Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

An accurate sparse-matrix based framework for statistical static timing analysis[☆]

Anand Ramalingam[a], Ashish Kumar Singh[d,*], Sani R. Nassif[c], Gi-Joon Nam[c], Michael Orshansky[b], David Z. Pan[b]

[a] Magma Design Automation, Austin, TX 78759, United States
[b] Department of Electrical and Computer Engineering, The University of Texas, Austin, TX 78712, United States
[c] Austin Research Lab, IBM, Austin, TX 78758, United States
[d] Terra Technology, Schaumburg, IL, 60173, United States

ARTICLE INFO

Article history:
Received 1 July 2010
Received in revised form
17 November 2010
Accepted 1 March 2011
Available online 21 March 2011

Keywords:
Statistical timing
Sparse matrix
Path-based timing

ABSTRACT

Statistical static timing analysis has received wide attention recently and emerged as a viable technique for manufacturability analysis. To be useful, however, it is important that the error introduced in SSTA be significantly smaller than the manufacturing variations being modeled. Achieving such accuracy requires careful attention to the delay models and to the algorithms applied. In this paper, we propose a new sparse-matrix based framework for accurate path-based SSTA, motivated by the observation that the number of timing paths in practice is sub-quadratic based on a study of industrial circuits and the ISCAS89 benchmarks. Our sparse-matrix based formulation has the following advantages: (a) it places no restrictions on process parameter distributions; (b) it can use an accurate polynomial-based delay model which takes into account slope propagation naturally; (c) it takes advantage of the matrix sparsity and high performance linear algebra for efficient implementation. Our experimental results are very promising.

1. Introduction

As technology has scaled, manufacturing variations have emerged as a major limiter of design performance. These variations exhibit themselves as systematic, spatial and random changes in the parameters of active (transistor) and passive (interconnect) components. Furthermore, these variations are increasing with each new generation of technology. Statistical static timing analysis (SSTA) has been proposed to perform full-chip analysis of timing under such types of uncertainty, and has been the subject of intense research recently [2–19]. The result of SSTA is the prediction of parametric yield at a given target performance for a design.

SSTA algorithms can be classified into two major groups:

(1) *Block-based* [2–7] approaches use a breadth-first traversal of the circuit to compute circuit delay [2]. The delay pdf is propagated from the primary inputs to the primary outputs. The major difficulty in block-based approaches is the introduction of the max operator at each block, and the need to accurately estimate the maximum of two random variables in the same form in which those two variables are defined.

(2) *Path-based* [8–12] approaches rely on an enumeration of all or a large number of the most critical paths in the circuit [8]. Considering the case where all paths are enumerated, the max operator is deferred to the end of the analysis (i.e. taking the maximum of all the paths) and therefore does not introduce any inaccuracy in the computation. A major problem with path-based approaches is the *perception* that typical circuits have an exponential number of paths, making the computational requirement for such approaches impractical.

While there has been much work on the algorithms for SSTA, there has been somewhat less work on the accuracy issues. Some of the sources of inaccuracy in SSTA are

- the basic assumptions underlying static timing analysis, such as treating a gate as a node without considering the functionality which gives rise to false paths,
- the delay models used for gates and wires, and
- the model for process variations and their spatial and/or temporal distributions.

The *algorithmic* error introduced by SSTA algorithms can be traced back to the application of the max operator, which is an

[☆] A preliminary version of this paper appeared in the Proceedings of the International Conference on Computer-Aided Design (ICCAD 2006) [1].

[*] Corresponding author. Tel.: +1 5126892111.

E-mail addresses: anandr@magma-da.com (A. Ramalingam), aksingh@cerc.utexas.edu (A.K. Singh), nassif@us.ibm.com (S.R. Nassif), gnam@us.ibm.com (G.-J. Nam), orshansky@cerc.utexas.edu (M. Orshansky), dpan@cerc.utexas.edu (D.Z. Pan).

0167-9260/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2011.03.002

approximation to the behavior of true circuits, and which is further approximated in SSTA algorithms [14–16]. While a direct assessment of that error is difficult, we propose that eliminating the max operation on parameterized form would aid in reducing the error. The algorithm we propose in this work applies max on a set of real numbers thus incurring no error.

The *model* error has been widely recognized and a number of researchers have made important contributions. The *parameterized delay form* expressed delays and arrival times as explicit linear functions of the process parameters [6]. It was later expanded to handle quadratic delay models that are able to improve the accuracy of delay estimates [13–16]. A related source of error, namely the modeling and handling of the slope of signals, has not received as much attention. In fact, current published approaches typically make a worst-case estimate of the slope or propagate the latest arriving slope [6] which can lead to significant error [20]. The polynomial models we propose in this work allow high accuracy by using higher order models, and naturally handling the slope and its propagation.

The *distribution* error, i.e. the error caused by lack of generality in the modeling of the statistical properties of the process variations has been the most difficult to deal with, due to the lack of published realistic manufacturing variability data. Earlier approaches assumed that process variables followed normal distributions [8], but recent work has shown how more general distributions can be handled, and how spatial and systematic correlation can be accommodated [19]. In this work, we make *no assumptions* about the character or distribution of any process parameter.

This paper proposes a new approach to parameterized path-based SSTA. The proposed method starts with a preprocessing step of path enumeration and delay computation of all the paths in a parameterized form, which we then efficiently represent using a *sparse matrix*. We model the delay and slope of each component in the circuit using a general parameterized polynomial form which can include the influence of

- input waveforms and output loading,
- manufacturing variations in parameters like threshold voltage and channel length,
- operating environment variations in parameters like power supply voltage and temperature.

Next, the path delays in this same parameterized form are computed by a natural extension to the gate delay formulation. Given a sample of values from the distribution of manufacturing variations, this computation is shown to be simply a matrix/vector multiply that produces a vector of delays of each path in the circuit. Finally, the maximum circuit delay is obtained by applying the max operator on the path delays. The major attributes of this work are

(1) We show that the number of paths in practice is sub-quadratic in number of gates by evaluating the number of paths in the ISCAS89 benchmarks as well as two different families of industrial circuits.
(2) It can handle global, spatial and intra-die variations in one unified framework.
(3) It can compute the delay based on an accurate propagation of slope along all paths.
(4) It minimizes the impact of the error caused by approximating the max function commonly used in SSTA.
(5) It is independent of the underlying distribution of the process parameters, and is not restricted to the usual Gaussian distribution.

The remainder of this paper is organized as follows. In Section 2 we first motivate the path-based approach by showing that it is indeed practical for many circuits. We then show the need for higher order (more than linear) delay models in Section 3, and describe our approach to the delay modeling of practical static CMOS gates. With those models in hand, we then describe our matrix based formulation for STA and SSTA without slope in Section 4 and SSTA with slope in Section 5. We demonstrate its application to the ISCAS family of sequential benchmark circuits in Section 6. We compare our method with an existing method in Section 7 and conclude in Section 8.

2. A case for path-based SSTA

The upper bound on the number of paths in an arbitrary network is exponential in the number of gates. A key observation in this paper, however, is that for the vast majority of practical circuits, the number of actual paths is far less than this theoretical upper bound, and is quite manageable. It should be noted that the theoretical upper bound is usually met by highly structured networks such as multipliers. With the easy availability of large amounts of memory in modern computers, storing and manipulating million of paths is eminently practical.

To test our conjecture, we enumerated *all* the latch-to-latch, primary input to latch, and latch to primary output paths in the ISCAS sequential circuit benchmarks [21] (see Fig. 1), and found that the paths $\approx 0.04 \times \text{gates}^{1.8}$. This is hardly the type of explosive growth that might cause one to completely discount a family of algorithms. But since the ISCAS benchmarks are small compared to modern designs, we further extended our analysis to two different families of industrial benchmarks, one for large circuits (much larger than the ISCAS benchmarks), and one for moderate sized circuits (comparable to the ISCAS benchmarks).

We enumerated all paths for the circuits in those two families. For the first and larger family, shown in Fig. 2, we saw that the number of paths $\approx 0.12 \times \text{gates}^{1.42}$. For the second and smaller family, shown in Fig. 3, we found paths $\approx 0.43 \times \text{gates}^{1.17}$.

Clearly, the demonstration above should not be taken as sufficient license to propose a purely path-based SSTA algorithm. However, it does demonstrate that such an algorithm can be practical for a significant number of cases. In the broader picture, one can imagine a pairing of path-based and block-based algorithms with one being applied when the enumeration of paths results in a manageable number of paths, while the other gets applied to those circuits where the number of paths exceeds some suitable threshold.

