# A General Graph Based Pessimism Reduction Framework for Design Optimization of Timing Closure

Fulin Peng[1], Changhao Yan[*1], Chunyang Feng[2], Jianquan Zheng[2], Sheng-Guo Wang[3], Dian Zhou[1,4], Xuan Zeng[*1]
[1] ASIC & System State Key Lab, Dept. of Microelectronics, Fudan University, Shanghai, China,
[2] Design Group, Synopsys Inc., Shanghai, China,
[3] Dept. of ECE, University of North Carolina at Charlotte, Charlotte, USA,
[4] Dept. of EE, University of Texas at Dallas, Dallas, USA

## ABSTRACT

In this paper, we develop a general pessimism reduction framework for design optimization of timing closure. Although the modified graph based timing analysis (mGBA) slack model can be readily formulated into a quadratic programming problem with constraints, the realistic difficulty is the size of the problem. A critical path selection scheme, a uniform sampling method with the sparse characteristics of the optimal solution, and a stochastic conjugate gradient method are proposed to accelerate the optimization solver. This modified GBA is embedded into design optimization of timing closure. Experimental results show that the proposed solver can achieve 13.82x speedup than gradient descent method with similar accuracy. With mGBA, the optimization of timing closure can achieve a better performance on area, leakage power, buffer counts.

## 1. INTRODUCTION

In the modern IC design flow, static timing analysis (STA) is a critical step towards timing validation and optimization of designs [1]. It is essential to identify critical paths and optimize timing of them, for achieving timing closure, determining the clock frequency and avoiding over-design. However, rapid growing complexities of designs and consideration of on-chip variations (OCV) make timing closure more and more difficult and time prohibition [2].

In practical engineering, conventional OCV timing analysis utilizes a penalty factor, termed *derating* factor, for example 1.2 multiplying the original delay of the gates, to increase delay of all paths on the entire circuit to increase design margin [3]. However, such simple method becomes too pessimistic at nanometer process node with requirements of high performance and low chip area. Statistical static timing analysis (SSTA) have been proposed for many years to process variations in STA [4][5][6]. The basic idea of SSTA is to statistically consider the variations and accurately analyze the circuit delay. While SSTA methods can sharply reduce design pessimism, it suffers the drawbacks of lack of silicon verification [7], the limited availability of statistical foundry data in a standard format [7], and huge size of statistical timing libraries [3].

Recently, an Advanced On-Chip Variation (AOCV) method is explored to handle on-chip variations in a more practical way [8]. It leverages on a pre-characterized derating table, in which the derating factors are depended on the number of logic cells of paths and the distance of two endpoints of the paths.

AOCV method can be roughly categorized into graph-based analysis (GBA) and path-based analysis (PBA). GBA analysis performs worst-case analysis of a circuit design, and it will choose the most conservative values to bound the worst-case path through a cell. However, PBA uses path-specific *cell-depth* and location-based bounding box values to determine derating values.

The delays of paths by PBA are calculated path by path and are accurate enough. The speed of PBA can be improved by a parallel framework with MapReduce [9]. However, for the fact that the number of paths in realistic designs can easily exceed 100 million, PBA is rarely applied for the optimization flow of timing closure and mainly for sign-off stage [10], due to its extremely high runtime and memory demand.

On the contrary, GBA is widely adopted by industrial tools because of its low computational cost. However, introducing too much pessimism significantly increases the difficulty for the following optimization of timing closure, which is the most critical phase in modern system-on-chip implementation [10], as process nodes shrink towards nanoscale. In order to alleviate the pessimism of GBA, a gradient AOCV methodology is proposed by Shaikh et al. [11]. This method can only remove the GBA pessimism caused by AOCV, while other features such as worst slew propagation and clock reconvergence pessimism removal (CRPR) [12] are left aside. Recently, Ritesh et al. [13] attempts to address the general GBA pessimism problem by using path margin techniques. Although it is a general solution, it also suffers from the runtime problem due to its path-based nature.

In this paper, we proposed a modified graph based analysis algorithm, which holds the speed advantage of GBA, and simultaneously is with extremely high accuracy compared with original GBA. This proposed method can significantly reduce the pessimism of path delays and accelerate the optimization of timing closure. The key idea is adding a *weighting* factor for every delay unit (gate) and optimizing these *weighting* factors with the objection of minimizing the timing slack differences of critical paths between our modified GBA's and PBA's (golden). The main contributions includes: 1) We firstly propose a modified GBA slack calculation model and integrate the novel and accurate slack model into the implementation flow of optimizing timing closure. 2) Although the modified GBA slack model can be readily formulated into a quadratic programming problem with constraints, however, the main challenge is the problem scale. A heuristic scheme is proposed to select critical paths, which decreases the path number from 100 million to several million. Then considering the sparse characteristics of the optimal solution, we propose a uniform row sampling algorithm and combine with a stochastic conjugate gradient method to accelerate the optimization solver. 3) Based on industrial designs, experimental results show that our proposed solver can achieve 13.82x speedup than gradient descent method with similar accuracy. The modified GBA approach can improve averagely

* Corresponding authors: yanch@fudan.edu.cn, xzeng@fudan.edu.cn

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3195973

43.79% on accuracy of correlation pass ratio compared with the original GBA. In addition, applying the general pessimism reduction framework to implementation flows, the circuits can achieve a better performance (e.g., area, leakage power, buffer counts).

