ICCAD: U: Heterogeneous Timing Estimation, Optimization, and Verification for VLSI Circuit Design Automation

Zizheng Guo, gzz@pku.edu.cn, ACM ID: 3011804
CS Department, Peking University, Beijing, P. R. China 100871

1 PROBLEM AND MOTIVATION

The semiconductor industry is experiencing both a widespread supply shortage and an unprecedented surge in demand for highly-customized chip designs ranging from data centers to consumer electronics. As the process node continues to shrink down below 5nm, tens of billions of transistors have been integrated into a single chip, introducing significant challenges to the chip performance analysis and optimization flow. A typical design flow optimizes chip performance through consecutive design stages by iteratively invoking performance estimation routines for guidance. The chip is then verified against its performance target. Such a flow requires months of effort even with experienced designers and the latest electronics design automation (EDA) software.

Despite the high demand for efficient chip performance estimation, optimization, and verification algorithms, many fundamental problems remain unsolved in this area. Firstly, the poor accuracy and speed of performance estimation algorithms leave performance issues undiscovered in early design stages and hinder the optimization of upcoming stages. Secondly, the chip performance cannot be directly optimized due to its complexity and non-differentiability, forcing EDA algorithms to use indirect surrogates that lead to suboptimal solutions. Moreover, the design verification requires error-free exact circuit timing analysis, which is extremely time-consuming for large circuits.

To boost the productivity of the chip design process, we propose a holistic framework for circuit performance (i.e., timing) estimation, optimization, and verification, consisting of three lines of research work. Our work gives fast and reliable timing estimation, optimization, and verification algorithms on a heterogeneous computing platform consisting of CPUs and GPUs, with both theoretical guarantees and orders-of-magnitude efficiency improvement.

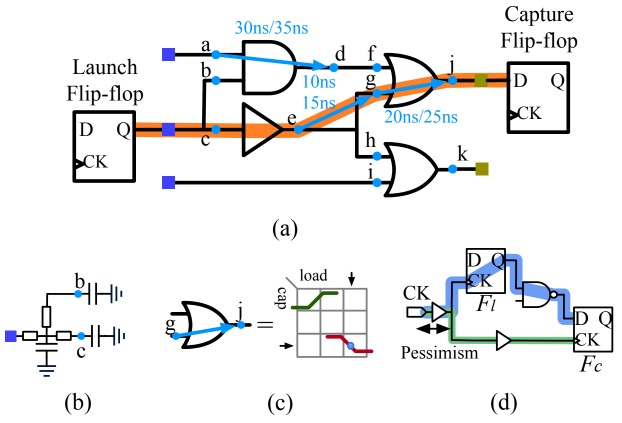

Figure 1: (a) Timing analysis computes signal delay and identifies timing-critical paths. (b) An interconnect resistor-capacitor (RC) tree. (c) A gate delay look-up table. (d) The common clock paths introduce pessimism.

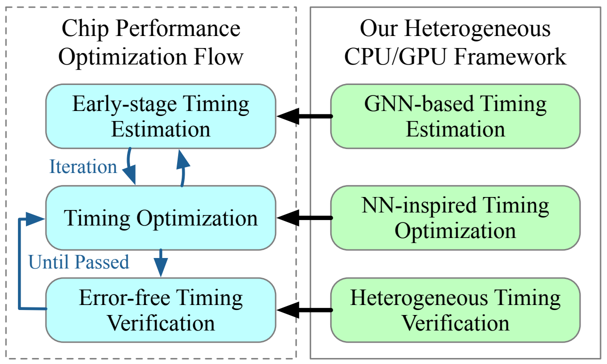
Chip Performance Optimization Flow
Early-stage Timing Estimation
Iteration
Timing Optimization
Until Passed
Error-free Timing Verification
Our Heterogeneous CPU/GPU Framework
GNN-based Timing Estimation
NN-inspired Timing Optimization
Heterogeneous Timing Verification

Figure 2: The three timing challenges in the chip design flow and our solutions.

