INSTA: An Ultra-Fast, Differentiable, Statistical Static Timing Analysis Engine for Industrial Physical Design Applications

Yi-Chen Lu, Zhizheng Guo, Kishor Kunal, Rongjian Liang, and Haoxing Ren
NVIDIA Research
yilu@nvidia.com

***Abstract*—Existing GPU-accelerated Static Timing Analysis (GPU-STA) efforts aim to build standalone engines from scratch but result in poor correlation with commercial tools, limiting their industrial applicability. In this paper, we present INSTA, a tool-accurate, differentiable, GPU-STA framework that overcomes these limitations by initializing timing graphs directly from reference STA tools (e.g., Synopsys PrimeTime). INSTA's core engine utilizes two custom CUDA kernels: a forward kernel for statistical arrival time propagation, and a backward kernel for gradient backpropagation from timing endpoints, enabling two unprecedented capabilities: (1) high-fidelity, rapid timing analysis for incremental netlist updates (e.g., gate sizing), and (2) gradient-based, global timing optimization at scale (e.g., timing-driven placement). Notably, INSTA demonstrates a near-perfect 0.999 correlation with PrimeTime on a 15-million-pin design in a commercial $3nm$ node with runtime under 0.1 seconds. In the experiments, we showcase INSTA's power through three applications: (1) serving as a fast evaluator in a commercial gate sizing flow, achieving 25x faster incremental update_timing runtime with almost no accuracy loss; (2) INSTA-Size, a gradient-based gate sizer that achieves up to 15% better Total Negative Slack (TNS) than PrimeTime's default engine by sizing 68% fewer amount of cells; and (3) INSTA-Place, a differentiable timing-driven placer that outperforms the state-of-the-art net-weighting placer by up to 16% in Half-Perimeter Wirelegnth (HPWL) and 59.4% in TNS.**

I. Introduction

Static Timing Analysis (STA) is the heart of Physical Design (PD) that drives the optimization at each stage from synthesis to signoff to achieve precise trade-offs among Power, Performance, and Area (PPA) objectives. With the relentless pursuit of high-performance and low-power designs, STA's role has become even more critical in advanced technology nodes to ensure reliable chip operation under stringent Process, Voltage, and Temperature (PVT) constraints. Despite its pivotal role, commercial STA tools face significant runtime challenge due to the handling of complex constraints and intricate delay calculations required for timing validation. Furthermore, every fundamental netlist transformation, such as gate sizing and buffering, demands iterative STA iterations to evaluate solution quality. While incremental STA methods [13, 27] aim to improve STA runtime by selectively re-propagating affected netlist portions, they often sacrifice precision in optimization, leading to sub-optimal results.

Given the inherently parallel nature of timing propagation and the growing accessibility of GPU programming, recent research has explored GPU-accelerated STA (GPU-STA) engines, utilizing GPU power for both graph-based (GBA) [7, 9, 25, 30] and path-based (PBA) [4–6] timing analysis. Despite their promise, these approaches often fail to meet the stringent accuracy requirements of industrial applications. They typically rely on simplified delay models, such as the Non-Linear Delay Model (NLDM) for cell delays and the Elmore model for interconnect delays, which lack the precision necessary for advanced technology nodes. In contrast, commercial tools like Synopsys PrimeTime employ proprietary delay models and robustly manage complicated timing constraints. Hence, existing GPU-STA engines demonstrate limited correlation with commercial tools, making them impractical for industrial PD applications.

In this paper, we address these limitations by introducing INSTA, a tool-accurate, differentiable, GPU-STA framework tailored for industrial PD flows. INSTA is GBA-based and adopts a fundamentally different philosophy from previous works [7, 9, 25, 30] by dividing STA into two distinct stages: delay calculation and timing propagation. Rather than constructing an STA engine from scratch to manage intricate delay computations as prior works, INSTA directly "clones" arc delays from a reference tool such as Synopsys PrimeTime and focuses exclusively on ultra-fast, differentiable timing propagation using custom CUDA kernels. This approach combines the strengths of both worlds: (1) unlike prior GPU-STA methods, INSTA achieves high-fidelity correlation with commercial tools, and (2) unlike commercial tools, INSTA enables rapid timing analysis of incremental netlist change and supports gradient-based, truly-global timing optimization subject to various leaf variables.

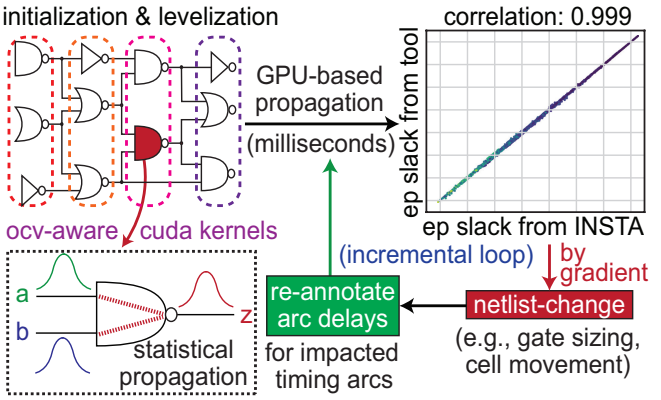

Fig. 1: Overview of INSTA which begins with a one-time initialization from a reference timing engine and performs fast, differentiable, and tool-accurate STA propagation. Notably, INSTA enables gradient computation of global timing metrics (e.g., TNS) with respect to leaf variables (e.g., gate sizes and cell locations) to drive critical PD optimization.

Figure 1 presents an overview of INSTA. Following a one-time initialization from a reference STA engine, INSTA utilizes a forward CUDA kernel for On-Chip Variation (OCV)-aware timing propagation and a backward kernel for gradient back-propagation. Experimental results demonstrate that INSTA achieves a near-perfect correlation (0.9999) in individual endpoint slack values with an industry-leading signoff tool (tool name is withheld in compliance with license agreement) across several real-world high-performance designs in signoff mode. Note that while INSTA is initialized from a reference tool, achieving this level of correlation is far from trivial. INSTA meticulously handles distribution-based timing propagation, timing exceptions, and Common Path Pessimism Removal (CPPR) to manage rise/fall conditions and unateness constraints. Beyond tool-accurate correlation, INSTA supports gradient computations of leaf optimization variables (e.g., gate sizes, cell locations) with respect to global timing metrics including Worst Negative Slack (WNS) and Total Negative Slack (TNS). This capability enables fast and precise identification of timing bottlenecks across cells, nets, stages, etc., facilitating targeted and effective PD optimization. In this work, we