The remainder of this paper is organized as follows. Section 2 briefly introduces the timing analysis basics and gaps between GBA and PBA. Section 3 depicts the proposed method and implementation details. Section 4 shows experimental results. Section 5 gives a short conclusion.

# 2. PRELIMINARIES

## 2.1 Setup Slack and Hold Slack

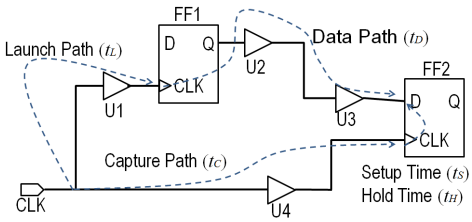


Fig. 1 Setup slack and hold slack.

Fig. 1 shows a simple synchronous circuit example. The timing constraints of the combinational circuit between two flip-flops should satisfy both setup slack $s_{setup}$ and hold slack $s_{hold}$ being larger than zero [1], i.e.

$$\begin{aligned} s_{setup} &= \min(t_C + T) - \max(t_L + t_D + t_S) > 0 \\ s_{hold} &= \min(t_L + t_D) - \max(t_C + t_H) > 0 \end{aligned}, \quad (1)$$

where capture path delay $t_C$, launch path delay $t_L$, data path delay $t_D$, setup time $t_S$ and hold time $t_H$ are illustrated in Fig. 1.

## 2.2 Background of GBA and PBA

Besides the tremendously huge number of paths, the on-chip variations make the accurate calculations of the setup and hold slack more difficult. Practical engineering chooses a *suitable* derating factor for every gate as the penalty of on-chip variations.

The suitable derating factor of a data path is depended on the number of logic gates of this data path, i.e. cell depth, and the distance of two endpoints of the data path [8]. The predefined lookup table of derating factor is supplied by foundries with the shape as shown in Table 1. From Table 1, we can see that as the number of gates increases, the derating factor decreases. This is attributed to the variation cancellation of logic gates.

Before looking up the derating factor table, one needs to calculate the cell depth of every gate. gives a simple case to demonstrate the differences of cell depth calculation between GBA and PBA.

Table 1 Example of lookup table of derating factor

| Distance \ Cell-Depth | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| 500 nm | **1.30** | **1.25** | **1.20** | **1.15** |
| 1000 nm | 1.32 | 1.27 | 1.23 | 1.18 |
| 1500 nm | 1.35 | 1.31 | 1.28 | 1.25 |

For PBA, the cell depth of a gate on a data path is simply the number of gates on this path, for example the green number 6 of the data path between FF1 and FF4 as shown in Fig. 1. Therefore, the data path delay from FF1 to FF4 by PBA is

$$d_{pba} = 100\text{ps} \times 1.15 \times 6 = 690\text{ps}, \quad (2)$$

where the delays of all gates are simply assumed to be 100*ps*.

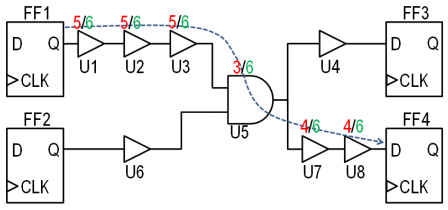


Fig. 2 Cell depth of GBA and PBA.

PBA analysis is accurate enough. However, we can see that for a given gate, its delay is variable with different paths, and it is nearly impossible to save all path related delays of every gate. Therefore, the delays of data paths for PBA can be fundamentally calculated only path by path, and it is only suitable for the sign-off stage of timing closure.

For GBA, the cell depth of a gate is the *worst* cell depth, i.e. choosing the minimal number of gates among all data paths through this gate *g*, for larger derating factor and design safety. Taking gate U1 for example, its cell depth is 5 from FF1 to FF3 and is 6 from FF1 to FF4. Therefore, GBA choose the minimal 5 as the cell depth of gate U1 as the red number shown in Fig. 2. Therefore, the data path delay from FF1 to FF4 by GBA is

$$d_{gba} = 100\text{ps} \times (1.20 + 1.20 + 1.20 + 1.30 + 1.25 + 1.25) = 740\text{ps}. \quad (3)$$

Finally, in this case, the delay gap between GBA and PBA is 50 ps. In fact, for each gate, the cell depth by GBA is always smaller or equal to that by PBA. Therefore, GBA chooses bigger derating factors and are more pessimistic on timing slack than PBA.

For GBA, if a circuit topology is given, the delays of gates are constant, with no respect to paths. Meanwhile, the output timing of a gate $t_{output}$ can be estimated efficiently as

$$t_{output} = max(t_{inputs}) + d_{gate}, \quad (4)$$

where $t_{inputs}$, $d_{gate}$ are the timing of all inputs and delay of this gate, respectively. Though the max function could introduce uncontrolled pessimism for some paths, the high efficiency of GBA still make it prevalent in the implementation stage of optimizing timing closure.