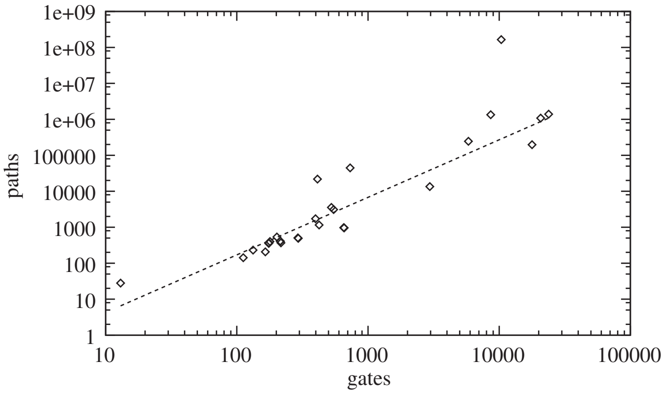

Fig. 1. The number of paths versus the number of gates in ISCAS'89 benchmarks. By linear regression we get the following relationship: paths $\approx 0.04 \times \text{gates}^{1.8}$.

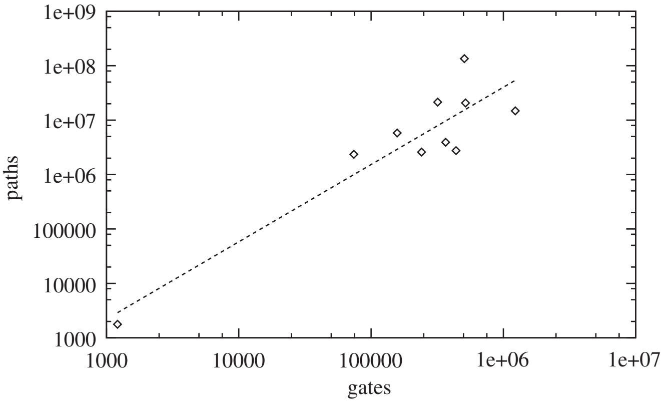

Fig. 2. The number of paths versus the number of gates for one family of 10 industrial benchmarks. By linear regression we get the following relationship: paths $\approx 0.12 \times$ gates$^{1.42}$.

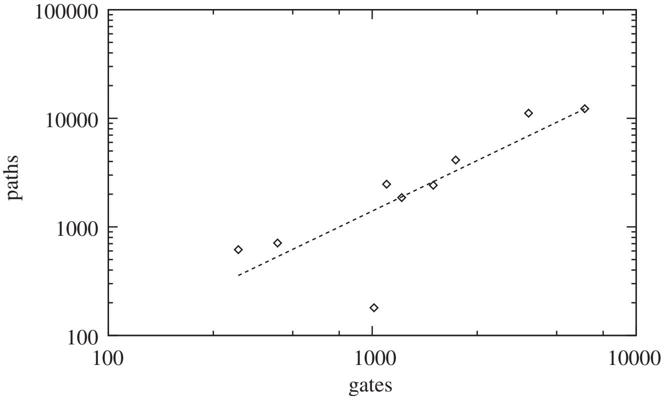

Fig. 3. The number of paths versus the number of gates for another family of nine industrial benchmarks. By linear regression we get the following relationship: paths $\approx 0.43 \times$ gates$^{1.17}$.

3. Parameterized gate delay modeling

The advantage of path-based SSTA is that it can naturally handle accurate nonlinear delay models. In this section, we present a parameterized gate delay model which explicitly takes slope propagation into account. In current published approaches, typically worst-case estimate of the slope or the latest arriving slope is propagated [6] which can lead to significant error [20]. By modeling the input slope in the gate delay equation we avoid this modeling error.

It has been observed that a delay model linear in process variations has a large amount of error; while a quadratic model fits the gate delay quite accurately [13–15]. The need for a *higher order* delay model is illustrated in Fig. 4, where we show a model of delay as a function of gate length (L), threshold voltage (V_{th}), the output capacitance C_{L} and input slope S_{in}. The samples of L and V_{th} used to create Fig. 4 were generated uniformly in the range $\mu \pm 3\sigma$ with $3\sigma = 0.2\mu$. The samples of S_{in} were generated in the range of 10–100 ps and samples of C_{L} were generated in the range of 1–10 fF.

The x-axis shows the HSPICE delay for various (L, V_{th}, C_{L}, S_{in}) tuples. The y-axis shows the values predicted by the two delay models. It is clear that our model is a much better predictor of delay than the linear model. In order to generate the cell delay model for every gate in the library, we simulate each gate varying the process parameters, load capacitance C_{L} and input slope S_{in} *uniformly* as described above, then fit to the delay equation given below:

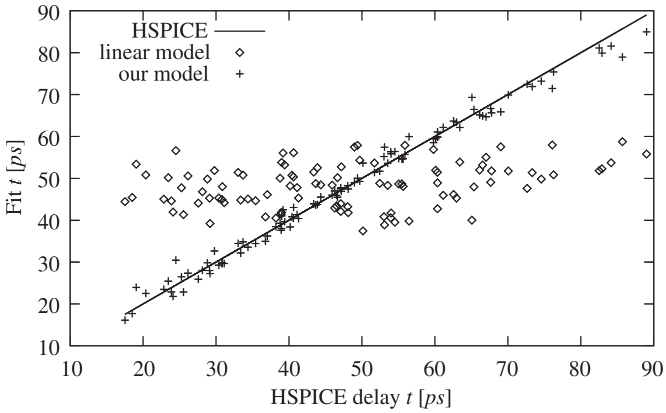

Fig. 4. Scatterplot of inverter delay and the values predicted by the linear and our higher order model for various (L, V_{th}, C_{L}, S_{in}) tuples. The variation in delay is due to variations in process parameters (L, V_{th}) as well as variations in operating conditions (C_{L}, S_{in}).

$$\begin{aligned} D = {} & a_0^d + a_1^d L + a_2^d L^2 + a_3^d V_{\mathrm{th}} + a_4^d V_{\mathrm{th}}^2 \\ & + C_{\mathrm{L}}(b_1^d L + b_2^d L^2 + b_3^d V_{\mathrm{th}} + b_4^d V_{\mathrm{th}}^2) \\ & + \alpha^d C_{\mathrm{L}} + \beta^d S_{\mathrm{in}} + \gamma^d S_{\mathrm{in}} C_{\mathrm{L}} \end{aligned} \tag{1}$$

Similarly, the output slope was also fit to the same canonical form as delay and is given below:

$$\begin{aligned} S_{\mathrm{out}} = {} & a_0^s + a_1^s L + a_2^s L^2 + a_3^s V_{\mathrm{th}} + a_4^s V_{\mathrm{th}}^2 \\ & + C_{\mathrm{L}}(b_1^s L + b_2^s L^2 + b_3^s V_{\mathrm{th}} + b_4^s V_{\mathrm{th}}^2) \\ & + \alpha^s C_{\mathrm{L}} + \beta^s S_{\mathrm{in}} + \gamma^s S_{\mathrm{in}} C_{\mathrm{L}} \end{aligned} \tag{2}$$

Note that both the output delay equation (1) and the output slope equation (2) are explicitly dependent on input slope S_{in}. The equations are valid only for a certain range of the parameters involved such as L. The canonical form presented in Eqs. (1) and (2) is equivalent to canonical forms presented in the literature in the form of the deviations from the nominal values. If we replace L with

$$L = L_{\mathrm{nominal}} + \Delta L$$

and V_{th} with

$$V_{\mathrm{th}} = V_{\mathrm{th,nominal}} + \Delta V_{\mathrm{th}}$$

in Eqs. (1) and (2) then we will get the canonical forms presented in the literature. Note that the constants a_0^d in Eq. (1) and a_0^s in Eq. (2) are intercepts obtained from linear regression. They should not be confused with nominal value of delay or output slope as they are usually denoted in the literature [6,22].

It should be noted that our formulation does not restrict the model order in any way, and higher order models are possible with no change to our methodology.

4. Sparse-matrix based SSTA without slope propagation

In this and next sections, we present the sparse-matrix based SSTA formulation. First, we calculate the path delays without considering slope propagation and in the next section we take the slope into account. Let the delay of gate j from input a to the gate output be $d_{j_a} \in \mathbb{R}$. Later we will generalize the gate delay as a function of parameters $\mathbf{z}$, $d_{j_a} = f(\mathbf{z})$.

4.1. Sparse-matrix based static timing analysis (STA)

Consider the circuit shown in Fig. 5. This circuit has four paths and three gates. The gates have two inputs which we distinguish

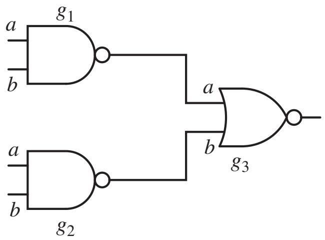

Fig. 5. Example circuit for illustrating the matrix formulation. The input pins are distinguished by the labels *a* and *b*.

by labeling them *a* and *b*. We define an incidence matrix where each row represents a path and each column represents a gate input. The columns are sorted by gate topological order. The *path-gate* incidence matrix for the example is given by

$$\mathbf{A} = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \begin{array}{c} \begin{matrix} g_{1_a} & g_{1_b} & g_{2_a} & g_{2_b} & g_{3_a} & g_{3_b} \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \tag{3}$$

Since each path only consists of a small number of gates, matrix **A** is a very sparse. The delay of the gates can be written as *gate delay* vector:

$$\mathbf{d}_{\text{gate}} = [d_{1_a}\ \ d_{1_b}\ \ d_{2_a}\ \ d_{2_b}\ \ d_{3_a}\ \ d_{3_b}]^\top \tag{4}$$

where d_{1_b} is the delay from input pin *b* of gate 1 to its output. The delay of a path is given by the addition of gate delays along that path. Thus the path delays are given simply by the multiplication of the path-gate incidence matrix with the gate delay vector:

$$\mathbf{d}_{\text{path}} = \mathbf{A}\mathbf{d}_{\text{gate}} \tag{5}$$

The overall circuit delay is given by the max of all path delays:

$$d_{\text{circuit}} = \max(\mathbf{d}_{\text{path}}) \tag{6}$$

Eq. (6) represents path-based static timing analysis (STA). We note that STA in this form is essentially a sparse-matrix-vector multiplication, and that it requires only a single max operator to find the circuit delay. There are many data structures and algorithms developed for efficient sparse-matrix manipulation which we can exploit [23]. Now we turn our attention to the statistical STA (SSTA).

