Gradient-Guided RC Weighting for Timing-Driven Global Routing

Liang Xiao
lxiao23@cse.cuhk.edu.hk
The Chinese University of Hong Kong
Hong Kong SAR

Qinkai Duan
qkduan25@cse.cuhk.edu.hk
The Chinese University of Hong Kong
Hong Kong SAR

Leilei Jin
leileijin6@gmail.com
The Chinese University of Hong Kong
Hong Kong SAR

Jinwei Liu
jinweiliu@comp.hkbu.edu.hk
Hong Kong Baptist University
Hong Kong SAR

Tsung-Yi Ho
tyho@cse.cuhk.edu.hk
The Chinese University of Hong Kong
Hong Kong SAR

Evangeline F.Y. Young
fyyoung@cse.cuhk.edu.hk
The Chinese University of Hong Kong
Hong Kong SAR

Martin D.F. Wong
mdfwong@hkbu.edu.hk
Hong Kong Baptist University
Hong Kong SAR

Abstract

As a critical step in electronic design automation (EDA), global routing provides a guide to subsequent steps and provides valuable feedback to previous steps, including congestion, timing, and power estimation. However, given the complexity of timing and power calculation, it is difficult to estimate the impact on timing and power during the routing process. To address this issue, we propose a gradient-guided framework that computes the "capacity sensitivity" and "resistance sensitivity" of each segment to estimate their influence on the timing objectives. Integrating these two values as weights to constrain the changes in capacitance and resistance of the wire segments, we develop a timing-driven global router with superior performance. Power is also considered by optimizing the cells' switching power. Tested on ISPD25 Contest benchmarks, we can achieve 14.3% and 18.5% improvements in worst negative slack and total negative slack, respectively, with comparable congestion. With power optimization, we can further improve switching power by 10.6%.

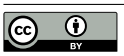
CCS Concepts

• **Hardware → Wire routing**.

Keywords

Global routing, Timing, Power

ACM Reference Format:
Liang Xiao, Qinkai Duan, Leilei Jin, Jinwei Liu, Tsung-Yi Ho, Evangeline F.Y. Young, and Martin D.F. Wong . 2026. Gradient-Guided RC Weighting for Timing-Driven Global Routing . In *Proceedings of the 2026 International Symposium on Physical Design (ISPD '26), March 15–18, 2026, Bonn, Germany.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3764386.3779588



This work is licensed under a Creative Commons Attribution 4.0 International License.
ISPD '26, Bonn, Germany
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2314-8/2026/03
https://doi.org/10.1145/3764386.3779588

1 Introduction

Modern integrated circuit design faces escalating challenges in timing closure and power efficiency with the continuous scaling of semiconductor technology [24]. Interconnect delays now play an increasingly critical role in determining timing, with switching-induced power consumption growing as a significant contributor to dynamic power dissipation [11]. These technological imperatives make performance-driven algorithmic approaches indispensable in modern IC design.

Global routing serves as a pivotal bridge between placement and detailed routing by connecting nets in a compressed 3D routing graph while optimizing various design objectives. Unlike detailed routing, which performs time-consuming processing of physical design rules on actual metal layers, global routing employs a coarse-grained abstraction that significantly accelerates the routing process. This efficiency makes global routing an invaluable tool for early-stage estimation and rapid design space exploration. For example, global routing may be invoked multiple times during the synthesis and placement phases to provide congestion and timing feedback for iterative refinement [20]. Therefore, huge efforts have been made to improve the quality and the runtime [5, 14–16, 18, 21, 27, 28]. What's more, global routing determines wire paths and layer assignments that serve as guides for subsequent detailed routing, affecting signal propagation through its decisions on both the 2D topology and the resistance-capacitance (RC) parasitics of interconnects, which directly influence circuit timing and power consumption. These factors collectively emphasize the critical need for performance-driven global routing algorithms that holistically address timing, power, and congestion constraints.

There are two fundamental challenges for enhancing the timing performance in global routing. First, the excessive runtime of timing calculation tools [1, 3, 10] remains a critical bottleneck— even with substantial advancement in acceleration techniques [7, 8, 13]. This computational overhead severely limits the feasibility of iterative strategies that run timing tools to obtain timing metrics in each iteration [22, 26]. Second, the inherent complexity of timing analysis

makes it difficult for routers to estimate the impact of individual net optimizations on entire timing paths. While these challenges pose significant barriers to timing-aware global routing, several studies have attempted to integrate timing considerations into routing through heuristic approaches. For example, Ao *et al.* propose a layer assignment algorithm [4] to balance between via count and net delay. TILA-S [17] proposes a post-optimization step to avoid slew violations. [9, 25] introduced a min-max resource sharing model that natively incorporates timing constraints into global routing. [9, 25] model timing constraints natively in global routing by using resource sharing weights to guide the routing process.

In this paper, we propose our timing-driven global routing algorithm with contributions listed below.

- We propose a gradient-based method to quantify the influence of the net arc's delay on the timing objective, which gives our router a global view to identify critical and subcritical net arcs.
- Within a net, we use a gradient-based method to analyze the influence of segments' capacitance and resistance on the net arcs' delay.
- Based on the above techniques, we implemented a timing-driven global router with consideration of power.
- Tested on the benchmark, we outperform the ISPD 2025 contest winner by 14.3% and 18.5% improvement on worst negative slack and total negative slack, respectively, with comparable congestion.

2 Preliminaries

2.1 DAG-based routing

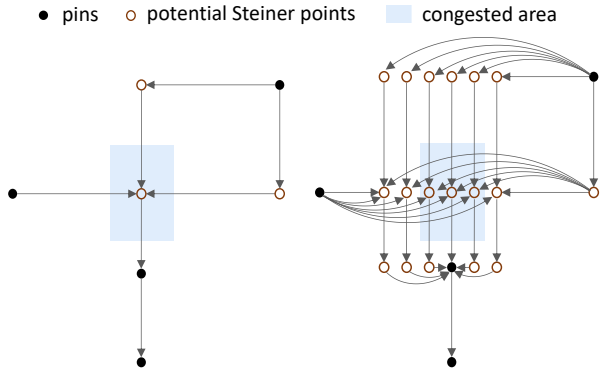
Our work is based on the DAG-based routing introduced in [19]. This method first generates a 2D routing topology using a rectilinear Steiner minimum tree to connect the pins of a net and then explores minimum-cost routing paths according to the 2D topology. Figure 1 gives an example of DAG-based routing. Figure 1a is a basic routing DAG of a 4-pin net based on its rectilinear Steiner minimum tree. To avoid congestion, five alternative paths are added around the congested area. During the routing process, dynamic programming is applied to process nodes from the sink pins to the root pin. At each node, all incoming alternative paths will be considered, and the one with the minimum cost will be picked.

Our proposed algorithm consists of two phases. In the first phase, the nets are routed on the basic routing DAG to provide an initial routing solution to all nets rapidly. In the second phase, the nets with congestion will be ripped up and rerouted on the augmented routing graphs.

