GPU-accelerated Time Simulation of Systems with Adaptive Voltage and Frequency Scaling

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

***Abstract*—Timing validation of systems with adaptive voltage- and frequency scaling (AVFS) requires an accurate timing model under multiple operating points. Simulating such a model at gate level is extremely time-consuming, and the state-of-the-art compromises both accuracy and compute efficiency.**

This paper presents a method for dynamic gate delay modeling on graphics processing unit (GPU) accelerators which is based on polynomial approximation with offline statistical learning using regression analysis. It provides glitch-accurate switching activity information for gates and designs under varying supply voltages with negligible memory and performance impact. Parallelism from the evaluation of operating conditions, gates and stimuli is exploited simultaneously to utilize the high arithmetic computing throughput of GPUs. This way, large-scale design space exploration of AVFS-based systems is enabled. Experimental results demonstrate the efficiency and accuracy of the presented approach showing speedups of three orders of magnitude over conventional time simulation that supports static delays only.

***Keywords*— AVFS, voltage-dependent delay, logic level time simulation, GPU parallelization, statistical learning**

I. Introduction

Today's advanced nano-scaled CMOS technology systems often rely on parametrization and self-adaptation by *adaptive voltage and frequency scaling* (AVFS) [1–3] to actively control internal voltages and clock frequencies. The parametrization allows to adapt the system to current workloads, environmental conditions [4] and performance degradation caused by transistor aging [5]. This way, a fine-grained trade-off in power and performance is enabled, which has gained increasing interest for application to low-power designs [6], as well as embedded systems and automotive area [7]. With the increasing sensitivity of devices to process-, voltage- and temperature variations [8–10], the timing validation of parametrizable systems has become difficult since the delays change with the system parameters. Especially for AVFS-systems, design validation and exploration require thorough investigation of different operating conditions [11, 12].

To validate the timing of AVFS-designs under different supply voltages, simulation approaches with accurate voltage-dependent delay modeling are necessary. They have to reveal the glitch-accurate switching activity in the design, which is of particular interest for small delay fault testing [13, 14] as well as power estimation [15].

So far, voltage-dependent delays have been incorporated into gate level evaluations and worst-case analyses in various ways. Besides common look-up table based approaches, analytical models with enclosed delay formulas [16–19] and delay approximation techniques [20, 21] were developed. Yet, conventional timing-accurate logic level simulation already shows severe scalability issues when applied to evaluate many input patterns under different parameters in medium-sized and larger designs with glitch-accuracy. Hence, incorporating parameter-variation-aware models in conventional timing simulation also comes along with a further increase in runtime complexity of the simulations as well as their applications [22].

The simulation complexity can currently only be tackled by exploiting the inherent parallelism of circuit [23–25] and fault simulation [26–28] on general-purpose *graphics processing unit* (GPU) accelerators. GPUs are able to provide high arithmetic floating-point throughput and can handle thousands to millions of light-weight threads concurrently on a single die to achieve high application speed-ups. However, not all simulation models and algorithms are suitable for GPUs and usually many restrictions have to be conquered to enable an efficient parallelization. The currently existing GPU-accelerated logic level simulators either do not consider timing at all, or they utilize simplified timing models that do not capture the impact of supply voltage on the gate delays.

The paper at hand overcomes the challenges of GPU-parallelization and presents a massively parallel logic level time simulator with parametric voltage-dependent delay modeling for scalable glitch-accurate timing validation of AVFS-based systems on GPUs. The delay modeling utilizes polynomial approximation with regression analysis to formulate the voltage-impact on the propagation delays at logic level. Simultaneous exploitation of different dimensions of parallelism from varying operating points, gates and input stimuli, allow to fully utilize the high-arithmetic throughput capabilities of the GPUs to maximally speed up the simulations. This way, fast and efficient design and test validation as well as large-scale early design space exploration of systems with AVFS is enabled for the first time.

The remainder of the paper is organized as follows: The next section summarizes the background on parametric delay modeling and timing simulation on GPUs. In section III, the cell delay characterization and the parameter-variation-aware delay modeling are introduced. Section IV describes the implementation of the according simulation algorithm and its GPU-parallelization. Finally, experimental results regarding the accuracy of the model and the performance of the simulation are provided.

II. Background

Traditional validation approaches typically use parametrizable delay models based on linear interpolation within look-up tables for each gate type, which contain the propagation delays over different process- and parameter corners [8]. To achieve a reasonable accuracy, these tables have to provide sufficient resolution and their size can grow exponentially with the number of parameters. These approaches do not scale well for AVFS systems and accuracy has to be compromised. As a consequence, worst-case margins and guardbands have to be increased further to overcome the uncertainties due to both variations and reduced modeling accuracy.

