Differentiable Physical Optimization

Yufan Du[1,2], Zizheng Guo[2,3], Runsheng Wang[2,3,4], Yibo Lin[2,3,4*]
[1]School of EECS, Peking University, [2]School of Integrated Circuits, Peking University,
[3]Institute of EDA, Peking University, [4]Beijing Advanced Innovation Center for Integrated Circuits
nbsdyf@hotmail.com, {gzz, r.wang, yibolin}@pku.edu.cn

***Abstract*—Gate sizing and buffer insertion are crucial for VLSI physical optimization; however, conventional decoupled approaches often yield suboptimal solutions due to uncoordinated resource allocation. Existing simultaneous methods resort to oversimplified timing models or heuristic assumptions, failing to unify the two tasks mathematically rigorously. We present a differentiable physical optimization framework integrating both techniques with GPU acceleration. Key innovations include timing-aware buffer tree skeleton construction, physics-aware modeling, and discrete-aware optimization algorithms. Experiments demonstrate 23% total negative slack (TNS) improvement and 12% worst negative slack (WNS) improvement with similar power consumption and 30× speedup versus CPU-based optimization flow. This work establishes a new paradigm for co-optimizing interdependent physical design tasks with rigorous modeling and efficient computation.**

I. Introduction

Physical optimization is critical in the modern VLSI physical design flow. It plays a crucial role in resolving design violations and also enhances circuit performance by improving electrical characteristics. Of all the techniques that can be adopted during physical optimization, gate sizing and buffer insertion are arguably the most effective [1]. While the former adjusts circuit gate sizes to balance timing constraints against power and area trade-offs, the latter inserts the necessary buffers in circuit nets for similar goals. Common objectives make simultaneous optimization feasible [2], [3].

Furthermore, two optimization tasks substantially affect each other, which means that separate optimization efforts may result in suboptimal solutions [1]. Consider the example in Figure 1, an underpowered AND gate drives a long wire beyond its capacitance. If buffer insertion is applied only, only a small buffer can be inserted due to the weak AND gate drive strength. Therefore, the remaining wire is still beyond buffer driving capability, leading to another buffer insertion. On the contrary, by applying driver sizing only, the sizer will invariably choose an extremely large size to drive the large capacitive load. Although it improves the delay, the net will still likely require buffers to handle the resistive interconnect. The upstream driver also suffers from the increased input load of the AND gate. A more reasonably sized driver and reduced buffer usage can only be achieved through simultaneous consideration.

Unfortunately, extensive research is focused on gate sizing or buffer insertion independently [4]–[9], failing to unlock a larger optimization space. Though a few works [1]–[3], [10] consider them simultaneously, they still rely on simplified and obsolete timing models or heuristic assumptions based on experience due to the intricacy of rigorous timing modeling, especially for the complicated impact of buffer insertion and

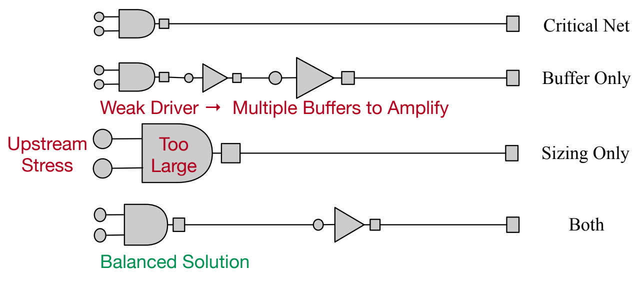


Fig. 1. A single-sink net with a long wire. Separate optimizations can lead to suboptimal solutions, such as additional powering-up buffers or an oversized driver. Simultaneous optimization provides the best solution.

gate sizing on timing. For example, a rigorous timing analysis should include both capacitance load and signal slew propagation in two bi-directional ways of the timing graph. Neither heuristic modeling methods [1], [3], [10] nor black box-based machine learning methods [2] manage to rigorously model the interdependency of different calculation parts in timing analysis. Therefore, they cannot accurately quantify how each optimization operation betters timing, let alone wisely allocate resources between the two optimization methods.

The strong discrete solution space of buffer insertion comes as yet another challenge. The dynamic programming method [11] focuses on local features and only considers one net across the entire timing path per iteration, thereby failing to explore a global optimal solution. Apart from that, the lack of physical constraints for upsizing and inserting buffers in prior works leads to local congestion or unreasonable buffer locations, such as hovering over a macro. Although they resort to detailed placement to legalize this, the significant displacement incurred sabotages the original placement, gate sizing, and buffer insertion effects. The last but not least challenge is the non-parallelizability of traditional gate sizing and buffer insertion. Therefore, they are mainly based on traditional CPU flow with limited room for further speedup.

To address those challenges, we propose a differentiable PPA optimization framework that quantitatively models how gate sizing and buffer insertion determine design quality. We construct a physical-aware objective to assist in avoiding congestion or buffering over obstacles. We proposed discrete-aware optimization algorithms that are compatible with the strong discrete solution space, ensuring all the gates and buffer candidates can be considered in a single iteration as well. Our framework's computationally intensive parts are deployed on the GPU, making it remarkably efficient.

Our contributions are summarized as follows:

- To the best of our knowledge, we propose the first differ-

entiable physical optimization framework that seamlessly integrates gate sizing and buffer insertion.
- We constructed a physical-aware objective to mitigate local congestion and illegal location buffering.
- Our method guarantees gradient precision and utilizes discrete-aware optimization algorithms to address challenging discrete problems effectively.
- Experimental results show marked improvements with an average enhancement of 23% in total negative slack (TNS), 12% in worst negative slack (WNS), and similar power consumption, while on average 30× speed up than OpenROAD [12].

We believe our work will certainly encourage VLSI physical optimization research from new perspectives. The rest of this paper is organized as follows. Section II shows the preliminaries and problem formulation; Section III details our differentiable physical optimization framework; Section IV demonstrates the results; Section V concludes the paper.

II. Preliminaries

This section reviews the background of VLSI physical optimization and prevalent differentiable optimization techniques, followed by problem formulation.

A. VLSI Physical Optimization

To achieve timing closure and power optimization in a placed design, gate sizing and buffer insertion are two of the most effective transforms that can be performed [1] for physical optimization.

1) Gate sizing: Gate sizing optimizes timing by selecting the drive strengths of each gate to ensure signal integrity on critical paths while minimizing power and area on non-critical paths. This NP-hard problem [13] exhibits inherent complexity from discrete sizing choices, non-convex delay models, and numerous near-critical paths [14]. Classical approaches include: (1) Dynamic programming methods [15]–[17] effective for tree-structured circuits but failing on general topologies; (2) Sensitivity-driven techniques [8], [18], [19] limited by heuristic quality; (3) Learning-based frameworks [20]–[23] requiring costly retraining across technology nodes; (4) Lagrangian relaxation (LR) methods [4], [5], [24]–[30] constrained by local optima.

Recent differentiable methods [7], [31]–[33] exhibit remarkable performance. However, they process benchmarks by simply minimizing gate sizes from well-optimized, both sized and buffered, circuits. It deviates from the normal flow and restricts exploration space due to already inserted buffers.