4.2. Sparse-matrix based statistical static timing analysis (SSTA)

In this section, we drop the input specific delay for the sake of convenience. Let the delay of gate *j* be a function of *r* parameters $\mathbf{z}_j \in \mathbb{R}^r$. Thus $d_j = f(\mathbf{z})$ is a symbolic function of parameters instead of a real number.

$$d_j = \sum_{k=1}^{r} c_{jk} z_{jk} = \mathbf{c}_j^\top \mathbf{z}_j \tag{7}$$

Note that **z** need not consist only of linear parameters. For example, a possible second order gate delay model in channel length *L* and load C_L might be:

$$\mathbf{z}_j = [1\ \ L\ \ L^2\ \ C_{\text{L}}\ \ C_{\text{L}}L]^\top \tag{8}$$

The same formulation can trivially handle a mixed model such as

$$\mathbf{z}_j = [1\ \ \sqrt{L}\ \ L\ \ C_{\text{L}}\ \ C_{\text{L}}e^L]^\top \tag{9}$$

The gate delays of the circuit in Fig. 5 can be written as

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \mathbf{diag}(\mathbf{c}_1^\top, \mathbf{c}_2^\top, \mathbf{c}_3^\top) \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \end{bmatrix}$$

$$\mathbf{d}_{\text{gate}} = \mathbf{C}^\top \mathbf{Z} \tag{10}$$

where **diag**() denotes the diagonal of a matrix.

The path delays are obtained by multiplying the path-gate incidence matrix in Eq. (3) and the gate delay vector in Eq. (10)

$$\begin{aligned} \mathbf{d}_{\text{path}} &= \mathbf{A}\mathbf{d}_{\text{gate}} \\ &= \mathbf{A}\mathbf{C}^\top \mathbf{Z} \end{aligned} \tag{11}$$

With Eq. (11) we have now extended the path delay calculation in Eq. (5) to include the dependence of delay on process parameters. Assuming that these process parameters are random variables with some well defined joint probability density function from which we can sample, our goal is to show how we can generalize this result to calculate the distribution of path delays, and by using the traditional max function, the distribution of overall circuit delay.

If the *k*th random sample is given by $\mathbf{Z}^{(k)}$ then the path delay vector corresponding to the *k*th random sample is given by

$$\mathbf{d}_{\text{path}}^{(k)} = \mathbf{A}\mathbf{C}^\top \mathbf{Z}^{(k)} \tag{12}$$

Now if we take ℓ samples then Eq. (12) can be generalized as

$$[\mathbf{d}_{\text{path}}^{(1)}\ \ \ldots\ \ \mathbf{d}_{\text{path}}^{(\ell)}] = \mathbf{A}\mathbf{C}^\top [\mathbf{Z}^{(1)}\ \ \ldots\ \ \mathbf{Z}^{(\ell)}] \tag{13}$$

To get the circuit delay distribution, we apply Eq. (6) to Eq. (13)

$$[d_{\text{circuit}}^{(1)}\ \ \ldots\ \ d_{\text{circuit}}^{(\ell)}] = [\max(\mathbf{d}_{\text{path}}^{(1)})\ \ \ldots\ \ \max(\mathbf{d}_{\text{path}}^{(\ell)})] \tag{14}$$

This is essentially a Monte Carlo simulation expressed in matrix form. A histogram of the circuit delay vector in Eq. (14) produces the circuit delay distribution. Thus Eq. (14) represents path-based statistical static timing analysis (SSTA) ignoring slope. In this form, SSTA is a natural extension of STA as written in Eq. (6) and is simply in the form of a matrix–matrix multiplication. We make a few remarks about the matrices. It is important to note that $\mathbf{AC}^\top$ matrix is a sparse matrix, which allows for efficient storage as well as fast computation. The **Z** vector, though dense, depends only on the number of gates and not on the number of paths.

4.3. Example

We will present an illustrative example to show how matrices are constructed and Monte Carlo simulations are done in our framework.

There are two paths in the circuit shown in Fig. 6. This can be captured in the incidence matrix **A** of Eq. (11):

$$\mathbf{A} = \begin{array}{c} \\ p_1 \\ p_2 \end{array} \begin{array}{c} \begin{matrix} g_1 & g_2 & g_3 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \end{array} \tag{15}$$

For the sake of illustrative purposes, consider the delay to be function of just length *L*. Let $d_1 = 70L_1$, $d_2 = 50L_2$ and $d_3 = 60L_3$. Then the coefficient matrix ($\mathbf{C}^\top$) and parameter matrix **Z** in Eq. (11) is given by

$$\mathbf{C}^\top = \begin{pmatrix} 70 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 60 \end{pmatrix} \tag{16}$$

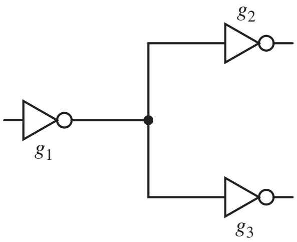

Fig. 6. Example circuit for illustrating how Monte Carlo simulation is done in our framework.

$$\mathbf{Z} = \begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} \tag{17}$$

We get the path delays using Eq. (11):

$$\begin{aligned}\mathbf{d}_{\text{path}} &= \mathbf{A}\mathbf{C}^{\top}\mathbf{Z} \\ &= \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}\begin{pmatrix} 70 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 60 \end{pmatrix}\begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} \\ &= \begin{bmatrix} 70 & 50 & 0 \\ 70 & 0 & 60 \end{bmatrix}\begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} = \begin{bmatrix} 70L_1 + 50L_2 \\ 70L_1 + 60L_3 \end{bmatrix}\end{aligned} \tag{18}$$

The Monte Carlo simulation is done by sampling L_1, L_2 and L_3 from their respective distributions. For example, after sampling it turns out that $L_1 = 1.6$, $L_2 = 1.4$ and $L_3 = 1.5$. Then substituting these values in Eq. (18) leads to path delays for this instance:

$$\mathbf{d}_{\text{path}} = \begin{bmatrix} 182 \\ 202 \end{bmatrix} \text{ps} \tag{19}$$

This corresponds to one Monte Carlo simulation as in Eq. (12). Repeating this ℓ times allows to obtain a distribution for path delays and circuit delay. The number of samples ℓ depends on the confidence interval we seek in the variance of the distribution and it is usually set to $\ell = 10{,}000$ in the literature.

5. Sparse-matrix based SSTA with slope propagation

We now extend our delay model to include slope propagation. It is important to note that the output slope of gate j cannot be specified unless we know which path it belongs to. For example, in Fig. 5, gate g_3 at input pin a will have two different slopes namely:

(1) due to path 1 ($g_{1_a} \rightarrow g_{3_a}$), and
(2) due to path 2 ($g_{1_b} \rightarrow g_{3_a}$).

We use the same canonical form to express both delay and slope, but we restrict the dependence of delay and output slope on the input slope to be *linear*. This linearity is required in order to preserve the canonical form as delays are accumulated along a path. However, we do not require a global linearity wherein the input slope is varying on order of magnitude. A locally linear model is sufficient and the sensitivity can be determined based on table look up from the range of input slope. In order to find the correct sensitivity, we propose a first step static timing analysis flow for each path to allow us to determine the three-sigma spread of each output slope for any fixed path. Based on the range of slope, we can determine the correct sensitivity coefficient to use for the SSTA analysis. We also observed a linear relationship between delay and input slope variation over a range for the cells in our library and we illustrate it in Fig. 7 for one of the cells in the library.

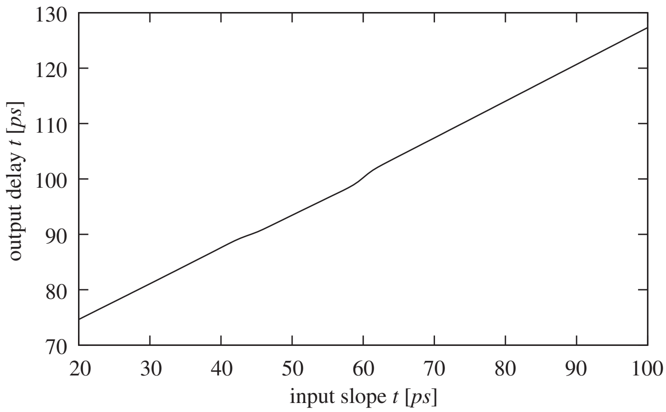

Fig. 7. Linear relationship between input slope and delay.