2 BACKGROUND AND RELATED WORK

Recent decades have witnessed the boost of customized computer chips ranging from artificial intelligence (AI) accelerators in datacenters to Internet of things (IoT) and system-on-a-chip (SoC) in consumer electronics. Chips highly-customized to specific applications are essential to optimize the computation performance and power with Moore's law slowing down. However, given tens of billions of transistors in typical modern chips, designing a high-performance chip is very challenging. The circuit performance is modeled by *timing analysis*. As shown in Figure 1(a), timing analysis starts by modeling signal transition delays inside the circuit. Then, it identifies a number of signal paths with maximum delay values, which are called timing-critical paths that limit the maximum chip performance.

Modern chip design involves a very long design flow, with many consecutive design stages essentially performing three tasks: *early-stage timing estimation*, *timing optimization*, and *exact* or *error-free timing verification*. As shown on the left of Figure 2, the chip timing optimization is performed iteratively given the guidance of timing estimations. After that, the chip undergoes error-free timing verification that validates its performance rigorously against the design goals. If any timing violation is found, the timing optimization is relaunched until the performance target is reached. We introduce the background of the three tasks in the following subsections.

2.1 Early-stage Timing Estimation

During early design stages, the chip design is still incomplete and we need to estimate the circuit timing based on the partial design information to provide guidance for optimization. For example, after the logic gates are placed on the circuit layout, but before the interconnect wires are routed, the signal delay cannot be accurately determined. However, a wire connecting two gates far away from each other is typically slow. Also, some regions might become congested and force crossing wires to take detours that introduce larger delay. Failure to address such issues can cause optimization challenges in the upcoming stages of the iterative design flow.

Despite the importance of timing estimation, current EDA flow only adopts simple yet inaccurate algorithms based on heuristics. This is because the timing behavior of a circuit is very difficult to be modeled accurately in an analytical way in early stages. Although the most recent studies have successfully investigated data-driven modeling techniques for single-timing-arc delay estimation using neural networks (NN) [1–4], it is still challenging to directly model global timing metrics like path delays and chip performance, due to the extraordinary depth of circuit logic graphs.

2.2 Timing Optimization

The chip design algorithms optimize the chip performance in an iterative flow, by adjusting the gate placement, wire routes, and other design parameters under the guidance of timing estimation. Due to limited chip resources, the algorithms pay special attention to the optimization of timing-critical circuit regions. For example, in gate placement stage, gates on timing-critical paths should be placed near each other to reduce the interconnect delay. Such considerations are reflected by the global timing metrics that we try to optimize.

Due to the indifferentiability of timing metrics, it is very difficult to directly optimize the global circuit timing. Instead, state-of-the-art gate placement algorithms adopt a technique called *net weighting* [5–12]. They assign a weight coefficient to each interconnect wire (i.e., net) and update the weights of nets on timing-critical paths based on timing analysis results. The timing improvement achieved by such an approach is limited, as it focuses on improving the timing of specific nets or paths, which is only an indirect surrogate of the global timing metric. Such a gap between the optimization objective and the performance target restrains their optimization capability and efficiency with the increase of design complexity.

2.3 Error-free Timing Verification

To validate the chip performance without error, timing analysis engines perform heavy delay computations and path searching on large circuit graphs, which makes it a major runtime bottleneck in the chip design flow. It has been reported that a single iteration of timing verification can take tens of minutes to hours for million-gate circuit designs [13].

Previous works on accelerating timing analysis like [13–17] only provide limited runtime improvements. One reason is that they are limited to CPU-based multi-thread parallelism which fails to scale beyond 16 CPU cores due to irregular memory access patterns and imbalanced workload in timing analysis computation. For example, the delay calculation in timing analysis has many different forms including RC trees in Figure 1(b) and look-up tables in Figure 1(c). Timing analysis also involves many compute-intensive algorithms and operations, including the search for timing-critical paths and the identification of common clock paths to remove pessimism as shown in Figure 1(d).

