Timing-Driven Global Placement by Efficient Critical Path Extraction

Yunqi Shi[1,2,3], Siyuan Xu[3], Shixiong Kai[3], Xi Lin[1,2], Ke Xue[1,2], Mingxuan Yuan[3], Chao Qian[1,2]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]School of Artificial Intelligence, Nanjing University, China
[3]Huawei Noah's Ark Lab, China
{shiyq, qianc}@lamda.nju.edu.cn

***Abstract*—Timing optimization during the global placement of integrated circuits has been a significant focus for decades, yet it remains a complex, unresolved issue. Recent analytical methods typically use pin-level timing information to adjust net weights, which is fast and simple but neglects the path-based nature of the timing graph. The existing path-based methods, however, cannot balance the accuracy and efficiency due to the exponential growth of number of critical paths. In this work, we propose a GPU-accelerated timing-driven global placement framework, integrating accurate path-level information into the efficient DREAMPlace infrastructure. It optimizes the fine-grained pin-to-pin attraction objective and is facilitated by efficient critical path extraction. We also design a quadratic distance loss function specifically to align with the RC timing model. Experimental results demonstrate that our method significantly outperforms the current leading timing-driven placers, achieving an average improvement of 40.5% in total negative slack (TNS) and 8.3% in worst negative slack (WNS), as well as an improvement in half-perimeter wirelength (HPWL).**

I. Introduction

In the field of very-large-scale integration (VLSI) design, the placement process is critical as it forms the bridge between logical design and physical layout [8], [28]. Traditional placement methods, while focusing on minimizing wirelength and reducing routing congestion, only implicitly address timing metrics [3], which may fail to satisfy the strict timing requirements of modern, large-scale chip designs. Direct optimization of timing is essential but typically demands considerable computational resources and turn-around time, emphasizing the need for more efficient timing-driven placement methods to improve design cycles and ensure timing closure.

Modern placement algorithms often consist of three main stages: global placement, legalization, and detailed placement [23]. Global placement distributes cells across the target layout, balancing the wirelength and density. The coarse result is then refined by legalization and fine-tuned by detailed placement. Among these three stages, global placement plays a crucial role in determining the overall distribution of cells, significantly influencing the quality of the final placement, including timing. As a result, timing-driven placement (TDP) for global placement has been extensively studied, focusing on optimizing key timing metrics such as total negative slack (TNS) and worst negative slack (WNS).

Such TDP techniques basically have three components: foundational placement algorithms, timing analysis, and interfaces between them [26]. The first component utilizes traditional global placement engines, which primarily focus on optimizing the trade-offs between wirelength and density. The second component involves either internal or external timing engines that assess the current layout of the placement to provide essential timing data, such as critical path delays or pin slacks. The third component translates timing metrics into certain weights or constraints to drive the foundational placement engines. Depending on the method of handling timing information, TDP techniques can be broadly categorized into two types: net-based and path-based approaches.

Net-based methods use timing analysis to adjust net weights [2], [5], [9], [10], [25] or net constraints [11], [16], [22] either dynamically or statically, indirectly guiding the placement to focus on critical nets. Since traditional placement algorithms primarily focus on minimizing wirelength, which inherently involves net considerations, only minimal modifications are required to adapt these for a timing-driven approach. Recently, Liao et al. upgraded the advanced nonlinear placer, DREAMPlace [20], to its timing-driven version 4.0 [18]. This new version dynamically adjusts net weights, utilizing a momentum-guided mechanism that interacts with a timing analysis engine, enhancing its focus on timing optimization.

Path-based methods [7], [15], [27] directly address paths extracted from the timing graph, typically formulated as a mathematical programming problem. These methods maintain an accurate view of timing during the optimization [4], thereby often ensuring high-quality results. However, they frequently encounter scalability issues as the number of paths grows exponentially with the increase of design size [18]. Recently, Guo and Lin introduced a novel differentiable-timing-driven placement framework [12], which incorporates a GPU-accelerated, differentiable timing engine into DREAMPlace, enabling efficient path-based analysis. This approach not only achieves state-of-the-art performance but also operates at competitive speeds, addressing traditional scalability challenges effectively.

Despite significant advancements, the timing-driven placement problem remains largely unsolved. Net-based approaches often suffer from an indirect optimization objective and underutilized timing information. For path-based methods, although Guo and Lin [12] have somewhat addressed the scalability issue, their approach potentially compromises accuracy by smoothing timing metrics. In this work, we introduce a timing-driven global placement framework that incorporates a fine-

978-3-9826741-0-0/DATE25/© 2025 EDAA

grained pin-to-pin attraction quadratic distance loss, directly targeting timing metrics. This is complemented by a path-level timing analysis module that extracts critical paths efficiently. We outline the key contributions as follows:

- We develop a GPU-accelerated, timing-driven placement flow that optimizes pin-to-pin attraction on critical paths, based on the leading placer DREAMPlace 4.0 [18]. Our code is available at https://github.com/lamda-bbo/Efficient-TDP.
- We introduce an efficient critical path extraction method that captures comprehensive timing information, enabling timing optimization at a high speed—achieving a 6× speed improvement over the default timer [14].
- We design a quadratic Euclidean distance loss for pin-to-pin attraction, which is closely aligned with timing metrics and significantly contributes to the superior performance, showing 50% (30%) improvements on TNS (WNS) compared to other distance metrics.
- Experimental results on the ICCAD2015 contest benchmark suites [17] show that we can achieve about 60% (30%) improvements on TNS (WNS), compared to DREAMPlace 4.0 [18] and about 50% (10%) improvements on TNS (WNS), compared to Guo and Lin's work [12].

