Hybrid Modeling and Weighting for Timing-driven Placement with Efficient Calibration

Bangqi Fu
CSE Department, CUHK
bqfu21@cse.cuhk.edu.hk

Lixin Liu
CSE Department, CUHK
lxliu@cse.cuhk.edu.hk

Martin D.F. Wong
CSE Department, CUHK
mdfwong@cse.cuhk.edu.hk

Evangeline F.Y. Young
CSE Department, CUHK
fyyoung@cse.cuhk.edu.hk

ABSTRACT

Placement is a crucial step in the physical synthesis flow that significantly determines the timing performance of a design. In this paper, we propose a timing-driven global placement framework with a hybrid pin-based weighting scheme that considers both graph and path information and an optimization-friendly RC tree and wirelength model. A calibration method is proposed to further improve the incremental timing. Experiment results show over 37% improvement on TNS and 15% improvement on WNS, with 4.2% less HPWL on the ICCAD 2015 benchmark compared to the state-of-the-art GPU-accelerated differentiable timing-driven placer, while also being around 2x faster.

KEYWORDS

Physical design, Global placement, Timing, GPU Acceleration

1 INTRODUCTION

Placement plays a central role in VLSI physical synthesis flow as it greatly influences the circuit's PPA (power, performance and area). The quality of the placement solution has a strong correlation with the effectiveness of subsequent stages, such as routing, power optimization, and timing optimization.

As VLSI designs become increasingly complex, it is becoming necessary to consider multiple objectives during the placement stage. Among these objectives, timing considerations are particularly crucial, as timing optimization is time-consuming in physical synthesis cycles and directly influences the performance and reliability of the circuit.

Timing-driven placement can be categorized into net-based, path-based, and pin-based methods. Net-based methods prioritize timing-critical nets and aim to reduce the timing delay by shortening the net wirelength in an implicit manner. Common net-based methods can further be categorized into static net weighting[1, 10, 16, 25] and dynamic net weighting[9, 18]. Path-based methods[4, 11, 27], on the other hand, conduct path-based timing analysis and optimize the critical timing paths. Although it achieves high quality, the number of paths grows exponentially and scalability issues arise as the design size increases, leading to substantial runtime costs. Pin-based methods offer a trade-off between the previous two approaches, employing graph-based timing analysis to directly optimize pin locations.

Timing-driven placement can be further categorized based on the optimization method into weight-based and gradient-based methods. In weight-based methods, timing optimization is achieved by assigning weights to wirelength objectives according to the criticality of timing attributes, whereas the gradient-based methods compute the gradient of timing objectives such as WNS (worst negative slack) and TNS (total negative slack) to optimize timing.

Recent works proposed advanced methods to achieve better quality and faster runtime compared to previous works. [18] proposed a momentum-based net weighting strategy that dynamically adjusts critical net weights according to historical steps. [8] introduced a differentiable timing-driven placement engine with GPU-accelerated static timing analysis that explicitly computes the pin gradients concerning timing metrics. [17] developed a timing calibration that correlates a simple timer to a reference timer and optimizes timing by a differentiable model for better quality and runtime balance. [20] presented a multilevel framework that optimizes design timing according to expected cell distribution. [22] proposed a virtual-buffer-based delay model that considers the path-sharing effect.

The previous works have limited performance due to the significant runtime overhead caused by frequent timing analysis invocation, insufficient consideration of abundant timing information, and the challenges associated with optimizing complex timing objectives based on Steiner RC tree construction. In this paper, we propose a novel hybrid RC tree and wirelength model designed to optimize efficiency, accompanied by a heterogeneous pin-based weighting scheme that accounts for both graph and path-based timing information. We further introduce a calibration method to enhance the incremental timing quality. Our main contribution can be concluded as:

- We propose a degenerated RC model for consistent timing estimation and a corresponding timing-driven wirelength model that enhances the intra-net topology for timing optimization.
- We present a heterogeneous graph/path-based timing weighting to provide abundant information for timing optimization.
- We propose dynamic weight decay that finely adjusts the pin weights.
- We introduce a timing calibration technique to enhance the estimated timing accuracy for incremental timing optimization.
- We achieve an average improvement of 37% and 15% and up to 43% and 78% improvement on WNS and TNS compared to the state-of-the-art GPU-accelerated differentiable timing-driven placer on ICCAD 2015 benchmarks, while also being 2x faster with better HPWL quality.

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10
https://doi.org/10.1145/3676536.3676803

2 PRELIMINARIES

2.1 Analytical Global Placement

Analytical VLSI placers aim at minimizing the wirelength of a circuit $G = (C, E)$, where C denotes the set of cells and E denotes the set of nets. Let $p = \{(x_1, y_1), ..., (x_{|C|}, y_{|C|})\} \in \mathbb{R}^{|C|\times 2}$ denote the cell positions, and $|C|$ is the number of cells. The objective of the global placement is to minimize the HPWL while ensuring the non-overlapping between cells. Typically, the analytical global placement models relax the non-overlapping constraints into a penalty term:

$$\min_{p} \sum_{e\in E} WL_e(p) + \lambda D(p) \tag{1}$$

where $WL_e(p)$ is the wirelength of net e and $D(p)$ is the density penalty. A widely used wirelength model is the weighted average (WA) [12], which is formulated as,

$$WL_e(p) = WL_e(x) + WL_e(y) \tag{2a}$$

$$WL_e(x) = \frac{\sum_{i\in e} x_i e^{x_i/\gamma}}{\sum_{i\in e} e^{x_i/\gamma}} - \frac{\sum_{i\in e} x_i e^{-x_i/\gamma}}{\sum_{i\in e} e^{-x_i/\gamma}} \tag{2b}$$

and similarly for $WL_e(y)$. Regarding the density penalty $D(p)$, the widely used density model is the electrostatic-based method [3, 24]. A smaller coefficient γ results in a more accurate approximation of HPWL. The parameter λ determines the weight of the cell spreading. The placer initially sets λ to a small value and gradually increases it to remove cell overlaps.