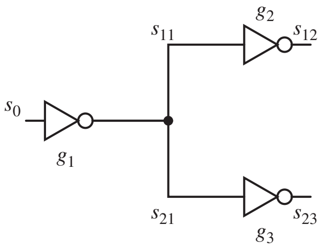

Fig. 8. A simple circuit to illustrate SSTA with slope propagation. Here s_0 denotes the slope at the primary input. The output slope at gate g_1 in path 1 is denoted as s_{11} and in path 2 is denoted as s_{21}.

We use the superscripts d and s to distinguish the coefficients of delay and slope. We delineate the input slope to a gate by the subscript in. The gate delay d_{ij} and the output slope s_{ij} of gate j in path i is given by

$$d_{ij} = \lambda_j^d s_{\text{in}} + (\mathbf{c}_j^d)^{\top}\mathbf{z}_j \tag{20}$$

$$s_{ij} = \lambda_j^s s_{\text{in}} + (\mathbf{c}_j^s)^{\top}\mathbf{z}_j \tag{21}$$

From Eq. (20), one can see that the input slope at all the gates is required to calculate the gate and path delays. One way to solve for the input slope is to look at each path p separately and obtain the slope of each gate in an individual path. This method is illustrated using Fig. 8, and this simple circuit consists of inverters which allows us to conveniently drop the input-pin specific subscripts.

Let $\mathbf{s}_p$ be the column vector in which the values of slopes along path p are listed. Assume the values are listed in the topological order of the gates along path p.

To illustrate, consider the path $p=1$, through gates g_1 and g_2 in Fig. 8. The column vector $\mathbf{s}_1$ is given by

$$\mathbf{s}_1 = \begin{bmatrix} s_0 \\ s_{11} \\ s_{12} \end{bmatrix} \tag{22}$$

and related by Eq. (21) as

$$\begin{bmatrix} s_0 \\ s_{11} \\ s_{12} \end{bmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ \lambda_1^s & 0 & 0 \\ 0 & \lambda_2^s & 0 \end{pmatrix}\begin{bmatrix} s_0 \\ s_{11} \\ s_{12} \end{bmatrix} + \begin{bmatrix} (\mathbf{c}_0^s)^{\top}\mathbf{z}_0 \\ (\mathbf{c}_1^s)^{\top}\mathbf{z}_1 \\ (\mathbf{c}_2^s)^{\top}\mathbf{z}_2 \end{bmatrix}$$

$$\begin{aligned}\mathbf{s}_1 &= \mathbf{\Lambda}_1^s\mathbf{s}_1 + \mathbf{diag}((\mathbf{c}_0^s)^{\top}, (\mathbf{c}_1^s)^{\top}, (\mathbf{c}_2^s)^{\top})\ \mathbf{Z}_1 \\ &= \mathbf{\Lambda}_1^s\mathbf{s}_1 + (\mathbf{C}_1^s)^{\top}\mathbf{Z}_1\end{aligned} \tag{23}$$

In general Eq. (23) is valid for any arbitrary path containing t gates. Thus, $\mathbf{s}_1 \in \mathbb{R}^{t+1}$, $\mathbf{\Lambda}_1^s \in \mathbb{R}^{(t+1)\times(t+1)}$ is *lower diagonal*, and $(\mathbf{C}_1^s)^\top \mathbf{Z}_1 \in \mathbb{R}^{t+1}$. If the circuit has p paths, then the Eq. (23) for all the p paths can be succinctly captured into one single equation shown below:

$$\begin{bmatrix} \mathbf{s}_1 \\ \cdots \\ \mathbf{s}_p \end{bmatrix} = \mathbf{diag}(\mathbf{\Lambda}_1^s, \ldots, \mathbf{\Lambda}_p^s) \begin{bmatrix} \mathbf{s}_1 \\ \cdots \\ \mathbf{s}_p \end{bmatrix} + \begin{bmatrix} (\mathbf{C}_1^s)^\top \mathbf{Z}_1 \\ \cdots \\ (\mathbf{C}_p^s)^\top \mathbf{Z}_p \end{bmatrix}$$

$$\begin{aligned} \mathbf{s} &= \mathbf{\Lambda}^s \mathbf{s} + \mathbf{diag}((\mathbf{C}_1^s)^\top, \ldots, (\mathbf{C}_p^s)^\top)\mathbf{Z} \\ &= \mathbf{\Lambda}^s \mathbf{s} + (\mathbf{C}^s)^\top \mathbf{Z} \end{aligned} \tag{24}$$

From Eq. (24) we can solve for the slope $\mathbf{s}$ in the circuit

$$\mathbf{s} = (\mathbf{I} - \mathbf{\Lambda}^s)^{-1} (\mathbf{C}^s)^\top \mathbf{Z} \tag{25}$$

Lemma 1. *The matrix* $(\mathbf{I} - \mathbf{\Lambda}^s)^{-1}$ *is non-singular. Thus its inverse exists.*

Proof. Assume path i contains $t_i - 1$ gates and $1 \le i \le p$. Now the $\mathbf{det}(\mathbf{I} - \mathbf{\Lambda}^s)$ can be written as

$$\begin{aligned} \mathbf{det}(\mathbf{I} - \mathbf{\Lambda}^s) &= \mathbf{det}(\mathbf{I} - \mathbf{diag}(\mathbf{\Lambda}_1^s, \ldots, \mathbf{\Lambda}_p^s)) \\ &= \mathbf{det}(\mathbf{diag}((\mathbf{I}_{t_1} - \mathbf{\Lambda}_1^s), \ldots, (\mathbf{I}_{t_p} - \mathbf{\Lambda}_p^s))) \\ &= \prod_{i=1}^{p} \mathbf{det}(\mathbf{I}_{t_i} - \mathbf{\Lambda}_i^s) \\ &= 1 \end{aligned}$$

where $\mathbf{\Lambda}_i^s \in \mathbb{R}^{t_i \times t_i}$ and is a *lower diagonal* matrix

Next, we reason why $\mathbf{\Lambda}_i^s$ is a lower diagonal matrix. The output slope of a gate is a function of the input slope of the gate. But the input slope of a gate is given by the output slope of its predecessor gate on the same path. Thus, in a *topologically ordered* vector of slopes $\mathbf{s}$, every element is *only* related to its previous element by some transformation. This transformation is affine in case of Eq. (21) where the coefficients were real-valued. Since the transformation relates an element to the element before it, the matrix describing this transformation is lower diagonal.

Since $\mathbf{\Lambda}_i^s$ is a lower diagonal matrix, the matrix $(\mathbf{I}_{t_i} - \mathbf{\Lambda}_i^s)$ turns out to be a lower triangular matrix with diagonal entries 1. Since for a lower triangular matrix $\mathbf{L} \in \mathbb{R}^{m \times m}$ the determinant is given by

$$\mathbf{det}(\mathbf{M}) = \prod_{i=1}^{m} \ell_{ii}$$

Thus we can conclude that

$$\mathbf{det}((\mathbf{I}_{t_i} - \mathbf{\Lambda}_i^s)) = 1, \quad 1 \le i \le p$$

Since the determinant of $(\mathbf{I} - \mathbf{\Lambda}^s)$ is non-zero it is invertible. □

Lemma 2. *The matrix* $(\mathbf{I} - \mathbf{\Lambda}^s)^{-1}$ *can be computed in linear time in number of non-zero entries of* $(\mathbf{I} - \mathbf{\Lambda}^s)$ *and the inverse is also sparse.*

Proof. First we exploit the block diagonal matrix structure of $(\mathbf{I} - \mathbf{\Lambda}^s)$ to represent its inverse in a block diagonal form as follows:

$$\begin{aligned} (\mathbf{I} - \mathbf{\Lambda}^s)^{-1} &= (\mathbf{I} - \mathbf{diag}(\mathbf{\Lambda}_1^s, \ldots, \mathbf{\Lambda}_p^s))^{-1} \\ &= (\mathbf{diag}((\mathbf{I}_{t_1} - \mathbf{\Lambda}_1^s), \ldots, (\mathbf{I}_{t_p} - \mathbf{\Lambda}_p^s)))^{-1} \\ &= \mathbf{diag}((\mathbf{I}_{t_1} - \mathbf{\Lambda}_1^s)^{-1}, \ldots, (\mathbf{I}_{t_p} - \mathbf{\Lambda}_p^s)^{-1}) \end{aligned} \tag{26}$$