The rest of the paper is organized as follows: Section II gives the preliminaries for timing-driven global placement. Section III presents details of our proposed timing-driven placement flow. Section IV provides empirical studies and discussions. Section V concludes this paper.

II. PRELIMINARIES

A. Nonlinear Global Placement

Global placement is a critical phase in physical design, aiming to determine the locations of millions of cells within a specified chip layout. The goal of optimization is to minimize the wirelength connecting all relevant components while adhering to density constraints. To facilitate efficient optimization, the constrained problem is transformed into an unconstrained nonlinear optimization problem:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{e\in E} WL_e(\boldsymbol{x},\boldsymbol{y}) + \lambda \cdot D(\boldsymbol{x},\boldsymbol{y}), \tag{1}$$

where $\boldsymbol{x}$, $\boldsymbol{y}$ are the cell locations, E represents the set of all nets, WL is typically a smoothed half-perimeter wirelength (HPWL) function (e.g., weighted-average method [13]), D is a density metric, and λ is the density penalty factor.

B. Static Timing Analysis

Static timing analysis (STA) [24] evaluates circuit timing by modeling it as a directed acyclic graph, where edges represent timing arcs that indicate signal propagation directions. In this graph, a timing path starts from a source and ends at a sink, with the arrival time Arr propagated forward and the required arrival time Req backward along the path. The difference between these times at any point t defines the slack:

$$Slack(t) = Req(t) - Arr(t). \tag{2}$$

A negative slack at an endpoint indicates a timing violation, necessitating further optimization. To quantify these violations, the metrics worst negative slack (WNS) and total negative slack (TNS) are used. WNS identifies the largest magnitude of violation, while TNS sums all the negative slacks:

$$\text{WNS} = \min\nolimits_{t\in V} Slack(t), \tag{3}$$

$$\text{TNS} = \sum\nolimits_{t\in V} Slack(t), \tag{4}$$

where V represents the set of all violated endpoints. If $V = \emptyset$, both WNS and TNS are zero, indicating that all timing constraints are met.

C. Timing-Driven Placement by Net Weighting

With WNS and TNS defined to measure the timing performance, vanilla nonlinear placement may be inefficient in optimizing these metrics, because wirelength does not directly target timing metrics. An intuitive idea is dynamically adjusting weight of nets using timing information. Among timing-driven placers, DREAMPlace 4.0 [18] is the state-of-the-art open-source implementation. Built upon DREAMPlace [20], version 4.0 makes the most of the GPU acceleration framework and integrates the popular open-source timer OpenTimer [14] for STA, further extracting timing information for net weighting. The nonlinear placement objective (1) is thus formulated as:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{e\in E} w_e \cdot WL_e(\boldsymbol{x},\boldsymbol{y}) + \lambda \cdot D(\boldsymbol{x},\boldsymbol{y}), \tag{5}$$

where w_e is the weight assigned to net e. Nets that are timing critical are assigned larger weights.

III. OUR ALGORITHM

In this section, we propose a GPU-accelerated timing-driven global placement framework shown in Fig. 1. We introduce a fine-grained pin-to-pin attraction objective, which directly targets timing metrics, facilitated by an efficient critical path extraction method and a quadratic Euclidean distance loss.

A. Fine-Grained Weighting Scheme

As described in Sec.II-C, traditional net weighting methods aim to enhance the timing performance by assigning additional weights to critical nets. However, given the complexity of modern designs, which often feature large fan-out nets and shared data paths, this approach has notable shortcomings. Specifically, it may apply unnecessary weights to non-critical pin pairs and overlook the effects of path-sharing, which disable efficient optimization of timing performance.

To address these issues, we propose incorporating pin-to-pin attraction as a fine-grained objective, replacing the traditional method of applying extra net weights for timing optimization. The revised objective function is presented as follows:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{e\in E} WL_e(\boldsymbol{x},\boldsymbol{y}) + \lambda \cdot D(\boldsymbol{x},\boldsymbol{y}) + \beta \cdot PP(\boldsymbol{x},\boldsymbol{y}), \tag{6}$$