2.2 Static Timing Analysis

Static Timing Analysis (STA) is essential for evaluating the timing performance of a circuit. The graph-based STA is conducted on a timing graph which is modeled as a directed acyclic graph (DAG) as shown in Fig. 1. The signals on the primary inputs forward-propagate through timing arcs on the nets and cells to the primary outputs. The net delay and cell delay are accumulated during the propagation. The net delay is computed using the Elmore delay model [6] and cell delay is queried from the look-up tables (LUTs) defined in nonlinear delay model (NLDM) libraries. The arrival time (AT) is the accumulation of net and cell delays along the timing path. The arrival time at a pin is the *min* and *max* quantity of all of its fan-in arrival time for early and late condition calculation, which are the best and worst case signal arrival time estimations respectively. Apart from net and cell arcs, the constrained arc tests the arrival between the clock and data signals on flip-flops so that the data signal can arrive at the proper time for setup and hold constraint of the clock. The required arrival time (RAT) is the valid arrival time that can satisfy the timing constraints [14]. The slacks are computed at the endpoints and back-propagated to the primary inputs as:

$$\begin{aligned} Slack_{hold} &= AT_{early} - RAT_{late} \\ Slack_{setup} &= RAT_{early} - AT_{late} \end{aligned} \tag{3}$$

where the AT and RAT are the actual arrival time and required arrival time at endpoints under early and late conditions. A positive slack means a valid arrival time, whereas a negative slack means a timing violation. The larger the negative slacks are, the worse a design performs. The timing quality of a design is usually evaluated according to WNS and TNS, which are the worst slack of all endpoints and the sum of all endpoint slacks:

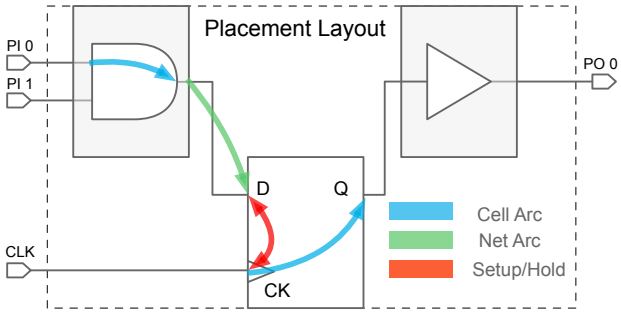

Figure 1: Illustration of timing arcs in static timing analysis.

$$\begin{aligned} WNS &= \min_{ep\in endpoints} Slack(ep) \\ TNS &= \sum_{ep\in endpoints} \min(Slack(ep), 0) \end{aligned} \tag{4}$$

2.3 Timing-driven Placement

Timing-driven placement aims to optimize both the timing metrics and the wirelength of a design. To achieve such targets, net-based weighting approaches and differentiable approaches have been proposed.

The net weighting technique analyzes the placement solution and reports the most timing-critical nets. The corresponding net wirelength objective in the original objective function is adjusted accordingly:

$$\min_{p} \sum_{e\in E} \omega_e \cdot WL_e(p) + \lambda D(p) \tag{5}$$

where the ω_e is the timing weight associated with net e.

The differentiable timing-driven placement [8] computes the timing gradients on each pin explicitly and back-propagates the gradients through the timing graph. The gradients are integrated into the objective function to optimize timing:

$$\min_{p} \sum_{e\in E} WL_e(p) + \lambda D(p) + t_1 WNS(p) + t_2 TNS(p) \tag{6}$$

where t_1 and t_2 are the timing objective weights. Both methods rely on explicit and precise timing analysis, typically performed on a Rectilinear Steiner Minimal Tree (RSMT) constructed by Flute [5] to represent the RC tree. The non-continuous nature of RC Steiner tree construction brings challenges for optimization and introduces a significant runtime bottleneck.

3 PROPOSED FRAMEWORK

The overall framework of our proposed timing-driven placer is illustrated in Fig. 2. We first build the level list for timing propagation and then start the placement iteration. The placement flow can be divided into timing analysis, timing calibration, and placement optimization, distinguished by color. We integrate a degenerated RC model for timing estimation and invoke a timing calibration in the late placement stages when the placement optimization becomes incremental. The calibration constructs a Flute-based Steiner tree and caches the RC parameters for the RC calibration. Reported

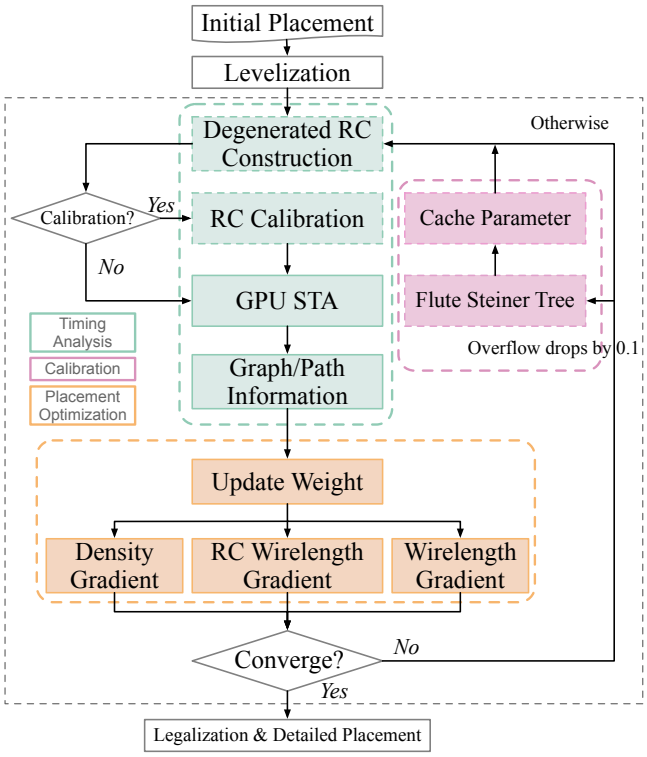

Figure 2: The overall flow of our proposed algorithm.

hybrid timing information from a GPU STA is utilized to update pin-weights for RC wirelength optimization. The proposed methods will be discussed in detail in the following sections.

3.1 Degenerated RC Tree Model

RC tree construction is essential in STA for delay and slew calculation. The capacitance and resistance of each net are extracted according to the tree structure.