Using the special structure of each of sub-matrix above $\mathbf{A}_{t_i} \equiv (\mathbf{I}_{t_i} - \mathbf{\Lambda}_i^s)$, we can write out closed form expression for its inverse. The special structure can be described by the presence of one in all diagonal entries, lower triangular matrix and only entries next to diagonal are non-zero. Assuming that the entries next to diagonal are $\lambda_1, \lambda_2, \ldots$, we can arrive at the following closed form expression for its inverse $\mathbf{B}_{t_i}$. (Following shows the entry in a-th row and b-th column):

$$\begin{aligned} \mathbf{B}_{t_i}[a][b] &= 0 \quad \text{if } (a < b) \\ &= 1 \quad \text{if } (a = b) \\ &= \lambda_b \cdots \lambda_{a-1} \quad \text{if } (a > b) \end{aligned}$$

By setting $\alpha_k = \lambda_1 \cdots \lambda_k$, we can re-express above as

$$\begin{aligned} \mathbf{B}_{t_i}[a][b] &= 0 \quad \text{if } (a < b) \\ &= 1 \quad \text{if } (a = b) \\ &= \frac{\alpha_{a-1}}{\alpha_{b-1}} \quad \text{if } (a > b) \end{aligned}$$

It is easy to see thus that the matrix $\mathbf{B}_{t_i}$ is lower triangular matrix and the inverse can be computed in $O(t_i\,(t_i + 1)/2)$, which is same as the number of non-zero entries in $\mathbf{B}_{t_i}$. We recall that, t_i is number of rows of square matrix $\mathbf{A}_{t_i}$, which is 1 plus the logic depth of path i. The overall complexity of computing the inverse in Eq. (26) is $O(\sum_{i=1}^{p} t_i(t_i+1)/2)$. Since t_i is bounded by 1 plus the logic depth of circuit, the overall inverse can be computed in linear time in number of paths $O(p)$.

It is also clear that the number of non-zero entries are $\sum_{i=1}^{p} t_i(t_i+1)/2 = O(p)$, while the total number of entries in the matrix are $(\sum_{i=1}^{p} t_i)^2 = O(p^2)$, again using the fact that t_i is bounded by logic depth, we deduce that the inverse is also sparse. □

The equation for path delays is similar to Eq. (24) and can be calculated as

$$\begin{aligned} \mathbf{d}_{\text{path}} &= \mathbf{\Lambda}^d \mathbf{s} + (\mathbf{C}^d)^\top \mathbf{Z} \\ &= \mathbf{\Lambda}^d (\mathbf{I} - \mathbf{\Lambda}^s)^{-1} (\mathbf{C}^s)^\top \mathbf{Z} + (\mathbf{C}^d)^\top \mathbf{Z} \quad \text{(using Eq. (25))} \\ &= \mathbf{\Lambda}^d (\mathbf{I} - \mathbf{\Lambda}^s)^{-1} (\mathbf{C}^s)^\top \mathbf{Z} + (\mathbf{C}^d)^\top \mathbf{Z} \\ &= (\mathbf{\Lambda}^d (\mathbf{I} - \mathbf{\Lambda}^s)^{-1} (\mathbf{C}^s)^\top + (\mathbf{C}^d)^\top) \mathbf{Z} \\ &= \mathbf{D}^\top \mathbf{Z} \end{aligned} \tag{27}$$

Since path delays are a function of process parameters, by taking a sample of all process parameters one can generate delay of all paths in the circuit as captured in the following equation:

$$\mathbf{d}_{\text{path}}^{(k)} = \mathbf{D}^\top \mathbf{Z}^{(k)} \tag{28}$$

The circuit delay is given by the max of all path delays. Now if we take ℓ samples then we get ℓ samples of circuit delay captured in the following equation:

$$[d_{\text{circuit}}^{(1)} \ \cdots \ d_{\text{circuit}}^{(\ell)}] = [\max(\mathbf{d}_{\text{path}}^{(1)}) \ \cdots \ \max(\mathbf{d}_{\text{path}}^{(\ell)})] \tag{29}$$

A histogram on these ℓ samples gives us the circuit delay distribution. Thus SSTA can be performed considering the slope and process variations.

6. Experimental results

We implemented our algorithm using a combination of awk/perl scripts and C++. We report the results of experiments run on the ISCAS89 benchmarks using a 64-bit Linux machine with 16 GB RAM and running at 3.4 GHz. The delay models were generated using the 90 nm Berkeley Predictive Technology Model [24]. In the experiments only latch-to-latch paths were considered for timing. Thus in Table 1 only the latch-to-latch paths and the number of gates between the latches are listed. We modeled the effect of variations in channel length and threshold voltage, and assumed that the variance of these parameters was such that $3\sigma = 0.2\mu$. We modeled the impact of spatial correlation on

Table 1
Path-gate statistics of ISCAS89 benchmarks and runtime for 10,000 simulations.

Circuit	Gates	Paths	Sparsity[a] (%)	Runtime (s)				Percentage runtime (%)			Time per matrix multiply (s)
				Generating[b]		Matrix[c] multiply	Total	Generating[b]		Matrix[c] multiply	
				Paths	Matrix			Paths	Matrix		
s27	8	9	46.13	0.02	0.02	0.07	0.11	18	18	63	7.00e−06
s1196	73	43	11.61	0.15	0.07	0.60	0.82	18	8	73	6.00e−05
s1238	73	43	11.61	0.12	0.04	0.39	0.55	21	7	70	3.90e−05
s208	50	72	10.48	0.07	0.04	0.49	0.60	11	6	81	4.90e−05
s386	92	86	8.56	0.08	0.07	0.57	0.72	11	9	79	5.70e−05
s820	187	207	3.42	0.15	0.13	1.14	1.42	10	9	80	1.14e−04
s298	98	212	4.97	0.10	0.11	1.04	1.25	8	8	83	1.04e−04
s832	188	219	3.41	0.15	0.12	1.07	1.34	11	8	79	1.07e−04
s510	162	230	4.02	0.15	0.19	1.39	1.73	8	10	80	1.39e−04
s641	237	238	12.82	0.41	2.14	4.80	7.35	5	29	65	4.80e−04
s344	154	323	6.21	0.17	0.43	2.33	2.93	5	14	79	2.33e−04
s349	155	333	6.11	0.21	0.40	1.70	2.31	9	17	73	1.70e−04
s382	133	353	4.21	0.16	0.17	1.18	1.51	10	11	78	1.18e−04
s1488	307	366	3.36	0.31	0.51	3.33	4.15	7	12	80	3.33e−04
s1494	306	375	3.37	0.36	0.52	3.33	4.21	8	12	79	3.33e−04
s526n	172	377	2.66	0.15	0.20	1.75	2.10	7	9	83	1.75e−04
s526	171	379	2.67	0.15	0.12	1.76	2.03	7	5	86	1.76e−04
s444	160	482	4.18	0.21	0.27	1.60	2.08	10	12	76	1.60e−04
s953	328	723	2.54	0.40	0.76	3.93	5.09	7	14	77	3.93e−04
s713	250	2650	17.54	5.13	36.47	50.42	92.02	5	39	54	5.00e−03
s5378	1938	6858	0.66	4.44	9.62	20.28	34.34	12	28	59	2.00e−03

[a] Sparsity = percentage of non-zero in the sparse matrix.

[b] We wrote awk/perl scripts to generate paths and build the sparse matrix.

[c] We wrote C++ program to do matrix multiplication.

parameter variations, and therefore required placement information for the circuits, which we obtained by placing the circuits using Dragon [25]. To properly account for random die-to-die (global) and within-die (intra) variations along with the spatial component mentioned above, we modeled each process parameter $z_{g,i}$ as

$$z_{g,i} = \sqrt{0.5}z_{g,i}^{\text{global}} + \sqrt{0.25}z_{g,i}^{\text{intra}} + \sqrt{0.25}z_{g,i}^{\text{spatial}} \tag{30}$$

where $z_{g,i}^{\text{intra}}$ models the random variation and $z_{g,i}^{\text{spatial}}$ models the spatial variation by introducing new grid random variables [5].

We performed 10,000 Monte Carlo simulations for each of the ISCAS benchmark circuits. The number of simulations performed in our experiment was set high in purpose to establish an accurate result. But a run with one tenth (1000) the number of samples would normally be sufficient to calculate the delay distribution to engineering accuracy. The results are summarized in Table 1. The table contains the number of gates and paths along with the runtime taken by the algorithm to compute the delay statistics of the circuit. Also shown is the breakdown of effort among

(a) path enumeration (implemented in awk),
(b) sparse-matrix generation (implemented in perl), and
(c) matrix multiplication (implemented in C++).

In Table 1, we have presented results for smaller benchmarks in ISCAS89. For the bigger benchmarks we implemented speedup techniques and they are discussed next.

6.1. Speedup techniques

In this section, we describe techniques to speedup the Monte Carlo simulation. The speedup technique is based on removing the non-critical paths to reduce the matrix size.

If we are concerned only about the circuit delay, the maximum of all path delays, then we can prune out non-critical paths based on their logic depth. To ensure fairness, complex gate like XOR can be assigned an equivalent logic depth of 2 instead of being treated as the same as a NAND gate.