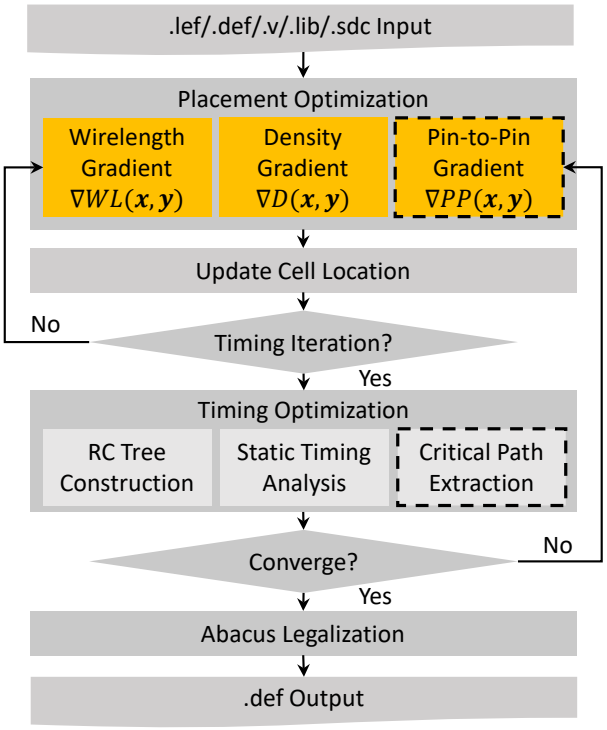

Fig. 1. Our timing-driven placement flow enabling GPU-acceleration. Gradients in orange are propagated on GPU.

where β represents the penalty multiplier, and PP denotes the pin-to-pin attraction loss.

Pin-to-pin attraction describes an attractive force that brings pins on critical path closer together, thereby reducing wire delay and improving timing performance. Fig. 2 compares the traditional net weighting scheme with the pin-to-pin attraction model for a three-pin net. We illustrate this with an example comprising three timing paths, indicated by green, yellow, and blue arrows. Traditionally, net weighting for timing optimization typically involves assigning a substantial weight to both pins B and C, based on the worst pin slack within the net (i.e., pin C in this case). But the weight is unnecessary for pin B, as positive slacks are disregarded in timing metrics. Furthermore, this heavy weighting could compromise the wirelength of other nets, potentially creating new critical paths. Additionally, pin C's slack is determined by the worst slack of the paths it belongs to, calculated as `min(-400,-500)`, which ignores the effects of path-sharing due to the nature of pin-level timing analysis. In contrast, the pin-to-pin attraction method assigns weights selectively to critical pin pairs based on their individual slacks, offering a more refined control that benefits both overall timing and wirelength. What's more, we can analyse critical paths one by one, thus define the pin C's slack as `sum(-400,-500)`, taking path-sharing into consideration.

Although a similar concept of 'pin-to-pin attraction' has been previously described as 'virtual path' in the literature [6], [21], the advantages of fine-grained weighting and path-sharing cannot be realized without extracting critical timing paths efficiently. This enables proper weights assigned to those critical pin pairs. This vital aspect has been insufficiently explored in prior studies due to its complexity and the potential for an exponential increase in the number of paths as chip scale grows.

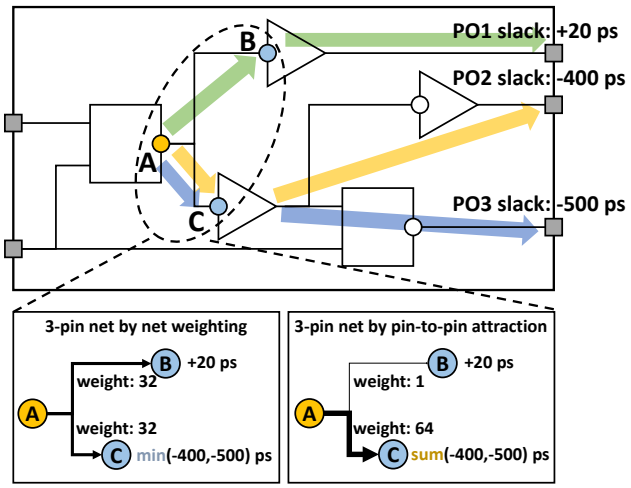

Fig. 2. Illustration of traditional net weighting and pin-to-pin attraction.

B. Critical Path Extraction

To address the necessity of efficiently extracting path-level timing information, we integrate OpenTimer [14], a leading timing engine widely adopted by open-source projects and adapted from DREAMPlace 4.0 [18]. OpenTimer provides an advanced feature, `report_timing(n)`, which identifies the worst n endpoints based on slack and retrieves the n worst critical paths for each, resulting in n^2 paths from which the top n worst paths are selected. While `report_timing(n)` is effective for identifying critical paths when n is small (e.g., 1), allowing quick and detailed analysis of specific paths, its efficiency decreases as n increases due to the quadratic growth in analyzed paths. Additionally, the extracted paths tend to concentrate on a few critical endpoints, which does not align well with the TNS metric that requires summing negative slacks across all endpoints.

To address the afore-mentioned problem, we present the `report_timing_endpoint(n,k)` method for critical path extraction. Here n represents the number of most critical endpoints we want to investigate and k means the number of critical paths we extract for each endpoint. The method returns $n \times k$ paths that ensure each mentioned endpoint is properly covered, thus comprehensively reflecting the timing issue of the entire chip and directly targeting the TNS metric.