To deal with the accuracy gap, designers usually use GBA to pessimistically identify all the violating paths in the design and then subject them to PBA to validate. If there still exist timing violations, some modifications such as inserting buffers, sizing and etc. are applied for timing closure. However, as the accuracy gap between GBA and PBA grows, the number of paths requiring PBA also grows and this is causing a large computational bottleneck.

# 3. PROPOSED APPROACH

## 3.1 Problem Formulation

A modified graph based timing analysis method is proposed in this paper. The key idea is adding a *weighting* factor for every delay gate and optimizing these *weighting* factors with the objection of minimizing the timing slack differences of all paths between our modified GBA's and the golden PBA's, i.e. the problem can be formulated as follow

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) = \left\| \mathbf{s}_{gba'}(\mathbf{x}) - \mathbf{s}_{pba} \right\|_2 \\ &s.t.\ \ s_{gba',i}(\mathbf{x}) \le s_{pba,i} + \varepsilon \,|\, s_{pba,i} \,|,\ i = 1, 2, ..., m,\ \varepsilon > 0 \\ &\mathbf{s}_{pba} = [s_{pba,1}, ..., s_{pba,m}]^{\mathrm{T}} \\ &\mathbf{s}_{gba'}(\mathbf{x}) = [s_{gba',1}(\mathbf{x}), ..., s_{gba',m}(\mathbf{x})]^{\mathrm{T}} \\ &\mathbf{x} = [x_1, ..., x_n]^{\mathrm{T}} \end{aligned}, \quad (5)$$

where for a given circuit, the total number of timing paths and gates are $m$ and $n$ respectively, and usually $m >> n$. $\mathbf{x}$ is the vector of weighting factor for all delay gates. $\mathbf{s}_{pba}$ and $\mathbf{s}_{gba'}(\mathbf{x})$, are the vectors of all path slacks by PBA and our modified GBA method respectively, and $\mathbf{s}_{pba}$ is taken as true results. The optimization object is minimizing the total timing slack error of all paths with the constraints that timing slack of modified GBA method should not less than the PBA's timing result, and $\varepsilon$ is the error tolerance. $\| \cdot \|_2$ is the 2-norm of a vector.

The constraints can be removed with adding the penalty factor $w$, and Eq. (5) becomes

$$\min_{\mathbf{x}} f(\mathbf{x}) = \left\| \mathbf{s}_{gba'}(\mathbf{x}) - \mathbf{s}_{pba} \right\|_2 + \sum_{i \in violative\ path\ set} w[s_{gba',i}(\mathbf{x}) - (s_{pba,i} + \varepsilon \mid s_{pba,i} \mid)]^2, \quad (6)$$

where *violated path set* is the set of paths which violate the constraints in Eq. (5) , and all other definition is the same as Eq. (5).

For GBA, if the derating factor $\lambda_j$ of the $j$-th gate is obtained advanced, the timing slack of the $i$-th path can be calculated simply with the summation of gate's delay multiply its derating factor as follow

$$s_{gba,i} = \sum_{j=1}^{n} \delta_{ij} d_j \lambda_j, i = 1, \ldots, m, \quad (7)$$

where $d_j$ is the original delay of the $j$-th gate, $\delta_{ij}$ is the indicator function where $\delta_{ij} = 1$, only if the $j$-th gate lies in the $i$-th path, and $n$ is the total number of gates.

For modified GBA, we add an extra weighting factor $x_j$ for the $j$-the gate, and therefore, the timing slack of the $i$-th path becomes

$$s_{gba',i}(\mathbf{x}) = \sum_{j=1}^{n} (\delta_{ij} d_j \lambda_j) \cdot x_j, i = 1, \ldots, m. \quad (8)$$

Eq. (8) can be rewritten in matrix form as

$$\mathbf{s}_{gba'}(\mathbf{x}) = \mathbf{A} \cdot \mathbf{x}, \mathbf{A} \in R^{m \times n}, \quad (a_{ij}) = \delta_{ij} d_j \lambda_j, i = 1, \ldots, m, j = 1, \ldots, n, \quad (9)$$

where $a_{ij}$ is the entry of matrix $\mathbf{A}$.

## 3.2 Critical Path Selecting Scheme

For a realistic design, the total number of timing paths $m$ can be easily greater than 100 million, so the number of equations in Eq. (5) or Eq. (6) is too much, and it is impractical to solve them directly. Fortunately, what we need to do in implementation flow is to eliminate the timing violations. Therefore, we should pay more attention to timing-violated paths, i.e. negative-slack paths, and can reasonably ignore other timing paths whose slacks are positive. However, the total number of violated paths is usually still big enough and we cannot select all the negative-slack paths out, so the suitable critical path selecting scheme becomes crucial for high accuracy of the modified GBA method.

A simple path selecting scheme is choosing the top $m'$ worst paths after sorting the path slack results of original GBA, where $m'$ is the number of paths we need. This scheme sounds reasonable for results of the original GBA should be a good initial solution. However, the following experimental shows that such scheme is not good enough.