Analytical delay models on the other hand express the delay behavior by closed-form expressions [16–19]. For example, a simple way of modeling supply voltage impact on the gate delay is the α-power law [16], which dictates that the time constant τ should be proportional to the supply voltage V_{DD} and some process parameter $\alpha \in [1, 2]$ such that

$$\tau \propto V_{\mathrm{DD}}/(V_{\mathrm{DD}} - V_{th})^{\alpha}. \quad (1)$$

A general analytical approach was proposed in [18], that relies on the logical effort delay model [17, 29]. The model is based on a simplified RC delay modeling in which the propagation delay is generally described as

$$d = \tau(gh + p), \quad (2)$$

where τ is the process-dependent delay constant, g is the gate-specific logical effort, h describes the fanout (electrical effort) and p represents some parasitic impact. Extensions in [18] consider process- as well as voltage- and temperature variations through a linear relation reflected in the logical effort. A delay model for static timing analysis under variations was proposed in [17], which deduces non-linear components and introduces derating coefficients for expressing the delay impact. In a similar way, topological correlation of gates was incorporated in [30]. Analytical models typically consider parameters as independent components for simplification and require thorough understanding of the low-level impact.

Delay approximation techniques [8] utilize extensive SPICE simulations to extract gate delays under different operating points. Linear regression is then applied to find a fitting hypersurface that closely matches the observed delay behavior. Another approximation approach based on machine-learning was proposed in [21]. Similarly, it learns the delay-dependencies of voltage- and temperature parameters by using neural networks. The derived functions are then utilized to calculate the gate delays approximatively.

While these models are suitable for worst-case analyses, scalability issues occur in glitch-accurate evaluation of large test sets. Timing-accurate logic simulation is a runtime-intensive task, and by adding more complex delay models the complexity of the evaluation further increases due to many additional real-valued arithmetic operations being required.

A first logic level time simulation with GPU-acceleration was proposed by [25], which efficiently computes the switching history (*waveforms*) in a circuit with full glitch support. It utilizes parallelism from structurally independent gates and data-independent input stimuli waveforms. The simulation kernels are organized as a two-dimensional array of threads each of which handles an individual gate for a particular stimuli in parallel. This way, massive simulation throughput is achieved enabling speedups of up to three orders of magnitude compared to conventional logic level time simulation thus overcoming the complexity of timing-accurate evaluation.

III. Cell Characterization

The paper at hand employs an intuitive and fine-grained approximative delay modeling for efficient and timing-accurate evaluation on GPUs. Given a standard cell library, the pin-to-pin propagation delays of the cells are analyzed for different operating points (e.g., parameter corners).

In this work, all cell delays are parametrized via supply voltage $v \in \mathbb{R}$ and load capacitance $c \in \mathbb{R}$ of the cell. The parameters are assumed to be constrained by intervals ($v \in [V_{min}, V_{max}]$ and $c \in [C_{min}, C_{max}]$) that define the range of the supply voltage and gate load capacitances in the design. Both, supply voltage and load capacitance form a continuous two-dimensional *parameter (sub-)space* $\mathcal{P} \subseteq \mathbb{R}^2$ in which each point corresponds to a distinct *operating point* $P := (v, c) \in \mathcal{P}$ with voltage v and capacitance c. It is assumed that under any given operating point $P_i \in \mathcal{P}$, a gate(-pin) has a propagation delay $d_i \in \mathbb{R}$. The *nominal* operating point is denoted as $P_{nom} \in \mathcal{P}$ which has a corresponding delay $d_{nom} \in \mathbb{R}$.

A. Overview

The overall flow of the cell delay characterization preprocess is illustrated in Fig. 1. For each cell type and input pin, the propagation delay of rising and falling transitions are extracted from SPICE transient analysis (step A) with parameter sweeps over a finite set of operating points. Linear interpolation and sub-sampling is employed on normalized data points to increase the density of the sample data-grid (step B). Eventually, multi-variable linear regression is applied to calculate a fitting hypersurface that represents the delay function (step C). Prior normalization of the parameters is used to avoid over-fitting during regression [21]. The surface parameters are then compiled (step D) for the use as delay kernels that allow to compute the propagation delays under different operating points during simulation on the GPU. This flow has to be repeated only once for each new cell *type* in the library as the computed functions are reused during simulation.

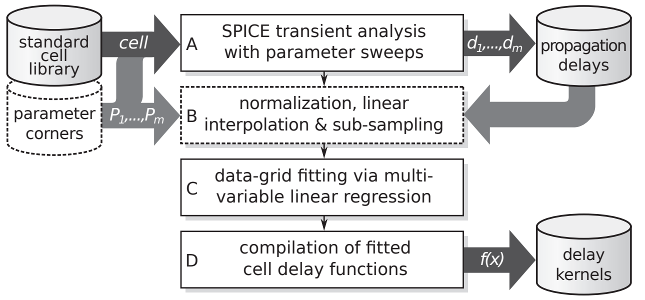

standard cell library
cell
A
SPICE transient analysis with parameter sweeps
$d_1,\ldots,d_m$
propagation delays
parameter corners
$P_1,\ldots,P_m$
B
normalization, linear interpolation & sub-sampling
C
data-grid fitting via multi-variable linear regression
D
compilation of fitted cell delay functions
f(x)
delay kernels

Fig. 1. Gate delay characterization and kernel generation pre-process.