To compute the RC parameters in the placement stage, an RSMT is required to be constructed as an early routing estimation, typically performed using Flute. Each Steiner point is regarded as a node in the RC tree. The RC parameters of a wire segment uv can be computed as [13]:

$$Load(u) = cap(u) + \sum_{v_i \in C} Load(v_i) \tag{7a}$$

$$Delay(v) = Delay(u) + res(uv) \cdot Load(v) \tag{7b}$$

$$LDelay(u) = cap(u) \cdot Delay(u) + \sum_{v_i \in C} LDelay(v_i) \tag{7c}$$

$$\beta(v) = \beta(u) + res(uv) \cdot LDelay(v) \tag{7d}$$

$$Impulse(v) = \sqrt{2\beta(v) - Delay^2(v)} \tag{7e}$$

$$Slew(v) = \sqrt{Slew^2(u) + Impulse^2(v)} \tag{7f}$$

where u is the parent node and v is the child node, and C is the set of all child node of u. The variables cap and res are the node capacitance and wire resistance derived from the parasitic parameters of the wire. $LDelay$ and β are intermediate variables for $Slew$ calculation. The term $Impulse$ is the degradation of a gate's input $Slew$ from its driving gate's output.

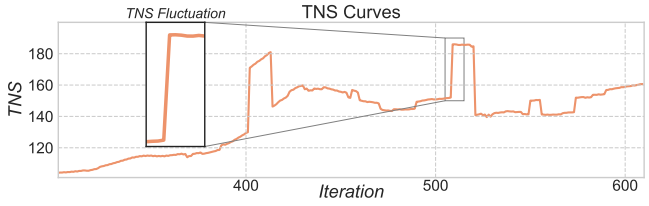

Figure 3: An example of TNS evaluated by Flute constructed RC tree in the placement iterations.

The RC parameters in Equation (7) are computed recursively along the internal Steiner nodes from the root pin to the sink pins, indicating the dependency of RC computation on the RSMT structure. When the placement changes during optimization, the change of pin locations will cause different RSMT structures, and this change of internal RSMT topology is non-continuous. Thus the routing topology between two global placement iterations might deviate, causing divergence in timing optimization based on the explicit RC tree.

Fig. 3 depicts an example of the inconsistency of a Flute-based Steiner tree in placement iterations. The large fluctuation of timing evaluation between iterations will lead to ineffective optimization. A consistent RC tree construction would be beneficial for optimizing the design's timing.

To achieve better timing optimization, we propose a consistent timing analysis with a degenerated RC tree model. Given a net with a set of pins, we assign the driver pin as the root r and the fan-out pins as sinks $v_i \in S$. We construct a rectilinear spanning shortest-path tree (SPT) [2] which directly connects the root pin to each sink pin as illustrated in Fig. 4. Under the spanning SPT, all the sink pins are driven directly by the root pin and share no common path parasitic parameters, thus the RC parameters of all pins are only dependent on the sink-root connection. Without constructing an RSMT, the RC parameters of pins in the degenerated RC tree can be computed explicitly as:

$$Load(r) = cap(r) + \sum_{v_i \in S} Load(v_i) \tag{8a}$$

$$Delay(v_i) = res(rv_i) \cdot Load(v_i) \tag{8b}$$

$$Impulse(v_i) = Delay(v_i) \tag{8c}$$

$$Slew(v_i) = \sqrt{Slew^2(r) + Impulse^2(v_i)} \tag{8d}$$

where r is the root pin and v_i are the sink pins. Since the node capacitance and wire resistance are linear to the segment wirelength, the RC parameters on the nodes are continuous functions of the pin locations. Thus the change in objectives $Load$ and $Delay$ through the placement iterations can be directly derived from $\Delta x, y$ consistently without constructing an RSMT.

The degenerated RC parameters are propagated through the timing graph to update the circuit slacks as an early estimation. We implemented our STA on GPU as in [7], which sorts all the pins according to their task dependencies and executes the forward propagation level-by-level on GPU. We further put the backward propagation on GPU by creating two level lists, with respect to the forward propagation and the backward propagation. The level lists

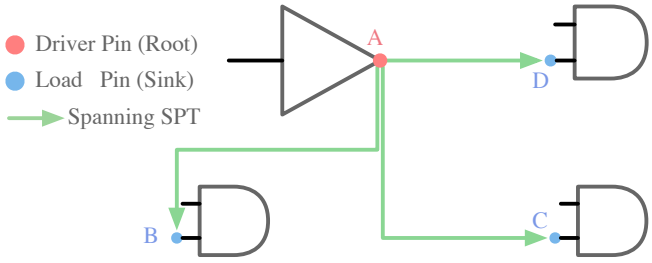

Figure 4: An example of our degenerated RC tree model for a 4-pin net. We regard the driver pin (A) as the root and fan-out pins (B, C, and D) as sinks, and construct a spanning SPT to directly connect the root pin to each sink pin.

ensure that different threads on the GPU will visit non-conflicting timing arcs in parallel. We initialize the level lists in memory so that they can be reused for timing propagation in different placement iterations.

3.2 Timing-driven Wirelength Model

Global placement optimizes wirelength as an objective, where many wirelength models have been adopted. The widely used Weighted Average (WA) wirelength [12] is a smoothed function that can be better optimized by gradient descent methods. The model in [19] further improves wirelength by introducing convexity to the objective function. The Steiner Wirelength (StWL) [26] aims to optimize the routing topology during placement. However, none of these wirelength models aim to improve timing. We introduced the degenerated RC model in Section 3.1 as an early estimation of timing performance in placement. To make the optimization objective consistent with the RC tree structure, we adopt an RCWL that measures the sink-root distance. Without loss of generality, the horizontal part of RCWL is given as follows,

$$RCWL_e(x) = \sum_{v_i \in S} |x_r - x_i| \tag{9}$$

where r is the root pin of net e, $v_i \in S$ is the set of sink pins and x_r and x_i are their corresponding horizontal coordinates. The RCWL decomposes a net into independent segments that connect sink pins directly to the root pin. To integrate the RCWL into a gradient optimizer, we relax the objective into a weighted average sum of all sink-root segments:

$$WL_i(x) = \frac{x_r e^{x_r/\gamma} + x_i e^{x_i/\gamma}}{e^{x_r/\gamma} + e^{x_i/\gamma}} - \frac{x_r e^{-x_r/\gamma} + x_i e^{-x_i/\gamma}}{e^{-x_r/\gamma} + e^{-x_i/\gamma}} \tag{10a}$$

$$RCWL_e(x) = \sum_{v_i \in S} \omega_i \cdot WL_i(x) \tag{10b}$$

where γ is the WA coefficient discussed in Equation (2), and ω_i is the weight applied on sink pin v_i. The smoothed WA RCWL aims to optimize the intra-net connection so that the total delay between the sink pins and the root pin can be minimized. The delay on the critical pins can be reduced by applying a large pin-root weight ω_i. As Equation (10) intuitively centers the root pin to minimize RCWL while considering the criticality by pin-root weight, the resulting TNS and WNS can be simultaneously optimized. The intuition