Table I details the timing analysis for the `superblue1` case [17] using different methods. Initially, we identify a total of 26,300 failing endpoints. Employing OpenTimer's `report_timing(26300)`, we find that out of these paths, only 6 unique endpoints and 748 unique pin pairs are extracted, which significantly deviates from the TNS metric's requirements that each failing endpoint's slack should be considered. Even increasing the path count to 26300×10, the extracted unique endpoints and pin pairs still fall short of the necessary criteria for a comprehensive TNS evaluation. In contrast, our method, `report_timing_endpoint(26300,1)`, efficiently covers all endpoints and includes a broader range of

TABLE I
TIMING STATISTICS COMPARISON AMONG VARIOUS CRITICAL PATH EXTRACTION METHODS.

Command	**Complexity**	**Number of Paths**	**Number of Endpoints**	**Number of Pin Pairs**	**Time** (sec)
`report_timing(26300)`	$O(n^2)$	26300	6	748	41.64
`report_timing(263000)`	$O(n^2)$	263000	20	2538	146.70
`report_timing_endpoint(26300,1)`	$O(n \times k)$	26300	26300	62811	7.00
`report_timing_endpoint(26300,10)`	$O(n \times k)$	135705*	26300	93740	21.46

* The number here is less than 263000, as not all endpoints can extract 10 critical paths.

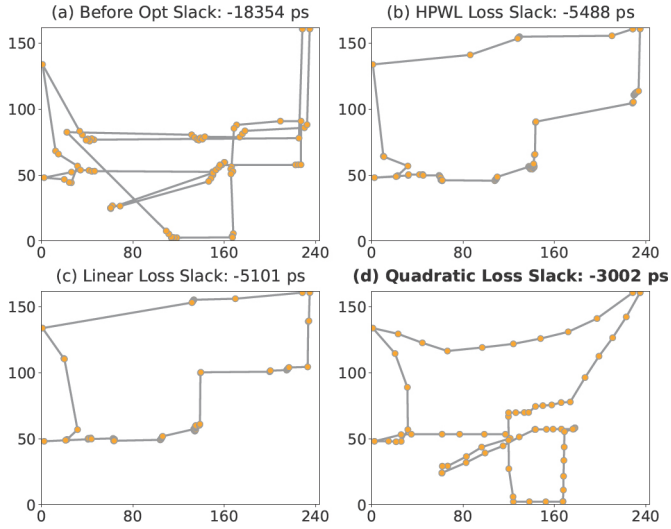

Fig. 3. Visualization of a specific critical path optimized using different distance losses. The slack of each path is given on the top of each figure.

pin pairs. Increasing the number of paths per endpoint to 10 triples the time while only increasing the number of pin pairs by 1.5 times, indicating that the former setting is sufficient for effective optimization. Further empirical evaluations and discussions are available in Table III and Sec. IV-B.

C. Quadratic Euclidean Distance Loss

To achieve effective optimization, it is desirable to design a loss function that aligns well with the final timing metrics. In practice, the RC delay model is fairly sufficient and widely adopted. Given the distributed RC network of a net, the delay from the net source s to sink t can be calculated as follows:

$$\text{Delay}_{s \to t} = R_{s \to t} C_t, \tag{7}$$

where $R_{s \to t}$ represents the equivalent resistance from s to t, and C_t represents the capacitance at node t. For net delay, R and C here are both linear to wirelength, making the delay quadratic in length. Thus, we choose the square of the pin-to-pin Euclidean distance – quadratic loss as the objective:

$$Q(i,j) = (x_i - x_j)^2 + (y_i - y_j)^2, \tag{8}$$

where $Q(i,j)$ represents the pin-to-pin loss of pin i and j.

Fig. 3 demonstrates the effectiveness of our quadratic loss design, comparing it to HPWL loss and Euclidean distance loss using the `superblue16` case [17]. We first identify the most critical path using `report_timing(1)` from the coarse placement before timing optimization, as shown in Fig. 3(a). Fig. 3(b) and (c) show the corresponding path optimized to convergence by HPWL loss and linear Euclidean distance loss, respectively. Both paths and slacks appear similar in these two figures due to the nearly linear relationship both loss functions have with distance. Such linear loss fails to differentiate effectively between longer and shorter wires, with gradients indicating direction but not magnitude. Consequently, many cells may cluster together, while some wire segments become excessively long, as shown in Fig. 3(b) and (c). In contrast, Fig. 3(d), which utilizes a quadratic distance loss, demonstrates improved path slack despite an increase in total wirelength. This improvement is attributed to the quadratic loss fostering a more uniform distribution of cells and maintaining more consistent wire segment lengths. It also implies that there are fewer excessively long wire segments, which typically necessitate the insertion of buffers that can escalate area, power, and thermal concerns, underscoring the utility in modern chip design [1]. As this study focuses on academic cases which are not suited for post-CTS evaluations, optimizing area, power, and thermal is identified as a crucial future work.