2.2 Timing analysis

Timing analysis is a critical step in IC design, which examines signal paths between specific start points and end points to confirm timing requirements. Start points typically include primary inputs (PIs) and the output ports of sequential elements like flip-flops (FFs), while end points include primary outputs (POs) and the input ports of sequential elements.

To quantify timing requirement violations on these paths, two key metrics are commonly used: worst negative slacks (WNS) and



(a) Basic routing DAG. **(b) Augmented routing DAG.**

Figure 1: Global routing using routing DAG.

total negative slacks (TNS):

$$Slack(e) = \min_{l \in Path(e)} (Require(l) - Arrival(l)) \tag{1}$$

$$WNS = \min_{e \in T} \min(Slack(e), 0) \tag{2}$$

$$TNS = \sum_{e \in T} \min(Slack(e), 0) \tag{3}$$

where $Reqiure(l)$ and $Arrival(l)$ are respectively the require time and arrival time of path l. $Slack(e)$ is the slack of a given endpoint e, which is the worst slack of all the signal paths ($Path(e)$) reaching it, T is the set of end points, WNS indicates the worst timing violation, and TNS reflects total violations.

The ISPD 2025 Contest [12] utilizes OpenROAD [2] to give contestants pre-routing estimated pin slacks, which give our algorithm important information to identify end points that have timing violations.

For a net with the driver pin p_0 and any sink p_{out}, we call the directed driver-to-sink relation a net arc and denote its delay by $Delay(Arc_{p_0 \to p_{out}})$. We estimate the delay using the Elmore delay model [6]. Since global routing operates on a coarsened routing graph and must remain efficient, the Elmore model offers a good trade-off between accuracy and computational cost during the global routing phase [4, 22]. The Elmore delay of a wire segment s is:

$$Delay(s) = R(s) \cdot (C(s)/2 + C_{\text{down}}(s)) \tag{4}$$

where $R(s)$ and $C(s)$ are the resistance and capacitance of the segment respectively. $C_{\text{down}}(s)$ is the total capacitance of all the downstream segments of s.

2.3 Evaluation metrics

The ISPD 2025 Contest evaluates both congestion and performance metrics, with the total score calculated as follows:

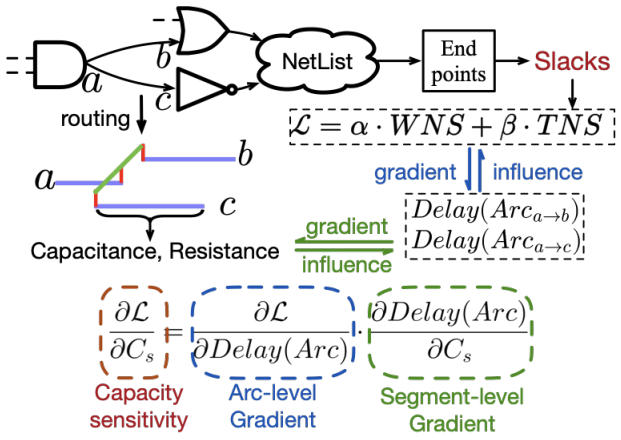


Figure 2: Example of the partial gradient of timing objective with respect to the capacitance of segment s.

$$\begin{aligned} Score = {} & w_{WNS} \cdot (WNS - WNS_{ref}) \\ & + w_{TNS} \cdot \frac{TNS - TNS_{ref}}{N_{endpoint}} \\ & + w_{Pow} \cdot (TotalPower - TotalPower_{ref}) \\ & + w_{Ovf} \cdot OverflowScore \end{aligned} \tag{5}$$

where w_{WNS}, w_{TNS}, w_{Pow}, and w_{Ovf} are the weights for respective metrics. WNS_{ref}, TNS_{ref}, and $TotalPower_{ref}$ are the median values of respective metrics across all contest submissions. The overflow cost of a G-Cell edge with capacity c and routing demand d at the l-th layer is computed as:

$$\begin{aligned} OverflowCost(c, d, l) &= OFWeight[l] \cdot e^{s(d-c)} \\ s &= \begin{cases} 0.5, & \text{if } c > 0, \\ 1.5, & \text{if } c \le 0. \end{cases} \end{aligned} \tag{6}$$

where s controls the penalty for capacity violations, with a higher value (1.5) assigned to edges with zero capacity to discourage routing through these regions. The $OFWeight$ represents the predefined overflow weight for different layers. The total $OverflowScore$ is the sum of all $OverflowCost$ values across all the edges in the design.

3 Performance-driven DAG-based routing

In performance-driven routing, we trade off between performance metrics like WNS, TNS and power, and the routing congestion, it is appropriate to allocate resources preferentially to the most "timing-critical" nets to optimize the performance metrics even with a small congestion overhead.

To efficiently account for this, we introduce a gradient-based framework to quantify the contribution of a segment's capacitance and resistance to the timing objective, and use this information to guide the dynamic programming process of the DAG-based router.

Figure 2 is an example. The wire segment's capacitance and resistance influence the Delay of the net arc $Arc_{a\to b}$ and $Arc_{a\to c}$. Then the net arcs' delay influences the downstream end points' slack and the timing objective $\mathcal{L}$. To quantify this influence, we first estimate the contribution of these two net arcs' delay to the timing objective $\mathcal{L}$ by calculating the gradient of the timing objective with respect to the delay of these two net arcs, using the method introduced in Section 3.1. Then, we estimate the contribution of this segment's capacitance and resistance to the two net arcs' delay by calculating the two net arcs' delay with respect to the capacitance and resistance using the method introduced in Section 3.2. After these steps, we can obtain the capacity sensitivity $\partial\mathcal{L}/\partial C_s$ and the resistance sensitivity $\partial\mathcal{L}/\partial R_s$, which can be used as weights giving penalties to segment capacitance and resistance, thereby optimizing the timing objective when routing.

3.1 Arc-level Gradient

To estimate the influence of a net arc's delay on our timing object $\mathcal{L}$, we have two steps.

- **Estimate the influence of endpoints' arrival time**. Since the negative slack is directly related to the arrival time of end points, we first estimate their influence on $\mathcal{L}$. Take Figure 3 for example. The slack of the input pin $FF2$ is worse than that of $FF1$, so we assign a higher weight to improve the arrival time of the input pin of $FF2$ and a smaller weight for $FF1$.
- **Backpropagate**. We trace this influence from end points to start points, weighting net arcs by their contribution to critical paths. Take Figure 3 for example. To optimize the arrival time of $FF2$, the red path has the most contribution to the arrival time of $FF2$, so we give higher weights to the net arcs on the path. However, if we only optimize the path with the worst slack (colored in red), the signal from cell B will gradually dominate the arrival time of cell C, thereby influencing the arrival time of $FF2$. Therefore, a more scientific way is that we pay more attention to the path from cell A, but also put some effort into optimizing the sub-critical path from cell B. The Backpropagation is performed **at cells and nets**.