The idea behind the pruning technique is illustrated using s1423. The histogram of logic depth of paths in s1423 is shown in Fig. 9(a). In Fig. 9(b), the histogram of the logic depth of the path with the maximum delay in a Monte Carlo simulation is plotted. The histogram was obtained after doing Monte Carlo simulation for 10,000 runs. s1423 is representative of the ISCAS89 benchmarks and it is clear that the paths with the higher logic depth tend to be the one which have maximum path delay. So we employ the strategy of eliminating paths with smaller logic depth when we are computing the circuit delay, the maximum of all path delays.

We have applied this pruning technique to the biggest benchmarks in the ISCAS89 benchmark suite. Our pruning strategy is to eliminate paths whose logic depth is less than 90% of the maximum logic depth of the circuit.[1] For example, if the circuit's maximum logic depth is 60 then all paths whose logic depth is less than $0.9 \times 60 = 54$ are eliminated.

In Table 2, benchmark s38417 seems a little anomalous. This is because there is a long tail in the logic depth of s38417 as shown in Fig. 10. This means that after pruning we are left with just 280 paths to analyze. To validate our approach we compare the pdf obtained from golden simulation (where no paths are eliminated) to the pdf obtained after pruning paths in Fig. 11. Note that both the pdf's, one obtained after pruning paths and the other obtained by considering all the paths, labeled exact, are nearly identical. The mean which was obtained from both the methods was the

[1] We found that retaining paths whose logic depth was greater than 90% of the maximum logic depth of the circuit provided the best tradeoff between accuracy and speed for our synthesized ISCAS89 benchmarks.

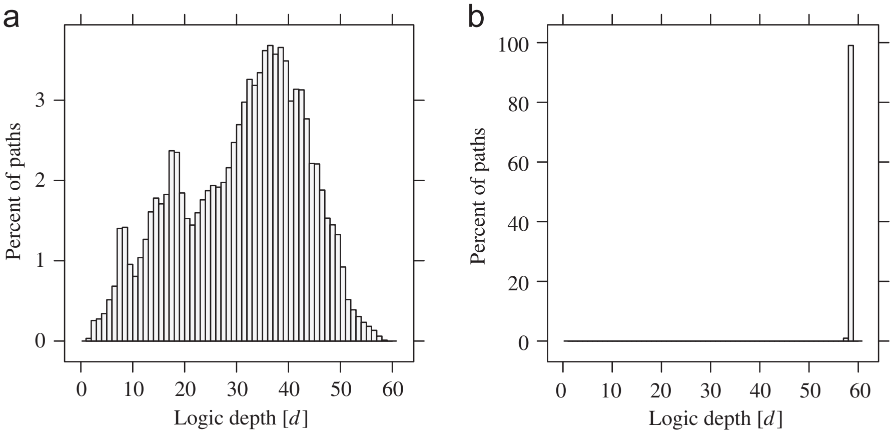

Fig. 9. Histogram was obtained after doing Monte Carlo simulation for 10,000 runs. Note that the logic depth of paths with maximum delay in (b) is always either 59 or 60. (a) Histogram of logic depth of paths in S1423 and (b) Histogram of logic depth of paths with the maximum delay in S1423.

Table 2
Runtime after logic depth based pruning for the three biggest benchmarks in ISCAS89. All paths whose logic depth was less than 90% of the maximum logic depth were pruned. The number of simulations were set to 10,000.

Circuit	Original paths[a]	Pruned paths[b]	Pruned runtime (s)			
			Generating		Matrix multiply	Total
			Paths	Matrix		
s1423	35,990	334	22.20	4.83	19.31	46.34
s9234	227,837	33,536	227.89	477.36	1691.61	2396.86
s35932	122,997	39,168	91.70	150.53	1012.51	1254.74
s38584	850,422	35,904	530.90	471.37	1732.91	2735.18
s13207	1,005,680	78,082	715.66	1138.58	3946.16	5800.40
s38417	1,389,348	280	541.31	3.75	12.47	557.53

[a] This column shows the total number of paths in the circuit.
[b] This column shows the number of paths left after logic depth based pruning.

same while the standard deviation obtained after pruning paths was differed by less than 0.9% from the exact method.

Note that the delay distribution of s38417 deviates from normality by having long tails in Fig. 11. The deviation from normality is because of two reasons:

(1) max operation;
(2) the quadratic terms in delay equation in Eq. (1). Even if the parameters follow normal distribution, the quadratic terms make the distribution non-normal.

The results from the other two benchmarks s13207 and s38584 are similar to the results from s38417.

7. Comparative studies

In this section we compare our proposed method with a graph based Monte Carlo method and a block-based method [6].

7.1. Comparison with path-based Monte Carlo method

In this section, we compare the runtimes of the proposed method with a path-based Monte Carlo (MC) since a block-based MC cannot handle slope propagation accurately. By path-based MC simulation we mean evaluation of path delays without resorting to sparse matrix method. To understand the difference between the

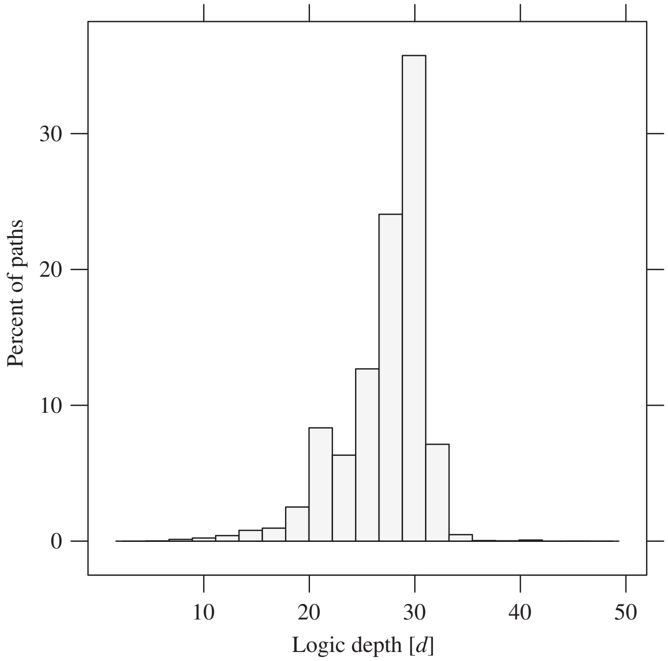

Fig. 10. Histogram of logic depth of paths in s38417. Note that there is a long tail and there are paths with logic depth of 47. Thus our pruning strategy of eliminating paths with logic depth less than 0.9 × 47 leaves us with just 280 critical paths.

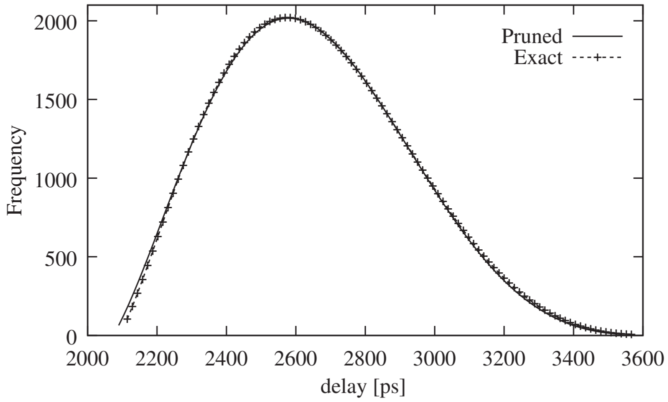

Fig. 11. Delay pdf of s38417 obtained after pruning paths plotted against the one obtained without pruning paths. Note that both the pdf's are virtually indistinguishable.

proposed sparse-matrix method and the graph based MC method let us compare the algorithms used to implement them.

First we summarize the steps involved in a sparse-matrix based method as a pseudocode in Algorithm 1. Next we summarize the steps involved in a path-based MC as a pseudocode in Algorithm 2.

Algorithm 1. Sparse-matrix-based-SSTA.

```
Input: Circuit description after it has been mapped to a library
Output: Timing distribution of the circuit
1: Enumerate all latch-to-latch paths in the circuit using
   depth first search (DFS) [26].
2: Calculate the parameterized delay for each of the paths and
   store it in sparse matrix. This process is captured in Eq.
   (27).
3: for $i=1$ to $\ell$ do
4:    Generate a sample of process parameters, pre-multiply it
      by the sparse matrix to get a vector of path delays. This is
      captured in Eq. (28). Apply the max operator to get the
      circuit delay. This constitutes a single Monte Carlo
      simulation.
5: end for
6: The above for loop results in a vector of circuit delays of
   length $\ell$. This is captured in Eq. (29). From this one can
   generate circuit delay statistics and estimate the timing
   yield of the circuit.
```

Algorithm 2. Graph-based-SSTA.

```
Input Circuit description after it has been mapped to a library
Output: Timing distribution of the circuit
1: Enumerate all latch-to-latch paths in the circuit using
   depth first search (DFS) [26].
2: for $i=1$ to $\ell$ do
3:    Generate a sample of process parameters for every gate
      in the circuit.
4:    for $p=1$ to $n$ paths do
5:       Get the output slope and delay for every gate in the
         path based on the process parameters and input slope. Add
         the delays of all the gates in the path to get the path delay.
6:    end for
7:    Apply the max operator over all path delays to get the
      circuit delay. This constitutes a single Monte Carlo
      simulation.
8: end for
9: The outer for loop results in a vector of circuit delays of
   length $\ell$. From this one can generate circuit delay statistics
   and estimate the timing yield of the circuit.
```

The algorithms presented in Algorithms 1 and 2 look deceptively similar. The major difference between the two lies in step 2 of the sparse-matrix based method (Algorithm 1). We generate path delays in a parameterized form and store it as a sparse matrix.[2] In the path-based method, the delay model evaluation is done inside the `for()` loop as shown in step 5 (Algorithm 2). Thus we need to do sample L and V_{th} for each simulation and then evaluate all the gates for their delays and output slope. Then gate delays along a path are added up to get the path delay. Our sparse-matrix approach calculates delay and slope in a parameterized (symbolic) form and avoids this delay evaluation inside a `for()` loop. This is the reason behind the efficiency of our approach compared to a path-based Monte Carlo method.

