Invited Paper: Accelerating Routability and Timing Optimization with Open-Source AI4EDA Dataset CircuitNet and Heterogeneous Platforms

Xun Jiang[1†], Zizheng Guo[1†], Zhuomin Chai[1,2], Yuxiang Zhao[1], Yibo Lin[1,3,4*], Runsheng Wang[1,3,4], Ru Huang[1,3,4]
[1]School of Integrated Circuits, Peking University, Beijing, China
[2]School of Physics and Technology and the School of Microelectronics, Wuhan University, Wuhan, China
[3]Institute of Electronic Design Automation, Peking University, Wuxi, China
[4]Beijing Advanced Innovation Center for Integrated Circuits

***Abstract*—Routability and timing are two critical metrics for modern VLSI circuits. With increasing design complexity and continuous shrinking of technology nodes, optimizing routability and timing become extremely expensive due to high computational overhead for analysis. It is reported that conventional CPU-based parallelization strategies can no longer scale beyond 8-16 threads. In this talk, we introduce how to accelerate routability and timing optimization leveraging AI-enabled GPU acceleration. To break the inter-stage information dependency in conventional physical design flow, we build AI for EDA models with an open-source dataset, CircuitNet, to enable ultrafast design optimization on GPU. We hope our study can shed lights to future development of EDA tools with AI-enabled heterogenity.**

I. Introduction

Due to the relentless technology scaling and ever-increasing integration, physical design for VLSI circuits needs to go through a complicated design flow and becomes extremely time-consuming. For a million-cell design, it takes days to weeks for a single physical design iteration, and it is common to run multiple iterations for design convergence. Let alone the design sizes for modern SoCs scale to tens of millions of cells. The semiconductor industry urgently calls for faster solutions for physical design automation.

Unfortunately, it is very challenging to accelerate physical design. Firstly, physical design follows a long and iterative design flow consisting of floorplanning, power planning, placement, clock tree synthesis, routing, routability/timing analysis, etc., as shown in Figure 1. The information dependency between design stages pushes away the possibility of task-level parallelism and raises difficulties in fast cross-stage performance analysis. Secondly, the algorithms in each physical design stage are often combinations of heuristics, which are sequential and iterative in nature, causing low parallelism even in fine granularity. It is observed that CPU-based parallelization does not scale beyond 8-16 threads [1]. Therefore, insisting on the existing physical design methodology can no longer accelerate design closure. New design methodologies with algorithm reorganization, redesign, and reformulation are desired for faster physical design automation.

†Equal contribution. *Corresponding author: yibolin@pku.edu.cn

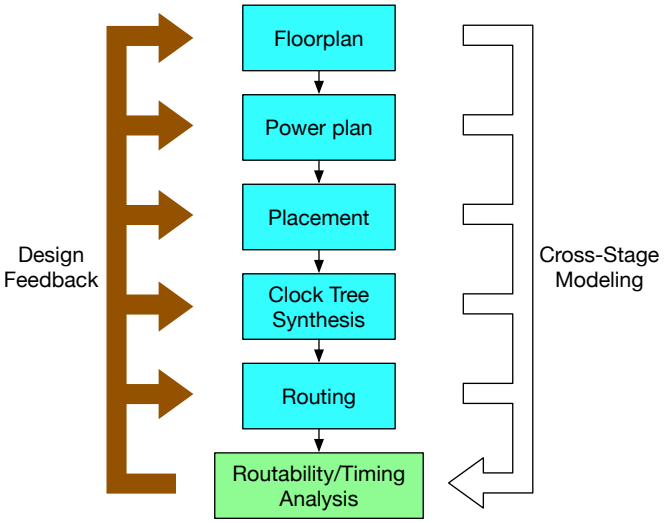


Fig. 1: Example of iterative physical design flow.

Existing studies have explored to accelerate typical physical design stages, including placement [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], routing [16], [17], [18], [19], timing analysis [20], [21], [22], [23], [24], and so on, with heterogeneous computing resources like GPU and FPGA. These studies move one step forward to reorganize and redesign the existing algorithms for heterogeneous hardware. However, they still follow the conventional design iterations, where each design stage relies on performance feedbacks from the output of succeeding stages to guide optimization. Such an iterative procedure can significantly slow down the entire design optimization.

To speed up cross-stage performance analysis, artificial intelligence (AI) techniques have been adopted for early-stage performance prediction [25]. The literature has extensively explored cross-stage performance modeling for routability [26], [27], [28], [29], [30], [31], [32], timing [33], [34], [35], [36], [37], power [38], [39], [40], etc. As typical AI models like deep neural networks are naturally parallelizable on GPU, they break the inter-stage information dependency and significantly reduce the runtime. However, most studies

have been only focused on building accurate models, while very few studies explore the effectiveness of integrating AI models into optimization.

In this work, we introduce recent efforts to leverage AI-enabled GPU acceleration for routability and timing optimization in physical design. The main idea lies in two aspects: 1) AI-assisted cross-stage modeling; 2) AI-inspired design optimization. We take advantages of an open-source AI for EDA dataset, CircuitNet [41], [42], consisting of more than 20K+ data samples from commercial design flow and PDKs, to build accurate and efficient deep learning models. Experimental results demonstrate that our AI-enabled heterogeneous acceleration can contribute to over 18× speed-up with competitive performance compared with conventional CPU-based algorithms.

The rest of the paper is organized as follows. Section II reviews the related work. Section III provides a brief introduction about CircuitNet dataset. Section IV explains AI-enabled GPU acceleration for routability optimization. Section V explains AI-enabled GPU acceleration for timing optimization. Section VI concludes the paper.

II. Related Work

In this section, we review the recent efforts to accelerate physical design stages on heterogeneous platforms and AI-assisted cross-stage performance modeling.

A. Heterogeneous Acceleration of Physical Design

In this section, we review the recent efforts to accelerate physical design algorithms on heterogeneous platforms.