For a small design which has totally 8444 violated timing paths with 1437 variables or gates, *i.e.* $n$ = 1437. We simply measure the error as

$$\varphi(\mathbf{x}) = \frac{\left\| \mathbf{s}_{gba'}(\mathbf{x}) - \mathbf{s}_{pba} \right\|_2}{\left\| \mathbf{s}_{pba} \right\|_2}, \quad (10)$$

where slack of PBA $\mathbf{s}_{pba}$ is taken as golden result, and the measurement is always with 8444 violated timing paths.

After optimizing this small design as Eq. (5) directly with all 8444 violated timing paths, the error $\varphi$= 4.1%. Following the above scheme, we select the top $m'$ = 2000 critical timing paths, the error tremendously increase to $\varphi$ = 72.4%. The weak of this path selecting scheme is too concentrated to the critical gates, while not covering the less critical gates. The total number of delay gates is 1437, it covers only 682 about 47.46% of total gates.

Therefore, in order to catch the critical paths and cover more gates simultaneously, the second scheme is for every endpoint of paths, picking the top $k'$ critical timing paths by only sorting paths ending at this endpoint, which can also save the runtime compared to sort all the violated paths. Taking the endpoint $D$ port of FF3 in Fig. 2 for example, only path FF1-FF3 and FF2-FF3 are sorted. If there is $p$ endpoints, the total number of selecting paths $m' = k' p$. Applying the second selecting path scheme for the above case, we also select 2000 timing paths by traversing the endpoint in timing graph with $k'$ = 20, the error $\varphi$ = 5.11%, which is much better. The total number of delay gates is 1437, it covers 1370 about 95.34% of total gates. In the experimental section, we set $k'$ = 20, $m' \leq 5 \times 10^6$.

## 3.3 Efficient Solver

After applying the above critical paths selecting scheme, the scale of the original problem can be reduced to the order of millions. However, the aforementioned optimization problem is still huge enough, and a direct optimization solver may need a long time for big designs. Considering the extreme sparseness of the optimal solution $\mathbf{x}^*$, we propose a novelty uniform rows sampling technique to further decrease the scale of problem to 100 thousands, which is then solved by a stochastic conjugate gradient method based on randomized Kaczmarz algorithm [14][15] for acceleration.

### *A. Uniform Rows Sampling Method*

Considering that in each timing path, the number of cells is often less than one hundred, which means matrix $\mathbf{A}$ in Eq. (9) is sparse and many columns only have several non-zero elements. It is intuitive to consider that the optimal solution $\mathbf{x}^*$ of Eq. (5) is sparse. For a small case, we can obtain the optimal solution $\mathbf{x}^*$ and plot the histogram of $\mathbf{x}^*$ in Fig. 3. It can be concluded that the solution $\mathbf{x}^*$ is extremely sparse, where 95.9% entries of $\mathbf{x}^*$ is near around zero within the interval of [-0.01, 0.01]. For iterative methods, the initial of $\mathbf{x}$ is given as $\mathbf{0}$, and 95.91% of them are nearly correct indeed. Thus, a small, manageable set of equations, i.e. the rows and corresponding right hand, can be selected out to approximate the original problem.

Then how to select the effective rows become the key to this problem. According to [16], random sampling has become a critical tool in solving massive matrix problem. For theoretical performance guarantees, each row must be sampled with probability proportional to its *statistical leverage score*. Unfortunately, *leverage scores* are difficult to calculate because finding them involves computing $(\mathbf{A}^T\mathbf{A})$, which is as slow as solving our problem. In practice, rows are sampled uniformly under the assumption that data has low coherence [17]. Based on the facts, we propose an algorithm which uniformly and randomly selects $m''$ rows of equations from $\mathbf{A}$ to form a smaller $\mathbf{A}''$.

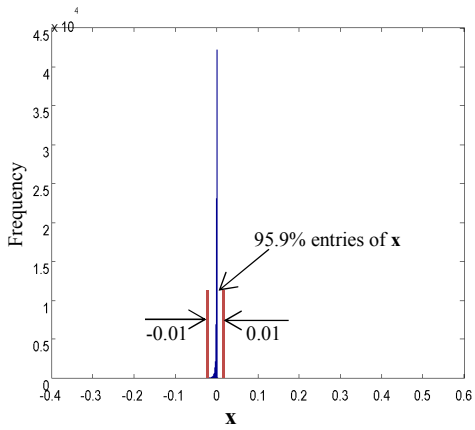

Fig. 3 Example to illustrate the distribution of optimal solution $\mathbf{x}^*$

How many equations should be chosen is another important issue. Theoretically, the number of selection rows $m''$ should at least be larger than the number of non-zero elements of optimal solution $\mathbf{x}^*$. However, the optimal solution $\mathbf{x}^*$ is unknown in advance. We use a simple doubling iteration strategy of $m''$ to automatically obtain the optimal parameter $m''$.