In what follows, we define $Arr(p)$ as the arrival time of a pin, which can be computed as the worst-case arrival at pin p among all paths reaching p.

Estimate the influence of endpoints' arrival time. We next respectively estimate how endpoint arrivals influence TNS and WNS, then combine the results. To calculate the influence of the end points' arrival time on TNS, we first recall that the total negative slack is the sum of the negative slack of all end points (Equation (3)). The gradient of TNS with respect to the arrival time of an end point e is simple:

$$\frac{\partial TNS}{\partial Arr(e)} = \begin{cases} 1, & \text{if } Arr(e) > Req(e), \\ 0, & \text{else.} \end{cases} \tag{7}$$

This formula means that every negative slack endpoint has the same contribution to the total negative slack. If any of the negative slack endpoints improve its slack by δt, the total negative slack will also improve by δt.

To calculate the influence of the end points' arrival time on WNS, we also recall the definition as shown in Equation (2). However, in

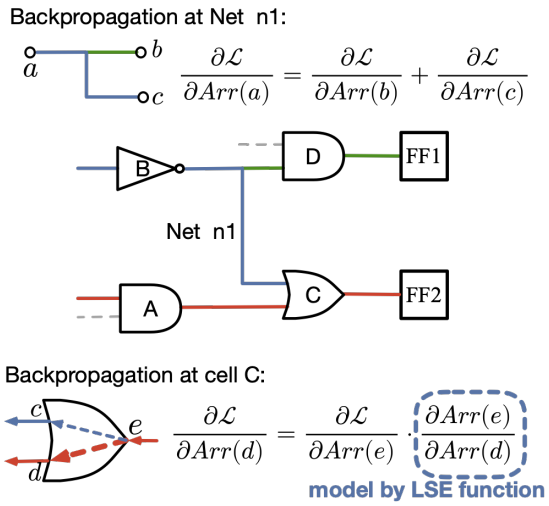


Figure 3: The process of backpropagation. The red, blue, and green paths are negative slack paths with the first, second, and third worst slack values, respectively.

the original definition, only the worst slack endpoint contributes to the WNS, which may overlook subcritical endpoints and is not smooth. Inspired by [23], we use the Log-Sum-Exponential (LSE) operator to approximate the min function among negative slack endpoints as follows. The gradient of this function is then used as the influence of the end points' arrival time on WNS.

$$\begin{aligned} WNS &= \min_{e \in END}(Slack(e)) \\ &= -\max_{e \in END}(-Slack(e)) \approx -LSE(-Slack(END)) \\ &= -(M + log(\sum_{e \in END} \tau \cdot exp(\frac{-Slack(e) - M}{\tau}))) \end{aligned} \tag{8}$$

where τ is a temperature to control the smoothness of the function, and END is the set of negative slack end points, and $Slack(END)$ is the set of end points' slack. When $\tau \to 0$, $LSE(-Slack(END)) = \max_{e \in END}(-Slack(END))$. The parameter M in the function is set to 0 for numerical stability. Using this function, we can calculate the gradient of WNS with respect to the slack of an endpoint as shown in Equation (9). According to the relationship between slack and arrival time in Equation (1), the gradient of WNS with respect to the arrival time is the negative of the gradient of WNS with respect to the slack.

$$\begin{aligned} \frac{\partial WNS}{\partial Arr(e)} &= -\frac{\partial WNS}{\partial Slack(e)} \approx -\frac{\partial - LSE(-Slack(END))}{\partial Slack(e)} \\ &= \frac{exp(\frac{-Slack(e_i) - M}{\tau})}{\sum_{e_i \in END} exp(\frac{-Slack(e_i) - M}{\tau})} \end{aligned} \tag{9}$$

After this step, we can obtain the gradient of WNS with respect to the arrival time of the end points, which will then be used as the weight of the timing-related cost. This format of gradient ensures that we optimize the worst slack endpoints with the biggest weight, but also gives some weight to optimize the subcritical endpoints.

Given the gradient as the relative importance of different endpoints' arrival time, we still need to analyze the contribution of different net arcs' delay to the end points' arrival time. Therefore, we need to model the propagation at nets and cells. For nets, we need to calculate the contribution of the arrival time of the driver pin to the sink pins. For cells, we need to estimate the contribution of different input pins' arrival time to the arrival time of the output pin. We then introduce the calculation method as follows.

Backpropagation at cells. Given the gradient of the timing objective with respect to endpoints' arrival time $\frac{\partial \mathcal{L}}{\partial Arr(e)}$, we can perform backpropagation from the end points to the starting points, as shown in Figure 3. For a cell c, we need to calculate the contribution of the input pins' arrival time to the output pin's arrival time. To achieve this, we model the cell delay as the maximum arrival time of all inputs with a cell delay D_c as follows:

$$Arr(p_{out}) = \max_{p_i \in In}(Arr(i)) + D_c \tag{10}$$

where In is the set of input pins of the cell c.

Since this formula has a max function, which is not smooth. We thus also use the LSE operator to approximate the max function as follows:

$$\max_{p_i \in In}(Arr(p_i)) \approx LSE(Arr(In)) \tag{11}$$

$$= M + log(\sum_{p_i \in In} \tau \cdot exp(\frac{Arr(p_i) - M}{\tau})) \tag{12}$$

Similar to calculating the gradient of WNS with respect to endpoints' arrival time, we can propagate the gradient from a output p_{out} pin to a input pin $p_i \in In$ as follows:

$$\frac{\partial Arr(p_{out})}{\partial Arr(p_i)} \approx \frac{\partial LSE(In)}{\partial Arr(p_i)} = \frac{exp(\frac{Arr(p_i) - M}{\tau})}{\sum_{p_j \in In} exp(\frac{Arr(p_j) - M}{\tau})} \tag{13}$$

Since the estimated slacks of the pins are provided in the contest, we can make use of this information without the need to calculate the arrival time again. Recall that the slack of an input pin p can be represented as $Slack(p) = Req(p) - Arr(p)$. In our model as shown in Equation (10), all input pins share the same delay to the output pin. Therefore, for all input pins of a cell c, they share the same downstream paths, thus have the same required time. Therefore, among the input pins of the cell c, the difference of slack is the negative of the difference of their arrival time (i.e., $Slack(p_1) - Slack(p_2) = (Req(p_1) - Arr(p_1)) - (Req(p_2) - Arr(p_2)) = -(Arr(p_1) - Arr(p_2))$). Suppose M and M' are the biggest value of $Arr(p)$ and $-Slack(p)$, respectively, then $Arr(p) - M$ equals to $-Slack(p) - M'$. Therefore, in our implementation, we use Equation (14) to calculate the value in Equation (13).

$$\frac{\partial Arr(p_{out})}{\partial Arr(p_i)} \approx \frac{exp(\frac{-Slack(i) - M'}{\tau})}{\sum_{j \in In} exp(\frac{-Slack(j) - M'}{\tau})} \tag{14}$$

Using this formula, we can perform backpropagation at cells using the estimated slack given by the contest organizer. This emphasizes the worst slack input pin while still assigning some gradient to the subcritical inputs.