1) Acceleration of Placement: Placement determines the locations of cells in a circuit, which can be divided into global placement, legalization, and detailed placement. Global placement roughly determines the locations of cells by numerical optimization. Legalization removes the overlap between cells to satisify design rules, as the result of global placement does not guarantee legality. Detailed placement further refines the locations of cells by incremental movement. Cong et al [2] present an early work to accelerate the nonlinear optimization part of global placement with GPU, while leaving the clustering part on CPU. Lin et al [5] discover the analogy between solving placement and training neural networks, and then propose *DREAMPlace* with deep-learning-toolkit-enabled GPU acceleration. It can achieve more than 30× speedup on wirelength-driven global placement compared with the recent CPU-based placer [43] without quality degradation. This work stimulates many follow-up studies like Xplace [8] for ultra-fast ASIC placement, and elfPlace [13] and OpenPARF [14], [15] for GPU-accelerated FPGA placement.

Besides global placement, Yang et al [10] investigate to accelerate mixed-cell-height legalization on GPU and achieve 2-4× speedup. Lin et al [6] and Dhar et al [11] propose many GPU-accelerated algorithms for detailed placement, including independent set matching, global swap, local reordering, and intra-row interleaving. Over 10× speedup can be achieved compared with CPU-based detailed placement algorithms [44]. Dhar et al [12] also develop an FPGA-accelerated wirelength computation kernel for FPGA placement.

Note the above studies mostly focus on wirelength optimization, which have not yet taken routability and timing optimization into consideration.

2) Acceleration of Routing: Routing needs to find shortest paths for each net on a large routing grid graph. It is extremely time-consuming for the expensive search of paths for millions of nets on million-node grid graphs. Routing involves various algorithms like pattern routing and maze routing. Pattern routing constrains the routing topology for a net to be specific patterns like L-shape or Z-shape to reduce the search space. Maze routing usually performs exhaustive search to find the shortest path for a net. A typical global routing algorithm often performs pattern routing first and then invokes maze routing iteratively to route nets failed during pattern routing. Liu et al [16] propose *FastGR* with GPU-accelerated L-shape and Z-shape pattern routing based on dynamic programming as well as an efficient task scheduling algorithm. It is reported that pattern routing can be accelerated by 10× on GPU, and the entire global routing ends up with over 2× speedup. Lin et al [17] propose a GPU-friendly maze routing algorithm called *GAMER* based on iterative sweeping on a matrix, and later they put both pattern routing and maze routing on GPU with carefully designed scheduling and data structures [18]. Eventually, 10× speedup for global routing can be achieved. Besides GPU acceleration, Jiang et al [19] investigate to implement a dijkstra-based maze routing algorithm on FPGA and report 3× speedup on the routing kernel.

3) Acceleration of Timing Analysis: Timing analysis aims at evaluating how fast a digital circuit can run. It computes delay and propagates timing information throughout the circuit graph to analyze whether timing constraints are satisfied. Static timing analysis (STA) is a widely adopted approach to perform corner-based checking, which is often invoked at each design stage to guide optimization. STA becomes computationally expensive with growing circuit sizes and complexity of timing models. Guo et al [24] propose to put both delay calculation and timing propagation on GPU, and demonstrate up to 4× speedup on single corner analysis, and up to 25× speedup on multi-corner analysis. Later, Guo et al [45] propose a GPU-based critical path generation algorithm that can achieve 25-45× speedup in reporting 100K critical paths compared with the state-of-the-art CPU-based algorithm. Guo et al [23] propose *HeteroCPPR* to resolve a specific kind of timing exception from common path pessimism. They improve the theoretical complexity of common path pessimism removal (CPPR) and report up to 16× speedup with 4 GPUs for completing the analysis of 10K post-CPPR critical paths in a million-cell design.

These studies have explored to reorganize and redesign specific algorithms in each design stage to achieve massive parallelism on heterogeneous platforms. However, they still cannot break the inter-stage information dependency and rely on expensive design iterations for convergence, which is not enough to fundamentally accelerate the physical design flow.

B. AI-Assisted Cross-Stage Performance Modeling

In this section, we review the recent efforts to AI-assisted cross-stage performance modeling.

1) Routability Modeling: Many studies have investigated to build ML models for routability prediction in physical design. As a layout in physical design can be naturally viewed as an image, existing studies leverage convolutional neural networks (CNNs) as the backbone of ML models to learn the geometric correlation [26], [27], [29], [30]. They extract image-like features from the layout as the input of ML models and output congestion maps or distribution maps of DRV hotspots. Another stream of studies start by viewing the circuit netlist as a graph, and leverage graph neural networks (GNNs) for routability prediction [28], [31], [32]. The layout information like the locations of cells is encoded into the attributes of graph nodes. The topological correlation can be learned through message passing between graph nodes. Recent studies also try to combine CNN and GNN to better capture both geometric and topological information [46].

2) Timing Modeling: Timing modeling aims at predicting net delay and slacks at early design stages without full information for exact timing analysis. A typical problem formulation in physical design is to predict post-routing timing information at placement stage. Barboza et al [33] propose to utilize random forest for net delay prediction. He et al [35] propose to leverage GNN to predict net delay and integrate into timing propagation to obtain the global timing information in static timing analysis. Guo et al [34] propose a level-by-level GNN to jointly learn net delay and global timing information. Ye et al [36], [37] explore to leverage GNN to close the gap between static timing analysis and sign-off timing information.

Most studies have been focusing on building accurate ML models, while very few work looks into effective performance optimization with ML models.

III. The CircuitNet Dataset

CircuitNet[1] is an open-source dataset for AI applications in chip design. With the development of AI for EDA in recent years [25], various machine learning (ML) models are invented to assist the traditional EDA flow to perform better. However, most researchers have to sacrifice a large amount of time to generate private datasets to evaluate the effectiveness of their ML models, which raises the bar for other people, especially researchers from the ML community, to conduct interdisciplinary studies and reproduce existing works.