B. Efficient Parameter-Aware Delay Modeling

The parametric delay behavior of a cell under a given operating point $P \in \mathcal{P}$ is derived from the *deviation* of the resulting propagation delays from the *nominal* operating point $P_{nom} \in \mathcal{P}$. The *delay deviation* forms a continuous surface in the (also continuous) parameter space. For this surface, a function $f : \mathcal{P} \rightarrow \mathbb{R}$ is constructed to approximate the *delay deviation* of a cell under different voltages in $\mathcal{P}$ with small error:

$$\forall P_i \in \mathcal{P} : f(P_i) \approx \frac{d_i}{d_{nom}} - 1. \tag{3}$$

In this work, the surface functions are expressed as a multi-variable higher-order polynomials. Polynomials are able to approximate any continuous differentiable hypersurface within constrained intervals with variable degree of accuracy, which typically increases with the order N of each variable.

The generalized form of a two-dimensional polynomial function with order $2 \cdot N$ is defined as $f : \mathcal{P} \rightarrow \mathbb{R}$,

$$f(P) := \sum_{i=0}^{N} \sum_{j=0}^{N} \left(\beta_{i,j} \cdot v^i c^j\right), \text{ with } P := (v, c) \in \mathcal{P}. \tag{4}$$

Each product term consists of the powers of the predictor variables $v^i c^j$ and has a corresponding coefficient $\beta_{i,j} \in \mathbb{R}$.

C. Defining Functions by Multi-variable Linear Regression

The surface expressed by a polynomial $f : \mathcal{P} \rightarrow \mathbb{R}$ is completely determined by its coefficients $\beta_{i,j}$, which are *unknown* a priori. Multi-variable linear regression allows to quickly find suitable coefficients for a surface polynomial that fits the delay behavior of a cell. This is achieved by setting up and solving an equation system based on the data grid samples obtained from the SPICE simulation. Given a data set $S := \{P_i \in \mathcal{P} | i = 1, ..., m\}$ of $m \in \mathbb{N}$ samples, the linear regression model is formulated in vector-form as

$$y = \mathbf{X}\beta + \varepsilon \tag{5}$$

with

$$y = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_m \end{pmatrix}, \mathbf{X} = \begin{pmatrix} v_1^0 c_1^0 & v_1^0 c_1^1 & v_1^1 c_1^0 & \dots & v_1^N c_1^N \\ v_2^0 c_2^0 & v_2^0 c_2^1 & v_2^1 c_2^0 & \dots & v_2^N c_2^N \\ v_3^0 c_3^0 & v_3^0 c_3^1 & v_3^1 c_3^0 & \dots & v_3^N c_3^N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_m^0 c_m^0 & v_m^0 c_m^1 & v_m^1 c_m^0 & \dots & v_m^N c_m^N \end{pmatrix}, \beta = \begin{pmatrix} \beta_{0,0} \\ \beta_{0,1} \\ \beta_{1,0} \\ \vdots \\ \beta_{N,N} \end{pmatrix} \tag{6}$$

and $\varepsilon \in \mathbb{R}^m$ as the residual. The entries of any row k in the matrix $\mathbf{X} \in \mathbb{R}^{m \times (N+1)(N+1)}$ correspond to the power terms $v_k^i c_k^j$ of the polynomial $f(v_k, c_k)$ for the k-th sample. All entries in a column l correspond to the l-th power term for the different sample polynomials. The first column typically reflects the zero-degree powers whose values are equal to 1.

All values v_k, c_k and d_k in the equation system are normalized prior to the regression to evenly weight the parameters and prevent overfitting. For the normalization $\phi_V : [V_{min}, V_{max}] \rightarrow [0,1], \phi_V(v) := \frac{v - V_{min}}{V_{max} - V_{min}}$ is used for the voltages, $\phi_C : [C_{min}, C_{max}] \rightarrow [0,1], \phi_C(c) := \frac{\log_2(c) - \log_2(C_{min})}{\log_2(C_{max}) - \log_2(C_{min})}$ is used for the capacitances. The propagation delays are also normalized with respect to the nominal delay d_{nom} under the nominal operating point P_{nom} through $\phi_D : \mathbb{R} \rightarrow \mathbb{R}, \phi_D(d) := \frac{d}{d_{nom}} - 1$. This way, the vector y provides the relative delay deviation with respect to P_{nom}.

The equation system in Eq. (5) is solved to find fitting coefficients $\hat{\beta} \in \mathbb{R}^{(N+1)(N+1)}$ for the polynomial. For this, the *ordinary least squares* criterion is followed that minimizes the sum of squared residuals in the Euclidean L^2-norm $|| \cdot ||_2$ by

$$\hat{\beta} = \arg\min_{\beta} \{||y - \mathbf{X}\beta||_2^2\}. \tag{7}$$

The solution of this problem is obtained by solving the following normal equation [31]:

$$\hat{\beta} = \left(\mathbf{X}^\mathsf{T}\mathbf{X}\right)^{-1} \mathbf{X}^\mathsf{T} y, \tag{8}$$

which eventually provides suitable coefficients $\hat{\beta}_{i,j}$ for the delay polynomial. The resulting polynomial can then be represented solely by the $(N+1) \cdot (N+1)$ coefficients.