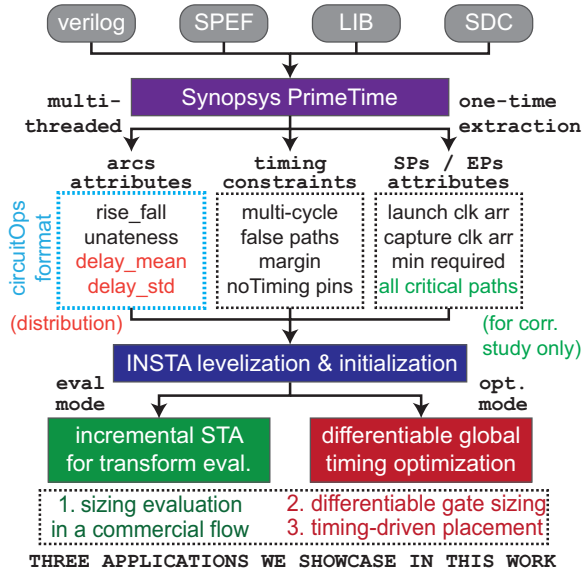

Fig. 2: Overview of INSTA's initialization process from a reference commercial STA tool, along with its evaluation and optimization capabilities.

present related applications in gate sizing and global placement.

We envision INSTA as a powerful tool to reshape existing industrial PD flows. In this paper, we specifically highlight three novel applications of INSTA. First, we utilize INSTA's fast timing analysis capability within an iterative gate-sizing scheme, demonstrating significant runtime improvements in each sizing iteration with almost no compromise on accuracy or Quality of Results (QoR). Then, we apply INSTA to differentiable gate sizing and differentiable timing-driven placement, illustrating its potential for driving timing optimization in PD tasks. The goal of this work is to demonstrate how INSTA can transform existing CPU-centric PD flows. We believe this paves the way for future works to revisit traditional PD tasks with INSTA.

Our main contributions are as follows:

- We present INSTA, the first-ever differentiable GPU-STA framework that performs statistical timing propagation with near-perfect endpoint slack correlation to Synopsys PrimeTime, an industry-standard timing signoff tool, in a commercial $3nm$ node with OCV enabled. On our largest block with **15 millions** pins, INSTA performs full-graph timing propagation in less than **0.1 seconds** with 0.999 correlation to Synopsys PrimeTime.
- We develop a GPU-accelerated Top-K statistical arrival propagation CUDA kernel that efficiently manages CPPR, a must-handle timing pessimism in advanced technology nodes.
- We demonstrate INSTA's capability as a fast timing evaluator in a commercial gate sizing flow, where it brings **25x** runtime improvement over PrimeTime's incremental timing analysis.
- We propose the concept of "timing gradients" to enable truly-global, differentiable timing optimization in traditional PD tasks. We introduce INSTA-Size and INSTA-Place, showing that INSTA-Size outperforms the signoff timing optimization engine in PrimeTime by up to **15**% in TNS with sizing **68**% less amount of cells, while INSTA-Place outperforms the state-of-the-art net-weighting-based timing-driven placer [19] by up to **59.4**% in TNS and **16.2**% in Half-Perimeter Wirelength (HPWL).

II. Related Work

A. GPU-Accelerated Timing Propagation

GPU-STA engines have been proposed for graph-based [7, 9, 25, 30] and path-based [4–6] timing analysis, achieving notable speedups on academic benchmarks. However, they rely on over-simplified delay models [20], omit essential industrial requirements such as OCV and CPPR, and lack fidelity to commercial tools. While CPPR-focused methods [8, 12] attempt to address these gaps, their accuracy and runtime are impractical for industrial use. Furthermore, none of these frameworks [4–9, 12, 25, 30] support differentiable timing propagation, limiting their utility for modern PD applications.

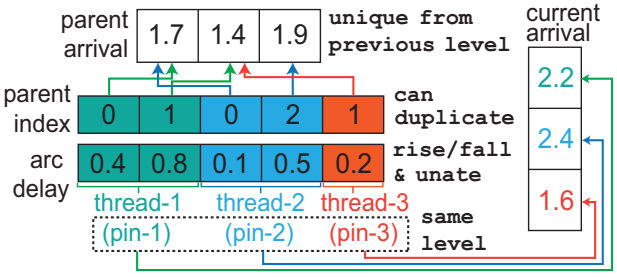

Fig. 3: Illustration (simplified) of arrival time merging in INSTA using GPU, where each pin on the same timing level is mapped to a CUDA thread. Refer to Algorithm 1 for the complete procedure.

B. Differentiable Timing Optimization

Differentiable STA has recently gained traction with the rise of Machine Learning (ML) frameworks like PyTorch [26]. [10] introduces a differentiable placement framework, but its reliance on simplified delay models again limits its usage. Another study [29] leverages ML-based timing predictions [11] to enable gradient computation, but its learning-based nature restricts generalizability, performing 60x slower than commercial tools even on an outdated $130nm$ node. These limitations [10, 29] underscore the need for a framework that combines tool-level accuracy with efficient optimization.

III. Algorithm

A. INSTA's Philosophy and Initialization

The core design principle of INSTA is to achieve tool-accurate, GPU-accelerated timing analysis (GBA), while enabling gradient-based optimization over various leaf variables. However, this alignment is far from trivial. While arc delays can be cloned, replicating the exact timing propagation procedure in a GPU-accelerated manner requires careful handling of OCV, CPPR, and timing constraints. Figure 2 provides an overview of INSTA's initialization process. Inputs include arc attributes in consideration of rise/fall conditions and unateness, timing exceptions (e.g., multi-cycle and false paths), and SP/EP attributes (e.g., clock arrival times, minimum required times). All are extracted using multi-threaded CPU processing in PrimeTime via custom TCL scripts into the CircuitOps [18] format. Even on million-gate industrial designs, this extraction only takes around 10 minutes. Following extraction, INSTA constructs and levelizes its timing graph for parallel arrival time propagation, where pins within the same level are processed independently. This levelization process is achieved by topological sort using Graph-Tool [28], which completes in just a few seconds on a million-node graph.

B. OCV Consideration in INSTA: Arrival Times as Distributions

In modern industrial PD flows, STA engines must account for OCV to ensure reliable chip operation under extreme PVT conditions. Although various OCV models are employed across different commercial tools, in this work, we choose to focus on replicating the Parametric OCV (POCV) propagation used in Synopsys PrimeTime [1]. To achieve this, INSTA models the arrival time at each pin as a Gaussian distribution that is characterized by a mean and a standard deviation (std). Based on the variational arc delay attributes extracted from PrimeTime as shown in Figure 2, our forward CUDA kernel propagates both the mean and standard deviation of these delay distributions at each timing level, where for an arc $j \rightarrow i$ with its

Algorithm 1 CUDA Kernel for Top-K Statistical **Unique** Arrival Propagation at Output Pins. Considering Rise/Fall and Unateness.