2) Buffer Insertion: Buffer insertion places buffers to reduce upstream load and mitigate interconnect net delay while balancing power-area tradeoffs. Its optimization complexity stems from the circuit structure transformation [34] and accurate timing modeling. Van Ginneken algorithm [35] pioneered dynamic programming for optimal buffer placement, later extended to handle noise-aware [36] and higher-order delay models [37]. However, existing methods predominantly operate on post-sized circuits, restricting their solution space to locally optimal regions. This sequential optimization fails to exploit the fundamental coupling between driver strength and buffer requirements, as demonstrated in Section I.

TABLE I
Summary of notations.

Notation	Description
$\mathcal{G}$	Set of original gates
$\mathcal{B}_{\text{cand}}$	Candidate buffer locations
$\mathcal{E}$	Timing endpoints set
$\mathcal{N}$	Interconnect net set
$\mathbf{s}_g$	Size for gate g
$\mathbf{bs}_u$	Size for buffer at $u \in \mathcal{B}_{\text{cand}}$
$\mathbf{b}_u$	Buffer insertion indicator at $u \in \mathcal{B}_{\text{cand}}$
$\Phi(x, y)$	Normalized electric potential at location (x, y)
$\Delta_{u \to v}$	Delay from node u to v
$a_v^{in/out}$	Arrival time at node v input/output pin
$\text{Slew}_v^{in/out}$	Slew rate at node v input/output pin
$\text{Leak}(d, \mathbf{s}_d)$	Leakage power of gate d with size $\mathbf{s}_d$
$\text{Area}(d, \mathbf{s}_d)$	Area of gate d with size $\mathbf{s}_d$
$L_u^{in/out}$	Load capacitance at buffer u input/output
$C_{\text{in}}^{\text{buf}}(\mathbf{bs}_u)$	Input capacitance of buffer with size $\mathbf{bs}_u$
$C_{\text{in},u}^{\text{gate g}}(\mathbf{s}_g)$	Input capacitance of gate g pin u with size $\mathbf{s}_g$
$R_{u \to v}$	Interconnect resistance between u and v
TNS	Total negative slack (Timing objective)
WNS	Worst negative slack
P_{total}	Unified power penalty
D_{total}	Unified density penalty

B. Differentiable Optimization Trend

Differentiable optimization has become a trend for VLSI physical design, converting traditionally combinatorial problems into continuous solutions with gradient-driven optimization techniques. Transforming discrete constraints into differentiable objectives enables gradient back propagation to guide optimization [38]–[40]. Unlike heuristic methods, differentiable frameworks systematically propagate timing, power, and area gradients across all reliant computing components and natively support GPU acceleration [41]. Those gradients directly relate the objectives to the inputs, thus guaranteeing the explainability and reliability. Its breakthrough potential is evidenced across placement [42], routing [39], and logic synthesis [43] while maintaining mathematical rigor absent in black-box machine learning.

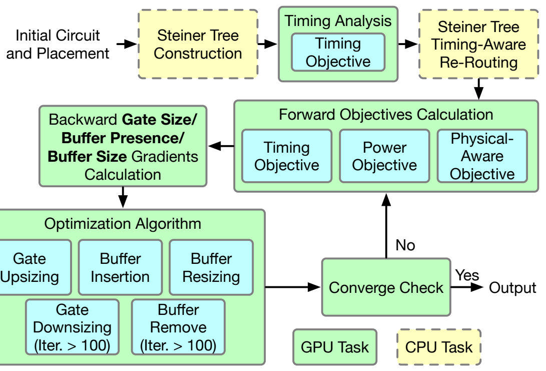
Recent successes in gate sizing [31]–[33] further establish this methodology as a trend for next-generation EDA tools. However, the completely discrete nature of buffer insertion makes applying differentiable methods incompatible. Additionally, circuit graph transformation triggered by buffer insertion is hard to model in a differentiable manner. Therefore, differentiable buffer insertion has not yet been proposed, let alone further integrated with gate sizing in a single differentiable physical optimization framework.

C. Problem Formulation

Problem. Given a set of gates and an initial placement layout, the objective is to mitigate the timing violation and minimize total power by simultaneously determining gate size set $\mathbf{s}$, buffer presence set $\mathbf{b}$, buffer size set $\mathbf{bs}$, buffer location set $\mathbf{bx}$ and $\mathbf{by}$, and the connection for each buffer tree.

III. Methodology

In this section, we present a novel, unified, differentiable framework for simultaneously optimizing gate sizing and buffer insertion. Section III-A strictly defines the discrete solution space, which includes gate size, buffer presence,



Initial Circuit and Placement
Steiner Tree Construction
Timing Analysis
Timing Objective
Steiner Tree Timing-Aware Re-Routing
Backward Gate Size/ Buffer Presence/ Buffer Size Gradients Calculation
Forward Objectives Calculation
Timing Objective
Power Objective
Physical-Aware Objective
Optimization Algorithm
Gate Upsizing
Buffer Insertion
Buffer Resizing
Gate Downsizing (Iter. > 100)
Buffer Remove (Iter. > 100)
Converge Check
No
Yes
Output
GPU Task
CPU Task

Fig. 2. The whole flow of our framework.

and buffer size variables. Based on these three sets of variables, sections III-B-III-D detail differentiable functions for timing, power, and physical-aware objectives, respectively. Section III-E proposes the unified optimization algorithm that conducts simultaneous gate sizing and buffer insertion. The whole framework is illustrated in Figure 2.

A. Solution Space Formulation

The solution space formulation has a critical impact on both the quality and efficiency of differentiable optimization for physical design. Our key insight is that buffer insertion and gate sizing require fundamentally different treatments. While gate sizing operates within a predefined discrete space $\mathcal{S}_g$ from the library, buffer insertion introduces multidimensional complexity across four variables: buffer presence, size, location, and connectivity (e.g., series vs. parallel connection for two or more buffers). Traditional approaches either oversimplify this space, thereby losing optimal solutions, or overexpand it, resulting in a computational explosion.

Our framework strategically constrains the solution space while preserving optimization flexibility:

- **Location Constraints**: Buffer candidate $\mathcal{B}_{\text{cand}}$ locations are restricted to early routing tree nodes and 20μm wire segments, reducing spatial complexity while maintaining sufficient solution space
- **Connectivity Constraints**: Automatic upstream connections based on the routing tree pattern, thus simplifying the degree of freedom on connectivity

The early routing tree should be carefully constructed to unlock a better solution while saving the wirelength resource. Some prior works [11], [37] simply leverage the Steiner tree skeleton, focusing solely on minimizing wirelength and failing to unlock a larger buffer insertion solution space. As shown in Figure 3, the most critical sink pin x is connected to the driver through a detour path. It leads to poor timing performance, thus requiring more buffers to be inserted. More buffers will, in turn, negatively prolong the pin x arrival time and also increase the area and power consumption.