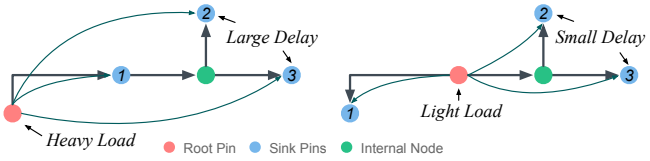

Figure 5: An example of two identical routing topologies of the same net. The right one in which the root pin is centered has better timing with light load and small delay.

behind RCWL is also illustrated in Fig. 5. Two Steiner trees are constructed for the same net, each with different placement layouts yet identical RSMTs. However, the right one in which the root pin is centered has better timing since the sink pins have a smaller capacitive load due to a shorter sink-root distance. We integrate the RCWL with a global weight t into the objective function in Equation (1) so that the timing is optimized as:

$$\min_p \sum_{e \in E} (WL_e(p) + t \cdot RCWL_e(p)) + \lambda D(p) \tag{11}$$

3.3 Hybrid Graph/Path-based Weighting

The work [18] proposed a momentum-based net weighting scheme to reduce the slacks on critical nets. This net-based weighting scheme is rough and cannot differentiate the pins in the same net. We propose a pin-based weighting scheme for a more fine-grained optimization. Section 3.2 introduced an RC wirelength model that improves the timing by shortening the sink-root connection. The weight assigned to each sink-root wire segment underscores its significance in influencing timing. To further facilitate the timing-driven wirelength, we propose a hybrid weighting scheme that considers abundant timing information.

Our weighting scheme consists of graph weights and path weights. The graph weights are derived from the graph-based timing analysis which reports the pin slacks information. The path weights are obtained from path extraction which reports the delay information of all timing arcs in a path. Given the circuit pins p_i and the reported pin slacks sl_i, we define the graph weights as:

$$\omega_{\mathcal{G},i} = \begin{cases} 0 & \text{, if } sl_{worst} \geq 0 \\ \frac{\max(-sl_i, 0)}{|sl_{worst}|} & \text{, otherwise} \end{cases} \tag{12}$$

where the sl_{worst} is the WNS value. A pin with a positive slack value has zero weight, and the pins with larger negative slacks will be assigned larger weights, which helps to reduce the critical pin slack by shortening the RCWL.

Regarding the path weight, we select K most critical endpoints and sort them according to their slacks. Then, we report the worst path of each endpoint and rank them, where the most critical path will have a rank of $k = 1$. A path P_k has a set of pins $\{i | i \in P_k\}$ counting sequentially from the primary input to the endpoint. In path P_k, each pair of pins i and j form a timing arc $i \rightarrow j$. There are two types of timing arcs, the net arc and the gate arc, which have net delay and gate delay respectively as shown in Fig. 6. The net delay is determined as described in Equation (8), depending on the sink-root wirelength. The gate delay is determined by the input pin slew and output pin load and is obtained from the LUTs

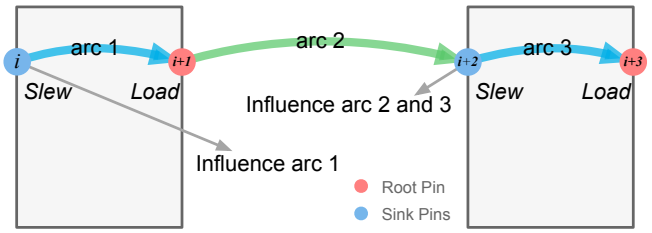

Figure 6: An example of net arc, gate arc and their influencing pins. The sink pin will influence both its parent arc delay and child arc delay.

defined in cell libraries. We define the influencing pin of a timing arc $(i \to j) \in P_k$ as:

$$f_k(i \to j) = \begin{cases} i & \text{, if } i \to j \text{ is gate arc} \\ j & \text{, if } i \to j \text{ is net arc} \end{cases} \tag{13}$$

We define the set of arcs that is influenced by pin i as:

$$\mathcal{S}_{k,i} = \{u \to v | i == f_k(u \to v), \forall (u \to v) \in P_k\} \tag{14}$$

As the example shown in Fig. 6, the sink-root weight applied on pin $i+2$ will influence both arc_2 $((i+1) \to (i+2))$ and arc_3 $((i+2) \to (i+3))$ on their delays and slews, thus $i+2$ is the influencing pin of arc_2 and arc_3, and $\mathcal{S}_{k,i+2} = \{\text{arc\_2}, \text{arc\_3}\}$.

Suppose the arc $u \to v$ on the path P_k has $delay_{k,uv}$, we will accumulate the weighted delay of all the K critical paths to the influencing pin i and define the path weight as:

$$\omega'_{k,i} = \sum_{(u \to v) \in \mathcal{S}_{k,i}} \frac{delay_{k,uv}}{k^2} \tag{15a}$$

$$\omega'_i = \sum_{1 \le k \le K} \omega'_{k,i} \tag{15b}$$

$$\omega_{\mathcal{P},i} = \frac{\omega'_i}{\max_{i \in V}(\omega'_i)} \tag{15c}$$

where k is the rank of a path and V is the set of all pins. The variable $\omega'_{k,i}$ represents the weight contributed by the k-th critical path. The path weight $\omega_{\mathcal{P},i}$ considers the delays of the reported critical paths and casts them on the influencing pins. As the criticality of a path increases and the arc delay grows, the value of $\omega'_{k,i}$ added to the influencing pins will also increase. A pin with a large path weight indicates that it influences timing critically, by either participating in numerous paths or constituting a critical arc on a critical path.

We dynamically compute the K with a threshold of $\frac{3}{4}$WNS. The endpoints with worse slacks than the threshold will be extracted their worst paths. The path analysis is executed on GPU, with each thread handling one path extraction. We record the *argmin* and *argmax* of all fan-in arcs of a pin in the forward propagation of STA so that the worst path extraction will not incur significant runtime overhead.

By incorporating the two weights, we obtain a hybrid pin weight:

$$\tilde{\omega}_i = \alpha \cdot \omega_{\mathcal{G},i} + \omega_{\mathcal{P},i} \tag{16}$$

where $\omega_{\mathcal{G},i}$ is the graph weight and $\omega_{\mathcal{P},i}$ is the path weight, and α is to normalize between the two.