Input: TopK: Number of unique arrivals to store per pin, μ_{arc}: Arc increment means, σ_{arc}: Arc increment standard deviations, $\mathcal{U}$: Arc unateness, $\mathcal{N}_\sigma$: Level of pessimism, N: Number of pins, $outPin\_parent\_start$: 1D output pins to input pins mapping array.
Output: Top-K arrival distributions per pin subject to **unique** startpoints.

Pre-Kernel Initialization of Final TopK Structures:
Initialize topK_{rise/fall}_{arrivals, means, stds, SPs} $\in \mathbb{R}^{N\times 2\times K}$
Pre-Kernel Initialization of Temporary Priority Queues:
$\{\mathcal{A}, \mu, \sigma, SP\}_{\text{top-K}} \leftarrow \{\} \in \mathbb{R}^{N\times K}$ ▷ arrival, mean, std, startpoint
CUDA Kernel Launch to Parallelize Arrival Merging of Arcs:
outPinID $\leftarrow$ blockIdx.x $\cdot$ blockDim.x + threadIdx.x
if outPinID $\geq N$ **then return** ▷ each maps to a CUDA thread
offsetL $\leftarrow$ $outPin\_parent\_start$[outPinID]
offsetR $\leftarrow$ $outPin\_parent\_start$[outPinID+1]
for each rf $\in$ {rise,fall}, $k \in \{0,\ldots,\text{TopK}-1\}$ **do**
  for each pID = offsetL **to** offsetR-1 **do** ▷ parent in prev level
    pRF $=\sim$ rf **if** $\mathcal{U}$[pID] == negative_unate **else** rf
    pMean $\leftarrow$ topK_means[pID][pRF][k] ▷ from previous level
    pStd $\leftarrow$ topK_stds[pID][pRF][k]
    arr_mean $\leftarrow$ pMean$+\mu_{\text{arc}}$[pID][rf]
    arr_std $\leftarrow \sqrt{\text{pStd}^2+\sigma_{\text{arc}}[\text{pID}][\text{rf}]^2}$
    arrival $\leftarrow$ arr_mean$+\mathcal{N}_\sigma\cdot$arr_std ▷ $\mathcal{A}_{new}$
    $\mu_{new} \leftarrow$ arr_mean, $\sigma_{new} \leftarrow$ arr_std
    $SP_{new} \leftarrow$ topK_SPs[pID][pRF][k]
    Update $\{\mathcal{A}, \mu, \sigma, SP\}_{\text{top-K}}$ at outPinID with **Algorithm 2**
  for each $k \in \{0,\ldots,\text{TopK}-1\}$ **do**
    topK_arrivals[outPinID][rf][k] $\leftarrow \mathcal{A}_{\text{top-K}}$[outPinID][$k$]
    topK_means[outPinID][rf][k] $\leftarrow \mu_{\text{top-K}}$[outPinID][$k$]
    topK_stds[outPinID][rf][k] $\leftarrow \sigma_{\text{top-K}}$[outPinID][$k$]
    topK_SPs[outPinID][rf][k] $\leftarrow SP_{\text{top-K}}$[outPinID][$k$]
  Free $\{\mathcal{A}, \mu, \sigma, SP\}_{\text{top-K}}$

specific variation attributes $arc\_mean_{j\to i}$ and $arc\_std_{j\to i}$, the final arrival time at the sink i is calculated as:

$$arrival\_mean_i = arrival\_mean_j + arc\_mean_{j\to i}, \quad (1)$$

$$arrival\_std_i^2 = arrival\_std_j^2 + arc\_std_{j\to i}^2, \quad (2)$$

$$arrival_i = arrival\_mean_i + N_\sigma * arrival\_std_i, \quad (3)$$

where N_σ controls the level of pessimism. Essentially, the arrival time at the pin i denotes the corner value of its propagated distribution. In the implementation, we set $N_\sigma = 3.0$ in alignment with PrimeTime.

C. CPPR Handling in INSTA via Priority Queues

CPPR is essential for accurate timing analysis in industrial design flows, as it mitigates pessimism introduced by clock paths to provide an accurate timing landscape. Prior CPU-based STA methods have attempted to manage CPPR through branch-and-bound techniques for search-space pruning [17, 31], block-based algorithms with alternative delay metrics to reduce pessimism [15], and Lowest Common Ancestor (LCA) path tracing [12]. However, these methods often suffer from substantial runtime overhead or accuracy trade-offs. Recent efforts to utilize GPU parallelism for CPPR path tracing [8] have improved performance but still fall short of industrial requirements.

In INSTA, CPPR is addressed through an efficient GPU-accelerated method that employs Top-K arrival time propagation while ensuring each arrival distribution is mapped to a distinct startpoint. Particularly, priority queues are dynamically maintained at each pin to efficiently track and update the Top-K critical arrival distributions with respect to unique timing startpoints. The key rationale is that clock pessimism arises when multiple timing startpoints converge at a single endpoint along distinct clock paths, resulting in a scenario, where the startpoint contributing to the maximum arrival time at an endpoint, differs from the startpoint introducing the minimum required time, which causes a shift in the endpoint's slack. Hence, with the Top-K arrival times that are mapped to unique startpoints and by initializing endpoint required times from a commercial reference tool that are also associated with respective startpoints (as illustrated in Figure 2), INSTA ensures that the correct arrival time is used to compute endpoints' slack values for tool-accurate timing analysis.

Algorithm 2 GPU-Accelerated Priority Queue Update for Top-K Arrival Propagation with **Unique** Startpoints. (Line 17 of Algorithm 1)

Input: (all from **Algorithm 1**): New arrival time $\mathcal{A}_{\text{new}}$ subject to startpoint $\mathcal{SP}_{\text{new}}$, top-K queues: arrival times $\mathcal{A}_{\text{top-K}}$, means $\mu_{\text{top-K}}$, standard deviations $\sigma_{\text{top-K}}$, startpoints $SP_{\text{top-K}}$.
Output: In-place update of Top-K queues.