D. Representation and Evaluation of Cell-Functions

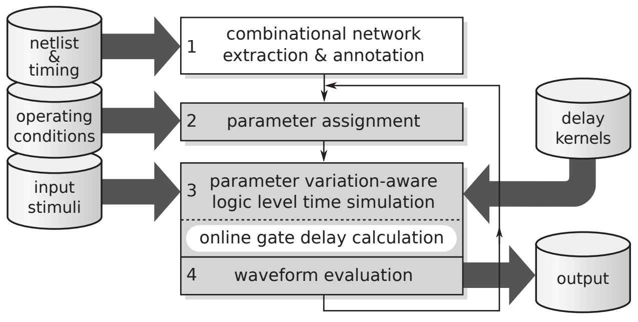
The propagation delay behavior of a cell differs for each input pin and output transition polarity (*rising*, *falling*). Therefore, for each gate type a set of delay polynomial coefficient vectors is generated, each of which covers a gate input pin for a transition polarity. When under a given operating point $P \in \mathcal{P}$ with normalized parameters, a signal transition at a gate input is propagated to the gate output, the delay deviation is calculated by evaluating the polynomial Eq. (4) for the respective coefficients. Since the polynomial approximation is highly prone to deviations in the coefficients, all calculations must be performed by floating-point operations in double-precision. As a final step the resulting propagation delay d' of the gate is then computed from the nominal propagation delay d_{nom} as follows:

$$d' := d_{nom} \cdot (1 + f(P)). \tag{9}$$

IV. High-Throughput Time Simulation

The coefficients of the delay polynomials are stored in a constant double-precision floating-point array structure in the global memory, which is indexed by the cell type, input pin and transition polarity. A delay computation kernel is implemented that evaluates the bare delay polynomial function efficiently on the GPU for a given operating point and provided delay coefficients. From Eq. (4) it is obvious that for the evaluation of the polynomial many floating point multiplications and additions are required. However, this complexity can be well handled by the GPUs with their arithmetic throughput. For this, the polynomial and normalization functions are compiled as device functions that are accessible by the threads. By following Horner's Method and reuse of previously calculated terms, the use of fused multiply-add (FMA) instructions is enforced which reduces the number of arithmetic operations [32].

The general flow of the parameter-aware time simulation is illustrated in Fig. 2. In step (1), the combinational network of the netlist is extracted, annotated with timing data from *standard delay format* files and stored on the GPU. Operating conditions of circuit instances are assigned (2) after which the test patterns are loaded for evaluation in the timing-accurate simulation (3). During the simulation, the threads of the simulation kernels independently compute modified delays on-the-fly according to the assigned conditions of the circuit instance. After the netlist has been processed, the waveforms are analyzed (4) to extract the output information, such as test responses, switching activity and transition times.



netlist & timing
operating conditions
input stimuli
1 combinational network extraction & annotation
2 parameter assignment
3 parameter variation-aware logic level time simulation
online gate delay calculation
4 waveform evaluation
delay kernels
output

Fig. 2. Flow of the parameter-aware time simulation.

The baseline time simulation assumes a *pin-to-pin* gate delay model with consideration of output transition polarities [25]. Static nominal delay annotations of the cells are extracted from *standard delay format files* and the load capacitances are obtained from *detailed standard parasitics format* that are assigned to the cell descriptions. Also, *inertial delay* is considered for pulse filtering of glitches and hazards. For the sake of simplicity, the inertial delays of the cells always equal their respective propagation delay values.

A. Online Delay Calculation

The operating points for evaluation (i.e., voltages) are assigned in a dedicated parameter memory on the GPU-device before the actual simulation. When a thread processes a gate g, it performs the following tasks as part of an initialization phase to generate the corresponding propagation delays:

1) load the gate description with the nominal delays from the global memory to the local private registers,
2) read the assigned gate- and circuit parameters P,
3) select delay d_{nom} of the locally stored nominal delays,
4) fetch corresponding delay coefficients β and evaluate polynomial $f(P)$ to determine the delay deviation,
5) adapt the locally stored delay d' accordingly (Eq. 9).

Steps (3) to (5) are repeated for each pin-to-pin propagation delay specified for the gate. Afterwards, the thread continues with the main waveform processing loop [25] to generate the gate output. Note that the kernel that computes the polynomial is always the same function call. Yet, depending on the selected coefficients, different functions are computed.

B. Parallelization

The implemented parameter-aware simulation simultaneously utilizes multiple dimensions of parallelism from gates, stimuli, and operating points of different circuit instances. The threads of the simulation kernels are organized as a three-dimensional grid of threads [32] as shown in Fig. 3.