Backpropagation at nets. We also perform backpropagation at nets. Suppose a net has a driver pin p_0 and several sink pins.

For a specific sink pin p_{out}, the arrival time relationship can be represented as:

$$Arr(p_{out}) = Arr(p_0) + Delay(Arc_{p_0 \to p_{out}}) \tag{15}$$

Obviously, $\frac{\partial Arr(p_{out})}{\partial Arr(p_0)} = \frac{\partial Arr(p_{out})}{\partial Delay(Arc_{p_0 \to p_{out}})} = 1$, which means that both the arrival time of the driver pin and the net arc delay between the two pins have the same one unit of contribution to the arrival time of this output pin. Therefore, we can formulate the backpropagation at a net n as follows, where the gradient with respect to the driver pin's arrival time is the sum of the gradient with respect to all sink pins.

$$\frac{\partial \mathcal{L}}{\partial Arr(p_0)} = \sum_{p_{out} \in Sinks(n)} \frac{\partial \mathcal{L}}{\partial Arr(p_{out})} \tag{16}$$

With the back propagation at cells and at nets, we can estimate the gradient of our timing objective with respect to the arrival time at pins.

However, recall the method to calculate the capacity sensitivity in Figure 2, what we need is the gradient of the timing objective with respect to the delay of net arcs, rather than the gradient with respect to arrival time as shown in Equation (13) and Equation (16). Therefore, we need one last step to convert gradient with respect to the arrival times to the gradient with respect to the net arcs' delay. Leveraging the relationship between the net arc delay and the arrival time at the sink pin (Equation (15)), we can see that the timing objective gradient with respect to the two are the same:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial Delay(Arc_{p_0 \to p_{out}})} &= \frac{\partial \mathcal{L}}{\partial Arr(p_{out})} \cdot \frac{\partial Arr(p_{out})}{\partial Delay(Arc_{p_0 \to p_{out}})} \\ &= \frac{\partial \mathcal{L}}{\partial Arr(p_{out})} \cdot 1 = \frac{\partial \mathcal{L}}{\partial Arr(p_{out})} \end{aligned} \tag{17}$$

With Equation (17), we have completed the computation of gradients at the net-arc level (blue part in Figure 2). Next, we utilize these results to obtain gradients with respect to segment resistance and capacitance, as detailed in Section 3.2.

3.2 Segment-level Gradient

Layer assignment is an important step in timing-driven global routing. Higher metal layers generally exhibit lower resistance and sometimes lower capacitance, but their resources are limited. Hence, we should prioritize assigning higher layers to the segments that more importantly affect the net's timing.

A commonly used approach [4, 22] augments the routing cost with a "delay cost" computed per segment to optimize towards the timing metrics.

$$Delay(s) = R(s)\left(C(s)/2 + C_{\text{down}}(s)\right) \tag{18}$$

which encourages choosing layers with smaller resistance and capacitance. While being effective to 2-pin nets, such per-segment delay costs cannot fully capture the mutual influence between different segments in a net.

Take Figure 4 as an example. Suppose branch D–E–F is not critical, so we don't need to optimize the delay of this branch and its delay cost is thus 0. However, this branch contributes to the downstream capacitance of segment AD, thereby affecting the delay of segment AD and the downstream sinks on some entirely different

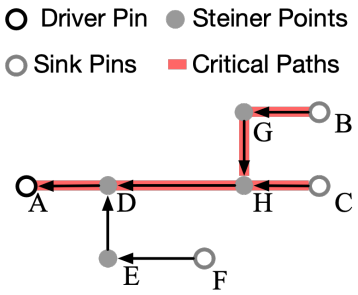


Figure 4: The routing graph of a 4-pin net.

branches (pins B and C). Simply penalize according to the delay of each segment omits this kind of coupling.

To capture this kind of mutual influence between wire segments, we compute a segment-level gradient by computing the derivatives of $Delay(Arc_{p_0 \to p_{out}})$ with respect to the capacitance $C(s_i)$ and resistance $R(s_i)$ of each segment s_i based on the Elmore delay model. The 2D structure of nets in the analysis is based on the basic routing DAG we introduced in Section 2.1.

These derivatives can reflect the influence of the segment capacitance and resistance on the net arc's delay. In Section 3.1, we have already obtained the gradient of the timing objective with respect to the net arc's delay (i.e., $\partial \mathcal{L}/\partial Delay(Arc_{p_0 \to p_{out}})$), we can combine the arc-level gradient and the segment-level gradient to estimate the influence of a segment's capacitance and resistance on the final timing objective, which can be represented as $\partial \mathcal{L}/\partial C(s)$ and $\partial \mathcal{L}/\partial R(s)$. We refer to these two values as capacitance sensitivity and resistance sensitivity, which will then be used as weights in the dynamic programming for layer assignment and global routing to constrain the increase in resistance and capacitance .

Capacitance sensitivity. We first recall that in the Elmore delay model, the delay from a driver pin p_0 to a sink pin p_{out} is the sum of segment delays along the path from p_0 to p_{out} (denoted as $P(p_0 \to p_{out})$):

$$Delay(Arc_{p_0 \to p_{out}}) = \sum_{s \in P(p_0 \to p_{out})} Delay(s). \tag{19}$$

Hereafter, we use U_s to present the set of segments on the upstream path from the driver pin to s, where s can be a segment or a sink pin. We use $\mathcal{S}$ to represent the set of all sinks and use $\mathcal{S}(s)$ to represent the set of sinks in the downstream of segment s. We also use the indicator $\mathbf{1}\{\cdot\}$ that equals 1 when the predicate holds and 0 otherwise.

Differentiating (18)–(19) and grouping terms yields

$$\frac{\partial Delay(Arc_{p_0 \to p_{out}})}{\partial C(s_i)} = \frac{R(s_i)}{2}\,\mathbf{1}\{s_i \in U_{p_{out}}\} + \sum_{s \in U_{p_{out}} \cap U_{s_i}} R(s). \tag{20}$$

where the first term is effective only if s_i lies on the $p_0 \to p_{out}$ path. The second term sums the resistances of the path segments that are upstream of s_i, reflecting that these upstream segments can "see" $C(s_i)$ as part of their downstream load.

Take Figure 4 as an example, the derivative of $Delay(Arc_{A\to B})$ with respect to the capacitance of segment EF is the resistance of segment AD. The reason is as follows. This segment is not on the path from A to B, thus the first term in Equation (20) is zero. Its upstream segments are AD and DE, but the part that overlaps with U_B is AD only, so the result is the resistance of segment AD.