**Algorithm 1: Uniform Sampling Scheme**

**Input**: $\mathbf{x}_0$ initial solution,
$r_0$ initial rows selecting ratio,
$\varepsilon_u$ tolerance error.
**Output**: optimal $\mathbf{x}^*$.

1. Sampling rows from matrix $\mathbf{A}$ in Eq. (9) with $r_0$ uniformly.
2. **While** $\|\mathbf{x}_{k+1}-\mathbf{x}_k\|_2 / \|\mathbf{x}_k\|_2 > \varepsilon_u$
3. Solve the reduced problem with Algorithm 2.
4. Update the selecting ratio $r_{k+1} = 2\, r_k$.
5. Sampling rows from $\mathbf{A}$ in Eq. (9) with $r_{k+1}$ uniformly.
6. **End While**

The proposed uniform row selection algorithm is list in Algorithm 1. First, we sample a small part $r_0$ of equations from matrix $\mathbf{A}$ and $\mathbf{b}$, which is solved by an efficient solver described in the next subsection. If solution $\mathbf{x}_k$ is not convergence, we increase the path selecting ratio doubly. In experiment section, we set $r_0 = 10^{-5}$ and $\varepsilon_u$=0.1.

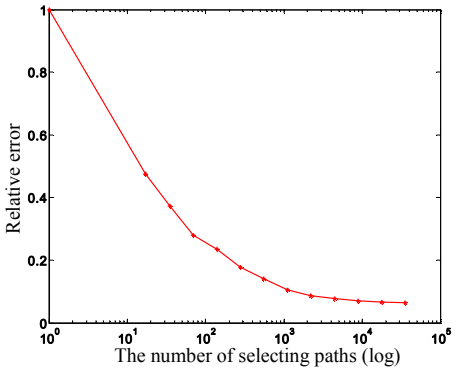

Fig. 4 The accuracy of $\mathbf{x}$ vs. the number of row selection.

Fig. 4 demonstrates the effectiveness of the proposed scheme in a small case. When the number of equations increases, the solutions $\mathbf{x}$ can be converged sharply.

*B. Efficient solver of stochastic conjugate gradient method*

After applying the uniform rows sampling scheme, we choose $m''$ equations out of the original optimization problem, where $m''$ is usually in the order of 10 thousands.

We propose an efficient solver based on randomized Kaczmarz algorithm, which belongs to a kind of stochastic gradient descent method. Unlike the conventional gradient methods computing all gradient directions at a descent step, it only calculates the derivative on $k''$ rows of randomly selected directions, where the probability of each row being selected follows the Euclidean norm of the row [14] as

$$P(j) = \|\mathbf{a}_j\|_2^2 / \sum_{l=1}^{m''} \|\mathbf{a}_l\|_2^2,\ j = 1, 2, ..., m'', \qquad (11)$$

where $m''$ is the total number of rows of equations as shown in line 4-6 of Algorithm 2 and $[\mathbf{a}_1{}^T, \mathbf{a}_2{}^T, \ldots, \mathbf{a}_{m''}{}^T]^T = \mathbf{A}''$.

**Algorithm 2: Efficient Solver with dynamic step size control**

**Input**: $f(\mathbf{x})$ objective function,
rows selected number $k''$,
$\mathbf{x}_0$ initial solution,
$\varepsilon_c$ convergence parameter,
$s$ step size.
**Output**: optimal $\mathbf{x}^*$.

1. Initialize $\mathbf{g}_0 = \mathbf{0}$ and $\mathbf{d}_0 = \mathbf{0}$.
2. **While** $\|\mathbf{x}_k - \mathbf{x}_{k-1}\|_2 / \|\mathbf{x}_{k-1}\|_2 > \varepsilon_c$
3. Calculate each row's selected probability as Eq. (11).
4. Take $k''$ rows with its own probability in $\{1, 2, \ldots, m''\}$.
5. Calculate gradient $\mathbf{g}_k$ of $\mathbf{x}$ with the $k''$ selected rows.
6. Normalize the $\mathbf{g}_k$.
7. Compute the Polak-Ribiere parameter
$$\beta_k = \frac{\mathbf{g}_k^T(\mathbf{g}_k - \mathbf{g}_{k-1})}{\|\mathbf{g}_{k-1}\|_2^2}.$$
8. Compute the conjugate directions
$$\mathbf{d}_k = -\mathbf{g}_k + \beta_k \mathbf{d}_{k-1}.$$
9. Compute the dynamic step size $\alpha_k = s / \|\mathbf{d}_k\|_2$.
10. Update the solution $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k$.
11. **End While**

We use conjugate gradient method with a carefully dynamic step-size control method [15] to find the optimal $\mathbf{x}^*$ as shown in line 7-9 of Algorithm 2. Note that the convergence condition is the relative variation of $\mathbf{x}$ as shown in line 2 of Algorithm 2. The default value of $k''$ is set to 2% of the total rows of matrix $\mathbf{A}''$, initial solution $\mathbf{x}_0 = \mathbf{0}$, tolerance error $\varepsilon_c$ =$10^{-3}$, and $s$ =0.02.