In the vertical dimension, the threads exploit structural parallelism in so-called simulation *slots* [28] through level-wise processing of the circuit with all gates of a level being handled concurrently by different threads. In each horizontal plane, all threads process the same gate, but for different input stimuli and operating points thereby exploiting data-parallelism in two dimensions (waveforms and circuit instances). The execution of the threads in the horizontal plane is organized in SIMD-thread groups which follow a uniform execution flow with all the threads of a thread group processing the same gate

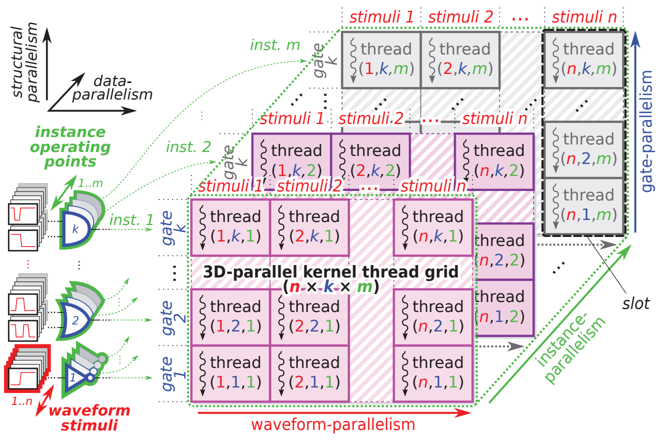


Fig. 3. Example of a three-dimensional parallel thread-grid organization for concurrent evaluation of n stimuli waveforms under m operating points.

function and the same delay kernels. In general, each slot can be assigned an individual input stimuli and operating point for evaluation. This way, the overall parallelization scheme allows to trade-off arbitrarily between simulation of multiple stimuli or multiple operating points. This provides the flexibility to maximize the slot utilization on the GPU and hence the simulation throughput for higher efficiency.

The modeling allows to generate timing annotations for many circuit instances on-the-fly and in parallel without having to transfer or store the gate delay descriptions of the individual instances on the GPU. Despite individually assigned operating points, the delay calculations of threads from parallel instances of a gate utilize the same coefficients and delay functions calls. Therefore no additional thread divergence is caused and a uniform execution flow among the parallel threads is sustained. Note that although this work utilizes polynomials for the delay calculation [20], analytical models [17, 18] and other types of approximations [21] can be applied as well.

V. Experimental Results

For the evaluation of the presented simulation, designs from ISCAS'89, ITC'99 and industrial benchmarks were investigated. The designs were synthesized using the NanGate 15nm Open Cell Library [33] in a commercial synthesis tool flow. All sequential elements were removed assuming full-scan and only the combinational logic remained. Transition delay test patterns were generated for each design using a commercial ATPG-tool. These were topped of with additional timing-aware patterns that target the 200 longest paths in each circuit. For the regression, the supply voltage was selected as $V_{\text{DD}} \in [0.55\text{V}, 1.1\text{V}]$ in steps of 0.05V (nominal 0.8V) with output loads $C_{\text{load}} \in [0.5\text{fF}, 128\text{fF}]$ in powers of two (i.e., 2^i for $i = -1, 0, ..., 7$). A commercial SPICE simulation tool was used to run the parameter sweeps which took few minutes for each cell. The regression analysis was implemented in Python. All experiments were conducted on an NVIDIA® Tesla$^{\text{TM}}$ P100 GPU (CUDA version 10.0) with 3584 cores and 16GB global device memory in a host system composed of two Intel®Xeon E5-2687W v2 processors clocked at 3.4GHz with 256GB of main memory.

A. Regression Analysis

Fig. 4 summarizes the resulting approximation error of the fitting for a subset of standard cells in the library (AND, NAND, BUF, INV, OR and NOR for all driving strengths). The error was measured for different polynomial orders by comparing a grid of 64×64 (4096) equidistant operating point samples to a linear approximation of the SPICE results. The distribution of the *mean* error is projected to the left axis, while the distribution of the standard deviation (*stddev*) and maximum error (*max*) are mapped to the right axis due to different magnitudes. As shown, the mean error is generally much less than a percent. Also, as expected the mean, standard deviation and maximum error generally decrease and their distributions narrow with increasing order of the polynomials. For polynomial orders $2 \cdot N$ with $N \geqslant 3$ the average standard deviation of the errors already falls below 1% and the average maximum error decreases below 2.7% (highest sample was 5.35%). However, this comes at the expense of an increased number of coefficients to store per pin-delay (4, 9, 16, 25, ...) and higher regression runtimes. Yet, the portion of memory required for storing the coefficients of a cell library is negligible compared to the waveforms. Also, in all cases obtaining the coefficients $\hat{\beta}$ by regression took between 1 and 40 milliseconds, which is considered as negligible overhead.

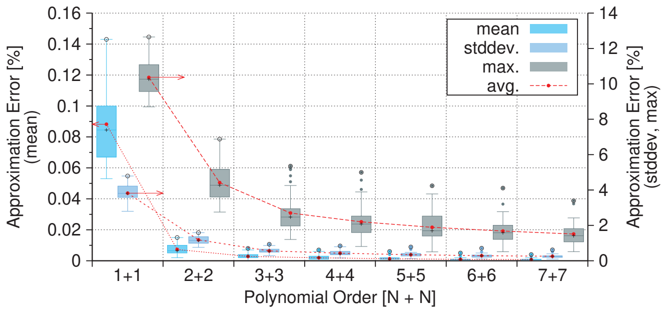


Fig. 4. Approximation error distribution of cell delay polynomials.