D. Workflow Summary

We detail the overall process for timing optimization as a summary. Initially, vanilla DREAMPlace [20] is run to distribute the cells within the layout. Subsequently, we perform a path-level timing analysis every m rounds to extract critical paths and update the pin-to-pin loss. This involves `report_timing_endpoint(n,1)`, where n denotes the number of all failing endpoints, to collect data on critical paths. As we traverse these paths, each pin pair (i,j) involved is added to a maintained set P, unless it has already been included. To address the path-sharing effect, the weight $w_{(i,j)}$ of each pin pair is dynamically updated as follows:

$$w_{(i,j)} = \begin{cases} w_0, & \text{if } (i,j) \notin P, \\ w_{(i,j)} + w_1 \cdot (slack/\text{WNS}), & \text{otherwise}, \end{cases} \tag{9}$$

where w_0 and w_1 are hyperparameters, and $slack$ indicates the negative slack of the respective critical path. The pin-to-pin attraction loss $PP(\boldsymbol{x}, \boldsymbol{y})$ of the layout is then computed as:

$$PP(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j) \in P} w_{(i,j)} \cdot Q(i,j), \tag{10}$$

with $Q(i,j)$ and $w_{(i,j)}$ defined in Eqs. 8 and 9, respectively. After defining the loss function properly, we implement the CUDA kernel of PP loss for GPU-acceleration.

TABLE II

COMPARING TNS ($\times 10^5$ ps), WNS ($\times 10^3$ ps), AND HPWL ($\times 10^6$) ACROSS DIFFERENT STATE-OF-THE-ART TIMING-DRIVEN PLACEMENT METHODS. THE BEST RESULTS ARE IN **BOLD**, AND THE RUNNER-UPS ARE COLORED BROWN.

Benchmark	DREAMPlace* [20]			DREAMPlace 4.0* [18]			Differentiable-TDP† [12]			Distribution-TDP§ [19]			Efficient-TDP (ours)		
	TNS	WNS	HPWL	TNS	WNS	HPWL	TNS	WNS	HPWL	TNS	WNS	HPWL	TNS	WNS	HPWL
superblue1	-262.44	-18.87	422.0	-85.03	-14.10	443.1	-74.85	-10.77	432.8	-42.10	-9.26	-	**-17.44**	**-7.75**	**418.8**
superblue3	-76.64	-27.65	478.2	-54.74	-16.43	482.4	-39.43	-12.37	478.4	-26.59	-12.19	-	**-20.40**	**-11.82**	**462.5**
superblue4	-290.88	-22.04	**312.0**	-144.38	-12.78	335.9	-82.92	**-8.49**	312.2	-123.28	-8.86	-	**-82.88**	-9.17	317.7
superblue5	-157.82	-48.92	488.3	-95.78	-26.76	556.2	-108.08	-25.21	488.7	-70.35	-31.64	-	**-62.18**	**-24.65**	**484.2**
superblue7	-141.55	-19.75	604.3	-63.86	**-15.22**	604.0	-46.43	**-15.22**	602.1	-95.89	-17.24	-	**-43.52**	**-15.22**	**597.5**
superblue10	-731.94	-26.10	935.9	-768.75	-31.88	1036.7	**-558.05**	**-21.97**	934.4	-691.10	-25.86	-	-558.14	-23.08	**911.6**
superblue16	-453.57	-17.71	**435.8**	-124.18	-12.11	448.1	-87.03	-10.85	485.1	-55.99	-12.21	-	**-22.90**	**-8.63**	471.6
superblue18	-96.76	-20.29	243.0	-47.25	-11.87	253.6	-19.31	-7.99	243.6	-19.23	**-5.25**	-	**-16.16**	-6.92	**234.4**
Average Ratio	6.90	2.07	1.004	2.75	1.40	1.06	2.00	1.09	1.02	1.68	1.11	-	**1.00**	**1.00**	**1.00**

* For DREAMPlace [20] and DREAMPlace 4.0 [18], we replicate the layout DEFs using the default configurations.

† For Differentiable-TDP [12], we acquire the DEFs from its authors to evaluate.

§ For Distribution-TDP [19], we borrow their results as our evaluation script is identical. The HPWL values are not provided.

TABLE III

ABLATION STUDY COMPARING TNS ($\times 10^5$ ps) AND WNS ($\times 10^3$ ps) ACROSS VARIOUS SETTINGS. THE BEST RESULTS ARE IN **BOLD**, AND THE RUNNER-UPS ARE COLORED BROWN.

Benchmark	w/ HPWL Loss		w/ Linear Loss		w/ rpt_timing(n*10)		w/ rpt_timing_ept(n,10)		w/o Path Extraction		Our Method	
	TNS	WNS	TNS	WNS	TNS	WNS	TNS	WNS	TNS	WNS	TNS	WNS
superblue1	-74.70	-13.85	-76.66	-11.94	-80.61	-9.84	**-12.69**	-8.79	-19.27	-14.43	-17.44	**-7.75**
superblue3	-47.42	-15.81	-47.29	-13.00	-37.10	**-11.71**	-20.93	-11.91	**-20.11**	-16.47	-20.40	-11.82
superblue4	-155.23	-16.35	-153.76	-14.32	-139.05	-9.15	-86.49	**-8.75**	-102.39	-10.23	**-82.88**	-9.17
superblue5	-93.21	-26.37	-91.96	-28.23	-123.13	-27.28	-58.78	-25.58	**-51.61**	-34.57	-62.18	**-24.65**
superblue7	-68.68	-16.19	-59.47	**-15.22**	-47.70	**-15.22**	-35.14	-19.57	**-34.96**	**-15.22**	-43.52	**-15.22**
superblue10	-657.95	-23.39	-707.27	-27.67	-629.28	-24.50	-570.37	-23.23	**-515.80**	**-21.94**	-558.14	-23.08
superblue16	-61.96	-9.93	-63.69	-13.81	-30.87	-9.11	-25.17	-12.57	-24.44	-9.90	**-22.90**	**-8.63**
superblue18	-51.62	-13.18	-48.21	-13.70	-34.69	-7.40	**-15.19**	-7.20	-15.38	-7.64	-16.16	**-6.92**
Average Ratio	2.33	1.39	2.31	1.39	1.97	1.07	**0.95**	1.12	0.99	1.25	1.00	**1.00**