### 3.4 Timing Closure Optimization Framework with modified GBA Flow

Fig. 5 gives the overview of timing closure optimization framework with our proposed modified GBA flow. The left of is the timing closure optimization framework. First, a timing graph is constructed from the circuit design and timing constraints specification (e.g., clock frequency, timing derating factors, external delays, etc.). Then the timing analysis is invoked with our proposed modified GBA flow, and the full timing graph is updated. Obtaining the full timing graph, millions of various modifications (e.g., gate sizing, wire sizing, inserting buffer, etc.) of the circuit design are applied for timing closure. Since many of the arrival times of the circuit design remain unchanged, it is

unnecessary and impractical to invoke the modified GBA flow for full timing analysis after every optimization transform. In contrast, we perform incremental timing update techniques [18] and evaluate the timing information after each modification, until it reaches timing closure.

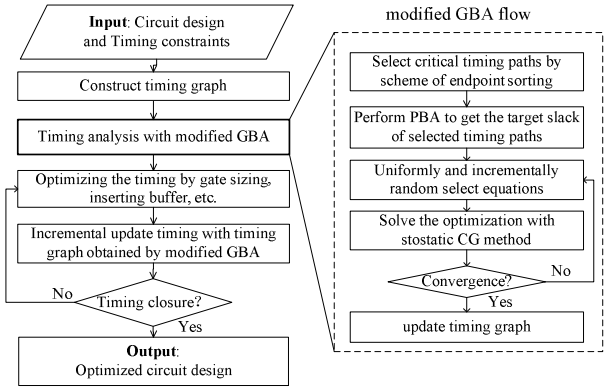


Fig. 5 Overview of timing closure optimization framework with modified GBA flow

The right of Fig. 5 is the modified GBA analysis flow. By the second endpoint scheme mentioned above, select the critical paths, whose timing is calculated by performing the GBA analysis. We use uniformly and incrementally random selection of equations to decrease problem size, and solving the optimization efficiently with stochastic CG method. The optimal weighting factors are obtained and the timing graph will be updated.

# 4. NUMERICAL EXAMPLES

In order to verify the effectiveness and efficiency of the proposed method, we implement the proposed modified GBA algorithms (mGBA) with C/C++ language, and run it on a server with 2.6GHz CPUs and 256GB memory. All test cases are from real industrial designs. Note that the technology nodes of designs are ranging from 65nm to 16nm, and the pessimism of these designs mainly comes from AOCV derating. The results of PBA analysis are adopted as *golden* results.

## 4.1 Speed and Accuracy Comparison of Optimization Solvers

In this experiment, we compare the speed and accuracy for gradient descent method without row selection (GD+w/o RS), the proposed stochastic conjugate gradient (SCG) method as Algorithm 2 depicts without row selection (SCG+w/o RS), and the uniform sampling method based on SCG as Algorithm 1 depicts with row selection (SCG+RS), where the accuracy is defined as the modeling squared error as (12)

$$mse = \left\| \mathbf{s}_{gba'}(\mathbf{x}) - \mathbf{s}_{pba} \right\|_2^2 / \left\| \mathbf{s}_{pba} \right\|_2^2 . \quad (12)$$

As we can see from Table 4, all three methods can obtain similar accuracy, where SCG+w/o RS and SCG+RS can slightly improve the accuracy by 0.052% and 0.098% on average, respectively. For speed comparison, both GD+w/o RS and SCG+w/o RS have no row selection, and their difference is only the gradient-like solver. From Table 4, the proposed SCG is 2.71x faster on average than conventional GD method. Besides, the difference of SCG+RS and SCG+w/o RS is with row selection or not. From Table 4, the proposed row selection method can achieve 5.10x speedup averagely. In all, the proposed SCG+RS is 13.82x speedup over the conventional GD method, which verifies the efficiency of proposed optimization solver.

## 4.2 Effectiveness of mGBA in Post-Route Optimization Framework

In order to demonstrate the effectiveness of the proposed mGBA model, we run the whole optimization Framework with original GBA and mGBA embedded.

Table 2 compares the quality of result (QoR) metrics, worst negative slack (WNS), total negative slack (TNS), chip area, leakage power and buffer inserted for conventional GBA and mGBA respectively. The baseline is flow with original GBA. As we can see, the post-route optimization framework with mGBA can averagely reduce 5.58% of chip area, 14.77% of leakage power, 4.84% of buffer insertion, 1.20% of WNS and 0.65% of TNS, compared with the optimization framework with GBA.