2.4 Our Solution

To tackle the fundamental challenges in circuit design flow regarding performance *estimation*, *optimization*, and *verification*, we propose a holistic and heterogeneous timing analysis framework. Our works give pioneering and comprehensive solutions to *accurate estimation*, *effective optimization*, and *ultrafast verification* of chip performance, as shown in Figure 2. By combining our three lines of work, we have opened up new opportunities for efficient and high-quality design of chips. Our contributions are summarized as follows.

- **GNN-based timing estimation**: *We are the first to present an end-to-end graph learning framework for chip performance estimation.* Inspired from timing analysis engine calculations, our GNN model incorporates a novel message-passing strategy that enlarges the GNN receptive field and improves model explainability and interpretability. Our delay prediction accuracy has significantly outperformed vanilla deep GNN models and previous local delay estimation works.
- **NN-inspired timing optimization in gate placement**: *For the first time, we propose a differentiable timing analysis engine for timing-driven gate placement optimization, inspired by the analogy between timing optimization and neural network training.* The gradient of global timing metrics can be directly used to improve the chip performance. Our gradient computation is ultrafast (GPU-accelerated) and highly extensible to different chip design scenarios. We have improved the circuit timing by up to 32.7% compared to the state-of-the-art gate placement algorithm [12], using only around 50% of their runtime.
- **Heterogeneous timing verification**: *For the first time, we present a GPU-accelerated timing verification engine with provably efficient algorithms.* We propose a novel depth-based pessimism removal algorithm with an *asymptotically lower time and memory complexity.* We design GPU-efficient graph data structures and CPU-GPU task scheduling flows that introduce timing analysis to manycore massive data parallelism. Our timing analysis engine achieves over 1000× speed-up compared to state-of-the-art [13].

3 APPROACH AND UNIQUENESS

3.1 GNN-based Timing Estimation

To estimate chip performance at early design stages, we propose the first end-to-end timing estimation model based on graph learning [18].

3.1.1 Problem Formulation. In our work, we focus on timing estimation before wire routing, where detailed interconnect wires are not available. The task of predicting a single interconnect delay (denoted as Task 1) is itself very difficult due to complex and non-linear routing effects on interconnect capacitance and resistance. Moreover, besides predicting interconnect delay, *we aim at directly predicting global timing metrics consisting of path delay and maximum frequency* (denoted as Task 2). This eliminates the need for path searching algorithms on top of the delay prediction models, making our framework an *end-to-end* solution for chip performance estimation.

3.1.2 Problem Analysis and Challenges. We note that Task 1 essentially learns the behavior of the wire routing stage, and Task 2 corresponds to the path searching stage. As a result, our framework crosses multiple chip design stages, depicting the complex effects of design choices arising from sophisticated mathematical formulations and EDA algorithms. In addition, learning from circuit graph data is required to complete the timing estimation. While graph data can be handled by GNNs, the considerable logic depths in circuit graphs often overreach the capability of the vanilla GNN models due to their narrow receptive field and limited numbers of stacked layers.

3.1.3 Timing Model Inspired from Timing Engines. Our timing estimation framework consists of two components. We first propose a wire routing and single-stage delay prediction model for all interconnect wires. We then propose a global chip performance estimation model based on the output of the single-stage delay prediction. These two stages are built with the analogy to the delay calculation and path searching algorithms in timing analysis, respectively.

To overcome the GNN receptive field problem, we propose an end-to-end GNN model with a novel message-passing strategy utilizing the domain knowledge in timing analysis, as depicted in Figure 3(b). Our central idea is to “propagate” the GNN node embeddings on the directed circuit graph, inspired by the similar process in timing analysis engines. Compared with vanilla GNNs where each node only receives information from its 1-hop neighbors (Figure 3(a)), our timer-inspired GNN model enlarges the receptive field for every node even using a single layer, enabling us to learn global timing metrics from large and deep circuit logic graphs without worrying about the depth of GNN models.