The weight will be dynamically updated in every placement iteration according to the STA information. However, there are cases where the placement layout rapidly changes, causing unstable updates of the pin weights, which may lead to insufficient optimization of the placement objectives. A rapid increase in pin weight means an urgent need to update the weight to avoid worsening of the critical paths, whereas a small change in pin weight means an unnecessity of updating the weight. To achieve the target of maintaining the nonsensitive weights and boosting the updates of critical pin weights, we propose a dynamic decay factor $\boldsymbol{\eta}$, which is a vector of size equal to the number of pins. Suppose we have a hybrid pin weight vector $\tilde{\omega}^{(m)}$ in iteration m, the updated weight in the next iteration can be written as:

$$\begin{aligned} \Delta\tilde{\omega}_i &= \tilde{\omega}_i^{(m+1)} - \tilde{\omega}_i^{(m)} \\ \eta_i &= \eta_0 \cdot \frac{a^{\max(\Delta\tilde{\omega}_i, 0)}}{b} \\ \omega_i^{(m+1)} &= \eta_i \tilde{\omega}_i^{(m+1)} + (1 - \eta_i)\omega_i^{(m)} \end{aligned} \tag{17}$$

where $\Delta\tilde{\omega}_i$ is the change of hybrid pin weights between two iterations, η_0 is the initial decay factor, and a, b are two decay hyperparameters. A small $\Delta\tilde{\omega}$ leads to a smaller decay factor, keeping a stable pin weight. A large $\Delta\tilde{\omega}$ will increase the decay factor so that the pin weight can react to the rapid change faster.

3.4 Timing Calibration

We have introduced the degenerated RC model in STA and the RCWL model as placement objectives, which serve as effective estimations of timing in the early stages of placement. However, when the placement solution converges and the timing analysis becomes incremental, the gap between the degenerated RC model and the accurate RSMT-based RC tree will lead to misguidance in the optimization objectives. To tackle this issue, we propose a calibration method aimed at enhancing the accuracy of the degenerated timing model.

Our calibration is based on the observation that when the adjustment of the placement layout is small, the structure of the Flute-constructed Steiner tree will remain similar, as shown in Fig. 7. The main parameters we are calibrating are the root pin *Load* and the sink pin *Delay* as described in Equation (8), which are most influential to the delay propagation. Note that in the following discussion we mainly focus on the wire capacitance, and we regard the cell static capacitance as 0 without loss of generality.

For a wire segment $i \to j$, the wire capacitive load will be evenly distributed to the two nodes i and j, which is proportional to the segment wirelength wl_{ij}. The wire resistance and node capacitance are:

$$\begin{aligned} cap_i = cap_j &= \frac{1}{2} wl_{ij} \cdot cap_{unit} \\ res_{ij} &= wl_{ij} \cdot res_{unit} \end{aligned} \tag{18}$$

where cap_{unit} and res_{unit} are the unit length wire capacitance and resistance. We decompose a net e into a set of wire segments from its tree structure $\{seg_{ij} | seg_{ij} \in e\}$, and flatten the recursive computation of the root pin *load* as:

$$Load_e(r) \simeq \sum_{seg_{ij} \in e} wl_{ij} \cdot cap_{unit} = WL_e \cdot cap_{unit} \tag{19}$$

which is proportional to the net wirelength WL_e. We define the calibration ratio between StWL and RCWL, denoted as $r_{cali} = \frac{StWL_e}{RCWL_e}$, as the ratio of the accurate $Load^*$ (derived from the RSMT-based RC tree) and our estimated $Load$ (derived from the degenerated RC model). The calibrated $Load_{cali}$ is:

$$Load_{cali} = r_{cali} \cdot Load \tag{20}$$

In the degenerated RC model, the sink pins are connected directly to the root, whereas in the RSMT-based RC, a pin might share a common path load with some neighboring pins. Consider a wire segment $i \to j$ between the root r and the sink v as shown in Fig. 7(b), we call the direct path $r \to v$ as trunk, and the branching nodes as neighbors. The value of $delay_{ij}$ can be computed as:

$$\begin{aligned} load_j &= (cap_1 + cap_2) + (cap_v + cap_j + cap_k) \\ &= cap_{j,N} + cap_{j,T} \\ delay_{ij} &= load_j \cdot res_{ij} = (cap_{j,N} + cap_{j,T}) \cdot res_{ij} \\ &= delay_{ij,N} + delay_{ij,T} \end{aligned} \tag{21}$$

where $cap_{j,N}$ and $cap_{j,T}$ are the neighbor capacitance and trunk capacitance, and $delay_{ij,N}$ and $delay_{ij,T}$ are the neighbor-contributed delay and trunk-contributed delay respectively. By decomposing the segment delay into trunk delay and neighbor delay, we can write the RSMT-based RC $Delay$ on pin v as:

$$Delay^*(v) \simeq \sum_{seg_{ij} \in (r \to v)} delay_{ij} = Delay_T(v) + Delay_N(v) \tag{22}$$

where $Delay_T$ and $Delay_N$ are the sum of all segments' $delay_{\cdot,N}$ and $delay_{\cdot,T}$ respectively. Note that the trunk delay $Delay_T(v)$ is the segmented wire delay of $r \to v$ without considering the neighboring effects, whereas the degenerated RC $Delay(v)$ is a non-segmented wire delay of $r \to v$, also without considering the neighboring effects. Let $\epsilon = Delay(v) - Delay_T(v)$ be a constant delay error between the segmented and non-segmented wire model. We compute the compensation as:

$$\begin{aligned} comp(v) &= Delay^*(v) - Delay(v) = Delay_N(v) - \epsilon \\ Delay_{cali}(v) &= comp(v) + Delay(v) \end{aligned} \tag{23}$$

where $comp(v)$ is the error between the estimated RC delay and the accurate RC delay for delay calibration. The $Impulse$ value has less variation and influence on delay propagation, so we record the RSMT-based $Impulse^*$ as the reference $Impulse$ in the subsequent iterations.

We start performing the Flute RC tree calibration once the placement overflow drops below 0.5, and we will update the cached RC parameters again every time the overflow decreases by 0.1, where the overflow is a value in $[0, 1]$ that measures the placement density.

4 EXPERIMENTS

We develop our algorithms using C++ and CUDA, and the experiments are conducted on a Linux machine with a 2.90GHz Intel Xeon CPU and a single Nvidia RTX 3090 GPU. The core algorithms of the electrostatic-based placer, the detailed placement and the static timing analysis are implemented on GPU based on the work [23] and the work [7].

