simulation using Graphics Processing Units," in *Proc. ACM/IEEE 45th Design Automation Conf. (DAC)*, Jun. 2008, pp. 822–827, Paper 45.1.

[17] D. Chatterjee, A. DeOrio, and V. Bertacco, "Event-Driven Gate-Level Simulation with GP-GPUs," in *Proc. ACM/IEEE 46th Design Automation Conf. (DAC)*, Jul. 2009, pp. 557–562.

[18] M. A. Kochte, M. Schaal, H.-J. Wunderlich, and C. G. Zoellin, "Efficient Fault Simulation on Many-Core Processors," in *Proc. ACM/IEEE 47th Design Automation Conf. (DAC)*, Jun. 2010, pp. 380–385, Paper 23.4.

[19] M. Li and M. S. Hsiao, "3-D Parallel Fault Simulation With GPGPU," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 10, pp. 1545–1555, Oct. 2011.

[20] S. Holst, E. Schneider, and H.-J. Wunderlich, "Scan Test Power Simulation on GPGPUs," in *Proc. IEEE 21st Asian Test Symp. (ATS)*, Nov. 2012, pp. 155–160.

[21] E. Schneider, S. Holst, X. Wen, and H.-J. Wunderlich, "Data-Parallel Simulation for Fast and Accurate Timing Validation of CMOS Circuits," in *Proc. IEEE/ACM 33rd Int'l Conf. on Computer-Aided Design (ICCAD)*, Nov. 2014, pp. 17–23.

[22] E. G. Ulrich, "Exclusive Simulation of Activity in Digital Networks," *Communications of the ACM*, vol. 12, no. 2, pp. 102–110, Feb. 1969.

[23] M. L. Bailey, J. V. Briner, Jr., and R. D. Chamberlain, "Parallel Logic Simulation of VLSI Systems," *ACM Computing Surveys*, vol. 26, no. 3, pp. 255–294, Sep. 1994.

[24] S. Holst, M. E. Imhof, and H.-J. Wunderlich, "High-Throughput Logic Timing Simulation on GPGPUs," *ACM Trans. on Design Automation of Electronic Systems*, vol. 20, no. 3, pp. 1–22, Article 37, Jun. 2015.

[25] L.-C. Chen, S. K. Gupta, and M. A. Breuer, "A New Gate Delay Model for Simultaneous Switching and Its Applications," in *Proc. 38th Design Automation Conf. (DAC)*, Jun. 2001, pp. 289–294, Paper 19.2.

[26] E. Schneider, S. Holst, M. A. Kochte, X. Wen, and H.-J. Wunderlich, "GPU-Accelerated Small Delay Fault Simulation," in *Proc. ACM/IEEE Conf. on Design, Automation Test in Europe (DATE)*, Mar. 2015, pp. 1174–1179.

[27] E. Schneider, M. A. Kochte, S. Holst, X. Wen, and H. J. Wunderlich, "GPU-Accelerated Simulation of Small Delay Faults," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 5, pp. 829–841, May 2017.

[28] E. Schneider and H.-J. Wunderlich, "High-Throughput Transistor-Level Fault Simulation on GPUs," in *Proc. IEEE 25th Asian Test Symp. (ATS)*, Nov. 2016, pp. 151–156.

[29] J. P. Hayes, "Digital Simulation with Multiple Logic Values," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 5, no. 2, pp. 274–283, Apr. 1986.