Our critical innovation lies in our timing-aware re-branching mechanism, which strikes a balance between wirelength advantage and timing enhancement. We begin with a single timing analysis (as described in III-B) based on initial Steiner tree routing results to assess the current performance of the

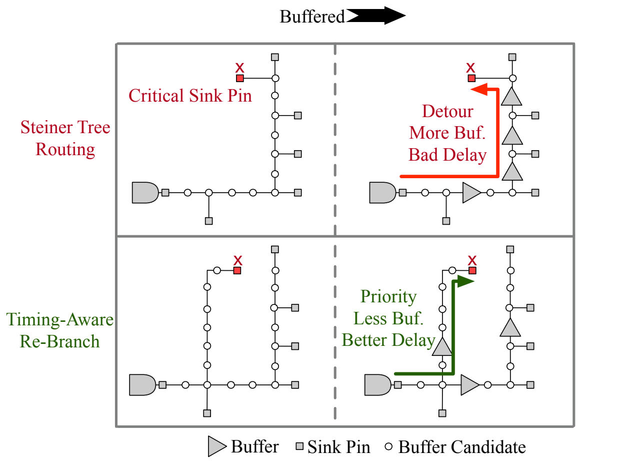


Fig. 3. Our timing-aware re-branching adapts the Steiner skeleton to create optimized buffer insertion points for critical sinks while saving wirelength.

circuit. Let $S = \{s_1, s_2, ..., s_n\}$ denote the set of sink pins in a net. For each sink pin $s_i \in S$, we define its *criticality* as:

$$C(s_i) = (-\text{slack}(s_i)) \times N_{\text{path}}(s_i) \tag{1}$$

where $N_{\text{path}}(s_i)$ counts how many critical paths pass through s_i. The timing slack and critical path number both determine the sink pin criticality.

The normalized criticality is computed as:

$$\hat{C}(s_i) = \frac{C(s_i)}{\max_{s_j \in S} C(s_j)} \tag{2}$$

Sinks are processed in descending order of $\hat{C}(s_i)$. For each sink s_i with routing level p_i (defined as its hop count from the Steiner root), we calculate the upstream jump level:

$$k_i = \left\lceil \hat{C}(s_i) \times p_i \right\rceil \tag{3}$$

where $\lceil \cdot \rceil$ denotes the ceiling function. This allows connecting s_i to an ancestor node k_i levels upstream in the Steiner tree.

To balance timing optimization with wirelength preservation, we constrain the number of re-branched sinks:

$$|\{s_i | s_i \text{ is re-branched}\}| \leq 0.2|S| \tag{4}$$

As shown in the lower part of Figure 3, the new re-branching skeleton reduces latency for critical sink x by k_x-level upstream connection. The algorithm prioritizes the most timing-critical sinks (the highest $\hat{C}$ values) while preserving most of the original Steiner topology for wirelength resource.

With timing-aware re-branching and buffer candidate set $\mathcal{B}_{\text{cand}}$, the unified solution space is defined as:

- Gate sizes $\mathbf{s}_g \in \mathcal{S}_g$
- Buffer presence $\mathbf{b}_u \in \{0, 1\}$ at candidate location $u \in \mathcal{B}_{\text{cand}}$
- Buffer sizes $\mathbf{bs}_u \in \mathcal{S}_b$ at candidate location $u \in \mathcal{B}_{\text{cand}}$

With the optimization variables defined, the unified optimization objective is formulated as:

$$\min_{\mathbf{s},\mathbf{b},\mathbf{bs}} |\text{TNS}| + \alpha_1 P_{\text{total}} + \alpha_2 D_{\text{total}} \tag{5}$$

with default coefficients $\alpha_1 = 0.001$ and $\alpha_2 = 0.005$ balancing timing/power/physical-aware density tradeoffs. The user can also adjust them based on their needs.

In the following sections, we will describe how to calculate these three differentiable objectives.

B. Timing Objective

Timing optimization is the cornerstone of VLSI physical design optimization. An effective optimization framework demands precise and rigorous timing objective analysis, which presents significant challenges in our work due to: (1) the inherent complexity of modern timing models; (2) the dynamic circuit topology changes introduced by buffer insertion; (3) the complex bi-directional impact of buffer insertion, buffer size, and gate size on upstream capacitance/delay and downstream slew/delay; and (4) the non-triviality to maintain the objective differentiable with respect to three set of optimization variables. These factors collectively make accurate timing analysis particularly demanding in the context of simultaneous buffer insertion and gate sizing optimization. In response to these challenges, we proposed a timing model to evaluate timing objectives through five systematic computational phases.

1) Load Capacitance Calculation: The load capacitance calculation is non-trivial, as the insertion of a buffer partitions the net, substantially impacting the overall load distribution. Also, the downstream gate size impacts the upstream load.

We propagate the load from sinks to sources through topological traversal. For each buffer candidate node u in the interconnect tree:

a) Capacitance Composition: The self capacitance for node u combines its own wire capacitance and gate input capacitance (only when u is a net sink node, i.e., the input pin of a downstream gate g):

$$C_u = C_u^{\text{wire}} + \begin{cases} C_{\text{in},u}^{\text{gate } g}(\mathbf{s}_g) & \text{if } u \text{ is gate } g \text{ input pin} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where the downstream gate g input pin u capacitance $C_{\text{in},u}^{\text{gate } g}(\mathbf{s}_g)$ uses linear interpolation between discrete sizes for differentiable gradient calculation:

$$C_{\text{in},u}^{\text{gate } g}(\mathbf{s}_g) = (1-\eta_g)C_{\text{in},u}^{\text{gate } g}(k) + \eta_g C_{\text{in},u}^{\text{gate } g}(k+1) \tag{7}$$

where $k = \lfloor \mathbf{s}_g \rfloor$[1] is the lower bound of the gate size and $\eta_g = \mathbf{s}_g - k$ is the fractional part of the gate size. In this way, the gradient of $C_{\text{in},u}^{\text{gate } g}(\mathbf{s}_g)$ with respect to $\mathbf{s}_g$ can be easily calculated, so does the gradient of C_u with respect to $\mathbf{s}_g$.

In the following equations, we also ensure that all timing calculation equations are differentiable and ignore gradient derivation for simplicity.

b) Load Recursion: The buffer candidate node u output pin load aggregates downstream capacitances:

$$L_u^{out} = C_u + \sum\nolimits_{v \in \text{children}(u)} L_v^{in} \tag{8}$$

Node u input pin load depends on buffer insertion $\mathbf{b}_u = 1$ or not $\mathbf{b}_u = 0$, as shown in the upper part of Figure 4.

$$L_u^{in} = \mathbf{b}_u \tilde{C}_{\text{in}}^{\text{buf}}(\mathbf{bs}_u) + (1-\mathbf{b}_u) L_u^{out} \tag{9}$$

[1]If $\mathbf{s}_g$ is already the largest size option, $k+1$ goes beyond the reasonable range. Therefore, in this paper, the maximal size interpolation is between the largest and the second largest size. Also, if there is only one size option, no interpolation is needed.

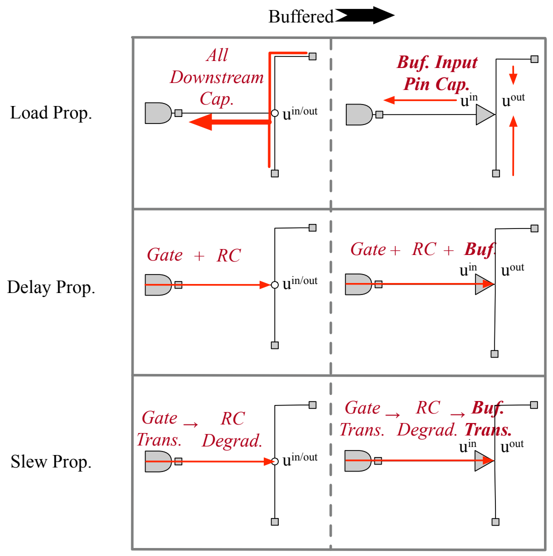


Fig. 4. The timing propagation through the interconnect net and gate.

where $\tilde{C}_{\text{in}}^{\text{buf}}$ applies similar linear interpolation for buffer size $\mathbf{bs}_u$. This equation also guarantees the differentiability of L_u^{in} with respect to $\mathbf{b}_u$ and $\mathbf{bs}_u$.