At Each GPU Parallel Thread Assigned to an Output Pin:
inserted $\leftarrow$ **false**, idx $\leftarrow -1$ ▷ Initialize control flags
Step 1: Check for Startpoint Uniqueness and Update if Found
for each $(A_j, SP_j) \in$ **zip**($\mathcal{A}_{\text{top-K}}$, $\mathcal{SP}_{\text{top-K}}$) **do**
  if $SP_j = SP_{\text{new}}$ **then**
    if $A_{\text{new}} > A_j$ **then**
      $\mathcal{A}_{\text{top-K}}[j] \leftarrow A_{\text{new}}$, $\mathcal{SP}_{\text{top-K}}[j] \leftarrow \mathcal{SP}_{\text{new}}$
      $\mu_{\text{top-K}}[j] \leftarrow \mu_{\text{new}}$, $\sigma_{\text{top-K}}[j] \leftarrow \sigma_{\text{new}}$
      inserted $\leftarrow$ **true**
    break ▷ Exit once an existing SP_{new} is found
Step 2: Insert New Startpoint if Unique
if not inserted **then** ▷ Only if SP_{new} is not found
  for $j = 0$ **to** $K-1$ **do**
    if $A_{\text{new}} > A_j$ **then** ▷ Found the first lower value
      for $l = K-1$ **down to** $j+1$ **do** ▷ Shifting in reversal!
        $\mathcal{A}_{\text{top-K}}[l] \leftarrow \mathcal{A}_{\text{top-K}}[l-1]$
      $\mathcal{A}_{\text{top-K}}[j] \leftarrow A_{\text{new}}$, $\mathcal{SP}_{\text{top-K}}[j] \leftarrow \mathcal{SP}_{\text{new}}$
      $\mu_{\text{top-K}}[j] \leftarrow \mu_{\text{new}}$, $\sigma_{\text{top-K}}[j] \leftarrow \sigma_{\text{new}}$
      inserted $\leftarrow$ **true**
      break

D. GPU Kernel Implementation of Top-K Arrival Propagation

Figure 3 shows how INSTA uses GPU parallelism to perform arrival time propagation in large-scale timing analysis. At each timing level, an index array in GPU shared memory maps each thread, representing a pin at the current level, to its parent pins. This mapping enables accurate merging of arrival distributions across multiple incoming arcs while handling rise/fall conditions and unateness constraints. Algorithm 1 provides the detailed steps for this process, focusing on output pins where multiple arcs converge. Note that for input pins, GPU kernels are not required, as each input pin is connected to a single parent in modern digital designs. Hence, INSTA employs a CPU-based vectorized approach, which avoids the overhead of data transfer to GPU while being computationally efficient.

The update of priority queues for CPPR handling is detailed in Line 17 of Algorithm 1, with the specific procedures outlined in Algorithm 2, which ensures that each Top-K entry corresponds to a unique startpoint while dynamically maintaining the Top-K arrival times. The update procedure works as follow. If the new incoming startpoint $\mathcal{SP}_{\text{new}}$ already exists in the queue, its arrival time $\mathcal{A}_{\text{new}}$ is compared with the existing value, and the entry is updated if $\mathcal{A}_{\text{new}}$ is larger. If $\mathcal{SP}_{\text{new}}$ is not found in the queue, the algorithm checks whether $\mathcal{A}_{\text{new}}$ exceeds the smallest arrival time in the current list. If so, the new values are inserted in the appropriate position to maintain descending order, with entries shifted as needed.

E. INSTA Complexity Analysis: Why not Heaps for Priority Queues?

In conventional priority queue implementations, heaps are commonly used due to their $O(\log K)$ complexity for insertions and deletions. However, implementing a heap on GPUs incurs substantial overhead because maintaining a binary tree structure requires frequent reordering and re-balancing of elements. These operations are computationally intensive and poorly suited to the highly parallel architecture of

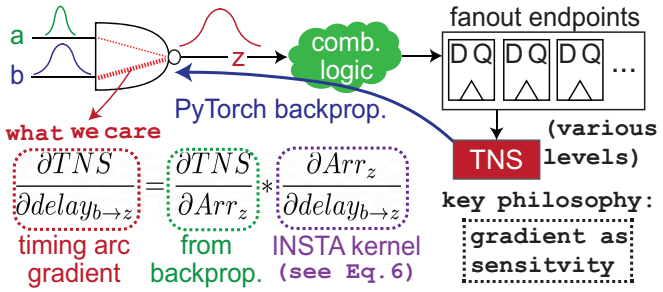

Fig. 4: Illustration of timing gradients in INSTA. For either cell arc or net arc, we can compute the "timing gradient" that quantifies its contribution to global timing metrics such as WNS and TNS.

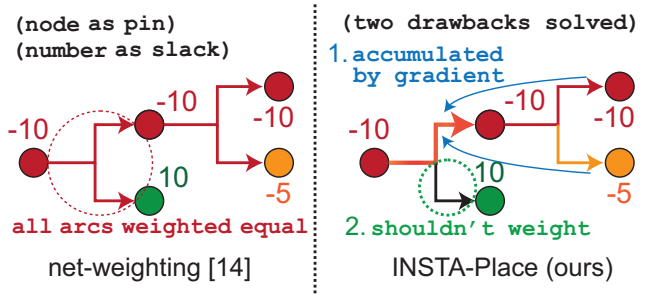

Fig. 5: Illustration of arc-based gradient for timing-driven placement in INSTA-Place that solves the two key drawbacks of the state-of-the-art net-weighting-based approach [19].

GPUs, where hundreds thousands of threads operate simultaneously. In INSTA, we use a fixed-size list to manage the Top-K elements, avoiding the overhead of heap-based operations. As shown in the two algorithms, each CUDA thread operates independently on its own lists, performing $O(K^2)$ comparisons and shifts to maintain the sorted order. Hence, with the GPU ensuring parallel processing across threads in $O(1)$ per level, the overall time complexity of INSTA's timing propagation is $O(K^2 \cdot L)$, where K denotes Top-K and L is the number of timing levels.

