Fusion of Global Placement and Gate Sizing with Differentiable Optimization

Yufan Du[1,2†], Zizheng Guo[2,3†], Yibo Lin[2,3,4*], Runsheng Wang[2,3,4], Ru Huang[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@stu.pku.edu.cn,{gzz,yibolin,r.wang,ruhuang}@pku.edu.cn

ABSTRACT

Gate sizing is critical in VLSI design because it significantly influences final design quality. Traditional design flows typically treat gate sizing as a separate step due to its discreteness nature. However, this approach not only undermines the optimization efforts of earlier stages like placement, but also restricts the exploration space for gate sizing. To address these challenges, we introduce an innovative design flow fusing gate sizing with the earlier global placement stage. Our method employs differentiable timing and leakage power objectives and leverages GPU-accelerated computation to enhance design quality directly and efficiently. Our experimental results demonstrate significant improvements in timing and power metrics, with an average improvement of 77.1% in total negative slack (TNS) and 43.5% in worst negative slack (WNS), and meanwhile achieving a reduction in leakage power consumption by 1% compared with one of the most popular design tools, OpenROAD. Our method can speedup the design process by up to 7×.

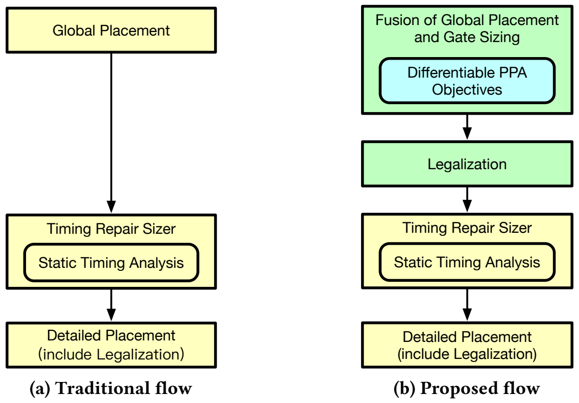

Global Placement
Timing Repair Sizer
Static Timing Analysis
Detailed Placement (include Legalization)
(a) Traditional flow
Fusion of Global Placement and Gate Sizing
Differentiable PPA Objectives
Legalization
Timing Repair Sizer
Static Timing Analysis
Detailed Placement (include Legalization)
(b) Proposed flow

Figure 1: Traditional design flow versus the proposed flow. Yellow boxes represent OpenROAD [3] steps, and green boxes represent our steps. For the completeness of the optimization, we also use OpenROAD sizer after our fusion framework because it includes additional optimization steps like pin swapping and buffer insertion for fair comparison.

1 INTRODUCTION

Gate sizing is critical for timing and power closure in VLSI design flow. It adjusts the drive strength of each gate within a circuit. It profoundly impacts the trade-offs between key factors like performance, power, and area (PPA) that determine the design's overall quality. With the increasing design complexity and the demand for high-performance and energy-efficient designs, gate sizing has become more challenging due to the NP-hard combinatorial optimization problem [1] for PPA trade-offs required in the large and discrete design space.

Complicating matters further, current methodologies typically explore gate sizing after the placement or routing is fixed. This approach introduces several notable drawbacks, including: (i) The exploration space is significantly constrained as the earlier stage outcomes are fixed, like gate positions from placement results. (ii) Adjustments to gate sizes will sabotage the optimization efforts during earlier stages since the resized gates may not fit the original placement or routing layout. (iii) The process is exceedingly time-consuming, often necessitating multiple iterations to achieve timing closure, thereby delaying the overall design cycles. For example, the cutting-edge commercial tool Innovus [2] takes several loops of timing violation fix, area reclaim, and placement refinement to optimize timing incrementally after the global placement stage. One of the most popular open-source design tools, OpenROAD [3], executes gate sizer and detailed placement separately as well.

Recent trends of design automation encourage a "shift-left" approach [4], suggesting that circuit constraints and performance should be considered in earlier stages of the design flow. This strategy is predicated on the premise that earlier stages can provide more flexibility and larger exploration space for holistic optimization. Gate sizing involves trade-offs between timing, power, and area, primarily dependent on the global placement step early in the design flow. As a result, the fusion of gate sizing and global placement has the potential to unlock a much larger search space and greatly unleash the design optimization opportunity. However, such a fusion is quite challenging. Global placement is formulated as a continuous optimization task, whereas gate sizing is discrete in nature. Modeling and optimizing the complex joint effects of placement and sizing on timing and power is also highly nontrivial.

This paper proposes a new design flow that fuses gate sizing with global placement, enabling simultaneous optimization of gate positions and gate sizes, as depicted in Figure 1. We carefully make optimization objective functions differentiable with respect to gate

† Equal contribution, * Corresponding author.

ICCAD '24, October 27–31, 2024, New York, NY, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1077-3/24/10…$15.00
https://doi.org/10.1145/3676536.3676670

positions and sizes. Therefore, jointly optimizing gate positions and sizes through the gradient descent method is adopted as a natural choice. We summarize our key contributions as follows.

- To the best of our knowledge, we propose the first framework that fuses the optimizations of gate positions and gate sizes with differentiable objectives.
- Our unified differentiable model for gate sizing and global placement leverages interpolation, gradient descent, and GPU-accelerated computation strategies to optimize timing and power objectives efficiently.
- We solve the discrete gate sizing problem by making discrete gate sizes continuous and leveraging the L1 loss function for discretization.
- Experimental results show marked improvements in timing and power metrics, with an average enhancement of 77.1% in total negative slack (TNS) and 43.5% in worst negative slack (WNS), while achieving a reduction in power consumption by 1% than OpenROAD [3] for the flow in Figure 1. Meanwhile, the proposed method can accelerate the design process by up to 7×.

We believe our work will inspire and stimulate further research into PPA optimization in the VLSI design flow from new perspectives. The rest of this paper is organized as follows. Section 2 gives the preliminaries and problem formulation; Section 3 explains the detailed framework; Section 4 demonstrates the results; Section 5 concludes the paper.

2 PRELIMINARIES

This section reviews the background of nonlinear global placement, gate sizing, and prior works in both fields.

2.1 Nonlinear Global Placement

Global placement determines the gate positions on the layout area. A widely-adpoted global placement method is nonlinear global placement [5–11]. The goal is to minimize the total wirelength under the constraints of gate density. Placement metrics such as half-perimeter wire length (HPWL) and layout density are expressed as functions of gate positions $\mathbf{x}, \mathbf{y}$. By treating gate positions as continuous variables and optimization objectives as continuous functions, nonlinear placement can be formulated as a continuous optimization problem. By transforming the density constraint into a density penalty term, the objective function of nonlinear global placement can be formulated as follows:

$$\min_{\mathbf{x},\mathbf{y}} \sum_{\text{net } e} WL(e;\mathbf{x},\mathbf{y}) + \lambda D(\mathbf{x},\mathbf{y}), \tag{1}$$

where $\mathbf{x}, \mathbf{y}$ denote gate positions and λ represents the density penalty weight. The function WL indicates the nets' HPWL, and D serves as the density penalty function. The optimization can be solved by the gradient descent method with gradually increasing λ to spread gates in the layout.