As shown in the upper part of Figure 4, if buffer u is not inserted ($\mathbf{b}_u = 0$), the input load L_u^{in} combines all of the downstream capacitances. If buffer u is inserted ($\mathbf{b}_u = 1$), the input load L_u^{in} is the input capacitance of buffer $\tilde{C}_{\text{in}}^{\text{buf}}(\mathbf{bs}_u)$.

2) Wire Elmore Delay Calculation: With each node's load computed, the Elmore delay from node u to its child node v is calculated as follows:

$$\Delta_{u \to v} = R_{u \to v} \cdot L_v^{in} \tag{10}$$

where $R_{u \to v}$ is the interconnect resistance between u and v. The Elmore delay $\Delta_{u \to v}$ is the product of the interconnect resistance $R_{u \to v}$ and the input load L_v^{in} at node v.

3) Net Signal Propagation: As buffer insertion might partition one net into several pieces, net signal propagation is also highly non-trivial. For each net, consider a net segment $u \to v$ with node v buffer presence $\mathbf{b}_v$ and buffer size $\mathbf{bs}_v$. The arrival time propagation from u to v is calculated as:

$$\begin{aligned} a_v^{in} &= a_u^{out} + \Delta_{u \to v} \\ a_v^{out} &= a_v^{in} + \mathbf{b}_v \cdot \text{LUT}_{\text{delay}}^{\text{buf}}(\text{Slew}_v^{in}, L_v^{out}, \mathbf{bs}_v) \end{aligned} \tag{11}$$

where the first equation computes input arrival time at v using Elmore wire RC delay $\Delta_{u \to v}$, and the second adds up the extra buffer delay $\text{LUT}_{\text{delay}}^{\text{buf}}$ at the buffer output if buffer inserted ($\mathbf{b}_v = 1$), as shown in the middle part of Figure 4. The LUT function also adopts linear interpolation [9] method to guarantee the differentiability of a_v^{out} with respect to $\mathbf{b}_v$, $\mathbf{bs}_v$, Slew_v^{in}, and L_v^{out}.

Slew propagation through the same net segment combines RC slew degradation and potential buffer slew transformation:

$$\begin{aligned} \text{Slew}_v^{in} &= \sqrt{\text{Slew}_u^{out\,2} + (\ln 10 \cdot \Delta_{u \to v})^2} \\ \text{Slew}_v^{out} &= (1-\mathbf{b}_v)\text{Slew}_v^{in} + \mathbf{b}_v \cdot \text{LUT}_{\text{slew}}^{\text{buf}}(\text{Slew}_v^{in}, L_v^{out}, \mathbf{bs}_v) \end{aligned} \tag{12}$$

The first equation models how input slew degrades passing through a wire $u \to v$, while the second handles buffer-induced slew transformation through $\text{LUT}^{\text{buf}}_{\text{slew}}$. If $\mathbf{b}_v = 1$, the output slew is determined by the buffer's output slew. If $\mathbf{b}_v = 0$, the slew will keep the RC degraded slew Slew^{in}_v. The slew propagation is shown in the lower part of Figure 4.

4) Gate Signal Propagation: Apart from interconnect nets propagation, for gate-level propagation, we handle gates through worst-case analysis. Each gate g with output v and inputs $u \in \text{fanin}(v)$ exhibits:

$$\begin{aligned}\Delta_{u\to v} &= \text{LUT}^{\text{gate }g}_{\text{delay}}(\text{Slew}^{in}_u, L^{out}_v, \mathbf{s}_g)\\ a^{out}_v &= \max_u \left(a^{in}_u + \Delta_{u\to v}\right)\\ \text{Slew}^{out}_v &= \max_u \left(\text{LUT}^{\text{gate }g}_{\text{slew}}(\text{Slew}^{in}_u, L^{out}_v, \mathbf{s}_g)\right)\end{aligned} \tag{13}$$

The max operations ensure timing conservatism by tracking the latest-arriving signal and the worst-case output slew. Gate lookup tables LUT^{gate} model input-to-output delay/slew dependencies on input slew, output load, and gate size $\mathbf{s}_g$, preserving technology-specific nonlinearities. The gate delay and slew transformations are also differentiable, as the max operation allows for gradient propagation.

5) Timing Endpoint Calculation: The timing analysis concludes with the analysis of timing endpoints. For each timing endpoint i synchronized by clock period T_{clk} with setup time T_{setup}, the timing margin is:

$$\text{Slack}_i = T_{\text{clk}} - T_{\text{setup}} - a_i \tag{14}$$

where a_i denotes the worst-case arrival time propagated to endpoint i. This formulation directly measures the safety margin between actual signal arrival and the clock edge, with positive slack indicating timing compliance and negative slack representing violations.

The global timing objective aggregates constraint violations through total negative slack (TNS):

$$\text{TNS} = \sum_{i\in\mathcal{E}} \min(0, \text{Slack}_i) \tag{15}$$

where $\mathcal{E}$ denotes the set of all timing endpoints. The TNS metric provides differentiable guidance for optimization by penalizing only violating paths, while satisfying endpoints contribute zero gradient. This formulation enables the efficient allocation of optimization resources to critical paths without over-constraining non-critical regions.

In summary, the timing objective calculation includes five different phases: net load calculation, wire Elmore delay calculation, net signal propagation, gate signal propagation, and timing endpoint calculation. The first, second, and fifth phases are performed only once in each iteration. In contrast, the third and fourth phases are iteratively performed in circuit topological order, from timing start points to end points. Each phase differentiates its outputs from its inputs, allowing the gradients to be backpropagated throughout the whole timing analysis process. Therefore, our proposed timing model integrates variables for buffer presence, buffer size, and gate size, while maintaining precision, differentiability, and efficiency.

C. Power Objective

Power optimization has become more crucial for modern low-power VLSI designs. Our approach carefully handles both sizing and buffer insertion through three key components:

a) Leakage Power Calculation: Leakage power is decomposed into gate and buffer components:

$$\begin{aligned}P_{\text{gate}}(\mathbf{s}) &= \sum_{g\in\mathcal{G}} \text{Leak}(g, \mathbf{s}_g)\\ P_{\text{buf}}(\mathbf{b}, \mathbf{bs}) &= \sum_{u\in\mathcal{B}_{\text{cand}}} \mathbf{b}_u \cdot \text{Leak}(\text{buf}, \mathbf{bs}_u)\end{aligned} \tag{16}$$

where $\mathcal{G}$ denotes original gates, $\mathcal{B}$ represents candidate buffer locations, and $\text{Leak}(\cdot)$ maps device sizes to library-characterized leakage values.

b) Unified Power Objective: The total leakage power aggregates both components:

$$P_{\text{total}} = P_{\text{gate}}(\mathbf{s}) + P_{\text{buf}}(\mathbf{b}, \mathbf{bs}) \tag{17}$$

c) Gradient Computation: Assuming the continuous relaxation through linear interpolation between discrete library sizes, the leakage gradients become:

$$\begin{aligned}\frac{\partial P}{\partial \mathbf{s}_g} &= \text{Leak}(g, \lfloor \mathbf{s}_g \rfloor + 1) - \text{Leak}(g, \lfloor \mathbf{s}_g \rfloor)\\ \frac{\partial P}{\partial \mathbf{b}_u} &= \text{Leak}(\text{buf}, \mathbf{bs}_u)\\ \frac{\partial P}{\partial \mathbf{bs}_u} &= \mathbf{b}_u \left[\text{Leak}(\text{buf}, \lfloor \mathbf{bs}_u \rfloor + 1) - \text{Leak}(\text{buf}, \lfloor \mathbf{bs}_u \rfloor)\right]\end{aligned} \tag{18}$$

In this way, we have gradients guiding the low-power optimization. Therefore, our model can balance the timing and power objectives. The buffer insertion and gate upsizing with low power consumption can be prioritized.