CircuitNet provides user-friendly image-based and graph-based data extracted from intermediate chip design stages for building multi-modal ML models. It includes over 20,000 samples with many kinds of chip designs (e.g., CPU, GPU, and ML accelerators). The chosen chip designs are from well-developed open-source hardware projects in academia and industry, whose design quality has been verified by many users of the hardware community. The CPU designs include three types of microcontroller-level CPUs from the

[1]https://circuitnet.github.io/

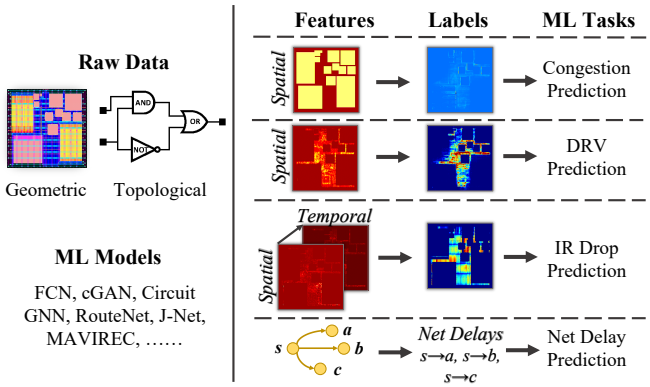


Fig. 2: Example of tasks supported by CircuitNet.

academic PULPino project and one high-performance CPU from the industrial OpenC910 project. The GPU designs in our dataset come from the academic Vortex project, where designs contain massive parallel processing units. The ML accelerator designs come from the industrial NVDLA project with specific dataflow for ML algorithms. All the designs can be further configured to generate different hardware samples. Besides, two types of widely-used technology nodes, including 28nm planar CMOS and 14nm FinFET, are applied throughout the complete commercial design flows to these chip designs.

The generation flow is divided into two parts, including synthesis and physical design. RTL designs are synthesized with different frequencies to generate netlists. These netlists are then put into the physical design flow with various settings, e.g., cell utilization, floorplan mode, etc. Between the intermediate stages of the physical design flow, we conduct the analysis procedures to evaluate their outcomes with different metrics, such as congestion, design rule violation (DRV), IR drop, timing, and power. These data will be viewed as labels in the ML optimization flow. The features used for these ML models are translated into image or graph format from the raw LEF/DEF files, reports, or netlists. It is worth mentioning that users have the freedom to generate richer features and labels based on the raw data provided in the dataset.

CircuitNet support multiple ML for EDA tasks, including routability prediction, IR drop prediction, and timing prediction, as shown in Figure 2. The routability prediction tasks the placement results as input to predict the severity of congestion after global routing and potential DRVs after detialed routing. The characteristics of these tasks are similar to image-to-image generation, where some works leverage GAN for prediction. The IR drop prediction is performed after the stage of CTS, which utilizes the spatial and temporal power data to predict the IR drop hotspots in the layout. One important factor in this task is that 3D CNN architecture can be used to handle the temporal axis of power data. The timing prediction is a multi-modal ML task, where the inputs of it include the geometric layout data and the graph-format netlists. To predict the timing data, the widely-used GNN model is employed on CircuitNet.

IV. Routability Modeling and Optimization

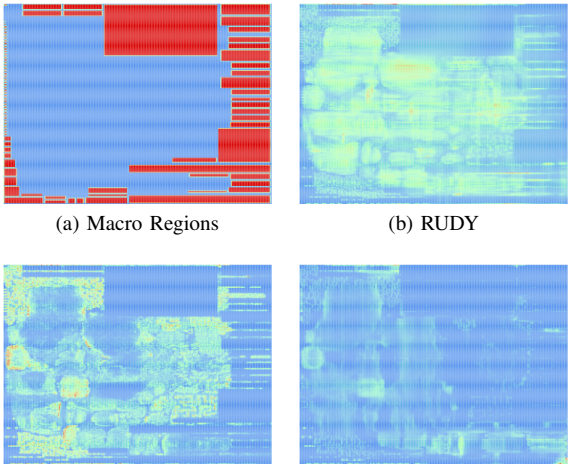
Routability relates to the potential routing quality provided by the current design solution, and it is largely affected by placement solutions. Routability estimation at pre-routing stages is challenging, as we need to model the complex heuristics in the routing algorithm. This section demonstrates our techniques to obtain fast and accurate routability prediction for guiding routability optimization using CircuitNet dataset on a heterogeneous platform.

A. Routability Modeling using FCN and CircuitNet

Routing is the notably time-consuming stage which can take over half of the cycle in physical design. Poor routability introduces more congestion in global routing or more DRVs in detailed routing, which must be totally solved before tape-out. Routability is strongly correlated to placement and cannot be fully optimized solely in routing stage.

Many routability-driven placement methods [47], [48], [49] leverage early-stage estimator like RUDY [50], or directly invoke global routing for congestion estimation, but these methods have either low accuracy or efficiency. To achieve fast and accurate routability estimation, ML techniques are adopted [26], [27], [28], [29], [30], [32]. We extract various features from placement, like cell density, pin density, RUDY [50], pin RUDY [26], and macro regions, and build ML models to predict global routing congestion, which is defined as the ratio of routing demand over routing resource in each global routing cell (GCell). As both features and labels can be encoded into images, as shown in Figure 3, we formulate an image-to-image translation task and adopt a fully convolutional network (FCN) with a encoder-decoder structure as the backbone of the ML model. Note that the congestion labels in CircuitNet come from global routing using Cadence Innovus.