Recent developments in machine learning have provided a fresh perspective on this challenge. Prior work [11] draws an analogy between placement challenges and ML paradigms, as summarized in Table 1. In this context, the global placement problem is analogous to training a neural network, where the analytical function of the placement can be expressed, and its gradients with respect to gate positions can be efficiently computed. This analogy, coupled with

Table 1: The analogy between placement and ML problems [11].

Machine Learning	Placement
Train a neural network	Solve global placement
Neural network weights	Gate positions
Training data	Netlist
Loss function	Wirelength objective
Regularization	Density constraint

Table 2: Summary of notations.

Notation	Description
$\mathcal{G}$	Set of gates
$\mathcal{S}_g$	Set of available sizes for gate g
$\mathcal{E}$	Set of edges in the timing graph
$\mathcal{N}_{\text{end}}$	Set of end nodes in the timing graph
$\mathbf{s}$	Set of gate sizes
$\mathbf{s}_g$	Size for gate g
$\mathbf{x}, \mathbf{y}$	Set of gate positions
$d_{ij}(\mathbf{s}_g)$	Delay from node i to node j
a_i	Arrival time at node i
t	Clock period
t_{setup}	Setup time
$\text{Leak}_g(\mathbf{s}_g)$	Leakage power of gate g with size $\mathbf{s}_g$
$\text{Area}_g(\mathbf{s}_g)$	Area of gate g with size $\mathbf{s}_g$
$\text{Cap}_i(\mathbf{s}_g)$	Capacitance of the pin i with its corresponding gate size $\mathbf{s}_g$
$\text{Load}_i(\mathbf{s})$	Load capacitance at output pin i
$\text{Slew}_i(\mathbf{s})$	Transition time at input pin i
$WL(e;\mathbf{x},\mathbf{y})$	Wirelength of net e with gate positions $\mathbf{x}, \mathbf{y}$
$D(\mathbf{x},\mathbf{y})$	Density penalty with gate positions $\mathbf{x}, \mathbf{y}$

the capabilities of modern neural network training toolkits, makes applying GPU acceleration a natural choice.

Timing optimization is necessary to achieve timing closure. The literature has explored two categories of techniques that can be integrated into nonlinear global placement: net-based methods and path-based methods.

Net-based methods [12–17] dynamically adjust weights of different nets during placement based on their timing criticality. The net weighting strategy assigns a higher weight to critical nets, thus encouraging the placement algorithm to shrink those critical nets by moving surrounding gates and reducing their delays. Path-based methods formulate the relation between gate positions and timing objectives during placement stage [18], among which the differentiable-timing-driven placement approach [19, 20] has achieved significant success. They establish the forward computation paths from gate positions to timing objectives and make these objectives differentiable, thus enabling the backward gradient propagation from objectives to gate positions. Therefore, they achieve direct optimization of timing metrics with respect to gate positions by the gradient descent method.

2.2 Gate Sizing

Gate sizing determines the drive strength of each gate within a circuit to meet different constraints, such as max load capacitance, max signal transition time, and timing constraints. Specifically, to achieve timing closure, a timing-driven gate sizer (hereafter referred to as "sizer" for simplicity) can upsize gates in critical paths to mitigate the timing violations at the cost of increased power consumption. The primary objective of the sizer is to minimize a design's total leakage power while satisfying timing constraints. Utilizing the notations summarized in Table 2, the timing-driven gate sizing problem can be formulated as follows:

$$\begin{aligned} \min_{\mathbf{s},a} \quad & \sum_{g\in\mathcal{G}} \mathrm{Leak}_g(\mathbf{s}_g) & & \\ \text{subject to} \quad & \mathbf{s}_g \in \mathcal{S}_g, & \forall g \in \mathcal{G}, & \\ & a_i + d_{ij}(\mathbf{s}) \le a_j, & \forall (i,j) \in \mathcal{E}, & \\ & a_k \le t - t_{\mathrm{setup}}, & \forall k \in \mathcal{N}_{\mathrm{end}}, & \end{aligned} \tag{2}$$

where minimization is over the set of discrete gate variable $\mathbf{s}$ and arrival time variable a. There are two constraints in this formulation. Firstly, gate sizes $\mathbf{s}_g$ must be selected from the available sizes $\mathcal{S}_g$ for each gate g. Secondly, signal arrival time at the endpoint of each timing path should not violate setup time constraints.

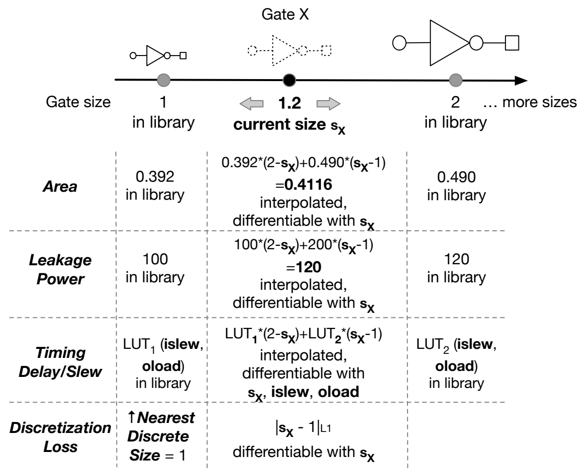
The existing timing-driven gate sizing methods can be broadly categorized into four types:

(1) Dynamic programming-based methods, such as [21–23]. Those works only achieve optimal solutions for tree-structured circuit topologies and have limitations with reconvergent paths.
(2) Sensitive-based methods. Works like [24–26] assess the sensitivity of each gate based on prior knowledge. Gate sizes are adjusted from the most sensitive ones until timing closure. This approach is entirely heuristic, with outcomes heavily reliant on the feasibility of the initial sensitivity knowledge.
(3) Learning-based methods. Works like the reinforcement learning-based methods [27], generative AI-based methods [28], graph convolutional methods [29, 30], and deep learning-based methods [31] stand out. While these methods employ the prevailing learning tricks, the performance of these data-driven models may be compromised once they are applied to other cell and timing libraries. Also, a huge amount of retraining time is unbearable for current fast-paced commercial design cycles.
(4) Heuristic methods improved by Lagrangian relaxation (LR)-based formulation [32–40], which have achieved remarkable success in the past decade. By relaxing the timing constraints in the objective function and employing the Karush-Kuhn-Tucker (KKT) optimality conditions, the search space can be greatly pruned. However, they still resort to heuristics and local search to derive a suboptimal solution, which can be slow on large designs due to the sequential nature of gate sizing adjustments. [39] introduced a learning-driven methodology that reduced the initial heuristic search space to accelerate the algorithm. [35, 37, 38] focused on enhancing the efficiency of these processes.