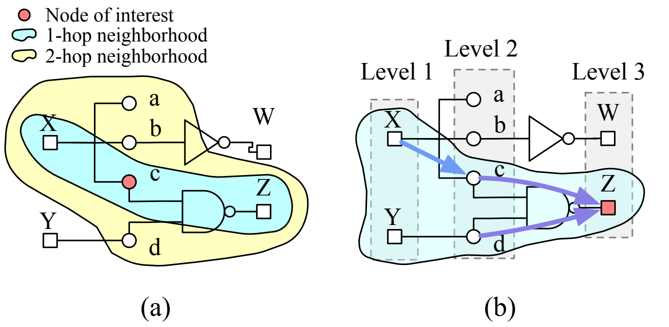

Figure 3: Comparison between the receptive field of (a) vanilla GNNs and (b) our feed-forward GNNs.

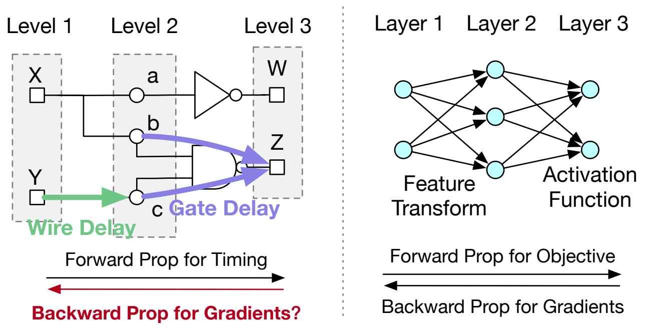

Figure 4: The analogy between a timing analysis engine and a deep neural network.

3.1.4 *Overall Contribution.*

We present the first end-to-end timing estimation framework for predicting chip performance before the wire routing stage. We formulate interconnect delay calculation and global timing estimation problems as graph learning tasks and propose a novel GNN message-passing strategy with a large receptive field which is necessary for processing large circuit graphs. Our framework and models are inspired from timing analysis engine calculations which improve model explainability and interpretability. Extensive experimental validations and ablation studies have shown superior model accuracy improvement compared with vanilla GNNs and prior works on interconnect delay estimation. *We have open-sourced both our models and our benchmark chip dataset to promote reproducibility of the machine learning and chip design community.*[1]

3.2 NN-inspired Timing Optimization in Gate Placement

We propose a new paradigm for timing-driven gate placement based on a differentiable timing engine [19].

3.2.1 *Analogy to Deep Neural Network Training.*

Our work is motivated by the analogy between the timing optimization problem in gate placement and the training of a neural network model, as illustrated in Figure 4. Essentially, the gate locations can be regarded as trainable model weights, and the gate interconnect topology can be regarded as the training dataset. By applying model weights to the training dataset, we obtain the training objective to optimize in the deep learning workflow. In the gate placement problem, we analogously obtain the interconnect delay penalty to optimize. We note that a deep neural network (DNN) can be efficiently trained with back-propagation (BP). Inspired by this, we intend to perform a similar back-propagation in timing analysis to obtain the gradients of path delays with respect to gate locations.

[1]https://github.com/TimingPredict/TimingPredict

3.2.2 *Timing Differentiability and Smoothness.*

One key challenge in differentiable timing analysis compared to neural networks is that timing analysis computation has highly non-smooth `max` and `min` operations to extract worst- and best-case signal delay. To avoid ill-conditioned sparse gradients induced by these operations, we replace them with their smoothed approximations using Log-Sum-Exp (LSE). An example for the max operation can be written as following,

$$\max(x_1, x_2, ..., x_n) \approx LSE_\gamma(x_1, x_2, ..., x_n) = \gamma \log\left(\sum_{i=1}^{n} \exp\frac{x_i}{\gamma}\right).$$

3.2.3 *GPU-Accelerated Back-propagation.*

During the placement optimization, the timing analysis computation and gradient analysis are performed inside the inner iteration loop which is repeated thousands of times before convergence. As a result, the speed of the differentiable timing analysis engine is critical for efficient chip optimization. To this end, we develop high-performance GPU-accelerated timing analysis kernels, including both forward computation and backward gradients. Our kernels are encapsulated as PyTorch CUDA extensions and can be easily integrated into existing analytical chip placement flow and back-propagation frameworks.