We evaluate our performance on the ICCAD 2015 contest benchmarks [15]. The statistics of benchmarks are shown in Table 1. We start the timing optimization at iteration 100, and the α, a, b, t, η_0 are 0.5, 5, 2, 0.25 and 0.4 respectively.

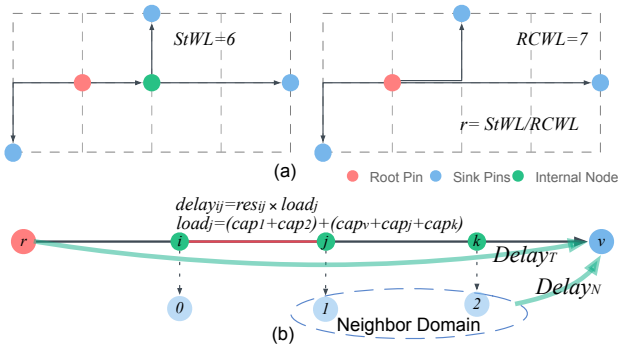

Figure 7: An illustration of the calibration of RC parameters.

Table 1: Benchmark statistics

Design	#Cells	#Nets	#Pins
superblue1	1209716	1215710	3767494
superblue3	1213253	1224979	3905321
superblue4	795645	802513	2497940
superblue5	1086888	1100825	3246878
superblue7	1931639	1933945	6372094
superblue10	1876103	1898119	5560506
superblue16	981559	999902	3013268
superblue18	768068	771542	2559143

4.1 Comparison with the State-of-the-art Timing-Driven Placers

We mainly compare with the state-of-the-art timing-driven placers, including the net-weighting-based timing-driven placer [18], differentiable timing-driven placer [8] and cell distribution-based timing-driven placer [20]. For reference, we also compare with two open-source placers [21, 23] that do not explicitly incorporate the timing optimization.

WNS and TNS. The results in Table 2 and Table 3 show that our work outperforms the state-of-the-art timing-driven placers significantly with better WNS, TNS, and HPWL, and is also much faster. Specifically, we achieve overall 14.9% improvement on WNS and 37% improvement on TNS compared with the differentiable timing-driven placer [8] and is around 2x faster when both of the timing-driven placers are GPU-accelerated. The improvement on TNS is significant, which can be up to 78% on the design superblue1 and superblue16, and the WNS can be reduced by up to 43% on design superblue16. The results show that we have obtained a much better solution closer to timing closure.

Our performance has shown better results in almost all the designs for both WNS and TNS. This is because our hybrid timing information considers the global and local path criticality simultaneously. Our RCWL is well-incorporated into the placement objectives and effectively improves the design timing by explicitly adjusting

Table 2: WNS ($\times 10^3$)ps and TNS ($\times 10^5$)ps results on the ICCAD 2015 contest benchmarks. The best results are highlighted in bold brown, and the second-best results are highlighted in bold.

Design	Xplace [23]		DREAMPlace [21]		DREAMPlace 4.0 [18]		Differentiable TDP [8]		ISPD24 [20]		Ours	
	WNS	TNS	WNS	TNS	WNS	TNS	WNS†	TNS†	WNS†	TNS†	WNS	TNS
superblue1	-27.884	-252.608	-26.468	-308.946	-14.953	-82.983	-10.770	-74.854	**-9.260**	**-42.100**	**-7.604**	**-16.479**
superblue3	-31.135	-67.671	-37.255	-66.901	-14.857	-52.039	-12.374	-39.430	**-12.190**	**-26.590**	**-11.127**	**-16.770**
superblue4	-20.378	-168.254	-21.707	-177.845	-12.339	-141.033	**-8.492**	**-82.924**	-8.860	-123.280	**-7.061**	**-70.462**
superblue5	-48.582	-150.952	-48.126	-195.090	-30.498	-96.372	**-25.212**	-108.076	-31.640	**-70.350**	**-24.375**	**-65.760**
superblue7	-19.435	-128.283	-20.477	-163.302	**-15.216**	-61.981	**-15.216**	**-46.426**	-17.240	-95.890	**-15.216**	**-27.080**
superblue10	-30.294	-802.622	-29.797	-737.663	-23.322	-658.546	**-21.974**	**-558.054**	-25.860	-691.100	**-20.070**	**-509.439**
superblue16	-16.175	-322.432	-14.093	-235.292	-13.646	-70.601	**-10.854**	-87.026	-12.210	**-55.990**	**-6.198**	**-19.596**
superblue18	-21.372	-84.740	-20.414	-90.677	-11.637	-49.355	-7.987	-19.314	**-5.250**	**-19.230**	**-6.575**	**-16.044**
Sum	-215.255	-1977.562	-218.337	-1975.717	-136.468	-1212.908	**-112.879**	**-1016.104**	-122.510	-1124.530	**-98.226**	**-741.631**
Ratio	2.191	2.667	2.223	2.664	1.389	1.635	**1.149**	**1.370**	1.247	1.516	**1.000**	**1.000**

† Reported WNS/TNS in [8] and [20] are shown.

Table 3: HPWL($\times 10^6$) and runtime (seconds) quality on the ICCAD 2015 contest benchmarks. The best results are highlighted in bold brown, and the second-best results are highlighted in bold.

Design	Xplace [23]		DREAMPlace [21]		DREAMPlace 4.0 [18]		Differentiable TDP [8]		Ours	
	HPWL	RT	HPWL	RT	HPWL	RT	HPWL†	RT‡	HPWL	RT
superblue1	**400.3**	**24.38**	410.2	**75.67**	481.3	536.35	423.8	246.37	**408.2**	133.64
superblue3	**453.1**	**23.10**	**457.1**	**72.59**	483.1	733.67	478.4	244.85	457.8	146.67
superblue4	**300.2**	**14.05**	314.1	**56.38**	334.2	248.27	312.2	143.58	**303.6**	89.72
superblue5	**466.2**	**21.88**	**468.3**	**111.60**	535.1	628.66	488.7	238.06	473.9	128.15
superblue7	**566.3**	**36.59**	599.0	**124.10**	604.0	814.20	602.1	413.99	**576.7**	186.29
superblue10	**893.8**	**38.66**	**904.8**	**198.28**	1086.7	1019.97	934.4	427.20	930.6	206.07
superblue16	**418.3**	**14.20**	**424.8**	**16.69**	461.9	381.79	485.1	199.85	424.9	93.79
superblue18	**226.9**	**11.49**	233.3	**27.83**	247.5	287.03	243.6	144.15	**231.8**	88.73
Sum	**3724.9**	**184.35**	3811.6	**683.14**	4233.8	4649.95	3968.3	2058.05	**3807.5**	1073.07
Ratio	**0.978**	**0.172**	1.001	**0.637**	1.112	4.333	1.042	1.918	**1.000**	1.000

† Reported HPWLs in [8] are shown. § The ISPD24 [20] did not report the HPWL and runtime.
‡ The runtime of [8] is computed by: reported runtime in [8] $\times \frac{\sum \text{Our DREAMPlace runtime}}{\sum \text{reported DREAMPlace runtime in [8]}}$ for a fair comparison.

the sink-root connection with the dynamic weight derived from the hybrid timing information.