Table 3
Runtime comparison for the proposed matrix method versus repeated path tracing method. The number of simulations were set to 10,000.

Circuit	Path MC[a] (s)	Sparse matrix[b] (s)	Alt DS[c] (s)
s27	220.03	0.11	1.12
s208	380.05	0.60	10.76
s1196	460.16	0.82	12.10
s298	880.09	1.25	29.68
s382	1360.13	1.51	56.00
s344	3060.17	2.93	81.05

[a] This column shows the runtime of a path-based MC whose pseudocode was presented in Algorithm 2. Monte Carlo simulation uses path-based approach since slope cannot be accurately propagated in a block-based method.

[b] This column shows the runtime of the proposed sparse matrix method presented in Algorithm 1.

[c] This column shows the runtime of the proposed method with an alternative data structure, array of hashes. The algorithm is the same as the one presented in Algorithm 1 but the data structure changes from sparse matrix to array of hashes.

The runtimes between the two approaches are compared for smaller benchmarks in Table 3. From the table one can observe that the proposed approach has a runtime which is orders of magnitude faster than the path-based method.

7.2. Implementation details

We presented the construction of sparse matrix (**D**) in Eq. (28) as a series of operations on matrices as shown in Eq. (27). This was done to present our analysis in a mathematically rigorous as well as an elegant fashion. It should be noted that one can construct the sparse matrix (**D**) using graph traversal method. In fact the sparse matrix construction in our implementation was done using graph traversal method.

The sparse matrix can be thought of as an efficient data structure to hold the parameterized path delays. An alternative data structure such as array of hashes to hold parameterized path delays lead to slower runtimes. In array of hashes one can visualize array consisting of all possible paths; each element in the array points to a hash which consists of all the gates in that path. The runtime comparison is shown in Table 3. It is clear that having sparse matrix as a data structure is superior since it is results in a regular access from caches leading to a faster runtime.

7.3. Comparison with block-based method

A comparison of the proposed method with a block-based method [6] is shown in Fig. 12 for s27, clearly showing the proposed method's accuracy compared with block-based method. The tails in the distribution are not captured by block-based method since its delay model is linear. Also the block-based method is restricted to work only with normal distributions. It also suffers from the fact that it approximates max of two random variables. We note that the biggest benchmarks in ISCAS89 ran in a few seconds using the block-based method showing the runtime superiority of the block-based method.

8. Conclusion and future work

This paper demonstrates that it is possible and practical to perform path-based statistical static timing analysis, and that such an analysis can be written compactly in matrix notation, allowing the use of standard highly optimized linear algebra techniques. The major advantage of this formulation is that it

[2] It should be noted that there is no restriction on the gate delay model except the need to have input slope appear linearly. The linearity restriction helps us to preserve the canonical form of the delay and slope models.

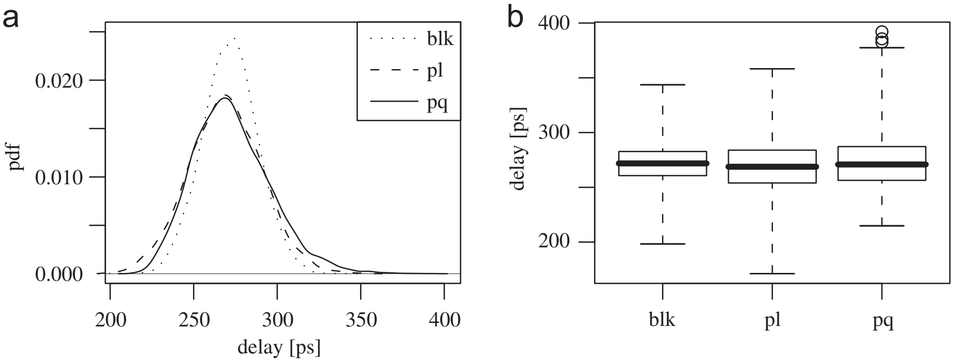

Fig. 12. Delay pdf of s27 obtained using block-based method [6] (denoted as *blk*), path based method with *linear* delay models (denoted as *pl*) and path-based method with *quadratic* models (denoted as *pq*). Results from path-based method with *linear* models can be thought of how much error is introduced by an analytical max() and using worst-case slope at the input of a gate. Results from path-based method with *quadratic* models can be thought of how much error is introduced when we use linear delay models. Note that the linear models are not adequate enough to model the tails of the distribution. (a) Density plot of circuit delay of s27. (b) Boxplot of circuit delay of s27.

places no restrictions on process parameter distributions. It embeds accurate polynomial-based delay model which takes into account slope propagation naturally. With the exception of the need to have the slope appear linearly, fairly arbitrary models can be trivially handled using this framework.

Data was presented to show that many practical circuits have a bounded number of paths, making such an analysis possible. It should be noted that this demonstration should not be taken as sufficient license to propose a purely path-based SSTA algorithm. We plan to extend the formulation to handle wires, and show how incremental computation may be done in the framework and also incorporate a more accurate waveform model [27].

Acknowledgments

This work is partially supported by SRC, IBM Faculty Award, Fujitsu, Sun, and Intel equipment donation.

References

[1] A. Ramalingam, A.K. Singh, S.R. Nassif, G.-J. Nam, M. Orshansky, D.Z. Pan, An accurate sparse matrix based framework for statistical static timing analysis, in: ICCAD '06: Proceedings of the 2006 ACM/IEEE International Conference on Computer-aided Design, 2006, pp. 231–236.

[2] J.-J. Liou, K.-T. Cheng, S. Kundu, A. Krstic, Fast statistical timing analysis by probabilistic event propagation, in: DAC '01: Proceedings of the 38th Conference on Design Automation, 2001, pp. 661–666.

[3] A. Agarwal, D. Blaauw, V. Zolotov, S. Vrudhula, Computation and refinement of statistical bounds on circuit delay, in: DAC '03: Proceedings of the 40th Conference on Design Automation, 2003, pp. 348–353.

[4] A. Devgan, C. Kashyap, Block-based static timing analysis with uncertainty, in: ICCAD '03: Proceedings of the 2003 IEEE/ACM International Conference on Computer-aided Design, 2003, pp. 607–614.

[5] H. Chang, S.S. Sapatnekar, Statistical timing analysis under spatial correlations, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24 (9) (2005) 1467–1482.

[6] C. Visweswariah, K. Ravindran, K. Kalafala, S.G. Walker, S. Narayan, First-order incremental block-based statistical timing analysis, in: DAC '04: Proceedings of the 41st Annual Conference on Design Automation, 2004, pp. 331–336.

[7] J. Le, X. Li, L.T. Pileggi, STAC: statistical timing analysis with correlation, in: DAC '04: Proceedings of the 41st Annual Conference on Design Automation, 2004, pp. 343–348.

[8] A.E. Gattiker, S.R. Nassif, R. Dinakar, C. Long, Timing yield estimation from static timing analysis, in: ISQED '01: 2nd International Symposium on Quality of Electronic Design, 2001, pp. 437–442.

[9] J.-J. Liou, A. Krstic, L.-C. Wang, K.-T. Cheng, False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation, in: DAC '02: Proceedings of the 39th Conference on Design Automation, 2002, pp. 566–569.

[10] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gala, R. Panda, Path-based statistical timing analysis considering inter and intra-die correlations, in: ACM/IEEE International Workshop on Timing Issues, 2002.

[11] J.A.G. Jess, K. Kalafala, S.R. Naidu, R.H.J.M. Otten, C. Visweswariah, Statistical timing for parametric yield prediction of digital integrated circuits, in: DAC '03: Proceedings of the 40th Conference on Design Automation, 2003, pp. 932–937.

[12] M. Orshansky, A. Bandyopadhyay, Fast statistical timing analysis handling arbitrary delay correlations, in: DAC '04: Proceedings of the 41st Annual Conference on Design Automation, 2004, pp. 337–342.

[13] Y. Zhan, A.J. Strojwas, X. Li, L.T. Pileggi, D. Newmark, M. Sharma, Correlation-aware statistical timing analysis with non-Gaussian delay distributions, in: DAC '05: Proceedings of the 42nd Annual Conference on Design Automation, 2005, pp. 77–82.