Table 2 QoR Improvement for Designs

| | WNS(%) | TNS(%) | area(%) | leakage(%) | buffer(%) |
|---|---|---|---|---|---|
| D1 | 0.68 | 0.50 | 3.27 | 17.59 | 2.16 |
| D2 | -2.94 | -0.02 | 2.17 | 9.85 | 1.88 |
| D3 | -0.01 | 0.14 | 8.08 | 17.63 | 5.04 |
| D4 | 0.15 | 0.07 | 2.60 | 2.14 | 1.65 |
| D5 | 1.36 | 1.58 | 2.54 | 0.03 | 1.10 |
| D6 | -0.93 | 0.45 | 10.15 | 23.31 | 10.14 |
| D7 | 1.43 | 0.04 | 2.87 | 9.66 | 0.86 |
| D8 | 14.19 | 2.89 | 8.76 | 14.97 | 6.32 |
| D9 | -1.83 | 0.75 | 4.81 | 29.99 | 5.82 |
| D10 | -0.06 | 0.12 | 10.51 | 22.53 | 13.45 |
| Avg. | **1.20** | **0.65** | **5.58** | **14.77** | **4.84** |

Some degradations of WNS and/or TNS (such as D2) are reasonable because our method is less pessimistic and may break the optimization loop earlier than original GBA. In engineering practice, it is not pre-requisite to waive every violated endpoint in the stage of post routing optimization, and usually no more than 100 violated endpoints is acceptable. If the timing optimization framework can exit normally, it must fulfill the timing requirement in the current stage.

## 4.3 Accuracy and Speed Comparison of GBA and mGBA within the optimization Framework

In the last subsection, we have demonstrated the effectiveness of mGBA in optimization flow. Compared to original GBA, this improvement comes from the more accurate timing analysis for critical timing paths.

Table 3 compares the accuracy of timing analysis results by GBA and mGBA. The results of PBA are adopted as *golden* results. The accuracy is measured with the metric of pass ratio $\varphi = n / N$, where $N$ is the total number of timing paths considered, and $n$ is the total number of *good* timing paths. Following the suggestions of engineers, a path is thought as *good*, if its relative error of path slack is less than 5% or the absolute error of path slack is less than 5ps compared with the golden results.

Table 3 Pass ratio comparison of GBA and mGBA

| | selected Timing Path($10^5$) | GBA (%) | mGBA(%) | pass ratio improvement (%) |
|---|---|---|---|---|
| D1 | 1.68 | 92.40 | 99.82 | 7.42 |
| D2 | 35.1 | 52.64 | 91.14 | 38.50 |
| D3 | 7.00 | 78.24 | 98.71 | 20.46 |
| D4 | 6.78 | 38.34 | 84.83 | 46.49 |
| D5 | 4.02 | 35.68 | 97.81 | 62.13 |
| D6 | 9.30 | 66.51 | 97.76 | 31.25 |
| D7 | 7.65 | 56.67 | 89.89 | 33.22 |
| D8 | 30.0 | 0.12 | 98.43 | 98.32 |
| D9 | 27.0 | 33.82 | 96.60 | 62.78 |
| D10 | 23.7 | 61.28 | 98.63 | 37.34 |
| Avg. | **15.2** | **51.57** | **95.36** | **43.79** |

Table 4 Accuracy and Speed Comparison of Optimization Solvers

| | GD + w/o RS | | | SCG + w/o RS | | | SCG + RS | | |
|---|---|---|---|---|---|---|---|---|---|
| | accuracy ($10^{-3}$) | time (s) | speedup | accuracy ($10^{-3}$) | time (s) | speedup | accuracy ($10^{-3}$) | time (s) | speedup |
| D1 | 1.06 | 150 | 1.00 | 0.93 | 108 | 1.39 | 1.48 | 59 | 2.53 |
| D2 | 7.86 | 4008 | 1.00 | 14.70 | 934 | 4.29 | 9.41 | 107 | 37.59 |
| D3 | 2.06 | 1045 | 1.00 | 1.27 | 412 | 2.53 | 0.95 | 98 | 10.63 |
| D4 | 1.34 | 734 | 1.00 | 0.60 | 310 | 2.36 | 1.26 | 120 | 6.12 |
| D5 | 1.02 | 298 | 1.00 | 0.71 | 153 | 1.95 | 1.04 | 57 | 5.26 |
| D6 | 1.20 | 1110 | 1.00 | 0.91 | 322 | 3.45 | 0.85 | 140 | 7.93 |
| D7 | 1.68 | 1078 | 1.00 | 1.38 | 304 | 3.54 | 1.56 | 102 | 10.60 |
| D8 | 6.14 | 2822 | 1.00 | 1.61 | 1027 | 2.75 | 1.55 | 177 | 15.98 |
| D9 | 3.56 | 2739 | 1.00 | 1.76 | 816 | 3.36 | 0.88 | 206 | 13.29 |
| D10 | 3.81 | 3797 | 1.00 | 0.59 | 2603 | 1.46 | 0.93 | 134 | 28.28 |
| Avg. | **2.97** | **1778** | **1.00** | **2.45** | **699** | **2.71** | **1.99** | **120** | **13.82** |

From Table 3, we can see that the proposed mGBA method can achieve on average 43.79% absolute improvement of pass ratio compared to the original GBA method. More importantly, no test case becomes worse than the original GBA method.

Although the post-route optimization framework need extra time to invoke the mGBA, with more accurate timing results, the optimization flow can converge more quickly. As shown in Table 5, the post-route optimization framework with mGBA is 1.21x speedup compared to that with original GBA.