All the above studies regard gate sizing as an individual step and solve the sizing problem with gate positions fixed, which may limit the exploration space for optimization.

2.3 Problem Formulation

Problem. Given a set of gates and an initial placement layout, the objective is to minimize total leakage power and the absolute values of TNS and WNS by simultaneously determining gate positions $\mathbf{x}, \mathbf{y}$ and gate sizes $\mathbf{s}$.



Gate X
Gate size
1 in library
1.2 current size s_X
2 in library
… more sizes
Area
0.392 in library
0.392*(2-s_X)+0.490*(s_X-1) =0.4116 interpolated, differentiable with s_X
0.490 in library
Leakage Power
100 in library
100*(2-s_X)+200*(s_X-1) =120 interpolated, differentiable with s_X
120 in library
Timing Delay/Slew
LUT_1 (islew, oload) in library
LUT_1*(2-s_X)+LUT_2*(s_X-1) interpolated, differentiable with s_X, islew, oload
LUT_2 (islew, oload) in library
Discretization Loss
↑Nearest Discrete Size = 1
|s_X - 1|_L1 differentiable with s_X

Figure 2: Illustration of linear interpolation on different metrics and discretization loss.

3 METHODOLOGY

In this section, we present our novel differentiable framework for simultaneous global placement and gate sizing. Section 3.1 presents an overview of our differentiable optimization framework. Section 3.2, 3.3, and 3.4 give the mathematical details of leakage power, timing, and wirelength/density objectives, respectively. Section 3.5 demonstrates the complete flow with GPU acceleration. Section 3.6 details the parameter configuration and the ideas behind it.

3.1 Framework Overview

Our key idea in the fusion of global placement and gate sizing is to introduce gate sizes ($\mathbf{s}$) as a new set of *continuous* variables in the optimization formulation of nonlinear global placement, along with the original gate position variables ($\mathbf{x}$ and $\mathbf{y}$). In addition to the added variables, we also add several new optimization objectives including timing, power, and area, along with the original HPWL and density penalty terms. These objectives are now functions of both gate positions and sizes.

To optimize the sizes and positions simultaneously, we *make all objectives differentiable* by carefully analyzing their analytical formulation and deriving their gradients to both positions and sizes using back-propagation. With the available gradients, gradient descent can be utilized for optimization, which fits naturally into the nonlinear global placement flow.

The gate sizes of a digital circuit should come from a standard cell library with a set of discrete choices. To define a continuous size variable, we use interpolations between adjacent sizes (Figure 2) to define gate area, leakage power, and timing behavior. These interpolations are differentiable and can propagate gradients back to both 2 adjacent sizes. As the output of our fused placer-sizer should still be discrete sizes, we encourage the discretization of size variables by an additional L1 loss term that we call *discretization loss.* This loss pushes a size variable to its nearest discrete choice. Its weight will be gradually increased along with placement iterations.

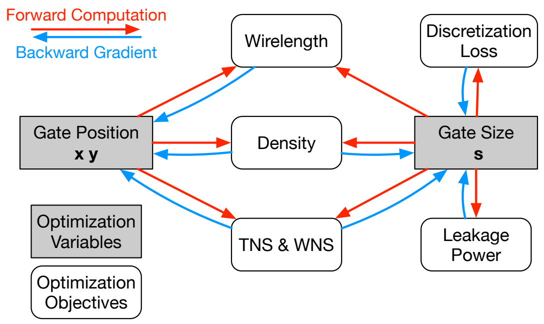


Figure 3: The formulation of our placement optimization problem considering gate sizes and PPA objectives.

The complete objective function of our placement problem is formulated as follows:

$$\begin{aligned}\min_{\mathbf{x},\mathbf{y},\mathbf{s}} \sum_{\text{net } e} & WL(e;\mathbf{x},\mathbf{y};\mathbf{s}) + \lambda_1 D(\mathbf{x},\mathbf{y};\mathbf{s}) \\ & + \lambda_2 \text{Leak}(\mathbf{s}) + \lambda_3 \text{L1Loss}(\mathbf{s}, \text{Round}(\mathbf{s})) \\ & + t_1 \text{WNS}(\mathbf{x},\mathbf{y};\mathbf{s}) + t_2 \text{TNS}(\mathbf{x},\mathbf{y};\mathbf{s}),\end{aligned} \quad (3)$$

where t_1, t_2, λ_1, λ_2 and λ_3 are weights for different objectives. WNS and TNS are represented by their absolute magnitudes in this formulation.

The dependency between 5 optimization objectives and 2 sets of variables is illustrated in Figure 3. The leakage power and discretization loss terms depend solely on and propagate gradients solely to gate sizes. The other 3 terms, wirelength, density, and timing, depend on both gate positions and sizes. In these 6 dependencies, we create 5 gradient channels for backward propagation except the wirelength–gate size channel because the effect of gate size on wirelength is quite minor compared to gate position.

In the following subsections, we will introduce the details of each objective, the overall GPU-accelerated framework, and parameter configuration.

3.2 Differentiable Leakage Power Objective

We begin with leakage power, which is the simplest objective. The total leakage power of a design is the sum of the leakage power of all gates. As a result, it only depends on the gate size $\mathbf{s}$. We express the leakage power of an interpolated gate (Figure 2) as follows:

$$\text{Leak}_g(\mathbf{s}_g) = \text{Leak}_g(\lfloor \mathbf{s}_g \rfloor)(\lceil \mathbf{s}_g \rceil - \mathbf{s}_g) + \text{Leak}_g(\lceil \mathbf{s}_g \rceil)(\mathbf{s}_g - \lfloor \mathbf{s}_g \rfloor). \quad (4)$$

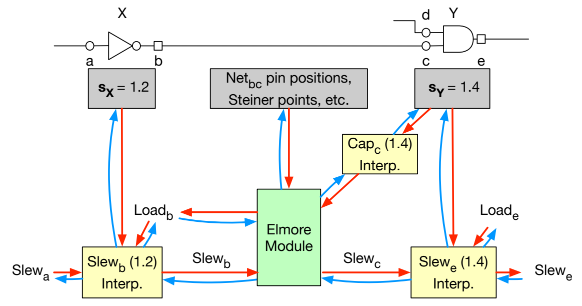
Taking the partial derivative of Equation (4) with respect to $\mathbf{s}_g$, we get

$$\frac{\partial}{\partial \mathbf{s}_g}\text{Leak}_g(\mathbf{s}_g) = \text{Leak}_g(\lceil \mathbf{s}_g \rceil) - \text{Leak}_g(\lfloor \mathbf{s}_g \rfloor).^{1} \quad (5)$$