We use the CircuitNet-ISPD15 dataset for training and evaluation, a visualization of the features and the label are shown in 3. This dataset is constructed based on ISPD 2015 detailed-routability-driven placement contest benchmarks [51]. We generate 150 data samples using the 5 million-cell `superblue` designs for congestion prediction, 30 for each design, through changing macro locations and core utilizations. We split the 30 samples of `superblue12` into six groups and use each group as a training set. The rest 120 samples of other `superblue` designs are used as testing set. Then we train on each group for 200 iterations and report the average testing accuracy on the testing set. As building ML models requires large amount of training data, we first pre-train a model using 10K+ data samples in CircuitNet-N28, and then fine-tune to the training set in CircuitNet-ISPD15 utilizing a teacher-student network [42]. We compare the results of training from scratch ("from scratch") and fine-tuning in Table I. With the knowledge transferred from CircuitNet-N28, much better performance can be achieved on the testing set, as shown in Table I, denoted as "fine-tuning". Compared with running global routing using Cadence Innovus for congestion evaluation, our FCN model can achieve 722,705× speedup on GPU in average with a stable inference time.



(a) Macro Regions (b) RUDY (c) pin RUDY (d) Congestion

Fig. 3: Visualization of the features (a-c) and label (d) in CircuitNet-ISPD15.

B. Heterogeneous Routability Optimization

Cell inflation is a typical strategy for routability-driven placement [47], [48], [49]. Figure 4 demonstrates our GPU-accelerated routability-driven placement flow, which performs cell inflation at congested regions based on the routability prediction model in Section IV-A. We extract features like macro regions, RUDY, and pin RUDY with GPU-accelerated kernels, and obtain the congestion map through model inference. The routability optimization is triggered when the density overflow is under a certain threshold, when cells have been spread enough for meaningful routability prediction.

We run the routability optimization flow on ISPD2015 benchmarks, and use Cadence Innovus for global routing to get congestion and wirelength. We demonstrate the results on the designs in the testing set in Table II. We evaluate routed wirelength (WL) and congestion rate (CR), defined as:

$$\text{H-CR} = \frac{\sum_{i=1}^{H}\sum_{j=1}^{V} Overflow\_h(i,j)}{HV}, \tag{1a}$$

$$\text{V-CR} = \frac{\sum_{i=1}^{H}\sum_{j=1}^{V} Overflow\_v(i,j)}{HV}, \tag{1b}$$

where H and V are the numbers of horizontal and vertical GCells. $Overflow\_h$ and $Overflow\_v$ are horizontal and vertical overflow values for GCell (i, j), respectively.

Our routability-driven placement flow can achieve CR reduction in all test designs with an average of 26.0% and 28.4% in both horizontal and vertical directions, respectively, with slight wirelength degradation, compared with the default wirelength-driven DREAMPlace. Figure 5 shows the runtime comparison under different design scales. To show the runtime benefits from AI-assisted routability optimization, we compare the results of other routability-driven placers reported from a previous AI-assisted optimization work [52]. Due to different experimental settings and benchmarks (we set target density to 0.9 to make challenging congestion), we cannot make direct

TABLE I: Accuracy of routability prediction and runtime comparison between FCN model inference and Innovus flow.

Benchmark	from scratch		fine-tuning		Runtime (s)		
	NRMSE↓	SSIM↑	NRMSE↓	SSIM↑	Innovus	FCN	Speed-up
superblue11_a	0.081	0.547	0.042	0.709	13002	0.011	1182000
superblue14	0.089	0.382	0.049	0.534	5455	0.011	495910
superblue16_a	0.082	0.524	0.062	0.611	8722	0.011	792910
superblue19	0.092	0.449	0.046	0.621	4620	0.011	420000
Average	0.086	0.476	**0.050**	**0.619**	31799	**0.011**	722705

comparison on routed wirelength and congestion, but we can still compare the runtime scalability with problem sizes. Note that such comparison may not be fair, but just to provide a sense of the potential speed-up we can get. We can see that AI-assisted approaches can achieve more than 43× speed-up on average over CPU-based algorithms like NTUplace4dr [53], as they rely on global routing for routability evaluation. Our method is also on average 3.4× faster than [52], as they leverage the gradient of routability prediction models to guide placement at each iteration, while we trigger the routability prediction model much less frequently for cell inflation.

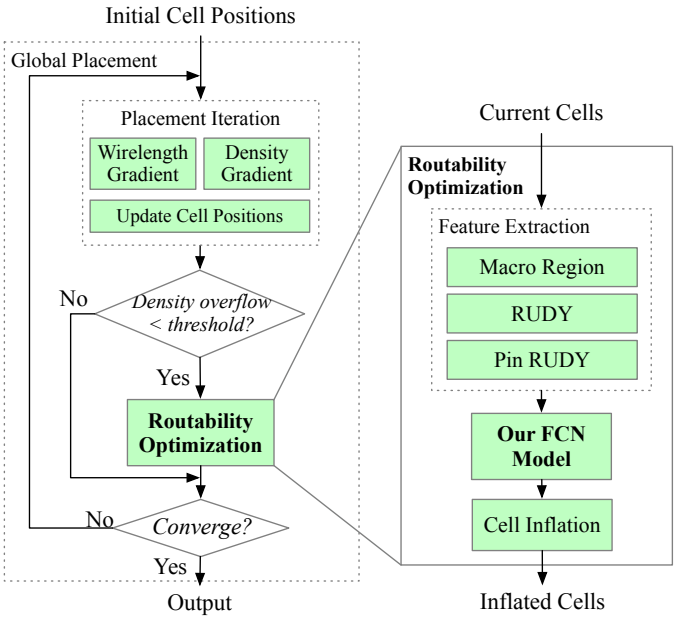


Fig. 4: The AI-assisted heterogeneous routability-driven placement flow.

V. Timing Modeling and Optimization

Timing is one of the ultimate goals for physical design due to its direct connection with circuit correctness and performance. While there has been plenty of room for circuit optimization during different design stages, it is highly nontrivial to place timing as one of our design targets, due to challenges in both *timing modeling* techniques and *timing optimization* algorithms. This section introduces our techniques to solve both challenges using the CircuitNet dataset and a heterogeneous platform.