Runtime and HPWL. We report the runtime and HPWL in Table 3. The results show that our placer only has small HPWL degradation compared with the wirelength-driven placers [21, 23]. It is also notable that our HPWL is shorter than the SOTA differentiable timing-driven-placer [8] by 4.2%, which means that our method has negligible wirelength overhead when optimizing the timing constraint.

Our placer also demonstrates efficiency with 2x faster runtime compared to the differentiable-timing-driven-placer [8] which is also GPU-accelerated. This is mainly achieved by avoiding frequent invocation of Steiner tree construction and by putting the forward propagation, backward propagation and path generation of STA on GPU. We illustrate the WNS, TNS, HPWL and overflow at different placement iterations in Fig. 8, in which the blue curves are non-timing-driven, and ours are in pink. We start the timing optimization at the 100^{th} iteration and we can see that the timing metrics are improved significantly. The HPWL and overflow curves are almost identical, showing good convergence of our placement objectives.

4.2 Effectiveness of Timing Calibration

To assess the effectiveness of our timing calibration, we calculate the normalized Mean Absolute Error (MAE) between the non-calibrated slacks and the calibrated slacks of our degenerated RC, in comparison with the ground-truth slacks generated from the Flute-constructed Steiner tree as shown in Fig. 9. At the beginning of the timing iterations when the timing is evaluated by the degenerated RC model, the error between the estimated slack and accurate slack is substantial, approximately 6%. The error drops to 0 when calibration is invoked and gradually rises in subsequent iterations. As we update the calibration parameters again whenever the overflow drops by 0.1, the placement solution remains similar so the calibrated slacks only deviate slightly (under 1%) from the ground-truth values.

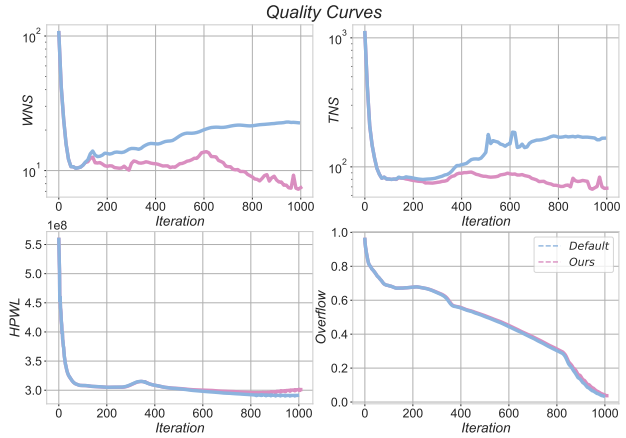

Figure 8: The quality curves of superblue4 in placement iterations.

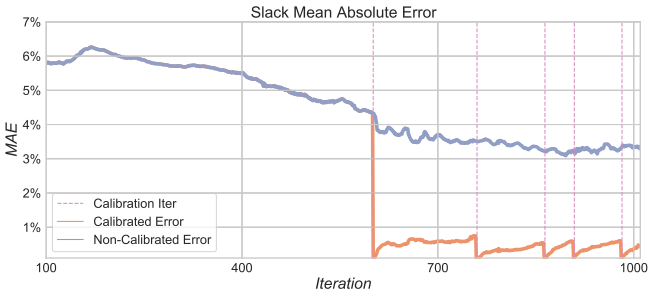

Figure 9: The MAE of endpoint slacks during placement iterations, the error is normalized by mean absolute ground-truth slack value.

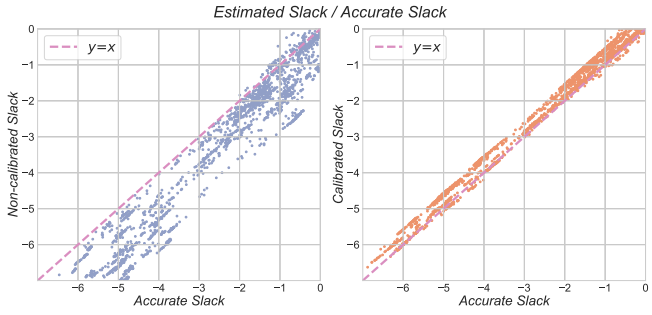

Figure 10: The ground-truth slack, non-calibrated slack, and calibrated slack at the 1000^{th} iteration.

Fig. 10 shows the scattered points of negative endpoint slacks at the 1000^{th} iteration. The calibrated slacks exhibit a strong correlation with the ground-truth slacks as depicted in the right figure, whereas the non-calibrated slacks deviate significantly from the accurate values. The correlated slacks offer precise guidance for timing optimization in the later stages when the change in placement becomes incremental.

Table 4: Ablation Studies of the proposed techniques on the ICCAD 2015 benchmarks[15]. GW, PW, DW, TC refer to graph weight, path weight, dynamic weight decay, and timing calibration as described in Section 3.

Method	GW	PW	DW	TC	WNS↑	TNS↑
Ratio	✓	✓	✓	✓	0.00%	0.00%
	✓	✓	✓	-	2.17%	6.30%
	✓	✓	-	-	5.34%	6.41%
	✓	-	-	-	8.94%	7.72%

4.3 Ablation Studies

To further illustrate the effectiveness of our proposed methods, we conduct the ablation study as shown in Table 4. We selectively enable the techniques and evaluate the overall timing quality loss compared to the baseline. Results show that our proposed techniques can effectively reduce the WNS and TNS.