IV. EXPERIMENTAL RESULTS

We have developed our timing-driven global placer based on the open-source placer DREAMPlace 4.0 released version[1]. We assess the efficacy of our placer using the well-established benchmark suite from the ICCAD 2015 contest [17]. All the evaluations are conducted on a Linux server equipped with a 52-core Intel Xeon CPU at 2.60 GHz, an NVIDIA RTX 2080S GPU, and 128GB of RAM. The hyperparameters are set as follows: $\beta = 2.5 \times 10^{-5}$, $m = 15$, $w_0 = 10$, and $w_1 = 0.2$. The updating rule for the Lagrange multiplier λ is adopted from DREAMPlace. Timing optimization commences at the 500th iteration, a stage where cell distribution has typically stabilized, in line with the configurations specified in DREAMPlace 4.0.

A. Main Results

Table II presents a comprehensive comparison of TNS, WNS, and HPWL metrics between our timing-driven placer and four baseline methods. All DEF results are assessed using the official evaluation kit from the ICCAD 2015 contest to ensure fair comparison. Our approach significantly outperforms the state-of-the-art timing-driven placers, notably Differentiable-TDP [12] and Distribution-TDP [19]. Specifically, our method achieves the best TNS results in seven out of eight test cases, showing an average improvement of 50.0% over Differentiable-TDP and 40.5% over Distribution-TDP. Our placer also shows a consistent 8.3% improvement in WNS compared to these two leading placers. Furthermore, when compared to DREAMPlace [20], including its version 4.0 [18], our results consistently outperform theirs in all eight cases for TNS and WNS. It is surprising that our method surpasses baselines, including the original DREAMPlace, in terms of HPWL in six out of eight cases. This improvement can be attributed to our targeted pin-to-pin attraction strategy, which minimizes the impact on non-critical pins and effectively preserves wirelength quality, unlike DREAMPlace 4.0 which applies weights to numerous nets. What's more, additional timing-driven optimization iterations may further optimize HPWL against density, compared to DREAMPlace with an earlier convergence.

[1]https://github.com/limbo018/DREAMPlace/releases/tag/4.0.0

B. Ablation Study

Table III summarizes the ablation study. The first two columns, 'w/ HPWL Loss' and 'w/ Linear Loss,' replace the quadratic distance loss with HPWL and linear Euclidean losses, respectively. They both fall short of the quadratic loss, consistent with the discussion in Sec. III-C. Nevertheless, they deliver a 15% improvement in TNS over DREAMPlace 4.0 [18], demonstrating the effectiveness of our pin-to-pin attraction modeling and critical path extraction. Furthermore, the superior performance of quadratic loss compared to HPWL/Euclidean loss suggests an advantage over Electrostatics-TDP [21], which relies on HPWL/Euclidean loss for its virtual path modeling. However, the differences in frameworks and datasets make direct comparisons with [21] impossible.

TABLE IV
COMPARISON ON RUNTIME (sec) WITH DREAMPLACE [20] AND DREAMPLACE 4.0 [18]. THE BEST RESULTS ARE IN **BOLD**, AND THE RUNNER-UPS ARE COLORED BROWN.

Benchmark	DREAMPlace	DREAMPlace 4.0	Our Method
superblue1	**122.95**	615.61	531.28
superblue3	**125.34**	798.10	699.86
superblue4	**80.88**	372.28	591.33
superblue5	**147.35**	660.42	714.27
superblue7	**190.32**	926.91	799.40
superblue10	**252.49**	1163.32	1113.07
superblue16	**62.61**	442.99	409.06
superblue18	**56.89**	368.06	301.42
Average Ratio	**0.20**	1.04	1.00

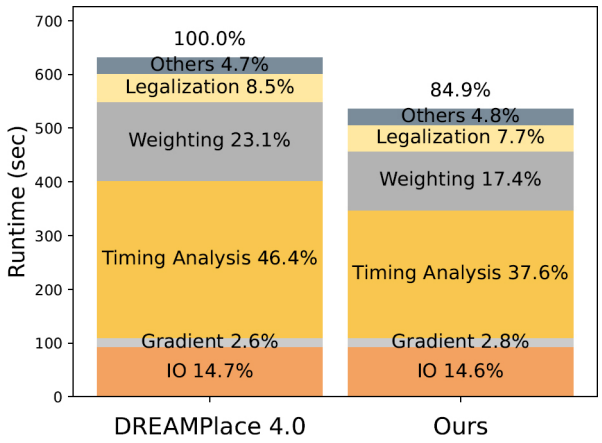

Fig. 4. Runtime breakdown comparison between DREAMPlace 4.0 and our method for case `superblue1`. The time spent by each component is normalized by 615 seconds, the total runtime of DREAMPlace 4.0.