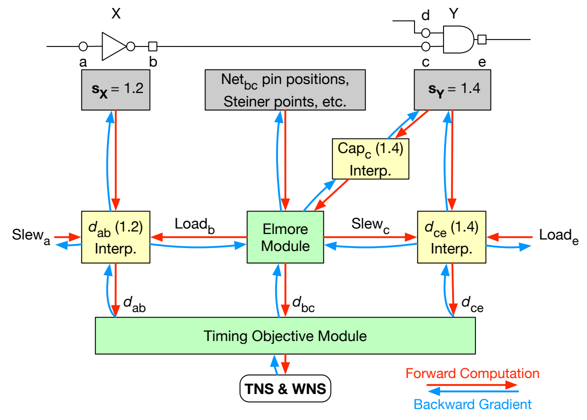
The total leakage power is the sum of gate leakage powers:

$$\text{Leak}(\mathbf{s}) = \sum_g \text{Leak}_g(\mathbf{s}_g). \quad (6)$$

[1]When $\mathbf{s}_g$ is a precise integer, this gradient might be zero. To avoid this, we redefine roundings in this paper as: $\lceil \mathbf{s}_g \rceil = \min(\mathbf{s}_{g\,\max}, \lceil \mathbf{s}_g + \epsilon \rceil)$. Here ϵ is a small constant ensuring the 2 roundings differ, and $\mathbf{s}_{g\,\max}$ is the largest available size for gate g. If the gate size is already the largest size, then $\lfloor \mathbf{s}_g \rfloor$ is $\max(1, \mathbf{s}_{g\,\max} - 1)$ and $\lceil \mathbf{s}_g \rceil$ is $\mathbf{s}_{g\,\max}$.



(a) Gradient flow in slew propagation.



(b) Gradient flow in delay propagation.

Figure 4: Gradient flows for differentiable timing objectives with respect to both gate sizes and locations. We split the flow into 2 figures due to its complexity.

Taking gradients of Equation (6) with respect to $\mathbf{s}$ and we get a vector of gradients:

$$\nabla_{\mathbf{s}}\text{Leak}(\mathbf{s}) = \left[\frac{\partial}{\partial \mathbf{s}_1}\text{Leak}_1(\mathbf{s}_1), \frac{\partial}{\partial \mathbf{s}_2}\text{Leak}_2(\mathbf{s}_2), \ldots\right]. \quad (7)$$

Each term can be calculated using Equation (5). The minimization of leakage power creates a trend to shrink the gate sizes.

3.3 Differentiable Timing Objectives

Contrary to leakage power, timing objectives are quite complex, especially when gate sizes are considered variables instead of constants. The gate positions $\mathbf{x}, \mathbf{y}$ determine the shape of interconnects between gates, thus determining the resistances and capacitances of the interconnect. This in turn affects the signal delay not only for interconnects, but also for cells because of the change in load. This dependency has been analyzed by Guo *et al* [19] to derive a differentiable timing-driven placement objective. In this work, gate sizes are considered variables as well, which makes deriving the gradients even more challenging.

We start with a brief review of major techniques introduced in [19] for completeness and then present our timing gradient calculation flow on top of them.

(1) *Differentiable Steiner Tree*: An early routing estimation using Steiner tree generation like FLUTE [41] is necessary for

timing analysis and optimization. The original FLUTE is not differentiable. To solve this, [19] proposes to link temporary Steiner points to related pin coordinates. These Steiner points move along with the move in related pins during a limited iteration window, which opens the backward gradient channel from routing to pin coordinates.

(2) *Differentiable Parasitics Extraction*: Given the coordinates of net pins and Steiner points, the resistances and capacitances (RCs) are extracted as input to signal delay models. The RCs follow a simple differentiable extraction model that assumes a linear relation between Manhattan distance and RC values, defined as part of the technology process. We note that the capacitances also include the load of sink pins determined by the type and size of the corresponding receiver gates, which are regarded as constants in [19].

(3) *Differentiable Net Delay Modeling*: With the parasitics information, signal delay on interconnects is calculated using the Elmore delay model. This model is differentiable through 4 tree-based dynamic programming [19] that propagates net delay gradients back to the parasitic values.

(4) *Differentiable Gate Delay Modeling and Propagation*: The gate delay is calculated using the non-linear delay model (NLDM), which consists of piecewise linear look-up tables (LUTs) indexed by input voltage slew and output capacitive load. The output load is obtained as part of the differentiable parasitics information. The input slew is recursively defined and propagated throughout the logic level of the entire design. We note that LUTs are part of the technology library and are determined by the type and size of gates.

(5) *Differentiable WNS and TNS Calculation*: With the available signal arrival time at the endpoint for each timing path, WNS and TNS can be calculated. Several techniques, such as max value smoothing, are involved in this process.

Our goal is to consider gate sizes $\mathbf{s}$ as differentiable variables along with the gate positions to optimize WNS and TNS. As can be seen in the timing calculation flow, a changing gate size affects nearly everything in delay modeling at the core of timing analysis. To derive the precise gradients, we analytically formulate this dependency and visualize it in Figure 4. The impact of gate size on timing objectives happens through 3 key channels: pin capacitance (part 1), gate delay (part 2), and gate output slew (part 3).

Gates of the same type but different sizes have different pin capacitance values that affect the upstream load. That load then changes the upstream net delay, cell delay, slew propagation, and finally, WNS and TNS. Similar to Equation (4), we interpolate the pin capacitances as a function of $\mathbf{s}$:

$$\mathrm{Cap}_i(\mathbf{s}_g) = \mathrm{Cap}_i(\lfloor \mathbf{s}_g \rfloor)(\lceil \mathbf{s}_g \rceil - \mathbf{s}_g) + \mathrm{Cap}_i(\lceil \mathbf{s}_g \rceil)(\mathbf{s}_g - \lfloor \mathbf{s}_g \rfloor), \quad (8)$$

where pin_i belongs to gate g and $\mathrm{Cap}_i(\mathbf{s}_g)$ is the interpolated pin i capacitance. We feed the derived pin capacitance to the differentiable Elmore delay model in [19]. As shown in Figure 4(b), this differentiable pin capacitance is chained with the Elmore delay module, passing the gradients from timing objectives all the way back to gate sizes $\mathbf{s}$. Specifically,

$$\begin{aligned}\left(\frac{\partial}{\partial \mathbf{s}_g} T\right)_{\text{part 1}} &= \sum_{\mathrm{pin}_i \in \text{gate } g} \frac{\partial \mathrm{Cap}_i}{\partial \mathbf{s}_g} \times \frac{\partial}{\partial \mathrm{Cap}_i} T \\ &= \sum_{\mathrm{pin}_i \in \text{gate } g} [\mathrm{Cap}_i(\lceil \mathbf{s}_g \rceil) - \mathrm{Cap}_i(\lfloor \mathbf{s}_g \rfloor)] \frac{\partial}{\partial \mathrm{Cap}_i} T,\end{aligned} \quad (9)$$

where T denotes either WNS or TNS.