3.2.4 *Overall Contribution.*

For the first time, we propose a differentiable timing analysis engine for timing-driven gate placement optimization, by establishing the analogy between timing optimization and neural network training. Instead of conventional methods that rely on timing improvement of specific nets or paths, we can directly optimize the global timing metrics such as worst path delay, enabled by our smoothed timing objective formulation and ultrafast GPU-accelerated forward and backward computation. Experimental results on large real-world circuit designs demonstrate up to 32.7% circuit performance improvement compared to the state-of-the-art gate placement algorithm [12], while also improving the runtime by around 50%.

3.3 Heterogeneous Timing Verification

In this section, we introduce our research on the acceleration of error-free timing analysis [20–24].

3.3.1 *Challenges.*

Our research focuses on solving both theoretical and practical challenges regarding timing analysis computation:

(1) In timing analysis, the algorithm to remove pessimism induced by common clock paths (see Figure 1(d)) has a quadratic time and complexity $O(n^2)$. Here n denotes the number of registers or flip-flops (FFs) in the circuit, which is proportional to the circuit size and typically exceeds hundreds of thousands. As a result, the algorithm scales poorly with the increasing chip design complexity.
(2) The timing-critical path finding algorithm used in timing analysis has time complexity $O(nk \log k)$ to report the top-k most critical paths. When reporting a large number of paths k on a large circuit size n, the timing analysis runtime becomes further unacceptable.
(3) Current timing analysis engines are poorly parallelized. The timing analysis algorithms involve complex delay calculation and graph traversal operations that lead to diverse computational patterns and irregular memory access, making it very difficult to design parallel algorithms that benefit from the computation power of modern hardware.

Within the above three challenges, (1) and (2) are theoretical problems, and (3) is a practical problem. Both theoretical problems and practical problems need to be addressed in order to achieve a remarkable runtime speed-up.

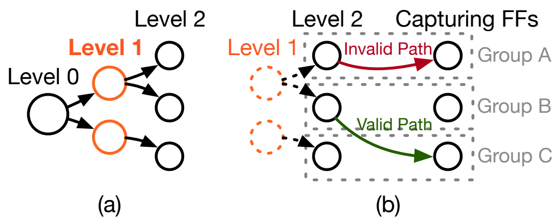

Figure 5: Our depth-based pessimism removal algorithm. (a) Select a current LCA depth. (b) Group nodes and search for cross-group paths.

3.3.2 *Provably Good Pessimism Removal [21, 23].* We solve challenge (1) by designing a novel *depth-based* pessimism removal algorithm. Instead of enumerating all clock tree node pairs to find their common paths, we group all possible pairs by the depth of their lowest common ancestor (LCA) on the clock tree, as shown in Figure 5. While it is difficult to locate the exact set of node pairs that have the desired LCA depth, we relax the constraint and propose an efficient linear-time algorithm to compute the pessimism for every single depth. *We give rigorous proof of the correctness and theoretical performance of our algorithm.* Our algorithm gives full-precision pessimism removal solutions, without any trade-off on accuracy. Our algorithm runs in $O(n \log n)$, where the term $\log n$ comes from the depth of the circuit clock tree. In practice, we can reduce the amount of computation for pessimism removal by 3 orders of magnitude for industrial benchmarks.

3.3.3 *Scalable Path Finding Algorithm [24].* We solve challenge (2) by proposing a new path searching algorithm that leverages two advanced data structures, *leftist heap* and *skew heap*. By using the two kinds of heaps, we split the path searching computation into a *preprocessing* step and a *path exploration* step. The preprocessing step prepares necessary path topologies and delay differences that are essential for path exploration, and arranges them into heaps. This preparation runs in $O(n \log n)$ and needs to be done only once. The path exploration step runs in $O(k \log k)$ for computing k critical paths. During the path exploration, circuit topology and delay information are reused from the preprocessing, significantly reducing redundant computation. The overall time complexity is then reduced to $O(n \log n + k \log k)$, where n and k are no longer multiplied with each other, solving the fundamental scalability problem for large-scale path searching in timing analysis.