5 CONCLUSION

In this paper, we propose a new framework for timing-driven placement with hybrid weighting, RC wirelength modeling, and efficient timing calibration that significantly improves the circuit timing compared with state-of-the-art works. We believe our work can be extensible for other timing models. Our future work would be on further improving the quality and runtime of timing-driven placement.

REFERENCES

[1] H. Chang, E. Shragowitz, J. Liu, H. Youssef, B. Lu, and S. Sutanthavibul. 2002. Net criticality revisited: an effective method to improve timing in physical design. In *Proceedings of the 2002 International Symposium on Physical Design* (San Diego, CA, USA) *(ISPD '02)*. New York, NY, USA, 155–160.

[2] Gengjie Chen, Peishan Tu, and Evangeline F. Y. Young. 2017. SALT: Provably good routing topology by a novel steiner shallow-light tree algorithm. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 569–576.

[3] Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, and Lutong Wang. 2018. Replace: Advancing solution quality and routability validation in global placement. *IEEE TCAD* 38, 9 (2018), 1717–1730.

[4] A. Chowdhary, K. Rajagopal, S. Venkatesan, Tung Cao, V. Tiourin, Y. Parasuram, and B. Halpin. 2005. How accurately can we model timing in a placement engine?. In *Proceedings. 42nd Design Automation Conference, 2005.* 801–806.

[5] Chris Chu and Yiu-Chung Wong. 2007. FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design. *IEEE TCAD* 27, 1 (2007), 70–83.

[6] William C Elmore. 1948. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of applied physics* 19, 1 (1948), 55–63.

[7] Zizheng Guo, Tsung-Wei Huang, and Yibo Lin. 2020. GPU-Accelerated Static Timing Analysis. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[8] Zizheng Guo and Yibo Lin. 2022. Differentiable-timing-driven global placement. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) *(DAC '22)*. New York, NY, USA, 1315–1320.

[9] Chrystian Guth, Vinicius Livramento, Renan Netto, Renan Fonseca, José Luís Güntzel, and Luiz Santos. 2015. Timing-Driven Placement Based on Dynamic Net-Weighting for Efficient Slack Histogram Compression. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design* (Monterey, California, USA) *(ISPD '15)*. New York, NY, USA, 141–148.

[10] Bill Halpin, C. Y. Roger Chen, and Naresh Sehgal. 2000. A sensitivity based placer for standard cells. In *Proceedings of the 10th Great Lakes Symposium on VLSI* (Chicago, Illinois, USA) *(GLSVLSI '00)*. New York, NY, USA, 193–196.

[11] T. Hamada, Chung-Kuan Cheng, and P.M. Chau. 1993. Prime: A Timing-Driven Placement Tool Using A Piecewise Linear Resistive Network Approach. In *30th ACM/IEEE Design Automation Conference*. 531–536.

[12] Meng-Kai Hsu, Yao-Wen Chang, and Valeriy Balabanov. 2011. TSV-Aware Analytical Placement for 3D IC Designs. In *Proceedings of the 48th Design Automation Conference* (San Diego, California). New York, NY, USA, 664–669.

[13] Jin Hu, Greg Schaeffer, and Vibhor Garg. 2015. TAU 2015 contest on incremental timing analysis. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 882–889.

[14] Tsung-Wei Huang and Martin D. F. Wong. 2015. OpenTimer: A high-performance timing analysis tool. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 895–902.

[15] Myung-Chul Kim, Jin Hu, Jiajia Li, and Natarajan Viswanathan. 2015. ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 921–926.

[16] T. Kong. 2002. A novel net weighting algorithm for timing-driven placement. In *IEEE/ACM International Conference on Computer Aided Design, 2002. ICCAD 2002.* 172–176.

[17] Wuxi Li, Yuji Kukimoto, Gregory Servel, Ismail Bustany, and Mehrdad E. Dehkordi. 2024. Calibration-Based Differentiable Timing Optimization in Non-linear Global Placement. In *Proceedings of the 2024 International Symposium on Physical Design (ISPD '24)*. New York, NY, USA, 31–39.

[18] Peiyu Liao, Dawei Guo, Zizheng Guo, Siting Liu, Yibo Lin, and Bei Yu. 2023. DREAMPlace 4.0: Timing-Driven Placement With Momentum-Based Net Weighting and Lagrangian-Based Refinement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 10 (2023), 3374–3387.

[19] Peiyu Liao, Hongduo Liu, Yibo Lin, Bei Yu, and Martin Wong. 2023. On a Moreau Envelope Wirelength Model for Analytical Global Placement. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[20] Jai-Ming Lin, You-Yu Chang, and Wei-Lun Huang. 2024. Timing-Driven Analytical Placement According to Expected Cell Distribution Range. In *Proceedings of the 2024 International Symposium on Physical Design (ISPD '24)*. New York, NY, USA, 177–184.

[21] Yibo Lin, Zixuan Jiang, Jiaqi Gu, Wuxi Li, Shounak Dhar, Haoxing Ren, Brucek Khailany, and David Z Pan. 2020. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. *IEEE TCAD* 40, 4 (2020), 748–761.

[22] Zhifeng Lin, Min Wei, Yilu Chen, Peng Zou, Jianli Chen, and Yao-Wen Chang. 2024. Electrostatics-Based Analytical Global Placement for Timing Optimization. In *2024 Design, Automation and Test in Europe Conference and Exhibition (DATE)*.

[23] Lixin Liu, Bangqi Fu, Shiju Lin, Jinwei Liu, Evangeline F.Y. Young, and Martin D.F. Wong. 2023. Xplace: An Extremely Fast and Extensible Placement Framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023), 1–1.

[24] Jingwei Lu, Hao Zhuang, Pengwen Chen, Hongliang Chang, Chin-Chih Chang, Yiu-Chung Wong, Lu Sha, Dennis Huang, Yufeng Luo, Chin-Chi Teng, et al. 2015. ePlace-MS: Electrostatics-based placement for mixed-size circuits. *IEEE TCAD* 34, 5 (2015), 685–698.

[25] Haoxing Ren, D.Z. Pan, and D.S. Kung. 2005. Sensitivity guided net weighting for placement-driven synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24, 5 (2005), 711–721.

[26] Min Wei, Xingyu Tong, Zhijie Cai, Peng Zou, Zhifeng Lin, and Jianli Chen. 2024. An Analytical Placement Algorithm with Routing topology Optimization. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 294–299.

[27] Carl Sechen William Swartz. 1995. Timing Driven Placement for Large Standard Cell Circuits. In *32nd Design Automation Conference*. 211–215.