D. Physical-Aware Objective

The physical-aware objective prevents excessive area inflation or usage caused by upsizing, while discouraging buffer insertion at illegal or congested locations. Unlike prior works, which often neglect these physical constraints due to the significant computational overhead of analyzing the impacts of gate sizing and buffer insertion on physical layout, our approach efficiently integrates these considerations through differentiable spatial and congestion-aware modeling. Our method introduces a density penalty that jointly handles both gate and buffer area impacts through three key components:

a) Electrostatic Density Model: We model gates and buffers as charged particles in an electrostatic system, where the density penalty corresponds to potential energy. For any device d (gate or buffer) at location (x_d, y_d) with area Area_d, the density contribution is:

$$D_d = \text{Area}_d \cdot \Phi(x_d, y_d) \tag{19}$$

where $\Phi(x, y) \in [0, 1]$ is the normalized electric potential reflecting local placement density, computed from gate locations.

b) Device Area Modeling: Device areas vary with discrete size options. We implement continuous relaxation through linear interpolation between adjacent sizes:

$$\text{Area}(\mathbf{s}_d) = \text{Area}(\lfloor \mathbf{s}_d \rfloor)(\lceil \mathbf{s}_d \rceil - \mathbf{s}_d) + \text{Area}(\lfloor \mathbf{s}_d \rfloor + 1)(\mathbf{s}_d - \lfloor \mathbf{s}_d \rfloor) \tag{20}$$

where $\mathbf{s}_d$ denotes the continuous size parameter for device d.

c) Unified Density Gradient: The total density penalty aggregates all devices:

$$D_{\text{total}} = \sum_{g \in \mathcal{G}} D_g + \sum_{u \in \mathcal{B}_{\text{cand}}} \mathbf{b}_u D_u \tag{21}$$

where $\mathcal{G}$ is the set of gates and $\mathcal{B}_{\text{cand}}$ is potential buffer locations.

The density gradients with respect to optimization parameters are:

$$\begin{aligned}
\frac{\partial D}{\partial \mathbf{s}_g} &= [\text{Area}_g(\lfloor \mathbf{s}_g \rfloor + 1) - \text{Area}_g(\lfloor \mathbf{s}_g \rfloor)]\Phi(x_g, y_g) \\
\frac{\partial D}{\partial \mathbf{b}_u} &= \text{Area}_u \cdot \Phi(x_u, y_u) \\
\frac{\partial D}{\partial \mathbf{bs}_u} &= \mathbf{b}_u[\text{Area}_{\text{buf}}(\lfloor \mathbf{bs}_u \rfloor + 1) - \text{Area}_{\text{buf}}(\lfloor \mathbf{bs}_u \rfloor)]\Phi(x_u, y_u)
\end{aligned} \tag{22}$$

This formulation discourages gate scaling in dense regions ($\Phi \to 1$) while permitting modifications in sparse regions ($\Phi \to 0$). The differentiable model enables simultaneous optimization of timing without compromising physical feasibility.

E. Unified Optimization Algorithm

This section processes gradients $\nabla \mathbf{s}_g$, $\nabla \mathbf{b}_u$, and $\nabla \mathbf{bs}_u$ derived from backward gradient propagation of the previous three objectives to simultaneously determine: (1) gate sizing, (2) buffer insertion/removal, and (3) buffer resizing. A key challenge arises from the discrete nature of buffer and sizing decisions, which conflicts with the continuous gradient-based formulation. Prior gradient-based methods (e.g., [9]) face fundamental limitations in addressing discrete problems:

First, the continuous relaxation of discrete variables leads to unrealistic intermediate states. Standard gradient descent may produce buffer presence values $\mathbf{b}_u \in (0, 1)$ that neither represent true insertion ($\mathbf{b}_u = 1$) nor removal ($\mathbf{b}_u = 0$), ultimately leading to the unreliability of objective calculation and degrading solution quality. This becomes particularly problematic when combined with:

Second, using raw gradients only provides limited guidance for the optimization. For instance, high fanout nets appearing on multiple critical paths may accumulate disproportionately large gradients, triggering excessive buffer insertion that starves less critical but still timing-sensitive nets.

To address these challenges, we develop a balanced co-optimization algorithm with two key features: 1) Discrete-Aware Gradient Processing: Specialized gradient projection techniques maintain feasible discrete states throughout optimization while preserving gradient guidance. 2) Adaptive Resource Allocation: Dynamic weighting adjusts optimization priorities based on a soft quota to prevent resource starvation.

The complete algorithm proceeds through four phases described in the following sections, each maintaining the discrete nature of decisions while enabling global optimization through careful gradient utilization. The detailed algorithm is shown in Algorithm 1. Description of each phase is as follows:

Algorithm 1 Balanced and Adaptive Optimization Algorithm

```
 1: $\mathbf{s}_g \leftarrow 1,\ \forall g \in \mathcal{G}$                          ▷ Gate Minimal Size
 2: $\mathbf{b}_u \leftarrow 0,\ \mathbf{bs}_u \leftarrow 4,\ \forall u \in \mathcal{B}_{\text{cand}}$  ▷ BUFx4 as Default Size
 3: $\text{quota}_n \leftarrow \max(1, 0.1|\mathcal{B}_n|),\ \forall n \in \mathcal{N}$   ▷ Net-Level Soft Quota
 4: for iter $\leftarrow 1$ to $\text{Iter}_{\max}$ do
 5:     Compute objectives $(\text{TNS}, P_{\text{total}}, D_{\text{total}})$
 6:     Compute gradients $(\nabla \mathbf{s}, \nabla \mathbf{b}, \nabla \mathbf{bs})$
                                                      ▷ Adaptive Gradient Scaling
 7:     for net $n \in \mathcal{N}$ parallel do
 8:         $k_n \leftarrow \sum_{u \in \mathcal{B}_n} \mathbf{b}_u$
 9:         $\gamma_n \leftarrow \exp(-\max(0, k_n - \text{quota}_n))$
10:         $\nabla \mathbf{b}_u \leftarrow \gamma_n \nabla \mathbf{b}_u,\ \forall u \in \mathcal{B}_n$
11:     end for
                                                      ▷ Gate Sizing
12:     $\mathcal{G}^+ \leftarrow \{g | \mathbf{s}_g < |\mathcal{S}_g| \wedge \nabla \mathbf{s}_g < 0\}$
13:     Upsize smallest 1% gradient gates in $\mathcal{G}^+$: $\mathbf{s}_g \leftarrow \mathbf{s}_g + 1$
14:     if iter $\geq 100$ then
15:         $\mathcal{G}^- \leftarrow \{g | \mathbf{s}_g > 1 \wedge \nabla \mathbf{s}_g > 0\}$
16:         Downsize largest 1% gradients gates in $\mathcal{G}^-$: $\mathbf{s}_g \leftarrow \mathbf{s}_g - 1$
17:     end if
                                                      ▷ Buffer Insertion
18:     $\mathcal{B}^+ \leftarrow \emptyset,\ \mathcal{B}^- \leftarrow \emptyset$
19:     for net $n \in \mathcal{N}$ parallel do
20:         $u_n^* \leftarrow \arg\max_{u \in \mathcal{B}_n} \nabla \mathbf{b}_u$
21:         if $\nabla \mathbf{b}_{u_n^*} < 0$ then
22:             $\mathcal{B}^+ \leftarrow \mathcal{B}^+ \cup \{u_n^*\}$
23:         end if
24:     end for
25:     Insert smallest 1% gradients buffers in $\mathcal{B}^+$: $\mathbf{b}_u \leftarrow 1$
26:     if iter $\geq 100$ then
27:         $\mathcal{B}^- \leftarrow \{u | \nabla \mathbf{b}_u < 0 \wedge \mathbf{b}_u = 1\}$
28:         Remove largest 1% gradients buffers in $\mathcal{B}^-$: $\mathbf{b}_u \leftarrow 0$
29:     end if
                                                      ▷ Buffer Resize
30:     $\mathcal{R}^+ \leftarrow \{u | \mathbf{b}_u = 1 | \mathbf{bs}_u < |\mathcal{S}_b| \wedge \nabla \mathbf{bs}_u < 0\}$
31:     $\mathcal{R}^- \leftarrow \{u | \mathbf{b}_u = 1 | \mathbf{bs}_u > 1 \wedge \nabla \mathbf{bs}_u > 0\}$
32:     Upsize smallest 5% gradient buffers in $\mathcal{R}^+$: $\mathbf{bs}_u \leftarrow \mathbf{bs}_u + 1$
33:     Downsize largest 5% gradient buffers in $\mathcal{R}^-$: $\mathbf{bs}_u \leftarrow \mathbf{bs}_u - 1$
34:     Break if Convergence
35: end for
```

Initialization establishes conservative starting points to prevent premature over-optimization. All gates initialize at minimum drive strength ($\mathbf{s}_g = 1$) to avoid false critical paths from oversized drivers. Buffer candidates $\mathcal{B}_{\text{cand}}$ initialize uninserted ($\mathbf{b}_u = 0$) with median default size $\mathbf{bs}_u = 4$ to balance load-driving capability and power preservation. The $4\times$ default buffer size addresses typical critical net requirements while maintaining both further upsizing and downsizing flexibility. Apart from that, we initialize net-level buffer quotas (Algorithm 1 line 3) to prevent local over-insertion.

Adaptive Gradient Scaling manages net-level resource competition through exponential decay:

$$\gamma_n = \exp\left(-\max(0, k_n - \text{quota}_n)\right) \tag{23}$$

where $k_n = \sum_{u \in \mathcal{B}_n} \mathbf{b}_u$ counts existing buffers for net n. Gradients scale as $\nabla \mathbf{b}_u \leftarrow \gamma_n \nabla \mathbf{b}_u$, dynamically suppressing insertions when $k_n > \text{quota}_n$ while permitting gradient-driven buffer insertion below quota.

Gate Sizing implements discrete gate size transitions between drive strengths. We partition gates into:

- $\mathcal{G}^+$: Non-maximal sizes with $\nabla \mathbf{s}_g < 0$ (upsizing candidates)
- $\mathcal{G}^-$: Non-minimal sizes with $\nabla \mathbf{s}_g > 0$ (downsizing candidates, activated post 100 iterations)

Updates follow strict thresholds - only gates with top-1% gradients in each category modify sizes. The 100-iteration delay for downsizing releases early over-sizing resources while preserving stability through 1% update rates.

Buffer Insertion/Removal handles buffer presence task. No more than one buffer is inserted per net per iteration. That is because spatially adjacent buffer candidates may have similar gradients, leading to potential multiple adjacent insertions in the same net. Therefore, for each net n, we first identify the most critical candidate:

$$u_n^* = \arg\min_{u \in \mathcal{B}_n} \nabla \mathbf{b}_u \tag{24}$$

The insertion set $\mathcal{B}^+$ contains all candidates with negative gradients:

$$\mathcal{B}^+ = \{u_n^* | \nabla \mathbf{b}_{u_n^*} < 0\} \tag{25}$$

From $\mathcal{B}^+$, we select the top 1% buffers with the strongest negative gradients for insertion. After 100 iterations, we activate removal for buffers with positive gradients:

$$\mathcal{B}^- = \{u \in \mathcal{B}_{\text{inserted}} | \nabla \mathbf{b}_u > 0\} \tag{26}$$

where $\mathcal{B}_{\text{inserted}}$ denotes currently inserted buffers. Removal follows the same top 1% selection on $\mathcal{B}^-$.

Buffer Resizing follows a similar upsizing and downsizing candidates selection method:

$$\begin{aligned} \mathcal{R}^+ &= \{u | \nabla \mathbf{bs}_u < 0, \mathbf{bs}_u < \mathbf{bs}^{\max}\} \quad \text{(upsizing candidates)} \\ \mathcal{R}^- &= \{u | \nabla \mathbf{bs}_u > 0, \mathbf{bs}_u > \mathbf{bs}^{\min}\} \quad \text{(downsizing candidates)} \end{aligned} \tag{27}$$

But buffer resizing employs wider 5% adjustment bands than 1% gate sizing in each iteration. The wider resizing range is due to the VLSI library providing more buffer size options than general gates, which increases the likelihood of resizing. Additionally, buffer insertion typically occurs on the critical path, where multiple adjustments take place.

Our algorithm resolves three key limitations of prior gradient methods: 1) Hard discrete constraints via discrete state transitions 2) Net-level resource balancing through γ_n-scaled gradients 3) Phased optimization with late-stage buffer removal and gate downsizing to avoid over-optimization and balance resource allocation.

Finally, convergence occurs when total negative slack (TNS) improvement plateaus.

TABLE II
OUR BENCHMARK STATISTICS.

Design	#Gate	WNS (ns)	TNS (ns)	Power (mW)
NV_NVDLA_partition_m	27,553	-1.15	-228.05	0.01
NV_NVDLA_partition_p	79,919	-0.29	-72.419	0.07
mempool_tile_wrap	145,776	-1.67	-20710.20	0.04
aes_256	187,851	-0.22	-87.60	0.32
ariane136	278,465	-0.38	-839.67	0.54
NV_NVDLA_partition_a	38,089	-1.07	-1136.05	0.03
ariane133	149,396	-0.56	-1279.20	0.51
NV_NVDLA_partition_c	184,863	-0.80	-12304.10	0.22
NV_NVDLA_partition_o	260,483	-2.99	-22094.52	0.10
aes_192	283,750	-0.26	-59.70	0.37

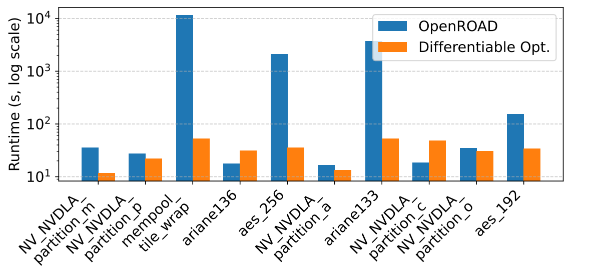


Fig. 5. Runtime (in log scale) of OpenROAD [12] method and our method.