3.3.4 *GPU-Accelerated Timing Analysis [20, 22].* We made a giant leap towards challenge (3) by leveraging the power of heterogeneous parallelism comprising manycore CPUs and GPUs. To compute signal delay values on GPU efficiently, we flatten the RC trees and map the RC delay updates as parallel scan steps on GPU, as shown in Figure 6. We sort the circuit graph nodes into levels to allow effective parallel delay computation. We apply GPU-specific design optimizations like arranging the data storage to allow global memory coalescing. We also overlap the execution of computation and memory transfer to minimize the computation overhead.

3.3.5 *Overall Contribution.* For the first time, we have proposed provably good timing analysis and verification algorithms for pessimism removal and path searching, which are two essential timing analysis components that used to have quadratic time complexity. We greatly improved their theoretical scalability by providing asymptotically lower time and memory complexities. We propose a GPU-accelerated timing analysis engine that breaks the parallelization obstacles and provides unprecedented computation efficiency. Our highly optimized timing analysis algorithms and implementations demonstrate the practicality of GPU acceleration for timing analysis beyond what has been done

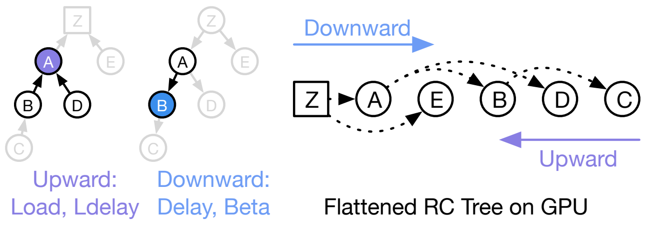

Figure 6: GPU-accelerated RC delay computation by flattening and scan-based dynamic programming.

in the literature. Our experimental results on large industrial benchmarks have shown over 1000× speed-up compared to a widely-used state-of-the-art timing analysis engine OpenTimer [13].

4 RESULTS AND CONTRIBUTIONS

4.1 GNN-based Timing Estimation

First, we present our experimental results on our GNN-based timing estimation model. We evaluate our methods using fully open-source chip designs, process technology data (known as process design kit or PDK), and EDA engines. Our dataset contains 14 training benchmarks and 7 test benchmarks each with tens or hundreds of thousands of circuit graph nodes. To our knowledge, this is the largest dataset used in similar machine-learning-assisted chip design works due to the small number of public hardware designs. We have released this dataset to the public along with our model.

Table 1 compares the R^2 score for two different tasks: estimation for single-stage delay and global timing metrics (worst path delay). On single-stage delay estimation, we compare with the statistical feature-based model in [3]. While their model fits the training data more effectively, our GNN shows better generalization to test circuits. On worst path delay estimation, we compare with up to 16 layers of GCNII, a vanilla GNN work [25]. Despite their optimization for deep models, no useful graph embeddings have been learned on our large and deep circuit logic graphs. On the other hand, our end-to-end GNN model with large graph receptive fields can achieve an R^2 score up to 0.89 on test circuits.

Table 1: Comparison on R^2 regression score for single-stage delay and worst path delay prediction.

Train/Test	Single-stage Delay			Worst Path Delay	
	RF [3]	MLP [3]	Our GNN	GCNII [25]	Our GNN
Avg. Train	**0.9944**	0.9550	0.9870	0.3586	**0.9493**
Avg. Test	0.9418	0.9357	**0.9552**	-0.7766	**0.8957**

RF: random forest. MLP: multi-layer perceptron

4.2 NN-inspired Timing Optimization

Next, we present our results on NN-inspired timing optimization for gate placement in Figure 7. The max delay and total delay in the table are measured on the benchmark `superblue18` from the ICCAD 2015 incremental timing-driven placement contest. We have demonstrated up to 32.7% max delay reduction and 59.1% total delay reduction compared to the state-of-the-art performance-driven gate placement algorithm [12] based on net weighting, setting up new records on the public benchmark suite. We ascribe this large improvement to our differentiable timing engine that directly computes the gradient of global timing metrics, allowing us to more effectively target chip performance refinement in the design iterations, which is visualized in the right-hand side figure in Figure 7.