A. AI-Assisted Timing Modeling using GNN and CircuitNet

Pre-routing design lacks detailed parasitics information of nets, which makes it difficult to accurately model net delay and cell loads. During the pre-routing design stages like placement, it is impossible to perform routing, parasitics extraction, and timing analysis on every optimization iteration due to the unacceptable runtime cost. While analytical timing estimation models can fail to account for complex interconnect effects, machine learning techniques have shown to be a promising way to model the timing of interconnects in a data-driven manner [55], [56], [57], [34], [36], [58]. This requires a flexible ML model trained under a large and diverse circuit dataset to yield enough accuracy.

To demonstrate the usefulness of CircuitNet on timing modeling, we use a graph neural network (GNN)-based model from a state-of-the-art pre-routing net delay prediction work [34]. Our basic idea is inspired by the calculation of Elmore delay model, which uses dynamic programming (DP) on net trees. During DP, the sum of subtree capacitive loads is collected from bottom to top, and the signal delay is propagated from top to bottom. We model this message-passing strategy as a GNN layer which we call the net convolution layer.

The net convolution layer is shown in Figure 6. One layer consists of a graph broadcast operation followed by a graph reduce operation. The graph broadcast operation sends the embedding of the net source to all net sinks, and the graph reduce operation collects the embeddings of net sinks at the net source. During the reduction, two independent channels are used with sum and max operations, respectively. We stack 3 of the net convolution layers to form our net convolution model. Our model takes the basic features of the unrouted input net including pin capacitances, pin directions, and input-output distances. The output embedding of the model on every sink is the net delay prediction from the source pin to this sink.

We use 10 samples from each design in CircuitNet-N28 and split them into training and testing sets based on designs. Our experimental results are shown in Table III. Overall, our GNN model can successfully learn the routing behavior that is transferable to test circuit designs. Meanwhile, our GNN is significantly faster than the traditional EDA flow including global routing and static timing analysis. For example, we are 7535–15024× faster than the Cadence Innovus flow. Thus when used in early placement stages, our AI-assisted timing model introduces low overhead inside optimization iterations.

TABLE II: Routability improvements on ISPD 2015 contest benchmarks [51] using our AI-assisted placement flow[1].

Benchmark	DREAMPlace H-CR	DREAMPlace V-CR	DREAMPlace WL (e+06 um)	Ours [42] H-CR	Ours [42] V-CR	Ours [42] WL (e+06 um)
superblue11_a	0.0222	0.0151	38.27	0.0239	0.0144	40.10
superblue14	0.1410	0.1508	25.77	0.1347	0.1481	25.87
superblue16_a	0.0719	0.0926	28.48	0.0595	0.0633	29.10
superblue19	0.1906	0.2638	19.65	0.1194	0.1811	20.16
Average	0.1064	0.1306	28.04	**0.0844**	**0.1017**	28.81
Ratio	1.260	1.284	0.973	**1.000**	**1.000**	1.000

[1] superblue12 is used for training, and the rest superblue designs are used to verify the placement results.

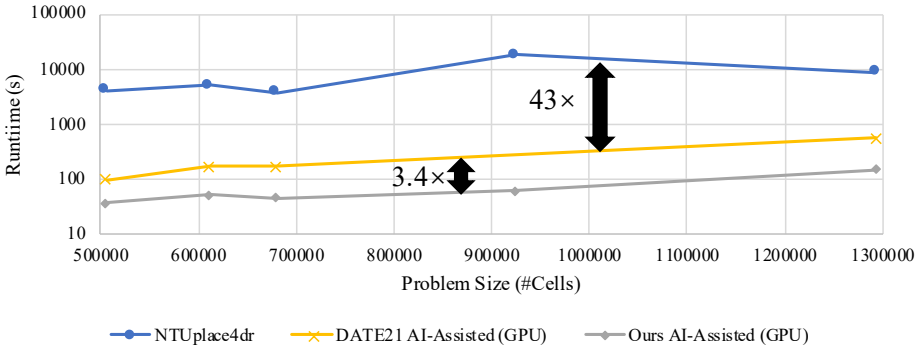


Fig. 5: Runtime comparison with NTUplace4dr [48] and gradient-based AI-assisted placement flow [52] on designs from the ISPD 2015 contest [54], based on data collected from [52]. Noted that the runtime for superblue11_a is not reported in [52], as it is used for training.

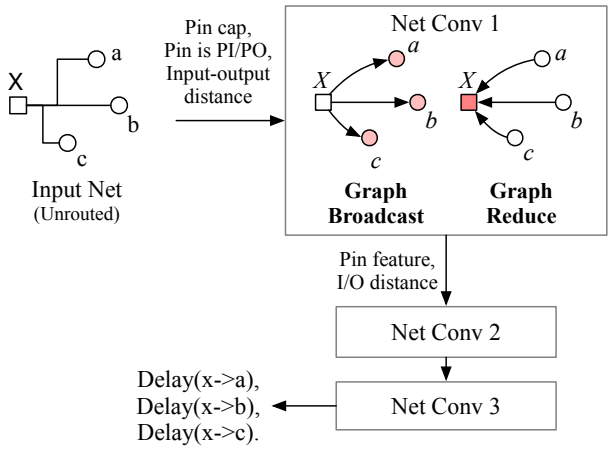


Fig. 6: The GNN-based net convolution model for pre-routing net delay prediction [34].

TABLE III: Accuracy of net delay prediction and runtime comparison between timing GNN model inference and the Innovus flow.