Resistance sensitivity. From equation (18), we can see that the derivative w.r.t. $R(s_i)$ is simply the effective capacitance seen by s_i when it is on the path:

$$\frac{\partial Delay(Arc_{p_0\to p_{out}})}{\partial R(s_i)} = \big(C(s_i)/2 + C_{\text{down}}(s_i)\big)\,\mathbf{1}\{s_i \in U_{p_{out}}\}. \quad (21)$$

Take Figure 4 as an example, for a wire segment s_i on path $A \to C$, the derivative of $Delay(Arc_{A\to C})$ with respect to the resistance is $\big(C(s_i)/2 + C_{\text{down}}(s_i)\big)$, but for other segments like EF or BG, the derivative is zero, meaning that their resistance do not contribute to the delay from A to C.

Gradient on multi-sink nets. For a net with multiple sinks, a segment may influence multiple driver-sink arcs. Thus, we need to accumulate the gradients of the arc delays with respect to the segment's capacitance and resistance for every sink. We use $w_{p_{out}}$ to denote $\frac{\partial \mathcal{L}}{\partial Delay(Arc_{p_0\to p_{out}})}$. We compute the sum of the partial derivatives of arcs' delay with respect to a segment s_i as follows.

$$\frac{\partial \mathcal{L}}{\partial C(s_i)} = \sum_{p_{out}\in\mathcal{S}} w_{p_{out}} \frac{\partial Delay(Arc_{p_0\to p_{out}})}{\partial C(s_i)} \quad (22)$$

$$= \sum_{p_{out}\in\mathcal{S}} w_{p_{out}} \Big(\frac{R(s_i)}{2}\mathbf{1}\{s_i \in U_{p_{out}}\} + \sum_{s\in U_{p_{out}}\cap U_{s_i}} R(s)\Big) \quad (23)$$

$$= \Big(\sum_{p_{out}\in\mathcal{S}(s_i)} w_{p_{out}}\Big)\cdot\frac{R(s_i)}{2} + \sum_{s\in U_{s_i}} \Big(R(s)\sum_{p_{out}\in\mathcal{S}(s)} w_{p_{out}}\Big) \quad (24)$$

$$\frac{\partial \mathcal{L}}{\partial R(s_i)} = \sum_{p_{out}\in\mathcal{S}} w_{p_{out}} \frac{\partial Delay(Arc_{p_0\to p_{out}})}{\partial R(s_i)} \quad (25)$$

$$= \Big(\sum_{p_{out}\in\mathcal{S}(s_i)} w_{p_{out}}\Big)\cdot\Big(\frac{C(s_i)}{2} + C_{down}(s_i)\Big). \quad (26)$$

These weighted aggregations yield the segment-level gradients $\frac{\partial \mathcal{L}}{\partial C(s_i)}$ and $\frac{\partial \mathcal{L}}{\partial R(s_i)}$, quantifying each segment's contribution to the timing objective $\mathcal{L}$. For each segment, to compute Equation (23), we need a nested loop, where the outer loop scans through all the sinks in $\mathcal{S}$ and the inner loop needs to find the set $U_{p_{out}} \cap U_{s_i}$ and iterate through all the segments in it. This will introduce a complexity of $O(n^2 \cdot |\mathcal{S}|)$ to compute the gradients of all segments, where n is the number of segments, and $|\mathcal{S}|$ is the number of sinks.

To reduce the complexity of computing the gradient, we rewrite the equation to a more computational-friendly way as shown in Equation (24). The correctness of the transformation can be explained as follows. In a tree structure, a segment s_i is on the upstream of a sink pin p_{out} if and only if p_{out} is in the downstream of s_i (i.e. $s_i \in U_{p_{out}} \Leftrightarrow p_{out} \in \mathcal{S}(s_i)$). With this simple observation, we can eliminate the $\mathbf{1}\{s_i \in U_{p_{out}}\}$ in the first term and eliminate the $s \in U_{p_{out}}$ in the second term. The resulting formula Equation (24) and Equation (26) are then easy to be implemented by two tree traversals as shown in Algorithm 1. In the algorithm, we divide all segments in the tree into D levels. We then accumulate the downstream capacitance $C_{down}(s)$ and the sum of the downstream sinks' weight $W(s) = \sum_{Sink\in\mathcal{S}(s)} w_{p_{out}}$ from depth D to level one. After that, we traverse from the root segment to the leaf segments, maintaining the term $RW(s)$ to record the sum of $R(s')\sum_{p_{out}\in\mathcal{S}(s')} w_{p_{out}}$ for all segment s' from the root segment to segment s.

Algorithm 1: Gradient Computation

Input: D levels of segments
Output: $\frac{\partial \mathcal{L}}{\partial C(s)}$ and $\frac{\partial \mathcal{L}}{\partial R(s)}$ for all segments s

```
for level ← D to 1 do
    parallel for s ← Segments(level) do
        if s connects to p_out then
            W(s) ← w_{p_out}; C_down(s) ← C(s);
        if s has parent segment s_p then
            W(s_p) ← W(s_p) + W(s);
            C_down(s_p) ← C_down(s_p) + C(s);
for level ← 1 to D do
    if s has parent segment s_p then
        RW_up ← RW(s_p);
    else
        RW_up ← 0;
    parallel for s ← Segments(level) do
        ∂L/∂C(s) ← W(s) R(s)/2 + RW_up;
        RW(s) ← RW_up + R(s) × W(s);
        ∂L/∂R(s) ← W(s) × (C(s)/2 + C_down(s));
```

Hence, with two Breadth-First traversals, we can obtain the gradient of our timing objective $\mathcal{L}$ with respect to the capacitance and the resistance of each segment. This algorithm finishes the gradient computation in just $O(n)$ time. What's more, we can use multithreading to compute the gradients for the segments at the same level concurrently, which can further reduce the running time. In the gradient computation step, the resistance and capacitance are obtained by multiplying the wire length and the average per unit length capacitance and resistance across all layers. The resulting gradient is then provided to guide the router as valuable information that quantifies the importance of a segment's capacitance and resistance on the final timing results.

3.3 Timing-aware DAG-routing

Similar to [15, 16, 28], we parallelize net routing on a GPU. Nets are first partitioned into batches, where nets in one batch do not overlap with each other. In one batch, nodes at the same level are routed together. For example, in Figure 4, there are four levels $\{A\}$, $\{D\}$, $\{E, H, G\}$ and $\{B, C, F\}$. At each level, k GPU threads are used to parallelly route k nodes at that level.

For a layer l on a node n, we maintain a variable $Cost(n, l)$ as the minimum cost to connect the node with the downstream pins. For the routing graph in Figure 4, to connect node H on layer 4 to downstream pins, one possible way is to connect C from layer 6, connect G from layer 5, and then use a via to connect layer 4, 5 and 6. At each node, we enumerate all possible via ranges and ways to

connect with downstream nodes and will record the best cost and connection scheme. After the cost of the root is updated, we then trace back from the root to the leaf nodes to obtain the best layer assignment.