Gate delay and output pin slew are also dependent on gate sizes. However, they are more involved as the delay and slew models are differentiable functions (NLDM LUTs) themselves, even for a single discrete gate size. Delay and slew outputs of an interpolated gate are thus a function of 3 variables: input slew, output load, *and* the interpolated gate size. Specifically, we always calculate 2 gate delays (respectively, slews) based on 2 adjacent discrete sizes $\lfloor \mathbf{s}_g \rfloor$ and $\lceil \mathbf{s}_g \rceil$, and then interpolate the results based on $\mathbf{s}_g$:

$$\begin{aligned} d_{ij}(\lfloor \mathbf{s}_g \rfloor) &= \mathrm{LUT}_{\lfloor \mathbf{s}_g \rfloor}(\mathrm{islew}, \mathrm{oload}), \\ d_{ij}(\lceil \mathbf{s}_g \rceil) &= \mathrm{LUT}_{\lceil \mathbf{s}_g \rceil}(\mathrm{islew}, \mathrm{oload}), \\ d_{ij}(\mathbf{s}_g) &= d_{ij}(\lfloor \mathbf{s}_g \rfloor)(\lceil \mathbf{s}_g \rceil - \mathbf{s}_g) + d_{ij}(\lceil \mathbf{s}_g \rceil)(\mathbf{s}_g - \lfloor \mathbf{s}_g \rfloor),\end{aligned} \quad (10)$$

where i and j determine the connected pins of a timing arc within gate g, and two LUT functions calculate two delay values for this timing arc with two gate sizes $\lfloor \mathbf{s}_g \rfloor$ and $\lceil \mathbf{s}_g \rceil$ by two indices: input slew value and output load capacitance value.

For the example shown in Figure 2, we first compute two sets of delay and slew values for two adjacent discrete sizes of a NOT gate, 1 and 2, through two different LUTs. A subsequent linear interpolation between these sizes yields the final delay and slew values. As depicted in Figure 4(b), with the interpolated gate delays d_{ab} and d_{ce}, the final timing objectives TNS and WNS can be calculated. Therefore, the delay channel of the timing gradient with respect to gate size $\mathbf{s}_g$ is as follows:

$$\begin{aligned}\left(\frac{\partial}{\partial \mathbf{s}_g} T\right)_{\text{part 2}} &= \sum_{\mathrm{arc}_{ij} \in \text{gate } g} \frac{\partial d_{ij}}{\partial \mathbf{s}_g} \times \frac{\partial}{\partial d_{ij}} T \\ &= \sum_{\mathrm{arc}_{ij} \in \text{gate } g} [d_{ij}(\lceil \mathbf{s}_g \rceil) - d_{ij}(\lfloor \mathbf{s}_g \rfloor)] \frac{\partial}{\partial d_{ij}} T.\end{aligned} \quad (11)$$

The slew calculation works similarly through differentiable linear interpolation:

$$\mathrm{Slew}_i(\mathbf{s}_g) = \mathrm{Slew}_i(\lfloor \mathbf{s}_g \rfloor)(\lceil \mathbf{s}_g \rceil - \mathbf{s}_g) + \mathrm{Slew}_i(\lceil \mathbf{s}_g \rceil)(\mathbf{s}_g - \lfloor \mathbf{s}_g \rfloor), \quad (12)$$

where pin i denotes one of the output pins of gate g. We chain the slew values throughout the design logic levels, as shown in Figure 4(a). We feed the output slew of one gate level to the Elmore model of the next level. Finally, we derive the slew channel of the timing gradient with respect to gate size $\mathbf{s}_g$ as follows:

$$\left(\frac{\partial}{\partial \mathbf{s}_g} T\right)_{\text{part 3}} = [\mathrm{Slew}_i(\lceil \mathbf{s}_g \rceil) - \mathrm{Slew}_i(\lfloor \mathbf{s}_g \rfloor)] \frac{\partial}{\partial \mathrm{Slew}_i} T. \quad (13)$$

Because of the partial derivative chain rule, the entire timing gradient is the sum of all 3 channels:

$$\frac{\partial}{\partial \mathbf{s}_g} T = \left(\frac{\partial}{\partial \mathbf{s}_g} T\right)_{\text{part 1}} + \left(\frac{\partial}{\partial \mathbf{s}_g} T\right)_{\text{part 2}} + \left(\frac{\partial}{\partial \mathbf{s}_g} T\right)_{\text{part 3}}. \quad (14)$$

By backward propagating the above gradient from timing objectives to gate size $\mathbf{s}$, the gradient descent method will create a trend to upsize the gates on critical timing paths to improve their timing.

3.4 Wirelength and Density Objectives

Wirelength objective depends on both gate positions $\mathbf{x}$ and $\mathbf{y}$ and gate sizes $\mathbf{s}$. Gate sizes affect pin offsets since a larger gate is likely to have more separate pins. We observe that pin offsets for different gate sizes are highly heterogeneous and cannot be soundly interpolated as a continuous function of gate size. But fortunately, the impact of pin offsets on the *optimization* of wirelength or PPA is quite minor compared to gate positions, capacitances, and LUTs. Therefore, we do not include the backward gradients of pin offsets in each placement and sizing iteration. We update the pin offsets to the nearest discrete sizes every few iterations to account for size changes during the optimization.

Density penalty objective encourages separation of gates. It also depends on both gate position $\mathbf{x}, \mathbf{y}$ and gate size $\mathbf{s}$. Previous nonlinear placement works [7, 8, 10, 11] exploit an analogy between placement and electrostatic system by modeling gates as electric charges, density penalty as the system potential energy, and density gradient as the electric field that pushes the gate charges apart. In our work, we model the gate size as a variable, which means the gate area is no longer a constant. In electrostatics-based placement, this implies a changing electrical charge *quantity*.