Table 5 Runtime(s) comparison for timing closure optimization framework with GBA and mGBA embeded

| | GBA flow | post-route flow | | | speedup |
|---|---|---|---|---|---|
| | | post-route | mGBA | total | |
| D1 | 11678 | 8841 | 116 | 8957 | 1.30 |
| D2 | 50735 | 42173 | 1713 | 43886 | 1.16 |
| D3 | 78079 | 63017 | 997 | 64014 | 1.22 |
| D4 | 40576 | 38296 | 475 | 38771 | 1.05 |
| D5 | 63215 | 52501 | 223 | 52724 | 1.20 |
| D6 | 63225 | 49659 | 842 | 50501 | 1.25 |
| D7 | 9564 | 8011 | 984 | 8995 | 1.06 |
| D8 | 105751 | 82813 | 1858 | 84671 | 1.25 |
| D9 | 30945 | 23918 | 1850 | 25768 | 1.20 |
| D10 | 46446 | 33430 | 329 | 33759 | 1.38 |
| Avg. | **50021** | **40266** | **939** | **41205** | **1.21** |

# 5. CONCLUSIONS

In this paper, we present a refined path-oriented slack calculation model to alleviate excessive pessimism of GBA timing analysis. Based on the proposed model, a pessimism reduction framework is integrated into optimization flows to improve design performance. Critical paths selecting scheme and uniform sampling method are adopted to accelerate the solver. Numerical experiments based on industrial designs show that our proposed method can achieve significant improvement in correlation pass ratio. And the proposed efficient solver can run much faster than the conventional GD method without any loss of accuracy. Applying the general framework into implementation flow can achieve a better design performance.

# 6. ACKNOWLEDGEMENTS

This research was supported partly by National Key Research and Development Program of China 2016YFB0201304, partly by the National Major Science and Technology Special Project of China (2017ZX01028101-003), partly by National Natural Science Foundation of China (NSFC) research projects 61674042, 61574046, 61574044, 61774045, and 61628402, partly by the Recruitment Program of Global Experts (the Thousand Talents Plan） and partly by NSF Grant 1115564.

# 7. REFERENCES

[1] R. Chadha and J. Bhasker, *Static timing analysis for nanometer designs*. Springer US, 2009.

[2] C. Visweswariah, "Death, taxes and failing chips," *DAC*, pp. 343-347, Jun. 2003.

[3] A. Dasdan, K. Santanu and Y. Mustafa, "Derating for static timing analysis: Theory and practice," *ISQED*, pp. 719-727, Mar. 2009.

[4] C. Visweswariah, K. Ravindran, K. Kalafala et al., "First-order incremental block-based statistical timing analysis," *IEEE TCAD*, vol. 25, no. 10, pp. 2170-2180, 2006.

[5] C. Amin, N. Menezes, K. Killpack et al., "Statistical static timing analysis: How simple can we get?" *DAC*, pp. 652-657, Jul. 2005.

[6] J. Singh and S. Sapatnekar, "Statistical timing analysis with correlated non-Gaussian parameters using independent component analysis," *DAC*, pp. 155-160, Jul. 2006.

[7] D. Blaauw, K. Chopra, A. Srivastava et al., "Statistical timing analysis: From basic principles to state of the art," *IEEE TCAD*, vol. 27, no. 4, pp. 589-607, 2008.

[8] Synopsys Advanced On-Chip Variation (AOCV) Application Note, Version 6.0, Jan. 2013.

[9] W. Huang and M. Wong, "Accelerated path-based timing analysis with MapReduce," *ISPD*, pp. 103-110, Mar. 2015.

[10] A. Kahng, "New game, new goal posts: A recent history of timing closure," *DAC*, p. 4, Jun. 2015.

[11] R. Shaikh and V. Rajan, "Gradient AOCV methodology enabling graph-based timing closure with AOCV timing models," U.S. Patent No. 8806, 413, 12 Aug. 2014.

[12] J. Zejda and P. Frain, "General framework for removal of clock network pessimism," *ICCAD*, pp. 632-639, Nov. 2002.

[13] R. Shyamsukha, C. Feng, S. Radhakrishnan and T. Craven, "Selectively reducing graph based analysis pessimism," U.S. Patent No. 14/480, 543, 8 Sep. 2014.

[14] D. Needell, "Randomized Kaczmarz solver for noisy linear systems," *BIT Numerical Mathematics*, vol. 50, no. 2, pp. 395-403, 2010.

[15] J. Lin, and D. Zhou, "Learning theory of randomized Kaczmarz algorithm," *The Journal of Machine Learning Research*, vol.16, no. 1, pp. 3341-3365, 2015.

[16] M. Cohen, Y. Lee, C. Musco et al., "Uniform sampling for matrix approximation," *ITCS*, pp. 181-190, Jan. 2015.

[17] A. Haim, M. Petar, and T. Sivan, "Blendenpik: Supercharging lapack's least-squares solver," *SIAM Journal on Scientific Computing*, vol. 32, no. 3, pp. 1217-1236, 2010.

[18] IC Compiler II Timing Analysis User Guide, version M-2016.12-SP2, Dec. 2016.