The third column, 'w/ rpt_timing(n*10),' uses OpenTimer's original `report_timing(n*10)` function for critical path extraction, where n is the number of failing endpoints. As shown in Table I, this approach provides insufficient coverage for comprehensive timing analysis, resulting in worse TNS performance. It also requires approximately $10\times$ more computation time compared to our path extraction method which uses `report_timing_endpoint(n,1)`.

The fourth column, 'w/ rpt_timing_ept(n,10),' extracts 10 critical paths (instead of one) per failing endpoint with `report_timing_endpoint(n,10)`. This adjustment improves TNS by incorporating more detailed timing information but slightly degrades WNS and increases computation time.

The fifth column, 'w/o Path Extraction,' replaces our path-level timing analysis with the pin-level timing information and momentum-based weighting scheme proposed by DREAMPlace 4.0 [18]. While this method achieves competitive TNS results, it performs worse in WNS, likely because pin-level analysis fails to consider path-sharing effects, overlooking some critical paths that significantly influence timing.

C. Runtime Analysis and Additional Results

Table IV compares runtime among DREAMPlace [20], DREAMPlace 4.0 [18], and our method across eight designs. DREAMPlace achieves the best runtime in all cases, as it focuses on wirelength without a timing engine which is time consuming. Our method surpasses DREAMPlace 4.0 in most cases thanks to our efficient timing analysis and weighting scheme, as illustrated in Fig. 4. For case `superblue1`, we break down the runtime into key components and normalize each against DREAMPlace 4.0's total runtime for clarity. The reductions in our runtime primarily result from our efficient critical path extraction and pin pair weighting techniques.

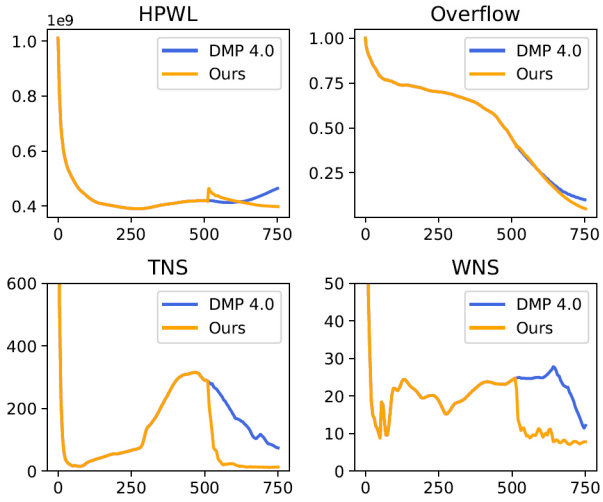

Fig. 5. Optimization iterations for case `superblue1`. The blue curve is DREAMPlace 4.0, and the yellow one is our method. Timing optimization of both methods starts from the **500th** iteration. TNS and WNS values are converted to their absolute values in the figure for better illustration.

Fig. 5 depicts the HPWL, overflow, TNS, and WNS throughout a placement run, comparing our method with DREAMPlace 4.0 [18]. The two curves align until the 500th iteration, at which point timing optimization commences. In the HPWL and Overflow sub-figures, the application of substantial net weights by DREAMPlace 4.0 leads to poorer HPWL performance and a slower convergence rate. Furthermore, our method rapidly enhances TNS and WNS performance, and maintains stability until the optimization fully converges, thereby demonstrating the efficacy of our timing objective design.

V. CONCLUSION

In this work, we introduce a GPU-accelerated, timing-driven global placement framework that integrates a pin-to-pin attraction objective within the popular open-source DREAMPlace framework. We develop a novel critical path extraction method for rapid, precise timing analysis and design a quadratic distance loss function to closely align with the specific timing metrics, thereby enhancing our framework's performance. Experimentation on the ICCAD2015 benchmark suite shows substantial improvements over leading timing-driven placers.

VI. ACKNOWLEDGEMENT

This work was supported by the National Science and Technology Major Project (2022ZD0116600), the National Science Foundation of China (62276124), and the Fundamental Research Funds for the Central Universities (14380020). Ke Xue was supported by National Science Foundation for PhD Students (624B2069). Chao Qian is the corresponding author.

REFERENCES

[1] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proceedings of ACM/IEEE Design Automation Conference*, Las Vegas, NV, 2019, pp. 1–4.

[2] M. Burstein and M. N. Youssef, "Timing influenced layout design," in *Proceedings of ACM/IEEE Design Automation Conference*, Las Vegas, NV, 1985, pp. 124–130.

[3] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov, and A. Zelikovsky, "On wirelength estimations for row-based placement," in *Proceedings of the International Symposium on Physical Design*, Monterey, CA, 1998, pp. 4–11.