Previously, placement engine takes the gradients of density penalty with respect to electric charge positions, which yields the electric force. In our work, we additionally take the gradients with respect to the electrical charge *quantity*, which yields the electric *potential*. A simplified version of density penalty is:

$$D(\mathbf{x}, \mathbf{y}; \mathbf{s}) = \sum_{\text{gate } g} \text{Area}_g(\mathbf{s}_g) \times \Phi(\mathbf{x}_g, \mathbf{y}_g), \tag{15}$$

where $\Phi(\mathbf{x}_g, \mathbf{y}_g)$ is the electrical potential[2] of gate g at position $\mathbf{x}_g, \mathbf{y}_g$. The partial derivative of the density penalty with respect to gate area Area_g is:

$$\frac{\partial}{\partial \text{Area}_g(\mathbf{s}_g)} D(\mathbf{x}, \mathbf{y}; \mathbf{s}) = \Phi(\mathbf{x}_g, \mathbf{y}_g). \tag{16}$$

For gate area, we also use linear interpolation, as depicted in Figure 2. The forward and backward gradient equations are as follows:

$$\text{Area}_g(\mathbf{s}_g) = \text{Area}_g(\lfloor \mathbf{s}_g \rfloor)(\lceil \mathbf{s}_g \rceil - \mathbf{s}_g) + \text{Area}_g(\lceil \mathbf{s}_g \rceil)(\mathbf{s}_g - \lfloor \mathbf{s}_g \rfloor), \tag{17}$$

$$\frac{\partial}{\partial \mathbf{s}_g} \text{Area}_g(\mathbf{s}_g) = \text{Area}_g(\lceil \mathbf{s}_g \rceil) - \text{Area}_g(\lfloor \mathbf{s}_g \rfloor). \tag{18}$$

We can derive the partial derivative of the density penalty with respect to gate size $\mathbf{s}_g$ as follows:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{s}_g} D(\mathbf{x}, \mathbf{y}; \mathbf{s}) &= \frac{\partial \text{Area}_g(\mathbf{s}_g)}{\partial \mathbf{s}_g} \frac{\partial}{\partial \text{Area}_g(\mathbf{s}_g)} D(\mathbf{x}, \mathbf{y}; \mathbf{s}) \\ &= [\text{Area}_g(\lceil \mathbf{s}_g \rceil) - \text{Area}_g(\lfloor \mathbf{s}_g \rfloor)] \times \Phi(\mathbf{x}_g, \mathbf{y}_g). \end{aligned} \tag{19}$$

[2]In practice, the absolute value for $\Phi(\mathbf{x}_g, \mathbf{y}_g)$ varies across designs of different scales and only the relative values make sense. As a result, we normalize the electric potential in sizing gradient calculation so that their sum equals 1, which will ease parameter selection.

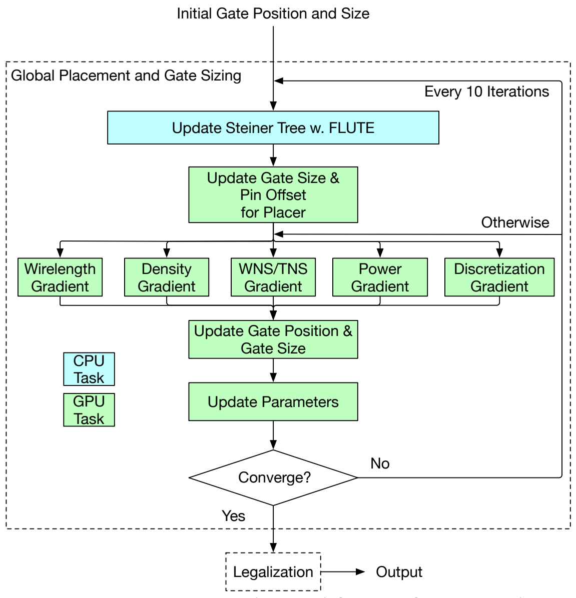


Figure 5: GPU-accelerated fusion framework.

Gates located in denser regions incur a higher density penalty, which discourages upsizing in these areas. Effectively, this promotes upsizing of gates in sparser regions on the critical paths. This differentiable fusion formulation of placement and sizing can thus explore a much richer search space and discover better joint placement and sizing solutions.

3.5 GPU-Accelerated Fusion Framework

Global placement is inherently time-consuming, particularly if we need to optimize gate sizes and positions simultaneously. Traditionally, handling large designs could require hundreds to thousands of iterations, often taking several hours on CPUs. Given the intense computational demands of such operations, GPUs can offer a significant advantage thanks to their capacity for efficient parallel processing. We employ a GPU-accelerated fusion framework to speed up the process significantly.

As depicted in Figure 5, our GPU-accelerated framework runs iteratively to update gate positions and sizes. In each iteration, we independently calculate 5 different objectives (wirelength, density, timing, power, discretization loss). We then back-propagate and merge their gradients with respect to gate positions and sizes. We use the final merged gradient to update gate positions and gate sizes. We note that every objective and gradient calculation is implemented using high-performance GPU kernels. This includes a GPU-accelerated differentiable STA engine as well as a GPU-accelerated differentiable electric force/electric potential solver.

During the iterations, we also do a number of auxiliary tasks, including the update of Steiner trees (every 10 iterations, see [19]), gradient-free pin offsets, and hyperparameters. The hyperparameter weights for different objectives change dynamically during the optimization process, which helps to balance the trade-offs between different objectives and achieve better convergence.

3.6 Parameter Configuration

In this section, we detail the selection of hyperparameters in this work. Without loss of generality, we assume the units are as follows:

timing slacks in nanoseconds (ns), area in square millimeters (mm^2), and leakage power in milliwatts (mW). WNS and TNS objective optimization weights t_1 and t_2 are initialized at 0.025 and 0.0005, respectively. The t_1 and t_2 parameters will increase by 1% every iteration to progressively enhance the importance of the timing objective. While these timing gradients include both gate positions and sizes, we observe that the gradient to sizes is typically much larger than to positions, which may cause increases in gate sizes in the early stage. As a result, we additionally scale down the gate size gradient by a factor of 1800. The weight λ_1 applied to the density penalty gradient to gate position $\mathbf{x}$ and $\mathbf{y}$ is set as the default value in DREAMPlace [11]. The weight λ_2 applied to leakage power objective is set at 1, increasing by 1‰ every iteration to encourage power reclaim in later stages. The discretization loss factor λ_3 is set at 0.01, which also increases by 1% every iteration to guide the gate sizes towards the nearest discrete values gradually. The timing-driven global placement and gate sizing kickstart at the 100th placement iteration.

In our experiments, uncontrolled expansion of gate sizes can trigger a vicious cycle. Specifically, the placer may disperse neighboring gates of one upsized gate, which could worsen timing metrics and compel the sizer to increase gate sizes further to counteract this degradation. To mitigate potential optimization divergence, we have implemented several safeguards:

Firstly, we clip the gradient of each gate size within the range $[-0.3, 0.3]$ to prevent gradient explosion. Additionally, gate sizes are constrained to remain within legal values each iteration, adjusted if they fall below 1 or exceed $\mathbf{s}_{g\,\max}$. Moreover, we employ a *preconditioning* strategy to moderate the influence of the gate size gradient. For each gate g with its size $\mathbf{s}_g$, the gradient is scaled down by a factor determined by the following formula:

$$\#\text{pins}(g) + \lambda_1 \times \text{Area}_g(\mathbf{s}_g), \tag{20}$$

where $\#\text{pins}(g)$ represents the number of pins of gate g, $\text{Area}_g(\mathbf{s}_g)$ denotes the area of gate g with size $\mathbf{s}_g$, and λ_1 is the density penalty weights for placer, a gradually increasing value that promotes gate spreading. Additionally, since the density penalty and leakage power objectives can directly counteract excessive upsizing, the weights of these objectives are increased if gate density is high or the circuit scale is large.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

We implement our proposed fusion flow combining global placement and gate size optimization using C++ and CUDA kernels, based on the open-source placer DREAMPlace [11]. Our environmental setup includes a server equipped with a 40-core Intel Xeon Gold 6230 CPU at 2.10 GHz, 256GB RAM, and an Nvidia A40 GPU with 48GB of memory. We evaluate our performance on large industrial designs sourced from eight open-source CircuitNet-N28 [42] datasets, all synthesized under a 28nm commercial technology node. Our wire unit RC also follows this library. Detailed benchmark statistics are presented in Table 3. These benchmarks vary in scale and complexity, reflecting a range of real-world design scenarios.