	Benchmark	R^2	Runtime (s) Innovus	Runtime (s) GNN	Runtime (s) Speed-up
Train	RISCY-a	0.9106	286	0.0212	13473.2
	RISCY-b	0.8815	280	0.0186	15024.5
	RISCY-FPU-a	0.9021	297	0.0292	10176.9
	RISCY-FPU-b	0.8818	292	0.0196	14879.7
Test	zero-riscy-a	0.8938	155	0.0217	7129.2
	zero-riscy-b	0.8706	149	0.0198	7535.7

B. AI-Inspired Heterogeneous Timing Optimization

Based on accurate timing modeling, timing optimization in the placement stage pays attention to timing-critical circuit regions and paths to prioritize their shortening. The performance of a timing-driven placement engine is determined by its explicitness in timing objective formulation, as well as its computational overhead compared to wirelength-driven placement flows. Most widely-used timing-driven optimization algorithms [59], [60], [61], [62], [63], [64], [65], [66] address the timing targets in indirect ways like net weighting due to the indifferentiable timing objectives, which harms their optimization outcome. Meanwhile, as the performance of placement engines has reached a new milestone thanks to GPU-based heterogeneous parallelism [5], the timing optimization algorithm is becoming the performance bottleneck due to its CPU affinity.

To improve the explicitness of the timing objectives, we proposed a differentiable timing-driven optimization framework [7]. We noted the analogy between the timing optimization problem in placement and the training of a multi-layer deep neural network (DNN) model, as illustrated in Figure 7. The trainable weights in a DNN are analogous to the movable cell locations in placement. The model structure of a DNN corresponds to the circuit topology of our placement instance. The output of a DNN is obtained by forward propagating the hidden representation vectors throughout the DNN layers, whereas the timing slacks of a circuit are obtained by forward propagating the arrival times throughout the directed acyclic

graph (DAG) of the design.

Inspired by these analogies, we learn from the training of a DNN using the back-propagation algorithm. In similar, we can calculate the gradient of timing objectives to the cell locations by back-propagating on the circuit topology. By smoothing the timing corner merges using the log-sum-exp formula, we build a differentiable timing engine that can be seamlessly integrated into an existing analytical placement flow.

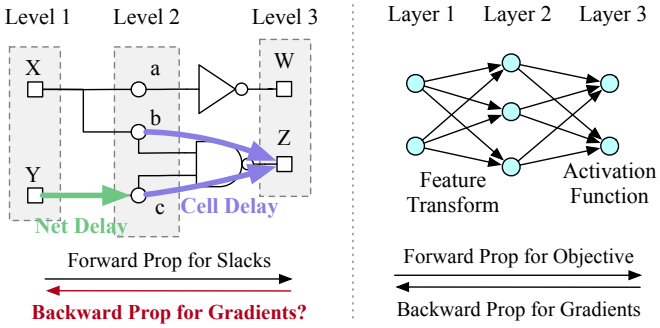


Fig. 7: The analogy between a timing analysis engine and a deep neural network [7].

The differentiable timing-driven placement algorithm can effectively reduce the timing violations in the placement stage, thanks to its direct modeling of the timing target. We have run the algorithm using 8 different netlists from the ICCAD 2015 contest [54], as well as 6 different netlists from CircuitNet. We list the result of worst negative slack (WNS), total negative slack (TNS), and wirelength in Table IV and Table V, respectively. Almost all the designs tested witness improvements in WNS and TNS, without notable wirelength degradation compared to a non-timing-driven flow. The average WNS and TNS improvements are 15.0% and 17.6% on CircuitNet, respectively.

During the placement optimization, the differentiable timing analysis and optimization are repeatedly invocated within the inner optimization loop which consists of hundreds to thousands of iterations. This makes the performance of timing analysis critical for efficient chip optimization. With the size of modern circuit designs reaching millions of logic gates and logic depth reaching hundreds, traditional CPU-based parallel timing analysis engines fail to deliver reasonable performance due to their poor scalability that saturates at around 16 CPU threads. We solve this scalability challenge using heterogeneous CPU-GPU parallelism [21]. We develop high-performance timing analysis kernels covering both forward slack computation and backward gradient computation.

Figure 8 shows the GPU-accelerated timing-driven placement flow. We implement the kernels as PyTorch operators to make use of the auto-differentiation algorithm from the existing ML framework. In addition to the wirelength and density kernels from DREAMPlace, we add differentiable WNS and TNS kernels in objective and gradient calculation. Thanks to the efficient heterogeneous kernels and our AI-inspired differentiable flow, our timing-driven placement engine outperforms

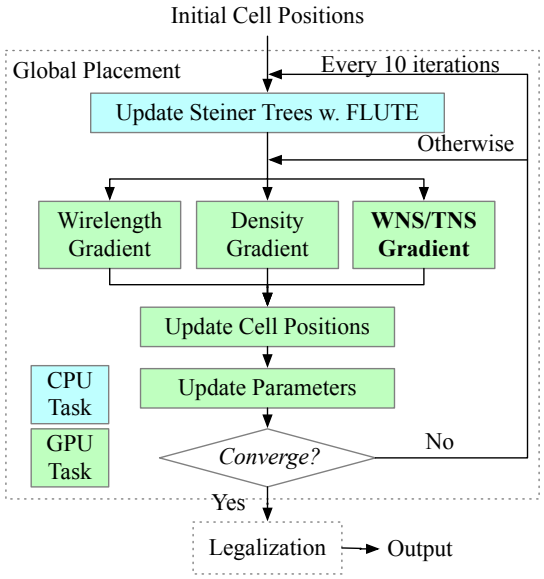


Fig. 8: The AI-inspired heterogeneous timing-driven placement flow [7].

the CPU-based OpenROAD flow [67], [43] and the GPU-based net weighting flow [66] by a notable gap on runtime. Figure 9 shows the runtime comparison under different design scales. We achieved an average 18.9× and 1.8× speed-up over OpenROAD and net weighting, respectively.