In Fig. 5 the resulting polynomial approximation of the rising propagation delay of the two-input NOR cell is compared with the linearly interpolated SPICE reference results. As shown, the contour lines of the polynomial surface closely follows the original data set. The average absolute error of the approximation over the 64×64 sample grid was ∼0.38% with a maximum deviation of 2.41%. This indicates that the delay can be well approximated by the implemented model.

B. Simulation Performance

Table I provides circuit statistics and performance results of the presented parallel simulation compared to a conventional serial commercial event-driven logic level time simulator. The delay kernel implements a polynomial of order $2 \cdot N$ over supply voltage and output load capacitance with $N = 3$. All nominal timings and load parameters were read from *standard delay format* and *standard-parasitics exchange format* files. The size of the circuits in nodes (cells, inputs and outputs) and the number of test pattern pairs are given in columns 2 and 3. For some designs all of the reported longest paths targeted by the timing-aware ATPG were *false paths* and no additional patterns were added to the original transition fault

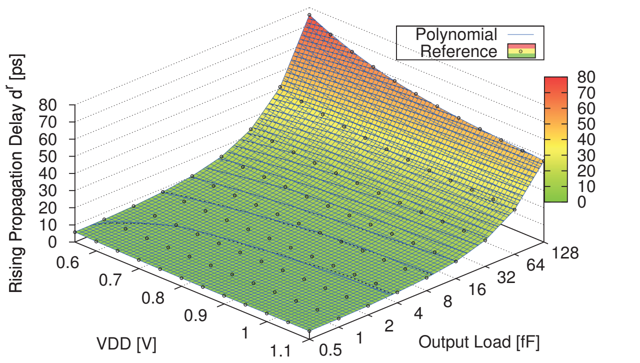


Fig. 5. Approximation of the rising propagation delay d^r of a NOR2_X2 cell by a surface polynomial of order $2 \cdot N$ with $N = 3$ compared to SPICE.

pattern set (marked by '*'). Columns 4 and 5 summarize the baseline simulation runtime for evaluating the respective pattern sets and the resulting throughput performance given in *million node evaluations per second* (MEPS). In column 6 the runtime of the GPU-accelerated time simulation from [25] without parametric delay support is given for comparison. The last three columns report the runtime (average out of 10 runs), throughput performance and the speedup of the presented approach over the serial event-driven solution.

As shown, the runtimes of the proposed approach ranged from 5ms to 10.35s for the circuits with speedups from 310× up to 1785× over the commercial solution with static delays. The average throughput performance was 1186 MEPS. Compared to the execution of the baseline GPU-algorithm with static delays [25], the runtime overhead of the gate delay calculations showed no significant impact even for higher degree polynomials, as the overall GPU-runtime is dominated by the memory overhead for storing the waveforms. The complete setup time of the proposed simulator typically took a few seconds only (up to 70 seconds for the largest design) without executing costly code-compilations of the netlists and their timing annotations or performing optimizations. Thus, to provide fair and unbiased comparisons of the performance, only the bare simulation times were considered. Note that the evaluation of a test stimuli under a given operating point is viewed as an independent simulation problem. Therefore, simulation problems could be grouped for distribution and execution on multi-GPU systems, which would provide additional speedup and scalability.

C. Voltage-dependent Delay Simulation

Table II summarizes the timing characteristics obtained from simulation of the previous pattern sets. Column 2 lists the longest path delay under nominal conditions as reported from a commercial timing analysis tool. Columns 3 through 8 show the arrival time of the latest transition that was observed at the outputs during simulation under different supply voltages. For the nominal case (0.8V) the relative deviation of the latest transition arrival time with respect to a simulation with static (nominal) delays is given next in parentheses.

As shown the latest arrival time in the nominal case is much lower than the pessimistic longest path delay reported by the timing analysis tool. As expected, lower supply voltages lead to higher delays and vice versa showing a non-linear depen-

TABLE I
CIRCUIT STATISTICS AND SIMULATION PERFORMANCE ($V_{DD} = 0.8V$).

Circuit(1)	Nodes(2)	Test	Event-Driven		[25]	Proposed		
		Pairs(3)	Time(4)	MEPS(5)	Time(6)	Time(7)	MEPS(8)	X(9)
s38417	18999	173	1.93s	1.70	6ms	5ms	557.1	328
s38584	23053	194	2.85s	1.57	6ms	9ms	486.1	310
b17*	42779	818	16.31s	2.15	18ms	25ms	1351.1	630
b18*	125305	961	2:20m	0.86	64ms	78ms	1528.1	1785
b19*	250232	1916	7:44m	1.03	207ms	267ms	1792.3	1737
b22	27847	692	16.22s	1.19	13ms	16ms	1204.4	1014
p35k	47997	3298	1:16m	2.08	69ms	86ms	1825.8	878
p45k	44098	2320	45.67s	2.24	56ms	69ms	1474.2	659
p100k	96172	2211	2:22m	1.49	100ms	126ms	1684.9	1133
p141k	178063	995	2:30m	1.18	100ms	117ms	1504.0	1279
p418k	440277	1516	8:11m	1.36	503ms	502ms	1329.3	979
p500k	527006	3820	0:49h	0.68	1.68s	1.91s	1052.4	1552
p533k	676611	1940	0:29h	0.74	1.62s	2.44s	538.0	729
p951k	1090419	4080	1:08h	1.09	7.97s	7.26s	612.6	564
p1522k*	1088421	8021	2:18h	1.05	9.72s	10.35s	843.2	802

TABLE II
CIRCUIT TIMING CHARACTERISTICS UNDER VOLTAGE SWEEP.