F. Enabling Differentiable Timing Propagation

In Algorithm 1 and Algorithm 2, the "greater than" operation identifies the maximum arrival time at a pin from multiple incoming paths. While effective for evaluation, it introduces non-differentiability, making it unsuitable for optimization. The limitation arises from its exclusivity, propagating only the arrival distribution of the most critical path and blocking gradient flow from other inputs. This approach is particularly problematic when multiple input paths provide near-critical arrival times, as it disregards opportunities to optimize other sub-critical paths to improve overall timing. To overcome this, we adopt the numerically stable Log-Sum-Exponential (LSE) operator, a differentiable approximation of the maximum function. The LSE operator ensures smoother gradient distribution across all inputs, enabling more comprehensive optimization. It is defined as:

$$\mathrm{LSE}(\{\mathcal{A}_i\}_{i=1}^n) = M + \tau \cdot \log\left(\sum_{i=1}^{n} \exp\left(\frac{\mathcal{A}_i - M}{\tau}\right)\right), \quad (4)$$

where $\{\mathcal{A}_i\}$ denotes the set of arrival times at a given pin, $M = \max(\{\mathcal{A}_i\}_{i=1}^n)$ ensures numerical stability, and $\tau > 0$ is a temperature parameter controlling the degree of smoothness. As $\tau \to 0$, the LSE operator converges to the standard maximum operator as:

$$\lim_{\tau \to 0} \mathrm{LSE}(\{\mathcal{A}_i\}_{i=1}^n) = \max(\{\mathcal{A}_i\}_{i=1}^n). \quad (5)$$

With Equation 4, the gradient of LSE can be computed in INSTA's backward kernel with respect to each arrival time $\mathcal{A}_i$ as:

$$\frac{\partial \mathrm{LSE}(\{\mathcal{A}_i\}_{i=1}^n)}{\partial \mathcal{A}_i} = \frac{\exp\left(\frac{\mathcal{A}_i - M}{\tau}\right)}{\sum_{j=1}^{n} \exp\left(\frac{\mathcal{A}_j - M}{\tau}\right)}, \quad (6)$$

which effectively assigns a softmax-like weight to each path. The term $\exp\left(\frac{\mathcal{A}_i - M}{\tau}\right)$ ensures that paths with larger arrival times receive higher contributions, while smaller paths contribute proportionally less. Notably, this formulation ensures full differentiability, allowing gradient descent to consider full timing graph during optimization.

G. Timing Gradient: Key for Truly-Global Timing Optimization

For the first time, in the realm of PD, INSTA introduces the concept of "timing gradients" to enable arc-based, truly-global timing optimization for industrial-grade PD applications. The key idea is that the timing gradient of each arc, accounting for rise/fall and unateness conditions, precisely quantifies its contribution to global timing metrics such as WNS and TNS, which enables targeted and fine-grained PD optimization. If we draw an analogy between INSTA and modern ML models, each pin in the timing graph corresponds to a neuron, and the pin-to-pin arc delays can be thought as weights. Similar to how forward propagation in a ML model that computes a differentiable loss function with respect to the weights, INSTA computes timing metrics in a fully differentiable manner.

Figure 4 illustrates the computation of timing gradients in INSTA. Leveraging PyTorch's C++/CUDA extension [26] (similar to DREAMPlace [21]), INSTA integrates PyTorch's auto-differentiation capabilities to perform efficient gradient backpropagation. This design choice allows INSTA to delegate the complicated global gradient flow to the PyTorch framework while focusing solely on gradient computations with a custom backward CUDA kernel at each timing level. As illustrated in the purple box of Figure 4, INSTA's backward kernel computes the gradient of merged arrival times at a destination pin to incremental changes in arc delays (e.g., $\frac{\partial Arr_z}{\partial delay_{b \to z}}$), following the formulation in Equation 6. The timing gradient of each arc is subsequently derived by combining this kernel's output with the backpropagated gradient contributions from the previous fanout level. In the experiments, we demonstrate how to leverage "timing gradient" to drive critical PD optimization including gate sizing and placement.

H. INSTA-Size: Pinpoint Gate Sizing with Timing Gradients

As INSTA computes the "timing gradient" of each arc in a GPU-accelerated manner, it enables **instantaneous** identification of timing-critical cells or nets within a design. This capability allows us to quickly pinpoint which elements contribute most significantly to global timing metrics such as TNS and prioritize them 'for timing improvement. Leveraging this concept, we introduce INSTA-Size, a fast, gradient-based gate sizing framework designed to optimize timing using the gradients calculated by INSTA. The core idea is that following the one-time initialization, a backward pass on the TNS metric computed by INSTA yields the timing gradient of each "stage" (i.e., the gradient sum of a cell arc and its driving net arc), which quantifies its contribution to the overall TNS. Stages with gradients above a pre-defined threshold are prioritized by magnitude. Starting with the most critical stage, PrimeTime's `estimate_eco` is used to determine the library cell that provides the most delay improvement for that stage, which is then committed and rolled back if TNS degrades. Note that `estimate_eco` is a highly parallelizable command, which computes local delay change estimates of gate sizing for millions of arcs within seconds [1]. A recent work [23] has shown that the accuracy of `estimate_eco` is good enough to drive commercial sizing optimization. Nonetheless, as `estimate_eco` operates under the assumption that the neighboring cells remain unchanged for estimation, INSTA-Size blocks the 3-hop neighborhood of a committed stage from further consideration to mitigate interference, which aligns with the strategies used in [23, 24].

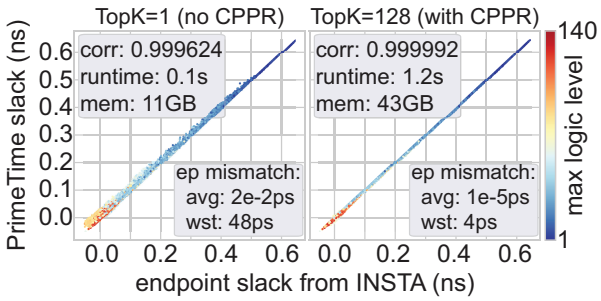

Fig. 6: INSTA vs. Synopsys PrimeTime endpoint slack correlation on block-1 with 4M cells and 15M pins. Top-K=1 and Top-K=128 propagation are compared, showing the trade-off of CPPR consideration.

I. INSTA-Place: INSTA for Timing-Driven Placement

Now, we introduce INSTA-Place, which is a timing-driven placer that leverages timing gradients from INSTA for global placement optimization. By integrating with DREAMPlace [21] and OpenTimer [14], INSTA-Place addresses the fundamental limitations in the state-of-the-art net-weighting approach [19], as shown in Figure 5, via an arc-specific weighting strategy guided by timing gradients. Particularly, the two critical issues that INSTA-Place solves are:

- **Locality of slack-based adjustment**: Current methods assign net weights primarily based on the most critical sink, leading to near-uniform weighting for arcs along the same worst critical path. However, instead of relying on "slack values", the weighting strategy should reflect each arc's contribution (i.e., gradient) to global timing metrics such as TNS.
- **Equal weighting for all arcs in a net**: Existing methods assign the same weight to all arcs in a net, which is inherently suboptimal. Often, only a subset of sinks in a net are timing-critical, while the others are not. Blindly forcing all pins closer together degrades overall placement quality.