VI. Conclusion

In this paper, we have introduced AI-enabled GPU acceleration for routability and timing optimization in physical design. The main idea is to leverage AI-assisted cross-stage prediction and AI-inspired design optimization to break the inter-stage information dependency and achieve massive parallelism on heterogeneous CPU-GPU platforms. AI-assisted cross-stage prediction based on an open-source AI for EDA dataset CircuitNet can achieve thousands of times of speedup compared with conventional iterative design feedbacks. AI-inpsired design optimization brings a new way to effectively optimize timing with GPU-friendly computation workloads. Our experimental results demonstrate that over 18× speedup can be achieved on heterogeneous CPU-GPU platforms with competitive performance compared with conventional CPU-based approaches. In the future, we plan to extend the methodology to other design stages like floorplanning, powerplanning, and clock tree synthesis.

Acknowledge

This work was supported in part by the National Key Research and Development Program of China (No. 2021ZD0114702).

References

[1] T. Huang, G. Guo, C. Lin, and M. D. F. Wong, "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," *IEEE TCAD*, vol. 40, no. 4, pp. 776–789, 2021.

[2] J. Cong and Y. Zou, "Parallel multi-level analytical global placement on graphics processing units," in *Proc. ICCAD*. ACM, 2009, pp. 681–688.

TABLE IV: Timing slack improvements on ICCAD 2015 contest benchmarks [54] using our differentiable timing-driven placement flow, compared with DREAMPlace, OpenROAD, and net weighting.

Benchmark	DREAMPlace [5]			OpenROAD [67], [43]			Net Weighting [68], [66]			Ours		
	WNS	TNS	HPWL	WNS	TNS	HPWL	WNS	TNS	HPWL	WNS	TNS	HPWL
superblue1	-18.87	-262.44	421.97	-28.01	-192.37	462.40	-14.10	-85.03	443.14	**-10.77**	**-74.85**	423.80
superblue3	-27.65	-76.64	478.18	-23.31	-70.21	598.70	-16.43	-54.74	482.40	**-12.37**	**-39.43**	478.37
superblue4	-22.04	-290.88	311.97	-21.65	-190.71	324.90	-12.78	-144.38	335.94	**-8.49**	**-82.92**	312.23
superblue5	-48.92	-157.82	488.29	-35.72	-140.44	518.80	-26.76	**-95.78**	556.18	**-25.21**	-108.08	488.72
superblue7	-19.75	-141.55	604.28	-19.02	-194.62	627.40	**-15.22**	-63.86	603.97	**-15.22**	**-46.43**	602.10
superblue10	-26.10	-731.94	935.93	-23.35	-575.78	946.60	-31.88	-768.75	1036.67	**-21.97**	**-558.05**	934.44
superblue16	-17.71	-453.57	435.76	-18.27	-523.53	455.50	-12.11	-124.18	448.11	**-10.85**	**-87.03**	485.12
superblue18	-20.29	-96.76	243.03	-9.95	-75.82	264.80	-11.87	-47.25	253.64	**-7.99**	**-19.31**	243.58
Avg. Ratio	1.897	3.125	0.987	1.712	2.889	1.066	1.282	1.472	1.043	1.000	1.000	1.000

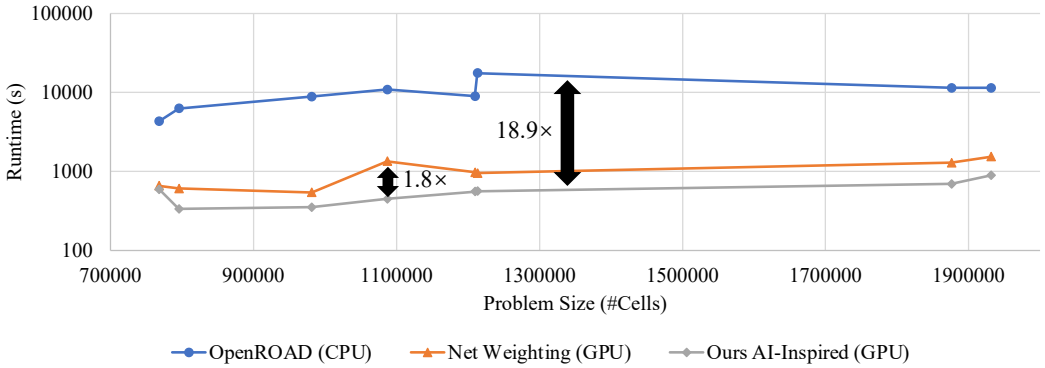


Fig. 9: Runtime comparison with OpenROAD and net weighting-based timing-driven placement flow on designs from the ICCAD 2015 contest [54]. Data for OpenROAD and net-weighting-based method is collected from [66] on a Linux host with 40 Intel Xeon CPU cores, 4 GeForce RTX 2080 Ti GPUs, and 256 GB RAM.

TABLE V: Timing slack improvements on CircuitNet using our differentiable timing-driven placement flow.

Benchmark	DREAMPlace			Ours		
	WNS	TNS	HPWL	WNS	TNS	HPWL
RISCY-a	-68.52	-246	1048.4	**-57.25**	**-223.3**	1051.2
RISCY-b	-51.45	-113.3	1132.9	**-46.28**	**-94.83**	1132.0
RISCY-FPU-a	-98.87	-823.7	1659.5	**-97.89**	**-783.9**	1663.4
RISCY-FPU-b	-226.7	-1038	1859.7	**-148.5**	**-687.6**	1845.2
zero-riscy-a	-26.77	-175.1	936.8	**-25.96**	**-156.5**	934.8
zero-riscy-b	-49.84	-49.14	830.5	**-48.56**	**-45.45**	824.4
Avg. Ratio	1.150	1.176	1.002	1.000	1.000	1.000

[3] C.-X. Lin and M. D. Wong, "Accelerate analytical placement with gpu: A generic approach," in *Proc. DATE*. IEEE, 2018, pp. 1345–1350.

[4] Z. Guo, J. Mai, and Y. Lin, "Ultrafast cpu/gpu kernels for density accumulation in placement," in *Proc. DAC*, San Francisco, CA, December 2021.