Circuit(1)	Longest	Latest Transition Arrival Times [s]					
	Path(2)	0.55V(3)	0.6V(4)	0.7V(5)	0.8V (vs. static)(6)	0.9V(7)	1.1V(8)
s38417	145.3p	164.5p	154.5p	139.3p	129.6p (-0.15%)	123.4p	115.0p
s38584	610.9p	846.0p	772.4p	661.9p	590.1p (-0.01%)	544.7p	485.0p
b17*	571.2p	548.5p	521.0p	479.7p	452.9p (+0.03%)	436.0p	413.8p
b18*	708.7p	736.2p	709.9p	670.4p	645.3p (-0.01%)	630.5p	611.1p
b19*	744.1p	741.5p	717.8p	683.6p	659.8p (+0.02%)	645.6p	627.3p
b22	606.2p	685.2p	651.8p	601.8p	569.5p (+0.04%)	549.2p	522.9p
p35k	275.5p	359.6p	333.7p	294.6p	268.8p (-0.21%)	252.1p	228.7p
p45k	2.234n	3.095n	2.847n	2.474n	2.231n (-0.14%)	2.078n	1.878n
p100k	2.234n	3.095n	2.847n	2.474n	2.231n (-0.14%)	2.078n	1.878n
p141k	640.0p	867.9p	795.8p	687.3p	616.5p (-0.10%)	581.8p	578.3p
p418k	1.537n	1.575n	1.539n	1.486n	1.452n (-0.03%)	1.430n	1.401n
p500k	660.8p	795.1p	734.4p	643.3p	584.2p (-0.25%)	547.0p	496.9p
p533k	2.348n	2.926n	2.760n	2.510n	2.347n (-0.06%)	2.244n	2.108n
p951k	708.0p	1.012n	924.3p	793.0p	707.8p (-0.03%)	653.9p	582.3p
p1522k*	2.335n	2.579n	2.406n	2.144n	1.972n (-0.04%)	1.862n	1.721n

dency. For the nominal case the parameter-dependent delay deviated in average by 0.1% from the static solution, which was due to the approximation error of the delay kernel. In comparison, the relative delay impact of the voltage deviation is much larger and thus the initial approximation error can be considered as uncertainty due to random process variations. Therefore, as shown the proposed simulation provides an efficient parametrizable delay modeling to simulate and explore AVFS-based designs.

VI. CONCLUSION

This work presented a massively parallel timing-accurate logic level simulation with parametric delay modeling on GPUs for large-scale validation and design space exploration of AVFS-based systems. The dynamic delay model is based on offline statistical learning and reflects the supply voltage impact on the gate propagation delays in a compact and accurate manner. Given the nominal timing specification of the circuit, the delay calculations are performed on the GPU during simulation with negligible memory and runtime overhead. By exploiting multiple dimensions of parallelism from gates, stimuli and circuit instances the simulation throughput is maximized for a fast and efficient evaluation of different operating conditions in AVFS-based systems. Experimental results have shown simulation speedups of up to 1785× over a commercial logic time simulation with conventional static delays. The average error of the gate delay approximation was less than 1% compared to electrical level simulation in SPICE.

ACKNOWLEDGMENT

This work is part of the project grant WU 245/19-1 funded by the German Research Foundation (DFG).