INSTA-Place elegantly resolves these limitations by using an arc-based, "gradient as sensitivity" approach by weighting pin-to-pin Manhattan distances by their respective timing gradients from INSTA. Specifically, at each placement iteration, given the current pin locations $\mathbf{P} = \{(x_i, y_i)\}_{i=1}^{N}$, the critical timing arc indices $\mathbf{I} = \{(f_k, t_k)\}_{k=1}^{M}$ (where f_k and t_k denote the indices of the driver and sink of the k-th arc), and the associated timing gradients $\mathbf{G} = \{g_k\}_{k=1}^{M}$ from INSTA, we define the timing objective as:

$$L_{\text{timing}} = \lambda_{RC} \cdot \sum_{k=1}^{M} \left(|x_{f_k} - x_{t_k}| + |y_{f_k} - y_{t_k}|\right) \cdot g_k, \tag{7}$$

where λ_{RC} denotes the global time constant scaling that reflects the RC delay per wirelength. To integrate timing-awareness into the global placement process, we combine L_{timing} with the default wirelength L_{WL} and density L_{DEN} objectives as:

$$L = \underbrace{L_{\text{WL}} + \lambda_1 L_{\text{den}}}_{\text{default objective}} + \underbrace{\lambda_2 L_{\text{timing}}}_{\text{INSTA-Place}}, \text{ s.t. } \lambda_2 = \frac{\|\nabla(L_{\text{WL}} + \lambda_1 L_{\text{den}})\|}{\|\nabla L_{\text{timing}}\|} \tag{8}$$

where λ_1 is the default density scaling factor, and λ_2 dynamically adjusts the contribution of L_{timing} relative to the default wirelength and density objectives by aligning the Euclidean norm $\|\cdot\|$ of their gradients. For computational efficiency, INSTA-Place refreshes the timing graph with OpenTimer [14] every 15 iterations as the practice in [19], while reusing last-computed timing gradients for the remaining 14 iterations. This approach ensures accurate timing analysis without incurring excessive computational overhead.

IV. Experimental Results

In this section, we first present a detailed correlation analysis between INSTA and Synopsys PrimeTime with 5 real-world million-gate industrial designs in a commercial $3nm$ technology node. Then, we demonstrate INSTA's capabilities across three PD applications as illustrated in Figure 2. INSTA is implemented using PyTorch [26] 2.1.2 and CUDA 12.1. All experiments are conducted on a Linux system equipped with a single NVIDIA A100 GPU (96GB memory) and an AMD EPYC 7742 64-core processor with 2TB RAM.

TABLE I: Timing correlation study between INSTA and PrimeTime (in signoff mode) on 5 industrial designs in a commercial $3nm$ node. UT refers to the runtime of PrimeTime's `update_timing` with 32 threads.

designs (TopK=32) # cells, # pins, UT(min)	ep slack corr. (top 5 digits)	runtime (second)	memory (GB)	ep mismatch (avg, wst) **ps**
block-1 (4M, 15M, 68)	0.99994	0.39	21.13	(7e-4, 17)
block-2 (2M, 6M, 32)	0.99999	0.35	5.98	(1e-4, 3)
block-3 (3M, 9M, 39)	0.99992	0.37	11.94	(1e-3 , 9)
block-4 (2M, 7M, 29)	0.99996	0.35	9.47	(2e-4, 6)
block-5 (2M, 6M, 33)	0.99999	0.33	5.81	(1e-4, 5)

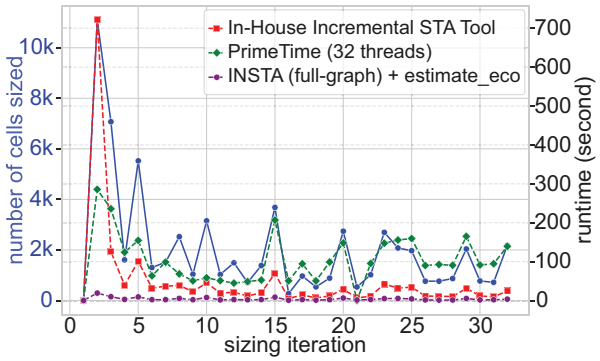

Fig. 7: Incremental STA runtime comparison (same changelist) on block-2 among an in-house CPU-based STA engine, Synopsys PrimeTime with 32 threads, and INSTA with `estimate_eco` for arc delay re-annotation. See Figure 8 for INSTA's correlation impact with `estimate_eco`.

A. Correlation Study: INSTA vs. PrimeTime in Signoff Mode

In this experiment, we validate INSTA's timing correlation against Synopsys PrimeTime. Figure 6 shows the endpoint slack correlation results as scatter plots on block-1 (4M cells, 15M pins), with the left plot achieved without CPPR handling (Top-K=1) and the right plot with consideration of CPPR (Top-K=128). Each dot in the figure represents an endpoint and is colored by its maximum level, highlighting INSTA's accuracy across varying timing depths. With CPPR handling, correlation is improved with expected trade-offs in runtime and memory. However, even without CPPR (Top-K=1), INSTA shows exceptional accuracy, with an average absolute mismatch per endpoint of only **0.02ps**, indicating near-perfect alignment with the 45-degree line. Table I presents detailed timing correlation results across 5 industrial designs with a fixed Top-K=32. Notably, the PrimeTime's `update_timing` runtimes are listed only for reference rather for direct comparison with INSTA, as INSTA focuses solely on timing propagation using arc delays initialized from PrimeTime to enable incremental timing evaluation and differentiable timing optimization. However, it is worth to note that since GPU parallelism amortizes the computation at each timing level to $O(1)$, INSTA's runtime scales with the number of timing levels under a fixed Top-K rather than the total number of pins or cells as PrimeTime.

B. Application-1: Timing Evaluation in a Commercial Sizing Flow

Now, we evaluate INSTA's capability as a timing evaluator within a commercial gate sizing flow for timing-constrained power optimization in a $3nm$ node, where incremental STA is used after each iteration to ensure timing consistency. In this experiment, we benchmark INSTA's runtime and correlation against two other STA engines in incremental mode: an in-house, highly-optimized STA

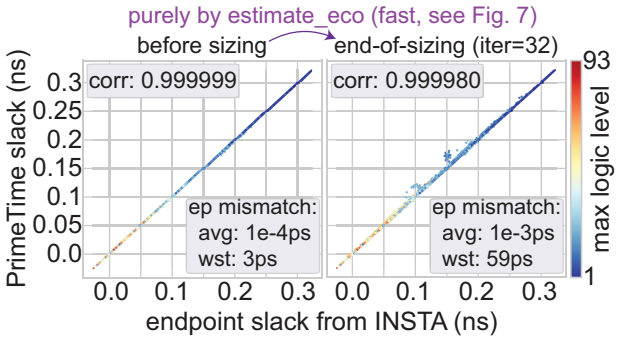

Fig. 8: INSTA correlation impact with `estimate_eco` re-annotation before and after a commercial gate sizing flow for power optimization. The correlation is still high enough to drive the optimization.