Table 3: CircuitNet [42] benchmark statistics.

Benchmark	Period (ns)	#Macros	#Cells	#Nets	#Pins
RISCY-a	2.0	3	54079	54635	211308
RISCY-FPU-a	2.0	3	81001	81777	303014
zero-riscy-a	2.0	3	46390	46089	172731
RISCY-b	2.0	13	34664	36420	129794
RISCY-FPU-b	2.0	13	63823	60428	210901
zero-riscy-b	2.0	13	21738	22902	81481
NVDLA-small	2.0	108	338280	353608	1236682
OpenC910-1	3.0	32	808686	818218	3265940

4.2 Results and Analysis

We compare our newly developed flow with the open-source OpenROAD [3] flow, representing a traditional design flow that separates global placement and gate sizing. In OpenROAD, we conduct global placement with `timing_driven` option enabled for the fairness of comparison. The OpenROAD timing-driven gate resizer[3] leverages local optimizations to adjust gate sizes based on timing analysis feedback incrementally. As discussed in Section 2, this represents a heuristic approach to gate sizing optimization. Additionally, the OpenROAD timing repairer includes not only a resizer but also other design steps like pin swapping and buffer insertion. Therefore, to ensure the fairness and authenticity of our experiments, we also apply an OpenROAD sizer after our flow, as shown in Figure 1. After the OpenROAD sizer, some gates may not fit into the original placement layout, requiring an additional OpenROAD detailed placement to ensure legal and compatible results.

Apart from OpenROAD flow, for a thorough evaluation, we also acquired the binary executable file from authors of [19] and conduct an ablation study[4] that performs timing-driven placement-only optimization. Gate size will only be optimized by the OpenROAD sizer to further prove that simultaneous optimization of gate positions and gate sizes is more effective than the traditional approach.

To summarize, OpenROAD's flow incorporates a timing-driven global placement, timing-driven gate sizing, and detailed placement. Our fusion flow consists of a novel fusion of global placement and gate sizing, followed by DREAMPlace's [11] greedy and abacus legalization, and then followed by OpenROAD timing-driven sizer and detailed placement. The ablation study includes the timing-driven global placement [19], OpenROAD timing-driven sizer, and detailed placement.

The metrics for our evaluation, including TNS, WNS, and leakage power, are reported by OpenSTA [45], one of the most widely used power and static timing analysis tools. Notably, our baseline OpenROAD sizer embeds OpenSTA as RC and timing calculation engine that is the same as our evaluator, whereas our approach employs a custom differentiable timing engine to enable gradient-based optimization. Given the inherent discrepancies between different timing engines for placement-stage timing estimation, the optimization directions might differ slightly. Therefore, the OpenROAD sizer added to our flow also plays a role in mitigating this discrepancy and ensuring both can be compared under OpenSTA reports.

[3]In OpenROAD, it is invocated using `repair_timing` command.
[4]The test result is different from [43] due to multiple differences in our experimental setup. We use OpenSTA for timing report while [43] uses evaluation timer and wire unit RC settings from ICCAD 2015 contest [44]. Furthermore, our flow introduces a detailed sizer in flow which is not present in [43].

Table 4: Timing and leakage power comparison between the OpenROAD [3] flow, an ablation study that includes timing-driven global placement [19] and gate sizing by OpenROAD, and our flow. The best results are highlighted in bold.

Benchmark	OpenROAD [3] Flow			Separate Flow ([19] and [3])			Ours		
	TNS (ns)	WNS (ns)	Power (mW)	TNS (ns)	WNS (ns)	Power (mW)	TNS (ns)	WNS (ns)	Power (mW)
RISCY-a	-515.51	-0.23	1.11	-611.05	-0.18	**1.10**	**-192.31**	**-0.15**	1.11
RISCY-FPU-a	-822.06	-0.23	1.88	-1730.32	-0.40	**1.86**	**-497.34**	**-0.21**	1.88
zero-riscy-a	-2.40	-0.05	1.19	-10.88	-0.05	1.17	**-2.31**	**-0.03**	**1.13**
RISCY-b	-456.28	-0.26	0.833	-310.40	**-0.13**	**0.822**	**-271.95**	**-0.13**	0.823
RISCY-FPU-b	**-803.78**	-0.71	1.56	-1147.10	-0.52	**1.54**	-881.1	**-0.44**	**1.54**
zero-riscy-b	-46.07	-0.21	0.574	-40.27	-0.12	**0.567**	**-23.92**	**-0.10**	**0.567**
NVDLA-small	-2850.55	-0.97	**6.28**	-780.79	**-0.32**	6.31	**-154.05**	**-0.32**	6.38
OpenC910-1	-7176.30	-0.71	**9.60**	-8535.54	-0.77	9.86	**-1097.02**	**-0.64**	9.92
Avg. Ratio	4.37	1.77	1.01	3.54	1.29	**1.00**	**1.00**	**1.00**	**1.00**

Table 5: Runtime comparison for different design benchmarks. Column names represent: GP (OpenROAD Global Placement), Sizer (OpenROAD Gate Sizing), DP (OpenROAD Detailed Placement), Fusion (Our Global Placement and Gate Sizing), Legal. (DREAMPlace Legalization), and Total (Total Runtime). The best results are highlighted in bold.

Design	OpenROAD Flow (s)				Ours (s)				
	GP	Sizer	DP	Total	Fusion	Legal.	Sizer	DP	Total
RISCY-a	237	18	2	257	95	1	14	1	**111**
RISCY-FPU-a	340	105	1	446	109	1	173	1	**284**
zero-riscy-a	180	7	10	197	89	1	8	1	**99**
RISCY-b	149	7	1	157	75	1	11	1	**88**
RISCY-FPU-b	244	115	1	360	89	1	125	1	**216**
zero-riscy-b	124	17	1	142	65	1	21	1	**88**
NVDLA-small	1511	97	7	1615	212	2	17	3	**234**
OpenC910-1	4678	917	8	5603	287	3	577	5	**872**
Avg. Ratio				3.03					**1.00**

* The running time for each step is rounded up to the nearest integer.