REFERENCES

[1] T. Kuroda, "CMOS Design Challenges to Power Wall," in *Proc. Int'l Microprocesses and Nanotechnology Conf. Digest of Papers.*, Oct. 2001, pp. 6–7.
[2] M. Horowitz, E. Alon, D. Patil *et al.*, "Scaling, Power, and the Future of CMOS," in *Proc. IEEE Int'l Electron Devices Meeting (IEDM)*, Dec. 2005, pp. 7 pp.–15.
[3] S. Borkar and A. A. Chien, "The Future of Microprocessors," *Communications of the ACM*, vol. 54, no. 5, pp. 67–77, May 2011.
[4] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A Dynamic Voltage Scaled Microprocessor System," *IEEE Journ. of Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, Nov. 2000.
[5] J. Tschanz, N. S. Kim, S. Dighe *et al.*, "Adaptive Frequency and Biasing Techniques for Tolerance to Dynamic Temperature-Voltage Variations and Aging," in *Proc. IEEE Int'l Solid-State Circuits Conf. (ISSCC)*, Feb. 2007, pp. 292–604.
[6] S. Kiamehr, M. Ebrahimi, and M. Tahoori, "Temperature-aware Dynamic Voltage Scaling for Near-Threshold Computing," in *Proc. Int'l Great Lakes Symp. on VLSI (GLSVLSI)*, May 2016, pp. 361–364.
[7] S. Mhira, V. Huard, A. Benhassain *et al.*, "Dynamic Adaptive Voltage Scaling in Automotive environment," in *Proc. IEEE Int'l Reliability Physics Symp. (IRPS)*, Apr. 2017, pp. 3A–4.1–3A–4.7.
[8] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical Analysis and Optimization for VLSI: Timing and Power*, 1st ed. Springer, 2005.
[9] S. Soleimani, A. Afzali-Kusha, and B. Forouzandeh, "Temperature Dependence of Propagation Delay Characteristic in FinFET Circuits," in *Proc. Int'l Conf. on Microelectronics (ICM)*, Dec. 2008, pp. 276–279.
[10] E. Amat, A. Calomarde, and A. Rubio, "Reliability Study on Technology Trends Beyond 20nm," in *Proc. 20th Int'l Conf. on Design of Integrated Circuits and Systems (MIXDES)*, Jun. 2013, pp. 414–418.
[11] S. Borkar, T. Karnik, S. Narendra *et al.*, "Parameter Variations and Impact on Circuits and Microarchitecture," in *Proc. Design Automation Conf. (DAC)*, June 2003, pp. 338–342.
[12] I. Polian, B. Becker, S. Hellebrand, H. Wunderlich, and P. Maxwell, "Towards Variation-Aware Test Methods," in *Proc. IEEE 16th European Test Symp. (ETS)*, May 2011, pp. 219–225.
[13] A. Czutro, M. E. Imhof, J. Jiang *et al.*, "Variation-Aware Fault Grading," in *Proc. IEEE 21st Asian Test Symp. (ATS)*, Nov. 2012, pp. 344–349.
[14] S. Hellebrand, T. Indlekofer, M. Kampmann *et al.*, "FAST-BIST: Faster-than-At-Speed BIST Targeting Hidden Delay Defects," in *Proc. IEEE Int'l Test Conf. (ITC)*, Oct. 2014, pp. 1–8, Paper 29.3.
[15] P. Girard, N. Nicolici, and X. Wen, Eds., *Power-Aware Testing and Test Strategies for Low Power Devices*. Springer New York, 2010.
[16] T. Sakurai and A. R. Newton, "Alpha-Power Law MOSFET Model and its Applications to CMOS Inverter Delay and Other Formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 584–594, Apr. 1990.
[17] B. Lasbouygues, R. Wilson, N. Azemard, and P. Maurine, "Temperature and voltage aware timing analysis: Application to voltage drops," in *Proc. Conf. on Design, Automation Test in Europe (DATE)*, Apr. 2007, pp. 1–6.
[18] C.-H. Wu, S.-H. Lin, and H. Chiueh, "Logical Effort Model Extension with Temperature and Voltage Variations," in *Proc. 14th Int'l Workshop on Thermal Inveatigation of ICs and Systems*, Sept 2008, pp. 85–88.
[19] K. Shinkai, M. Hashimoto, and T. Onoye, "A gate-delay model focusing on current fluctuation over wide range of process-voltage-temperature variations," *Integration, the VLSI Journal*, vol. 46, no. 4, pp. 345–358, 2013.
[20] H. Chang and S. S. Sapatnekar, "Statistical Timing Analysis Under Spatial Correlations," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Syst. (TCAD)*, vol. 24, no. 9, pp. 1467–1482, Sep. 2005.
[21] B. P. Das, V. Janakiraman, B. Amrutur, H. S. Jamadagni, and N. V. Arvind, "Voltage and Temperature Scalable Gate Delay and Slew Models Including Intra-Gate Variations," in *Proc. 21st Int'l Conf. on VLSI Design (VLSID)*, Jan. 2008, pp. 685–691.
[22] J. Mahmod, S. Millican, U. Guin, and V. Agrawal, "Special Session: Delay Fault Testing - Present and Future," in *Proc. IEEE 37th VLSI Test Symp. (VTS)*, Apr. 2019, pp. 1–10.
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] M. Li and M. S. Hsiao, "3-D Parallel Fault Simulation With GPGPU," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Syst. (TCAD)*, vol. 30, no. 10, pp. 1545–1555, Oct. 2011.
[28] 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 Syst. (TCAD)*, vol. 36, no. 5, pp. 829–841, May 2017.
[29] I. Sutherland, B. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.
[30] J. Shiomi, T. Ishihara, and H. Onodera, "Variability- and Correlation-Aware Logical Effort for Near-Threshold Circuit Design," in *Proc. 17th Int'l Symp. on Quality Electronic Design (ISQED)*, Mar. 2016, pp. 18–23.
[31] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer New York, 2009.
[32] NVIDIA Corporation, "CUDA C Best Practices Guide v9.1." http://www.nvidia.com, Mar. 2018.
[33] NanGate Inc., "NanGate15nm Open Cell Library." http://www.nangate.com/, 2017.