Performance	NW [12]	Ours	Improve
Max Delay	11.871	7.987	32.7%
Total Delay	4724.55	1931.43	59.1%
Avg. Runtime	515.2 s	280.1 s	1.84×

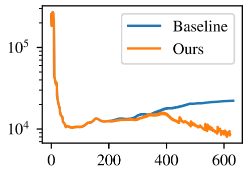

Figure 7: Left: performance comparison with net weighting (NW). Right: visualization of the maximum delay optimization process along with iterations.

4.3 Heterogeneous Timing Verification

Finally, we demonstrate the runtime and memory improvement of our GPU-accelerated timing verification engine. We implement our timing analysis engine using C++ and CUDA and profile it on the TAU 2015 timing contest benchmarks, which include large industrial circuit designs commonly used in the chip design community. Table 2 shows a detailed runtime comparison between OpenTimer [13] and our timing analysis engine running with CPU [21, 24] or GPU [20, 22]. When both running on CPUs, our timing analysis engine demonstrates up to 96.28× speed-up compared to OpenTimer, thanks to our novel pessimism removal and path searching algorithm with asymptotically lower time complexities. By using one Nvidia RTX 2080 Ti GPU, our timing analysis engine boosts the efficiency by over 1000× compared to OpenTimer, as a result of our highly-optimized GPU kernels and GPU-friendly task scheduling.

Table 2: Runtime comparison of timing analysis engines (in seconds). The CPU and GPU columns indicate our algorithm.

Benchmark	OpenTimer	CPU	Speed-up	GPU	Speed-up
Combo6v2	1333.36	33.85	39.39×	1.24	1075.29×
netcard	2749.46	41.59	66.11×	1.75	1571.12×
leon2	4320.00	44.87	96.28×	2.03	2128.08×
leon3mp	1348.55	31.11	43.35×	1.54	875.68×

Figure 8 draws the runtime and memory under different number of timing-critical paths on a large circuit benchmark `leon2` with 21M circuit graph nodes. Our timing analysis engine demonstrates superior runtime and memory scalability with the increasing circuit size in modern chip design flow.

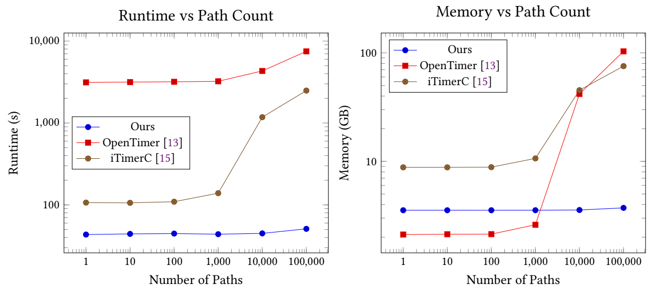

Figure 8: Runtime and memory values for reporting different numbers of timing-critical paths on the circuit `leon2`.

4.4 Research Impacts

My undergraduate research study on chip performance estimation, optimization, and verification has led to a total of **10 publications** [18–24, 26–28] in premier ACM/IEEE conferences and journals, including **7 first-authored conference papers** [18–22, 24, 26] accepted by top EDA conferences like DAC, ICCAD, and ASP-DAC, **1 first-authored journal paper** at the top EDA journal TCAD [23] (plus another TCAD submission under review), and 2 second-authored conference papers [27, 28].

We have proposed fast and reliable chip performance estimation, optimization, and verification algorithms leveraging heterogeneous computing platforms, with both theoretical guarantees and orders-of-magnitude improved performance. Our work has drawn attention from leading industrial companies like IBM, Xilinx, and academia. We have open-sourced our timing estimation models, code, and the entire dataset for reproducibility. We believe our contribution to circuit timing analysis can benefit a broad range of chip design and EDA applications.