We can guide the DAG-based routing by defining the wire cost and via cost. For routers without timing consideration, the cost consists of wirelength and via count, and overflow. To make our router timing aware, we integrate the gradient with respect to capacitance and resistance as a penalty to constrain the capacitance and resistance of wires as follows:

$$Cost(s) = Overflow(s) + \gamma \cdot \left(\frac{\mathcal{L}}{\partial C(s)} \cdot C(s) + \frac{\mathcal{L}}{\partial R(s)} \cdot R(s)\right) \quad (27)$$

In this formulation, γ is a weight to balance timing and routing overflow. For each routing layer, we can easily obtain the per-unit-length resistance $r(\Omega/m)$ and capacitance $c(F/m)$, so that the resistance and capacitance of a wire segment of length ℓ can be computed $R(s) = r \cdot \ell$ and $C(s) = c \cdot \ell$.

To balance efficiency and quality, we adopt several simplifications. It is found that an accurate via delay calculation has little influence on quality, so we use a simplified via cost that consists of a constant *unit_via_cost* per via plus the overflow cost. During the rip-up and reroute phase, we route on an augmented DAG (as shown in Section 2.1) to repair congestion, where we directly use the gradient at the driver pin $\frac{\partial \mathcal{L}}{\partial Arr(p_0)}$ as the resistance and capacitance sensitivities, ignoring the segment-level gradient. Since the number of ripped-up nets is only about 15% of the total number of nets, this simplification does not compromise the overall timing quality.

In summary, we tradeoff efficiency and accuracy of computing the timing aware costs in our routing algorithm. Compared with directly using a delay-based cost, our approach leverages gradients to account for how a wire segment's capacitance and resistance influence the timing of all branches more comprehensively. Besides, as the timing cost merely applies extra weights to $C(s)$ and $R(s)$, our approach adds negligible computational overhead.

3.4 Power Optimization

Besides timing and congestion, power is another critical optimization metric in chip design. The total power of a design consists of three parts: leakage power, internal power, and switching power. Leakage power is technology-dependent static power dissipation. Internal power is mainly influenced by the cell characteristics and the input transition time, where routing will affect the transition time indirectly. These two sources of power consumption cannot be significantly reduced by routing. On the other hand, switching power is proportional to the load capacitance of a cell as follows:

$$P_{\text{switching}} = \alpha \cdot C_{\text{load}} \cdot V^2 \cdot f \quad (28)$$

where α is a switching activity factor representing the probability of signal transitions ($0 \rightarrow 1$ or $1 \rightarrow 0$) per clock cycle, C_{load} is the load capacitance, V is the supply voltage, and f is the frequency of the clock. The switching activity factor α ranges from 0 to 1, with 0 indicating no switching activity and 1 indicating signal switches in every clock cycle. We can optimize power consumption by reducing the switching power.

Table 1: Details of the benchmark.

	Benchmark	w_{TNS}	w_{WNS}	w_{Pow}	w_{Ovf}	#ends	#nets
0	Ariane	-10	-100	300	3.0E-07	20K	123K
1	Bsg	-10	-100	25	4.0E-08	215K	737K
2	NVDLA	-0.05	-0.5	25	1.5E-07	46K	199K
3	MemPool-Tile	-1	-10	300	7.0E-07	13K	136K
4	MemPool-Group	-1	-10	20	3.0E-08	348K	3.3M
5	MemPool-Cluster	-1	-10	0.3	5.0E-09	1.1M	12M
6	Ariane_r	-0.2	-2	100	4.0E-07	20K	106K
7	BlackParrot_r	-0.1	-1	50	2.0E-08	215K	768K
8	NVDLA_r	-0.01	-0.1	100	1.0E-07	46K	158K
9	MemPool-Tile_r	-3	-30	100	1.0E-06	13K	136K
10	MemPool-Group_r	-0.5	-5	3	4.0E-08	348K	3.2M
11	MemPool-Cluster_r	-0.4	-4	2	1.0E-08	1.1M	12M

The load capacitance C_{load} is the sum of the downstream capacitances of all the pins and wire segments connected to a cell's output, which can be directly affected by the routing solution. We can optimize the switching power by minimizing the load capacitances of the cells. We use OpenRoad [2] to compute cells' activity data $w_p = \alpha \cdot V^2 \cdot f$ and store it in a file. When the power optimization is enabled, our router reads the activity data from the file. For a net n, we give an extra weight w_p (from its driver cell's activity) to the capacitance sensitivity for all wire segments.

By weighting segment-level capacitance sensitivities with driver activity, the router prioritizes reducing high-impact load capacitances, thereby lowering switching power with minimal impact on timing and congestion.

4 Experiments

Table 2: Experimental result of total score and runtime.

Benchmark	1st		2nd		3rd		Ours	
	Score	Runtime	Score	Runtime	Score	Runtime	Score	Runtime
0	0.897	6.51	0.640	7.04	0.095	15.88	-0.945	7.24
1	0.562	32.94	0.722	22.07	0.094	37.25	-0.409	24.55
2	-1.995	9.20	-2.077	22.37	-1.824	18.92	-2.144	12.36
3	-0.405	7.49	0.401	11.03	1.370	15.53	0.443	11.80
4	-2.029	133.53	-1.295	153.52	-3.599	266.88	-2.180	103.88
5	0.510	501.61	0.399	839.77	0.714	915.47	0.497	609.10
6	1.780	7.06	1.646	7.15	1.519	23.74	1.461	11.35
7	0.860	37.23	0.426	27.79	0.700	44.80	0.424	33.70
8	1.402	9.42	4.375	20.63	1.364	19.43	1.391	16.32
9	0.358	7.01	2.088	11.47	1.817	15.88	1.205	11.75
10	1.333	135.70	1.233	161.55	3.117	247.71	1.346	103.78
11	2.327	491.11	2.371	766.29	2.678	1965.67	2.313	550.58
Average	0.467	**114.90**	0.911	170.89	0.670	298.93	**0.284**	124.70
Ratio	1.645	**0.921**	2.209	1.370	1.191	2.397	**1.000**	1.000

We conducted our experiments on a 64-bit Linux workstation with Intel Xeon Silver 4114 CPU (2.20GHz) and 256GB memory. To be consistent with the ISPD2025 contest [12], we use one GPU (GeForce RTX 3090) and 8 CPU threads. The details of the contest benchmark are listed in Table 1. We obtained the binary files from the top 3 teams of the contest and ran them on our machine.

4.1 Comparison with the contest winners

For fair comparison with the top contest teams, we compute the weighted sums using the organizer-provided weights and method (Table 1, Equation (5)). Note that lower scores are better. Since our power optimization needs extra information (i.e., the activity of

Table 3: Experimental results of the top-3 teams of ISPD2025 Contest and our algorithm.