[4] C.-C. Chang, J. Lee, M. Stabenfeldt, and R.-S. Tsay, "A practical all-path timing-driven place and route design system," in *Proceedings of the Asia Pacific Conference on Circuits and Systems*, Taipei, Taiwan, 1994, pp. 560–563.

[5] H. Chang, E. Shragowitz, J. Liu, H. Youssef, B. Lu, and S. Sutanthavibul, "Net criticality revisited: An effective method to improve timing in physical design," in *Proceedings of the International Symposium on Physical Design*, San Diego, CA, 2002, pp. 155–160.

[6] W. Chen, H. Huang, M. Wei, P. Zou, and J. Chen, "Virtual-path-based timing optimization for VLSI global placement," in *Proceedings of the International Conference on Solid-State & Integrated Circuit Technology*, Nanjing, China, 2022, pp. 1–3.

[7] A. Chowdhary, K. Rajagopal, S. Venkatesan, T. Cao, V. Tiourin, Y. Parasuram, and B. Halpin, "How accurately can we model timing in a placement engine?" in *Proceedings of ACM/IEEE Design Automation Conference*, Anaheim, CA, 2005, pp. 801–806.

[8] C. Chu, "Placement," in *Electronic Design Automation*. Elsevier, 2009, pp. 635–685.

[9] A. Dunlop, V. Agrawal, D. Deutsch, M. Jukl, P. Kozak, and M. Wiesel, "Chip layout optimization using critical path weighting," in *Proceedings of ACM/IEEE Design Automation Conference*, Albuquerque, NM, 1984, pp. 133–136.

[10] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proceedings of ACM/IEEE Design Automation Conference*, San Francisco, CA, 1998, pp. 269–274.

[11] T. Gao, P. M. Vaidya, and C. L. Liu, "A performance driven macro-cell placement algorithm." in *Proceedings of ACM/IEEE Design Automation Conference*, Anaheim, CA, 1992, pp. 147–152.

[12] Z. Guo and Y. Lin, "Differentiable-timing-driven global placement," in *Proceedings of ACM/IEEE Design Automation Conference*, San Francisco, CA, 2022, pp. 1315–1320.

[13] M.-K. Hsu, Y.-W. Chang, and V. Balabanov, "TSV-aware analytical placement for 3D IC designs," in *Proceedings of ACM/IEEE Design Automation Conference*, San Diego, CA, 2011, pp. 664–669.

[14] T.-W. Huang, G. Guo, C.-X. Lin, and M. D. Wong, "Opentimer v2: A new parallel incremental timing analysis engine," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 4, pp. 776–789, 2020.

[15] M. A. Jackson and E. S. Kuh, "Performance-driven placement of cell based ic's," in *Proceedings of ACM/IEEE Design Automation Conference*, Las Vegas, NV, 1989, pp. 370–375.

[16] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer, 2011.

[17] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan, "ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, Austin, TX, 2015, pp. 921–926.

[18] P. Liao, D. Guo, Z. Guo, S. Liu, Y. Lin, and B. Yu, "DREAMPlace 4.0: Timing-driven placement with momentum-based net weighting and lagrangian-based refinement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 10, pp. 3374–3387, 2023.

[19] J.-M. Lin, Y.-Y. Chang, and W.-L. Huang, "Timing-driven analytical placement according to expected cell distribution range," in *Proceedings of the International Symposium on Physical Design*, Taipei, Taiwan, 2024, pp. 177–184.

[20] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern VLSI placement," in *Proceedings of ACM/IEEE Design Automation Conference*, Las Vegas, NV, 2019, pp. 1–6.

[21] Z. Lin, M. Wei, Y. Chen, P. Zou, J. Chen, and Y.-W. Chang, "Electrostatics-based analytical global placement for timing optimization," in *Proceedings of Design, Automation & Test in Europe Conference*, Valencia, Spain, 2024, pp. 1–6.

[22] W. K. Luk, "A fast physical constraint generator for timing driven layout," in *Proceedings of ACM/IEEE Design Automation Conference*, San Francisco, CA, 1991, pp. 626–631.

[23] I. L. Markov, J. Hu, and M.-C. Kim, "Progress and challenges in VLSI placement research," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, 2012, pp. 275–282.

[24] M. Naresh and S. Sachin, *Timing Analysis and Optimization of Sequential Circuits*. Springer, 1999.

[25] B. Obermeier and F. M. Johannes, "Quadratic placement using an improved timing model," in *Proceedings of ACM/IEEE Design Automation Conference*, San Diego, CA, 2004, pp. 705–710.

[26] D. Z. Pan, B. Halpin, and H. Ren, "Timing-driven placement," in *Handbook of Algorithms for Physical Design Automation*. Auerbach Publications, 2008, pp. 423–446.

[27] W. Swartz and C. Sechen, "Timing driven placement for large standard cell circuits," in *Proceedings of ACM/IEEE Design Automation Conference*, San Francisco, CA, 1995, pp. 211–215.

[28] Z. Wang, Z. Geng, Z. Tu, J. Wang, Y. Qian, Z. Xu, Z. Liu, S. Xu, Z. Tang, S. Kai *et al.*, "Benchmarking end-to-end performance of AI-based chip placement algorithms," *arXiv:2407.15026*, 2024.