[14] L. Zhang, W. Chen, Y. Hu, J.A. Gubner, C.C.-P. Chen, Correlation-preserved non-Gaussian statistical timing analysis with quadratic timing model, in: DAC '05: Proceedings of the 42nd Annual Conference on Design Automation, 2005, pp. 83–88.

[15] V. Khandelwal, A. Srivastava, A general framework for accurate statistical timing analysis considering correlations, in: DAC '05: Proceedings of the 42nd Annual Conference on Design Automation, 2005, pp. 89–94.

[16] H. Chang, V. Zolotov, S. Narayan, C. Visweswariah, Parameterized block-based statistical timing analysis with non-Gaussian parameters, in: DAC '05: Proceedings of the 42nd Annual Conference on Design Automation, 2005, pp. 71–76.

[17] K.R. Heloue, F.N. Najm, Statistical timing analysis with two-sided constraints, in: ICCAD '05 Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design, 2005, pp. 829–836.

[18] D. Sinha, H. Zhou, A unified framework for statistical timing analysis with coupling and multiple input switching, in: ICCAD '05: Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design, 2005, pp. 837–843.

[19] J. Singh, S. Sapatnekar, Statistical timing analysis with correlated non-Gaussian parameters using independent component analysis, in: DAC '06: Proceedings of the 43rd Annual Conference on Design Automation, ACM Press, New York, NY, USA, 2006, pp. 155–160.

[20] D. Blaauw, V. Zolotov, S. Sundareswaran, Slope propagation in static timing analysis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21 (10) (2002) 1180–1192.

[21] F. Brglez, D. Bryan, K. Koźmiński, Combinational profiles of sequential benchmark circuits, in: Proceedings of ISCAS, 1989, pp. 1929–1934.

[22] H. Chang, S.S. Sapatnekar, Statistical timing analysis considering spatial correlations using a single PERT-like traversal, in: ICCAD '03: Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design, 2003, pp. 621–625.

[23] Y. Saad, Iterative Methods for Sparse Linear Systems, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[24] Y. Cao, T. Sato, M. Orshansky, D. Sylvester, C. Hu, New paradigm of predictive MOSFET and interconnect modeling for early circuit simulation, in: Proceedings of Custom Integrated Circuits Conference, 2000, pp. 201–204.

[25] M. Wang, X. Yang, M. Sarrafzadeh, Dragon2000: standard-cell placement tool for large industry circuits, in: ICCAD '00: Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design, 2000, pp. 260–263.

[26] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, McGraw-Hill Higher Education, 2001.

[27] A. Ramalingam, A.K. Singh, S.R. Nassif, M. Orshansky, D.Z. Pan, Accurate waveform modeling using singular value decomposition with applications to timing analysis, in: DAC '07: Proceedings of the 44th Annual Conference on Design Automation, ACM Press, New York, NY, USA, 2007, pp. 148–153.

Anand Ramalingam received his B.E. degree in Electronics and Communication Engineering from P.S.G College of Technology, Coimbatore, India, in 2000, M.S. degree in Electrical Engineering from Stanford University in 2003 and Ph.D. degree in Computer Engineering from The University of Texas at Austin in 2007. He is currently a Member of Consulting Staff at Magma Design Automation Inc., Austin, TX. His research interests include statistical timing analysis, circuit simulation, and macro-modeling of analog cells.

Ashish Kumar Singh received his B.Tech. degree in Computer Science from the Indian Institute of Technology, India, in 2001. He has received his M.S. degree from Royal Institute of Technology, Stockholm and Ph.D. degrees in electrical engineering from University of Texas, Austin, in 2003 and 2007 respectively.

His research has been in the development of statistical and robust optimization algorithms for integrated circuit synthesis under manufacturing and operation uncertainty. He has received the Best Paper Award at 2006 International Conference for Computer Aided Design.

He has held postdoctoral research position with Professor Michael Orshansky at Univeristy of Texas, Austin from 2009 to 2010. Prior to joining the postdoctoral position he held the position of senior member of technical staff at Magma Design Automation. He is currently a Member of Research and Software Development Staff at Terra Technology, Schaumburg, IL.

Sani received his Bachelors degree from the American University of Beirut in 1980, and his Masters and Ph.D. degrees from Carnegie-Mellon University in 1981 and 1985, respectively. He worked at Bell Laboratories until 1996, then joined the IBM Austin Research Laboratory where he is currently. He has authored numerous conference and journal publications, received five Best Paper awards (IEEE Trans. CAD, ICCAD, DAC, ISQED and ICCD), authored invited papers to ISSCC, IEDM, ISLPED, HOTCHIPS, and CICC, and given Keynote and Plenary presentations at Sasimi, ESSCIRC, BMAS, SISPAD, SEMICON, PATMOS and VLSI-SOC. He is an IEEE Fellow (2008), a member of the IBM Academy of Technology, a member of the ACM, and is an IBM Master Inventor with more than 40 patents.

Gi-Joon Nam received his B.S. degree in Computer Engineering from Seoul National University in Seoul, Korea, and M.S. and Ph.D degrees in computer science and engineering from the University of Michigan, Ann Arbor, Michigan. Since 2001, he has been working with the IBM Austin Research Lab in Austin, Texas, where he is primarily working in the physical design space. His general interests include computer-aided design algorithms, combinatorial optimization, VLSI system design, and computer architecture. Dr. Nam has served on the technical program committees for various conferences including the International Conference on Computer Aided Design (ICCAD), the International Symposium on Physical Design (ISPD), the Asia and South Pacific Design Automation Conference (ASPDAC), etc., and was a general chair of ISPD 2009. He is the recipient of 1st place award for the 38th Design Automation Conference Student Design Contest and SIGDA technical leadership award from ACM.

Michael Orshansky is an Associate Professor of Electrical and Computer Engineering at the University of Texas, Austin. He received his Ph.D. degree in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 2001. Prior to joining UT Austin, he was a Research Scientist and Lecturer with the Department of EECS at UC Berkeley. His research interests include design optimization for robustness and manufacturability, statistical timing analysis, and design in fabrics with extreme defect densities. He is the recipient of the National Science Foundation CAREER award for 2004 and ACM SIGDA Outstanding New Faculty Award in 2007. He received the 2004 IEEE Transactions on Semiconductor Manufacturing Best Paper Award, as well as Best Paper Awards at the Design Automation Conference 2005, International Symposium on Quality Electronic Design (ISQED) 2006, and International Conference on Computer-Aided Design (ICCAD) 2006. He is the author, with Sani Nassif and Duane Boning, of the book "Design for Manufacturability and Statistical Design: A Constructive Approach."

David Z. Pan (S^{1}97-M^{1}00-SM^{1}06) received his B.S. degree from Peking University, M.S. and Ph.D. degrees from University of California, Los Angeles (UCLA). From 2000 to2003, he was a Research Staff Member with IBM T. J. Watson Research Center. He is currently an Associate Professor at the Department of Electrical and Computer Engineering, the University of Texasat Austin (UT Austin). He has published over 140 papers in international conferences and journals, and is the holder of 8 US patents. His research interests include nanometer physical design, design for manufacturability and reliability, vertical integration design and technology, and design/CADfor emerging technologies.

He has served as an Associate Editor for IEEE Transactions on ComputerAided Design of Integrated Circuits and Systems (TCAD), IEEE Transactions on Very Large Scale Integration Systems (TVLSI), IEEE Transactions on Circuits and Systems PART I (TCAS-I) and PART II (TCAS-II), Journal of Computer Science and Technology (JCST), and IEEE CAS Society Newsletter. He has served as the Chair of the IEEE CANDE Committee and the ACM/SIGDA Physical Design Technical Committee (PDTC). He is in the Design Technology Working Group of International Technology Roadmap for Semiconductor. He has served in the Technical Program Committees of major VLSI/CAD conferences, including ASPDAC (Track Chair), DAC (Track Chair), DATE, ICCAD, ISPD (Program Chair), ISQED (Topic Chair), ISCAS (CAD Track Chair), SLIP (Publication Chair), GLSVLSI, ACISC (Program Co-chair), ICICDT (Award Chair), and VLSI-DAT (EDA Track Chair). He is the GeneralChair of ISPD 2008, General Chair of ACISC 2009, and Steering Committee Chair of ISPD 2009.

He has received a number of awards for his research contributions and professional services, including ACM/SIGDA Outstanding New Faculty Award (2005), NSF CAREER Award (2007), SRC Inventor Recognition Award three times (2000 and 2008), IBM Faculty Award four times (2004–2006, 2010), UCLA Engineering Distinguished Young Alumnus Award (2009), Best Paper Award at ASPDAC 2010, Best IP Award at DATE 2010, SRC Techcon Best Paper in Session Award (1998 and 2007), Best Student Paper Award at ICICDT 2009, a number of Best Paper Award Nominations at DAC/ICCAD/ASPDAC/ISPD, Dimitris Chorafas Foundation Research Award (2000), ISPD Routing Contest Awards (2007), eASIC Placement Contest Grand Prize (2009).