IV. Experimental Results

A. Experimental Setup

We implement our unified framework for gate sizing and buffer insertion using Python, C++, and CUDA kernels. Our experiments are conducted on a server equipped with a 48-core Intel Xeon Silver 4124 CPU at 2.20 GHz, 256GB RAM, and an Nvidia A6000 GPU with 48GB of memory. We obtain 10 cases from the ICCAD'24 CAD contest [14] and perform a few necessary operations. Those cases are first synthesized using the ASAP7 [44] standard cell library without inserting buffers in this early stage, and further placement is performed by the OpenROAD [12] flow. Regarding timing and power metrics reporting, we utilize a widely used commercial tool, Innovus [45], to ensure accuracy. Our benchmark statistics and relevant initial metrics are presented in Table II. These benchmarks vary in scale, complexity, and timing performance, reflecting a range of real-world design scenarios.

B. Results and Analysis

For our evaluations, we compare three different methods: end-to-end OpenROAD [12] flow, OpenROAD buffering followed by the latest differentiable-based gate sizer [9], and our proposed differentiable physical optimization method. Following each of these three methods, we conduct detailed placement for legalization and early global routing before reporting the timing and total power metrics from Innovus [45] to ensure the accuracy and reliability of our evaluation.

The results are summarized in Table III. Compared to end-to-end OpenROAD [12] flow, our differentiable physical optimization framework achieves 23% improvement in TNS and 12% improvement in WNS, while only a 3% power increase. The slight power increase is reasonable since timing

TABLE III
COMPARISON OF TIMING AND POWER METRICS ACROSS OPTIMIZATION FLOWS.

Case Name	OpenROAD Buffering + Sizing			OpenROAD Buffering + Sizer [9]			Differentiable Physical Opt.		
	TNS (ns)	WNS (ns)	Power (mW)	TNS (ns)	WNS (ns)	Power (mW)	TNS (ns)	WNS (ns)	Power (mW)
NV_NVDLA_partition_m	-13.05	-0.27	**0.01**	-11.99	-0.30	**0.01**	**-11.92**	**-0.27**	**0.01**
NV_NVDLA_partition_p	-61.13	-0.23	**0.08**	-43.17	**-0.20**	**0.08**	**-39.05**	-0.22	**0.08**
mempool_tile_wrap	-21051.10	-1.66	**0.05**	-18423.10	-1.62	0.06	**-17661.10**	**-1.56**	0.06
ariane136	**0.00**	**0.00**	0.37	**0.00**	**0.00**	**0.33**	**0.00**	**0.00**	0.35
aes_256	-631.92	-0.58	**0.55**	-532.52	-0.32	0.57	**-409.44**	**-0.24**	0.58
NV_NVDLA_partition_a	**0.00**	**0.00**	**0.03**	**0.00**	**0.00**	**0.03**	**0.00**	**0.00**	**0.03**
ariane133	-554.41	-0.36	**0.52**	-551.60	-0.35	0.53	**-545.05**	**-0.35**	0.56
NV_NVDLA_partition_c	-12304.00	-0.80	**0.22**	-11245.80	-0.80	0.23	**-11185.10**	**-0.79**	0.23
NV_NVDLA_partition_o	**0.00**	**0.00**	**0.10**	**0.00**	**0.00**	**0.10**	**0.00**	**0.00**	**0.10**
aes_192	-54.90	-0.17	0.47	-53.27	-0.16	0.47	**-12.01**	**-0.13**	**0.46**
Norm. Avg	1.00	1.00	**1.00**	0.90	0.93	1.02	**0.77**	**0.88**	1.03

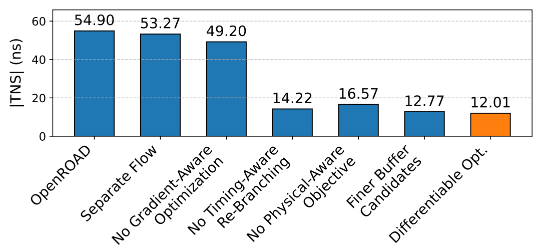


Fig. 6. Ablation study of our method based on the largest case (aes_192).

closure is the primary goal of our optimization, while power consumption is less important. We also compare our method with a separate OpenROAD buffering and the latest sizer [9] flow. Although this separate method outperforms end-to-end OpenROAD flow, it does not offer a better solution than our proposed method. As pointed out in Section I, the separate optimization of gate sizing and buffer insertion may get stuck in local optima for both gate sizing and buffer insertion, thus failing to unlock the mutual optimization space.

For runtime performance, as shown in Figure 5, our GPU-based method achieves a speedup of 30× compared to the OpenROAD [12] flow. It is also worth noting that our method has consistent runtime performance across benchmarks of different sizes. It highlights the scalability of our method and its potential for large-scale industrial designs. In contrast, the OpenROAD flow runtime is highly correlated with the design size and the severity of the original timing performance.

C. Ablation Study

To further validate the effectiveness of our method, we conduct an ablation study to evaluate the impact of each component in our framework. For this study, we select the largest case (aes_192) and compare the results of our method with and without each component.

As shown in Figure 6, the traditional or separate flow has the worst performance, highlighting the importance of simultaneous optimization. The discrete-aware gradient-based method is crucial for achieving optimal performance in our simultaneous optimization framework. The performance degrades significantly once we replace it with a simple gradient descent method followed by a rounding operation. This is predictable as relaxing discrete buffer presence variables to continuous variables will insert "semi-buffers" that mislead the optimization. In addition, the timing-aware buffer tree re-branching technique also contributes to further enhancing performance. With the timing-critical sink pins taking priority, our buffer tree skeleton unlocks better timing performance. Finally, the physical-aware objective is also important for achieving the best performance. Since oversizing, over-buffering, or infeasible buffer locations lead to local congestion or unreasonable buffer locations, the performance will degrade once the detailed placer displaces and legalizes these components.

In our ablation study, we also attempted finer-grained buffer candidate locations, but this did not substantially improve performance. This is because the slight displacement of buffer locations will not make a big difference. Therefore, although the finer-grained buffer tends to include better buffer locations, the performance improvement is limited.