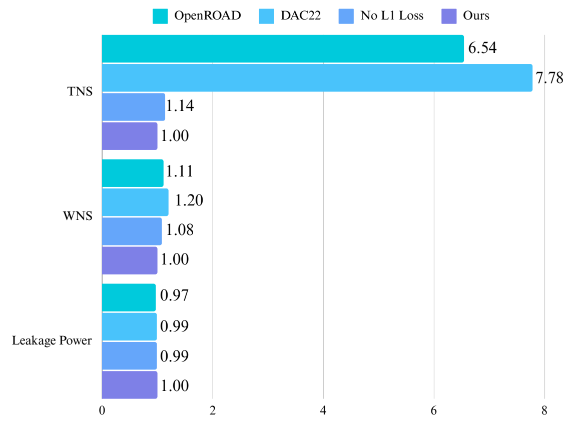


Figure 6: Comparison across different flows on the largest design, OpenC910-1. Metrics are normalized based on results from our proposed flow.

As depicted in Table 4, our approach yields a significant average improvement of **77.1%** (1-1/4.37) in TNS and **43.5%** (1-1/1.77) in WNS compared to the OpenROAD flow. Meanwhile, our method achieves an average reduction in leakage power consumption by **1%** (1-1/1.01), highlighting its capability to balance power and performance effectively. In addition, the ablation study, in which gate size remains static during timing-driven global placement and will only be optimized by OpenROAD, further proves the benefit of our fusion optimization strategy that explores a larger joint search space between placement and sizing. We conduct an additional ablation study to prove the effectiveness of our discretization loss by removing that L1 loss from the objective. The results shown in Figure 6 demonstrate that our overall performance will deteriorate for the largest design, OpenC910-1, once the discretization loss objective is removed.

For the large designs evaluated, including NVDLA-small and OpenC910-1, our approach offers substantial benefits in optimizing TNS and WNS despite exhibiting a higher leakage power than the conventional OpenROAD flow. This is crucial for high-performance designs, where timing closure is a critical challenge. Our method not only enhances timing and power metrics but also significantly improves the efficiency of the entire placement flow, achieving an average 3× speed-up compared to the baseline OpenROAD flow, as detailed in Table 5. As the scale and complexity of the designs increase, the acceleration ratio can reach up to approximately 7× for NVDLA-small and OpenC910-1 designs. This enhancement largely benefits from the heterogeneous GPU acceleration and the efficiency of the differentiable optimization framework. A notable bottleneck in our method is the OpenROAD sizer, a sequential process that takes much runtime. This runtime cost can be lowered with the fusion of sizing, buffering, and pin swapping into early design stages in future works.

5 CONCLUSION

In this paper, we introduce a pioneering approach to VLSI design that integrates global placement with gate sizing to optimize PPA metrics more effectively. Leveraging differentiable objective functions and GPU-accelerated computation, our method not only significantly enhances design quality but also provides a substantial acceleration, achieving up to a 7× acceleration in speed compared to the traditional OpenROAD flow. Our experimental results demonstrate remarkable improvements in timing and power metrics, with an average enhancement of 77.1% in TNS and 43.5% in WNS, while achieving a 1% reduction in leakage power consumption. Our future work includes adding more optimization strategies such as pin swapping within this framework. We believe our work opens up new possibilities for differentiable and fusion strategies in EDA and will inspire further research in this field.

ACKNOWLEDGE

This work is supported in part by STIC, the Natural Science Foundation of Beijing, China (Grant No. Z230002), and 111 project (B18001).

REFERENCES

[1] 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.

[2] Cadence Design Systems, *Innovus User Guide*, Cadence Design Systems, Inc., 2022, version 21.13.

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

[4] V. Bhardwaj, "Shift left trends for design convergence in soc: An eda perspective," *International Journal of Computer Applications*, vol. 174, no. 16, pp. 22–27, Jan 2021.

[5] T.-C. Chen *et al.*, "Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1228–1240, 2008.

[6] M.-K. Hsu *et al.*, "Ntuplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1914–1927, 2014.

[7] 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.

[8] C.-K. Cheng *et al.*, "Replace: Advancing solution quality and routability validation in global placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1717–1730, 2019.

[9] Z. Zhu *et al.*, "Generalized augmented lagrangian and its applications to vlsi global placement," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[10] J. Lu *et al.*, "eplace-ms: Electrostatics-based placement for mixed-size circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 5, pp. 685–698, 2015.

[11] Y. Lin *et al.*, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 4, pp. 748–761, 2021.

[12] H. Chang *et al.*, "Net criticality revisited: an effective method to improve timing in physical design," in *ACM International Symposium on Physical Design (ISPD)*. Association for Computing Machinery, 2002, p. 155–160.

[13] T. Kong, "A novel net weighting algorithm for timing-driven placement," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2002, pp. 172–176.

[14] B. Halpin, C. Y. R. Chen, and N. Sehgal, "A sensitivity based placer for standard cells," in *Proceedings of the 10th Great Lakes Symposium on VLSI*. Association for Computing Machinery, 2000, p. 193–196.

[15] H. Ren, D. Pan, and D. Kung, "Sensitivity guided net weighting for placement-driven synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 5, pp. 711–721, 2005.

[16] Z. Xiu and R. Rutenbar, "Timing-driven placement by grid-warping," in *Proceedings. 42nd Design Automation Conference, 2005.*, 2005, pp. 585–590.

[17] P. Liao *et al.*, "Dreamplace 4.0: Timing-driven global placement with momentum-based net weighting," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2022, pp. 939–944.

[18] A. Chowdhary *et al.*, "How accurately can we model timing in a placement engine?" in *ACM/IEEE Design Automation Conference (DAC)*, 2005, pp. 801–806.

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

[20] W. Li *et al.*, "Calibration-based differentiable timing optimization in non-linear global placement," in *ACM International Symposium on Physical Design (ISPD)*. New York, NY, USA: Association for Computing Machinery, 2024, p. 31–39.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

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

[32] 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.

[33] 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.

[34] 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*, vol. 18, no. 7, pp. 1014–1025, 1999.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 27, no. 1, pp. 70–83, 2008.

[42] Z. Chai *et al.*, "CircuitNet: An open-source dataset for machine learning in vlsi cad applications with improved domain-specific evaluation metric and learning strategies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

[43] X. Jiang *et al.*, "Accelerating routability and timing optimization with open-source ai4eda dataset circuitnet and heterogeneous platforms," in *2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2023.

[44] M. Kim *et al.*, "ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 921–926.

[45] The-OpenROAD-Project, "Parallax static timing analyzer," GitHub repository, 2024, available online: https://github.com/The-OpenROAD-Project/OpenSTA.