Bench	1st				2nd				3rd				Ours			
	TNS	WNS	Power	Congestion	TNS	WNS	Power	Congestion	TNS	WNS	Power	Congestion	TNS	WNS	Power	Congestion
0	-1325.78	-0.432	0.646	5846461	-1340.19	-0.398	0.646	5902618	-1138.37	-0.381	0.646	8003814	-1026.76	-0.379	0.646	6438399
1	-10366.19	-0.420	3.054	21332547	-10774.87	-0.424	3.053	20468992	-9344.33	-0.407	3.054	25412296	-8666.18	-0.401	3.053	22573155
2	-521344.56	-56.919	2.941	13315606	-514660.28	-55.841	2.940	13686713	-514674.25	-55.914	2.938	15687876	-515533.69	-55.926	2.941	13056103
3	-1610.01	-0.379	0.143	3019425	-2225.46	-0.489	0.145	2478317	-2524.33	-0.520	0.145	3469019	-2085.02	-0.461	0.145	2910471
4	-20473.81	-0.305	7.692	55005746	-21709.13	-0.347	7.734	49071539	-19338.70	-0.275	7.553	97506552	-19042.52	-0.283	7.673	65052484
5	-59712.49	-0.266	23.450	236920427	-62459.98	-0.315	23.578	192012708	-59533.47	-0.276	23.220	289712328	-54359.36	-0.264	23.320	252306571
6	-650.10	-1.756	0.156	4349152	-149.52	-0.737	0.156	4651055	-162.38	-0.633	0.156	4386728	-45.87	-0.431	0.156	4374734
7	-2859.79	-3.984	0.305	22567636	0.00	0.000	0.305	21307154	-2936.03	-2.747	0.305	20564539	0.00	0.000	0.305	21213727
8	0.00	0.000	0.136	14061156	0.00	0.000	0.136	16352692	0.00	0.000	0.136	13637302	0.00	0.000	0.136	13936159
9	-1356.07	-0.342	0.143	2232008	-1920.72	-0.472	0.145	2122543	-1806.79	-0.440	0.145	2168599	-1600.96	-0.392	0.144	2246898
10	-32016.10	-0.433	8.369	52783071	-34846.05	-0.460	8.359	49722314	-26801.44	-0.373	7.942	132027917	-29386.86	-0.421	8.349	55695359
11	-42564.08	-0.222	24.244	242737068	-50372.44	-0.294	24.380	214110807	-39516.23	-0.200	23.968	335020596	-37846.03	-0.216	24.152	261589161
Ratio	1.000	1.000	1.000	1.000	0.954	0.975	1.004	**0.956**	0.969	0.937	**0.995**	1.302	**0.815**	**0.857**	1.000	1.036

Table 4: Experimental results of power optimization.

Bench	w/o power optimization				w/ power optimization			
	P_s	P_t	Congestion	Score	P_s	P_t	Congestion	Score
0	0.0009	0.6461	6438399	-0.945	0.0009	0.6460	6443602	-0.977
1	0.0772	3.0531	22573155	-0.409	0.0773	3.0531	22576420	-0.409
2	0.1621	2.9406	13056103	-2.144	0.1561	2.9345	13175395	-2.285
3	0.0488	0.1447	2910471	0.443	0.0454	0.1417	3069599	-0.349
4	3.2985	7.6727	65052484	-2.180	3.0019	7.3692	72775512	-7.962
5	10.2331	23.3196	252306571	0.497	9.7320	22.8011	269961040	0.426
6	0.0001	0.1561	4374734	1.461	0.0001	0.1561	4376830	1.462
7	0.0024	0.3048	21213727	0.424	0.0024	0.3048	21180832	0.424
8	0.0013	0.1363	13936159	1.391	0.0013	0.1363	13972036	1.394
9	0.0476	0.1443	2246898	1.205	0.0447	0.1410	2416448	1.172
10	3.7161	8.3489	55695359	1.346	3.2413	7.8588	64429910	0.235
11	10.9162	24.1525	261589161	2.313	9.4767	22.6697	292718663	-0.332
Average	2.3754	5.9183	**60116102**	0.284	**2.1483**	**5.6843**	65591357	**-0.600**
Ratio	1.106	1.041	**0.917**		**1.000**	**1.000**	1.000	

cells), we see it as an extra optimization method and will show the experiment result in Section 4.2. Thus, we don't include it in this table for fair comparison. The total score and runtime comparison is shown in Table 2. On average, we can outperform the winning team by 64.5% in quality score with moderate runtime overhead.

Note that the negative scores in Table 2 and Table 4 are legitimate. In Equation (5), the first three terms are computed as differences from the median, so better WNS, TNS, or power contribute negative values to the total score.

The detailed performance across all metrics is shown in Table 3. Since the test cases vary significantly in size, we compute the ratio for each case individually by dividing our result by the first-place team's result, and then take the average of these ratios. This approach prevents larger cases from dominating the results. For TNS and WNS, larger values indicate better performance, whereas for the ratio, smaller values are better. For case 8, all teams achieve 0 TNS and WNS, so we mark the ratio for all teams as 1. Overall, our approach demonstrates 18.5%, 14.3% improvement on TNS, WNS respectively compared to the first-place solution, with a modest congestion overhead of 3.6%.

4.2 Power optimization

Table 4 compares switching power (P_s), total power (P_t) and contest scores between two versions of our algorithms. On average, our power optimization algorithm achieves a 10.6% reduction in switching power, which results in a 4.1% reduction in total power and a contest score improvement by 0.884. Since this optimization gives extra capacity weight to nets driven by high-activity cells, it brings 8.3% congestion overhead. Compared to the version without power optimization, the difference in timing results is very small (0.1% more WNS and 0.05% less TNS), so we don't show it in the table. The optimization requires negligible runtime overhead, only needing to read pre-generated cell activity data. Overall, our approach significantly improves power metrics and contest scores with acceptable congestion overhead, showing the effectiveness of the algorithm.

5 Conclusions

In this paper, we propose a gradient-based two-level analysis framework for timing-driven global routing. At the inter-net level, we calculate the gradient of the timing objective with respect to the net timing arcs' delay. At the intra-net level, we utilize the gradient of Elmore delay to estimate the influence of segments' capacitance and resistance on the timing arcs. Using the gradient as the weight of capacitance and resistance, we implement a timing-driven global router. What's more, we also effectively include power-cooptimization in our router. In comparison with the winner of the ISPD 2025 contest, our algorithm achieves better contest scores, improving the WNS and TNS by 14.3% and 18.5%, respectively. The power optimization technique can further reduce 10.6% switching power and significantly reduce the contest scores.

References

[1] 2022. PrimeTime SI: Crosstalk Delay and Noise. https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html/.

[2] 2024. OpenROAD. https://github.com/The-OpenROAD-Project/OpenROAD.git. Accessed: April 2025.

[3] 2024. OpenSTA, version: 2.5.0. https://github.com/abk-openroad/OpenSTA. Accessed: April 2025.