REFERENCES

[1] S.-S. Han, A. B. Kahng, S. Nath, and A. S. Vydyanathan, "A deep learning methodology to proliferate golden signoff timing," in *Proc. DATE*, 2014, pp. 1–6.
[2] W.-T. J. Chan, K. Y. Chung, A. B. Kahng, N. D. MacDonald, and S. Nath, "Learning-based prediction of embedded memory timing failures during initial floorplan design," in *Proc. ASPDAC*, 2016, pp. 178–185.
[3] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proc. DAC*, 2019, pp. 1–6.
[4] R. Liang, Z. Xie, J. Jung, V. Chauha, Y. Chen, J. Hu, H. Xiang, and G.-J. Nam, "Routing-free crosstalk prediction," in *Proc. ICCAD*, 2020, pp. 1–9.
[5] M. Burstein and M. N. Youssef, "Timing influenced layout design," in *Proc. DAC*. IEEE, 1985, pp. 124–130.
[6] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. Jukl, P. Kozak, and M. Wiesel, "Chip layout optimization using critical path weighting," in *Proc. DAC*. IEEE, 1984, pp. 133–136.
[7] 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 *Proc. ISPD*, 2002, pp. 155–160.
[8] T. Kong, "A novel net weighting algorithm for timing-driven placement," in *Proc. ICCAD*, 2002, pp. 172–176.
[9] B. Halpin, C. R. Chen, and N. Sehgal, "A sensitivity based placer for standard cells," in *Proc. GLSVLSI*, 2000, pp. 193–196.
[10] T.-Y. Wang, J.-L. Tsai, and C. C.-P. Chen, "Sensitivity guided net weighting for placement driven synthesis," in *Proc. ISPD*, 2004, pp. 124–131.
[11] Z. Xiu and R. A. Rutenbar, "Timing-driven placement by grid-warping," in *Proc. DAC*, 2005, pp. 585–591.
[12] 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 *Proc. DATE*, Antwerp, Belgium, March 2022.
[13] T. Huang, G. Guo, C. Lin, and M. D. F. Wong, "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," *IEEE TCAD*, pp. 1–1, 2021.
[14] T.-W. Huang and M. D. Wong, "UI-timer 1.0: An ultrafast path-based timing analysis algorithm for CPPR," *IEEE TCAD*, vol. 35, no. 11, pp. 1862–1875, 2016.
[15] 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 *Proc. ICCAD*. IEEE, 2015, pp. 890–894.
[16] B. Jin, G. Luo, and W. Zhang, "A fast and accurate approach for common path pessimism removal in static timing analysis," in *Proc. ISCAS*. IEEE, 2016, pp. 2623–2626.
[17] P.-Y. Lee, I. H.-R. Jiang, and T.-C. Chen, "Fastpass: fast timing path search for generalized timing exception handling," in *Proc. ASPDAC*. IEEE, 2018, pp. 172–177.
[18] 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 *Proc. DAC*. ACM, 2022.
[19] Z. Guo and Y. Lin, "Differentiable-timing-driven global placement," in *Proc. DAC*. ACM, 2022.
[20] Z. Guo, T.-W. Huang, and Y. Lin, "Gpu-accelerated static timing analysis," in *Proc. ICCAD*. ACM, 2020.
[21] ——, "A provably good and practically efficient algorithm for common path pessimism removal in large designs," in *Proc. DAC*. ACM, 2021.
[22] ——, "Heterocppr: Accelerating common path pessimism removal with heterogeneous cpu-gpu parallelism," in *Proc. ICCAD*. ACM, 2021.
[23] 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 TCAD*, pp. 1–1, 2021.
[24] K. Zhou, Z. Guo, T.-W. Huang, and Y. Lin, "Efficient critical paths search algorithm using mergeable heap," in *Proc. ASPDAC*. ACM, 2022.
[25] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. ICML*. PMLR, 2020, pp. 1725–1735.
[26] Z. Guo, J. Mai, and Y. Lin, "Ultrafast cpu/gpu kernels for density accumulation in placement," in *Proc. DAC*. ACM, 2021.
[27] Z. Zhang, Z. Guo, Y. Lin, R. Wang, and R. Huang, "Avatar: An aging- and variation-aware dynamic timing analyzer for application-based dvafs," in *Proc. DAC*. ACM, 2022.
[28] ——, "Eventtimer: Fast and accurate event-based dynamic timing analysis," in *Proc. DATE*, 2022.