V. Conclusion

In this paper, we develop a pioneering differentiable physical optimization framework that integrates gate sizing and buffer insertion. Based on differentiable optimization techniques, we successfully unify two physical optimization tasks via a rigorous mathematical formulation. Benefiting from GPU-acceleration techniques, our method achieves a remarkable speedup of 30× compared to the traditional CPU-based flow. To further improve the timing performance, we propose a timing-aware buffer tree skeleton re-branching method that unlocks better solutions. We also introduce a physical-aware objective to enhance the feasibility of our solutions upon the original placement layout. Our discrete-aware gradient-based optimization method effectively handles the strong discrete nature of gate sizing and buffer insertion. We demonstrate the effectiveness of our framework through extensive experiments, showing that our method significantly outperforms existing methods with a 23% improvement in total negative slack (TNS) and a 12% improvement in worst negative slack (WNS) while maintaining similar power consumption. Our future work includes even more steps, such as placement or even global routing, within this framework. We also plan to explore more extensive optimization strategies, such as pin swapping and logic resynthesis, to enhance the performance of our framework further. We strongly believe that our work will inspire and stimulate further research into PPA optimization in the VLSI design flow from brand new perspectives.

Acknowledgment

This work was supported in part by the National Science Foundation of China (Grant No. T2293701), the Natural Science Foundation of Beijing, China (Grant No. Z230002), and the 111 Project (B18001).

References

[1] C. Alpert *et al.*, "Simultaneous driver sizing and buffer insertion using a delay penalty estimation technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 1, pp. 136–141, 2004.

[2] H. Wu *et al.*, "Aito: Simultaneous gate sizing and buffer insertion for timing optimization with gnns and rl," *Integration*, vol. 98, p. 102211, 2024.

[3] A. Murugavel and N. Ranganathan, "Gate sizing and buffer insertion using economic models for power optimization," in *17th International Conference on VLSI Design. Proceedings.*, 2004, pp. 195–200.

[4] A. Sharma *et al.*, "Fast lagrangian relaxation-based multithreaded gate sizing using simple timing calibrations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 7, pp. 1456–1469, 2020.

[5] S. Daboul *et al.*, "Provably fast and near-optimum gate sizing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3163–3176, 2018.

[6] G. Flach *et al.*, "Effective method for simultaneous gate sizing and v th assignment using lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 4, pp. 546–557, 2014.

[7] Y. Du *et al.*, "Addressing continuity and expressivity limitations in differentiable physical optimization: A case study in gate sizing," in *2025 International Symposium of EDA (ISEDA)*, 2025.

[8] J. Hu *et al.*, "Sensitivity-guided metaheuristics for accurate discrete gate sizing," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 233–239.

[9] Y. Du *et al.*, "Fusion of global placement and gate sizing with differentiable optimization," in *2024 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024.

[10] K. Lowe and P. Gulak, "A joint gate sizing and buffer insertion method for optimizing delay and power in cmos and bicmos combinational logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 17, no. 5, pp. 419–434, 1998.

[11] L. van Ginneken, "Buffer placement in distributed rc-tree networks for minimal elmore delay," *IEEE International Symposium on Circuits and Systems*, pp. 865–868 vol.2, 1990. [Online]. Available: https://api.semanticscholar.org/CorpusID:109927706

[12] The-OpenROAD-Project, "Openroad," GitHub repository, 2024, available online: https://github.com/The-OpenROAD-Project/OpenROAD.

[13] W. Ning, "Strongly np-hard discrete gate-sizing problems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 8, pp. 1045–1051, 1994.

[14] B.-Y. Wu *et al.*, "2024 iccad cad contest problem c: Scalable logic gate sizing using ml techniques and gpu acceleration," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024.

[15] Y. Liu and J. Hu, "Gpu-based parallelization for fast circuit optimization," in *ACM/IEEE Design Automation Conference (DAC)*, 2009, pp. 943–946.

[16] S. Hu, M. Ketkar, and J. Hu, "Gate sizing for cell-library-based designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 6, pp. 818–825, 2009.

[17] Y. Liu and J. Hu, "A new algorithm for simultaneous gate sizing and threshold voltage assignment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 2, pp. 223–234, 2010.

[18] J. P. Fishburn, "Tilos: A posynomial programming approach to transistor sizing," *Proc. Int. Conf. On Computer-Aided Design*, vol. 33, no. 2, pp. 236–238, 2003.

[19] A. B. Kahng *et al.*, "High-performance gate sizing with a signoff timer," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '13. IEEE Press, 2013, p. 450–457.

[20] Y.-C. Lu *et al.*, "Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning," in *ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 733–738.

[21] S. Nath *et al.*, "Transsizer: A novel transformer-based fast gate sizer," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022, pp. 1–9.

[22] C.-K. Cheng *et al.*, "Dagsizer: A directed graph convolutional network approach to discrete gate sizing of vlsi graphs," *TODAES*, no. 4, 2023.

[23] P. Pham and J. Chung, "Agd: A learning-based optimization framework for eda and its application to gate sizing," in *ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[24] C.-P. Chen, C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 18, no. 7, pp. 1014–1025, 1999.

[25] ——, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 7, pp. 1014–1025, 1999.

[26] D. Chinnery and A. Sharma, "Integrating lr gate sizing in an industrial place-and-route flow," in *ACM International Symposium on Physical Design (ISPD)*, 2022, p. 39–48.

[27] M. M. Ozdal, S. Burns, and J. Hu, "Algorithms for gate sizing and device parameter selection for high-performance designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 31, no. 10, pp. 1558–1571, 2012.

[28] D. Mangiras, D. Chinnery, and G. Dimitrakopoulos, "Task-based parallel programming for gate sizing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 4, pp. 1309–1322, 2023.

[29] X. Zhou *et al.*, "Heterogeneous graph neural network-based imitation learning for gate sizing acceleration," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022, pp. 1–9.

[30] A. Sharma *et al.*, "Fast lagrangian relaxation based gate sizing using multi-threading," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 426–433.

[31] Y. Du *et al.*, "Fusion of global placement and gate sizing with differentiable optimization," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2024.

[32] Y. Ye *et al.*, "Learning-driven physically-aware large-scale circuit gate sizing," 2024. [Online]. Available: https://arxiv.org/abs/2403.08193

[33] Y.-C. Lu *et al.*, "Lego-size: Llm-enhanced gpu-optimized signoff-accurate differentiable vlsi gate sizing in advanced nodes," in *ACM International Symposium on Physical Design (ISPD)*, 2025.

[34] Y. Du *et al.*, "Powpredict: Cross-stage power prediction with circuit-transformation-aware learning," in *Proceedings of the 61st Annual Design Automation Conference 2024*. ACM, 2024.

[35] L. P. P. van Ginneken, "Buffer placement in distributed rc-tree networks for minimal elmore delay," 1990.

[36] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion for noise and delay optimization," in *Proc. ACM/IEEE DAC*, 1998.

[37] ——, "Buffer insertion with accurate gate and interconnect delay computation," in *Proc. ACM/IEEE DAC*, 1999.

[38] W. C. N. D. Sha, "Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer," Patent.

[39] W. Li *et al.*, "Dgr: Differentiable global router," in *ACM/IEEE Design Automation Conference (DAC)*, 2024.

[40] Z. Guo and Y. Lin, "Differentiable-timing-driven global placement," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, p. 1315–1320.

[41] G. Chen *et al.*, "Differentiable edge-based opc," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2024.

[42] J. Lu *et al.*, "eplace: Electrostatics-based placement using fast fourier transform and nesterov's method," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 2, mar 2015.

[43] X. Wang *et al.*, "Dasals: Differentiable architecture search-driven approximate logic synthesis," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[44] L. T. Clark *et al.*, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[45] "Cadence Innovus Implementation System," http://www.cadence.com.