[4] Jianchang Ao, Sheqin Dong, Song Chen, and Satoshi Goto. 2013. Delay-driven layer assignment in global routing under multi-tier interconnect structure. In *ACM International Symposium on Physical Design (ISPD)* (Stateline, Nevada, USA). 101–107. doi:10.1145/2451916.2451942

[5] Wing-Kai Chow, Liang Li, Evangeline F. Y. Young, and Chiu-Wing Sham. 2014. Obstacle-avoiding rectilinear Steiner tree construction in sequential and parallel approach. *Integr. VLSI J.* 47, 1 (Jan. 2014), 105–114. doi:10.1016/j.vlsi.2013.08.001

[6] W. C. Elmore. 1948. The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers. *Journal of Applied Physics (JAP)* 19, 1 (01 1948), 55–63. arXiv:https://pubs.aip.org/aip/jap/article-pdf/19/1/55/18307672/55_1_online.pdf doi:10.1063/1.1697872

[7] Zizheng Guo, Tsung-Wei Huang, Zhou Jin, Cheng Zhuo, Yibo Lin, Runsheng Wang, and Ru Huang. 2024. Heterogeneous Static Timing Analysis with Advanced Delay Calculator. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE).*

[8] Zizheng Guo, Tsung-Wei Huang, and Yibo Lin. 2023. Accelerating Static Timing Analysis using CPU-GPU Heterogeneous Parallelism. *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems (TCAD) (2023), 1–1. doi:10.1109/TCAD.2023.3286261

[9] Stephan Held, Dirk Müller, Daniel Rotter, Rudolf Scheifele, Vera Traub, and Jens Vygen. 2018. Global Routing With Timing Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 2 (2018), 406–419. doi:10.1109/TCAD.2017.2697964

[10] Tsung-Wei Huang and Martin D. F. Wong. 2015. OpenTimer: A high-performance timing analysis tool. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 895–902. doi:10.1109/ICCAD.2015.7372666

[11] Joon-Seok Kim, Joonyun Kim, Dae-Jin Yang, Jaewoo Shim, Luhing Hu, Chang-Seok Lee, Jeehwan Kim, and Sang Won Kim. 2024. Addressing interconnect challenges for enhanced computing performance. *Science* 386, 6727 (2024), eadk6189.

[12] Rongjian Liang, Anthony Agnesina, Wen-Hao Liu, Matt Liberty, Hsin-Tzu Chang, and Haoxing Ren. 2025. Invited: ISPD 2025 Performance-Driven Large Scale Global Routing Contest. In *ACM International Symposium on Physical Design (ISPD)*. 252–256. doi:10.1145/3698364.3715706

[13] Shiju Lin, Guannan Guo, Tsung-Wei Huang, Weihua Sheng, Evangeline Young, and Martin Wong. 2024. GCS-Timer: GPU-Accelerated Current Source Model Based Static Timing Analysis. In *ACM/IEEE Design Automation Conference (DAC)*. Article 71, 6 pages. doi:10.1145/3649329.3655983

[14] Shiju Lin, Jinwei Liu, Evangeline F. Y. Young, and Martin D. F. Wong. 2023. GAMER: GPU-Accelerated Maze Routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 42, 2 (2023), 583–593. doi:10.1109/TCAD.2022.3184281

[15] Shiju Lin and Martin D. F. Wong. 2022. Superfast Full-Scale GPU-Accelerated Global Routing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Article 51, 8 pages. doi:10.1145/3508352.3549474

[16] Shiju Lin, Liang Xiao, Jinwei Liu, and Evangeline F. Y. Young. 2025. InstantGR: Scalable GPU Parallelization for Global Routing. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design* (Newark Liberty International Airport Marriott, New York, NY, USA) *(ICCAD '24)*. Association for Computing Machinery, New York, NY, USA, Article 148, 8 pages. doi:10.1145/3676536.3676787

[17] Derong Liu, Bei Yu, Salim Chowdhury, and David Z. Pan. 2018. TILA-S: Timing-Driven Incremental Layer Assignment Avoiding Slew Violations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 37, 1 (2018), 231–244. doi:10.1109/TCAD.2017.2652221

[18] Jinwei Liu, Gengjie Chen, and Evangeline F.Y. Young. 2021. REST: Constructing Rectilinear Steiner Minimum Tree via Reinforcement Learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 1135–1140. doi:10.1109/DAC18074.2021.9586209

[19] Jinwei Liu and Evangeline F.Y. Young. 2023. EDGE: Efficient DAG-based Global Routing Engine. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[20] Lixin Liu, Bangqi Fu, Shiju Lin, Jinwei Liu, Evangeline F. Y. Young, and Martin D. F. Wong. 2024. Xplace: An Extremely Fast and Extensible Placement Framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 43, 6 (2024), 1872–1885. doi:10.1109/TCAD.2023.3346291

[21] Siting Liu, Peiyu Liao, Rui Zhang, Zhitang Chen, Wenlong Lv, Yibo Lin, and Bei Yu. 2022. FastGR: Global Routing on CPU-GPU with Heterogeneous Task Graph Scheduler. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 760–765. doi:10.23919/DATE54114.2022.9774606

[22] Vinicius Livramento, Derong Liu, Salim Chowdhury, Bei Yu, Xiaoqing Xu, David Z. Pan, José Luís Güntzel, and Luiz C. V dos Santos. 2017. Incremental Layer Assignment Driven by an External Signoff Timing Engine. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 36, 7 (2017), 1126–1139. doi:10.1109/TCAD.2016.2638450

[23] Yi-Chen Lu, Zhizheng Guo, Kishor Kunal, Rongjian Liang, and Haoxing Ren. 2025. INSTA: An Ultra-Fast, Differentiable, Statistical Static Timing Analysis Engine for Industrial Physical Design Applications. In *2025 62th ACM/IEEE Design Automation Conference (DAC)*.

[24] Gracieli Posser, Evangeline FY Young, Stephan Held, Yih-Lang Li, and David Z Pan. 2022. Challenges and approaches in vlsi routing. In *ACM International Symposium on Physical Design (ISPD)*. 185–192.

[25] Rudolf Scheifele. 2016. RC-aware global routing. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. doi:10.1145/2966986.2967067

[26] Gang Wu and Chris Chu. 2017. Two Approaches for Timing-Driven Placement by Lagrangian Relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 12 (2017), 2093–2105. doi:10.1109/TCAD.2017.2697947

[27] Liang Xiao, Shiju Lin, Jinwei Liu, Qinkai Duan, Tsung-Yi Ho, and Evangeline F. Y. Young. 2025. InstantGR: Scalable GPU Parallelization for 3-D Global Routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2025), 1–1. doi:10.1109/TCAD.2025.3573685

[28] Chunyuan Zhao, Zizheng Guo, Rui Wang, Zaiwen Wen, Yun Liang, and Yibo Lin. 2024. HeLEM-GR: Heterogeneous Global Routing with Linearized Exponential Multiplier Method. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.