TABLE II: Gate sizing for timing optimization results between INSTA-Size and Synopsys PrimeTime on the IWLS benchmark [2]. bRT denotes INSTA's runtime of identifying critical stages via the backward kernel.

design (# pins) (bRT, RT_{pt})	method	WNS (ps)	TNS (ps)	# vio. eps	# cells sized
aes (24k) (0.02s, 33s)	initial state	-27.59	-1115.81	149	–
	PrimeTime	-27.59	-1206.12	152	182
	INSTA-Size	-27.59	-1025.15	137	113 (-38%)
cipher_top (50k) (0.02s, 42s)	initial state	-69.24	-11232.28	317	–
	PrimeTime	-69.00	-11187.22	314	431
	INSTA-Size	-68.07	-10924.20	311	254 (-41%)
des (11k) (0.03s, 14s)	initial state	-114.61	-5546.59	113	–
	PrimeTime	-114.50	-5498.41	112	186
	INSTA-Size	-113.54	-5360.59	110	121 (-35%)
mc_top (25k) (0.02s, 22s)	initial state	-203.01	-77653.10	776	–
	PrimeTime	-203.00	-77694.79	774	156
	INSTA-Size	-203.01	-75307.32	776	49 (- 68%)

tool and Synopsys PrimeTime with 32 threads. Figure 7 shows the incremental STA runtime comparison at each sizing iteration over the exact same changelist across these three tools. Particularly, it is important to note that the runtime of INSTA includes both timing propagation time and the arc delay re-annotation time from PrimeTime's `estimate_eco`. Figure 8 further evaluates INSTA's correlation before and after the complete gate sizing flow against PrimeTime. While `estimate_eco` introducing minor inaccuracies which can be observed in the right figure, INSTA maintains extremely high correlation throughout the entire optimization process for the vast majority of endpoints, delivering a **14x** speed-up over the highly-optimized in-house STA engine, and a **25x** runtime improvement over PrimeTime's incremental `update_timing`. We want to emphasize that in practice, any significant accuracy concerns can be addressed by re-synchronizing INSTA with PrimeTime-calculated arc delays, as shown in Figure 2, which takes only 10 minutes for million-gate industrial designs. Note that in this experiment, we do not perform this re-synchronization for benchmarking purpose.

C. Application-2: INSTA-Size for Timing Optimization

In this experiment, we assess INSTA-Size's capability in gate sizing for timing optimization using the "timing gradients" as described in Section III-H. A head-to-head comparison between INSTA-Size and PrimeTime is conducted on the IWLS benchmark suite [2] in the ASAP $7nm$[3] node, with $\tau = 0.01$ set in Equation4 for INSTA-Size. Table III-H presents the optimization results, demonstrating that INSTA-Size achieves up to 15% improvement in TNS compared to PrimeTime's default timing optimization engine, while requiring significantly fewer cell modifications across all benchmarks (**42%** less on average). We attribute this significant success to the effectiveness of INSTA's timing gradients, which offer a precise and instantaneous identification of the most critical stages in the design and their corresponding sensitivities to TNS (i.e., gradients as sensitivities).

TABLE III: Timing-driven placement results **after legalization** between INSTA-Place and the state-of-the-art net-weighting-based approach [19] on ICCAD 2015 benchmarks [16]. The unit for TNS is $10^5 ps$. DP denotes DREAMPlace. Refer to Figure 5 for detailed runtime comparison.

ICCAD'15 benchmark	DP [21]		DP 4.0 [19]		INSTA-Place (ours)	
	HPWL	TNS	HPWL	TNS	HPWL	TNS
Superblue1	410.4	-312.5	481.6	-85.3	443.1(-8.0%)	-34.6(-59.4%)
Superblue3	469.3	-70.3	482.7	-50.4	472.2(-2.2%)	-40.4(-19.8%)
Superblue4	317.3	-191.1	343.7	-144.0	333.9(-2.9%)	-114.5(-21%)
Superblue5	487.9	-211.6	533.5	-102.1	478.5(-10.3%)	-93.7(-8.2%)
Superblue7	600.2	-167.3	604.1	-64.7	593.0(-1.8%)	-57.6(-10.9%)
Superblue10	916.0	-756.0	1088.3	-671.3	911.9(-16.2%)	-628.2(-6.4%)
Superblue16	424.4	-250.1	460.2	-71.4	458.7(-0.3%)	-37.6(-47.3%)
Superblue18	237.1	-92.2	248.6	-47.5	242.6(-2.4%)	-35.8(-24.6%)
					avg. -5.5%	avg. -24.7%

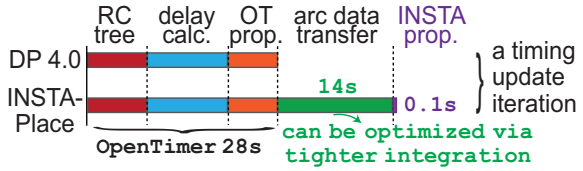

Fig. 9: Detailed runtime comparison between [19] and INSTA-Place on Superblue10 (largest benchmark, 5.6M pins) in a timing update iteration.

D. Application-3: INSTA-Place for Timing-Driven Global Placement

In this experiment, we evaluate INSTA-Place against DREAMPlace 4.0 [19], the state-of-the-art open-source net-weighting-based timing-driven placer, using the well-established ICCAD 2015 benchmarks [16]. The methodology of INSTA-Place is detailed in Section III-I, with post-legalization results (by ABCDPlace [22]) summarized in Table III with λ_{RC} set to around 0.001. It is shown that INSTA-Place achieves up to 59.4% TNS improvement and 16.2% wirelength reduction compared to [19]. Figure 9 details the runtime comparison between INSTA-Place and [19] on Superblue10 over a timing update iteration. While INSTA-Place incurs a 50% runtime overhead due to the data transfer between OpenTimer [14] and INSTA, this overhead can be minimized through tighter integration (e.g., shared data structures). Notably, runtime is not the focus of this experiment. What we aim to demonstrate is that by addressing the fundamental shortcomings of [19] as highlighted in Figure 5 through the arc-based weighting strategy (Equations 7 and 8), INSTA-Place can significantly advance the timing quality of global placement. Finally, while prior work [10] also attempts differentiable timing-driven placement by approximating gradient propagation over non-differentiable variables such as Steiner points, this approach is error-prone and poorly correlates with reference STA tools. In contrast, INSTA-Place employs arc-based timing gradients for distance-based weighting, providing a more elegant and robust approach. On benchmarks like Superblue1 and Superblue16, INSTA-Place achieves over 50% TNS improvement compared to [10], while achieving parity results on other benchmarks.

V. Conclusion and Future Work

We have proposed INSTA, a GPU-accelerated, differentiable STA engine tailored for industrial PD flows. INSTA achieves near-perfect correlation with PrimeTime in signoff mode while delivering exceptional computational efficiency. By utilizing timing gradients, INSTA enables highly effective optimization in critical PD tasks, including gate sizing and timing-driven placement, achieving significant improvements in QoR and resource utilization. We believe INSTA shall pave the way for transforming existing CPU-centric PD flows with high-fidelity GPU-STA. In the future, we aim to investigate INSTA for buffering and restructuring.

REFERENCES

[1] Primetime user guide: Advanced timing analysis. V-2023.03, Synopsys online Documentation, 2023.

[2] C. Albrecht. Iwls 2005 benchmarks. In International Workshop for Logic Synthesis (IWLS), volume 9, 2005.

[3] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric. Asap7: A 7-nm finfet predictive process design kit. Microelectronics Journal, 53:105–115, 2016.

[4] G. Guo, T.-W. Huang, Y. Lin, Z. Guo, S. Yellapragada, and M. D. Wong. A gpu-accelerated framework for path-based timing analysis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 42(11):4219–4232, 2023.

[5] G. Guo, T.-W. Huang, Y. Lin, and M. Wong. Gpu-accelerated critical path generation with path constraints. In 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–9. IEEE, 2021.

[6] G. Guo, T.-W. Huang, Y. Lin, and M. Wong. Gpu-accelerated path-based timing analysis. In 2021 58th ACM/IEEE Design Automation Conference (DAC), pages 721–726. IEEE, 2021.

[7] Z. Guo, T.-W. Huang, and Y. Lin. Gpu-accelerated static timing analysis. In Proceedings of the 39th international conference on computer-aided design, pages 1–9, 2020.

[8] Z. Guo, T.-W. Huang, and Y. Lin. Heterocppr: Accelerating common path pessimism removal with heterogeneous cpu-gpu parallelism. In 2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). ACM, 2021.

[9] Z. Guo, T.-W. Huang, and Y. Lin. Accelerating static timing analysis using cpu–gpu heterogeneous parallelism. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 42(12):4973–4984, 2023.

[10] Z. Guo and Y. Lin. Differentiable-timing-driven global placement. In Proceedings of the 59th ACM/IEEE Design Automation Conference, pages 1315–1320, 2022.

[11] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin. A timing engine inspired graph neural network model for pre-routing slack prediction. In Proceedings of the 59th ACM/IEEE Design Automation Conference, pages 1207–1212, 2022.

[12] Z. Guo, M. Yang, T.-W. Huang, and Y. Lin. A provably good and practically efficient algorithm for common path pessimism removal in large designs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(10):3466–3478, 2021.

[13] 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, 40(4):776–789, 2020.

[14] T.-W. Huang and M. D. Wong. Opentimer: A high-performance timing analysis tool. In 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 895–902. IEEE, 2015.

[15] B. Jin, G. Luo, and W. Zhang. A fast and accurate approach for common path pessimism removal in static timing analysis. In 2016 IEEE International Symposium on Circuits and Systems (ISCAS), pages 2623–2626. IEEE, 2016.

[16] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan. Iccad-2015 cad contest in incremental timing-driven placement and benchmark suite. In 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 921–926. IEEE, 2015.

[17] P.-Y. Lee, I. H.-R. Jiang, C.-R. Li, W.-L. Chiu, and Y.-M. Yang. itimerc 2.0: Fast incremental timing and cppr analysis. In 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 890–894. IEEE, 2015.

[18] R. Liang, A. Agnesina, G. Pradipta, V. A. Chhabria, and H. Ren. Circuitops: An ml infrastructure enabling generative ai for vlsi circuit optimization. In 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), pages 1–6. IEEE, 2023.

[19] P. Liao, S. Liu, Z. Chen, W. Lv, Y. Lin, and B. Yu. Dreamplace 4.0: Timing-driven global placement with momentum-based net weighting. In 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 939–944. IEEE, 2022.

[20] S. Lin, G. Guo, T.-W. Huang, W. Sheng, E. F. Young, and M. D. Wong. Gcs-timer: Gpu-accelerated current source model based static timing analysis. In Proceedings of the 61th ACM/IEEE Design Automation Conference, 2024.

[21] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 40(4):748–761, 2020.

[22] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan. Abcdplace: Accelerated batch-based concurrent detailed placement on multithreaded cpus and gpus. IEEE transactions on computer-aided design of integrated circuits and systems, 39(12):5083–5096, 2020.

[23] Y.-C. Lu, K. Kunal, G. Pradipta, R. Liang, R. Gandikota, and H. Ren. Lego-size: Llm-enhanced gpu-optimized signoff-accurate differentiable vlsi gate sizing in advanced nodes. In Proceedings of the 2025 International Symposium on Physical Design. ACM, 2025.

[24] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim. Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning. In 2021 58th ACM/IEEE Design Automation Conference (DAC), pages 733–738. IEEE, 2021.

[25] K. E. Murray and V. Betz. Tatum: Parallel timing analysis for faster design cycles and improved optimization. In 2018 International Conference on Field-Programmable Technology (FPT), pages 110–117. IEEE, 2018.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 2019.

[27] C. Peddawad, A. Goel, B. Dheeraj, and N. Chandrachoodan. iitrace: A memory efficient engine for fast incremental timing analysis and clock pessimism removal. In 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 903–909. IEEE, 2015.

[28] T. P. Peixoto. The graph-tool python library. figshare, 2014.

[29] P. Pham and J. Chung. Agd: A learning-based optimization framework for eda and its application to gate sizing. In 2023 60th ACM/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2023.

[30] H. H.-W. Wang, L. Y.-Z. Lin, R. H.-M. Huang, and C. H.-P. Wen. Casta: Cuda-accelerated static timing analysis for vlsi designs. In 2014 43rd International Conference on Parallel Processing, pages 192–200. IEEE, 2014.

[31] Y.-M. Yang, Y.-W. Chang, and I. H.-R. Jiang. itimerc: Common path pessimism removal using effective reduction methods. In 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 600–605. IEEE, 2014.