[5] 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 TCAD*, 2020.

[6] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan, "ABCD-Place: Accelerated batch-based concurrent detailed placement on multi-threaded cpus and gpus," *IEEE TCAD*, 2020.

[7] Z. Guo and Y. Lin, "Differentiable-timing-driven global placement," in *Proc. DAC*. ACM, 2022.

[8] L. Liu, B. Fu, M. D. F. Wong, and E. F. Y. Young, "Xplace: An extremely fast and extensible global placement framework," in *Proc. DAC*, 2022.

[9] P. Liao, H. Liu, Y. Lin, B. Yu, and M. Wong, "On a moreau envelope wirelength model for analytical global placement," in *Proc. DAC*. San Francisco, CA: ACM/IEEE, July 2023.

[10] H. Yang, K. Fung, Y. Zhao, Y. Lin, and B. Yu, "Mixed-cell-height legalization on cpu-gpu heterogeneous systems," in *Proc. DATE*, Antwerp, Belgium, March 2022.

[11] S. Dhar and D. Z. Pan, "GDP: GPU accelerated detailed placement," in *Proc. HPEC*, Sept 2018.

[12] S. Dhar, L. Singhal, M. Iyer, and D. Pan, "Fpga accelerated fpga placement," in *Proc. FPL*. IEEE, 2019, pp. 404–410.

[13] Y. Meng, W. Li, Y. Lin, and D. Z. Pan, "elfplace: Electrostatics-based placement for large-scale heterogeneous fpgas," *IEEE TCAD*, vol. 41, pp. 155–168, January 2021.

[14] J. Mai, Y. Meng, Z. Di, and Y. Lin, "Multi-electrostatic fpga placement considering slicel-slicem heterogeneity and clock feasibility," in *Proc. DAC*. San Francisco, CA: ACM/IEEE, July 2022.

[15] J. Mai, J. Wang, Z. Di, G. Luo, Y. Liang, and Y. Lin, "Openparf: An open-source placement and routing framework for large-scale heterogeneous fpgas with deep learning toolkit," in *Proc. ASICON*. Nanjing, China: IEEE, October 2023.

[16] S. Liu, Y. Pu, P. Liao, H. Wu, R. Zhang, Z. Chen, W. Lv, Y. Lin, and B. Yu, "Fastgr : Global routing on cpu-gpu with heterogeneous task graph scheduler," *IEEE TCAD*, vol. 42, pp. 2317–2330, October 2022.

[17] S. Lin, J. Liu, E. F. Young, and M. D. Wong, "Gamer: Gpu-accelerated maze routing," *IEEE TCAD*, vol. 42, no. 2, pp. 583–593, 2022.

[18] S. Lin and M. D. Wong, "Superfast full-scale cpu-accelerated global routing," in *Proc. ICCAD*, 2022, pp. 1–8.

[19] X. Jiang, Y. Lin, and Z. Wang, "Fpga-accelerated maze routing kernel for vlsi designs," in *Proc. ASPDAC*. Virtual Conference: IEEE, January 2022.

[20] Y. Shen and J. Hu, "GPU acceleration for PCA-based statistical static timing analysis," in *Proc. ICCD*. IEEE, 2015, pp. 674–679.

[21] Z. Guo, T.-W. Huang, and Y. Lin, "Gpu-accelerated static timing analysis," in *Proc. ICCAD*. ACM, 2020.

[22] G. Guo, T.-W. Huang, Y. Lin, and M. Wong, "Gpu-accelerated path-based timing analysis," in *Proc. DAC*. ACM, 2021.

[23] Z. Guo, T.-W. Huang, and Y. Lin, "Heterocppr: Accelerating common path pessimism removal with heterogeneous cpu-gpu parallelism," in *Proc. ICCAD*, Virtual Conference, November 2021.

[24] ——, "Accelerating static timing analysis using cpu-gpu heterogeneous parallelism," *IEEE TCAD*, 2023.

[25] M. Rapp, H. Amrouch, Y. Lin, B. Yu, D. Z. Pan, M. Wolf, and J. Henkel, "Mlcad: A survey of research in machine learning for cad," *IEEE TCAD*, vol. 41, pp. 3162–3181, November 2021.

[26] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *Proc. ICCAD*. IEEE, 2018, pp. 1–8.

[27] C. Yu and Z. Zhang, "Painting on placement: Forecasting routing congestion using conditional generative adversarial nets," in *Proc. DAC*, 2019, pp. 1–6.

[28] R. Kirby, S. Godil, R. Roy, and B. Catanzaro, "Congestionnet: Routing congestion prediction using deep graph neural networks," in *Proc. VLSI-SoC*. IEEE, 2019, pp. 217–222.

[29] M. B. Alawieh, W. Li, Y. Lin, L. Singhal, M. Iyer, and D. Z. Pan, "High-definition routing congestion prediction for large-scale fpgas," in *Proc. ASPDAC*. IEEE/ACM, January 2020.

[30] R. Liang, H. Xiang, D. Pandey, L. Reddy, S. Ramji, G.-J. Nam, and J. Hu, "Design rule violation prediction at sub-10nm process nodes using customized convolutional networks," *TCADICS*, 2021.

[31] B. Wang, G. Shen, D. Li, J. Hao, W. Liu, Y. Huang, H. Wu, Y. Lin, G. Chen, and P. A. Heng, "Lhnn: Lattice hypergraph neural network for vlsi congestion prediction," in *Proc. DAC*. San Francisco, CA: ACM/IEEE, July 2022.

[32] Y. Chen, J. Mai, X. Gao, M. Zhang, and Y. Lin, "Macrorank: Ranking macro placement solutions leveraging translation equivariancy," in *Proc. ASPDAC*, Tokyo, Japan, January 2023.

[33] 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. 106:1–106:6.

[34] 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.

[35] X. He, Z. Fu, Y. Wang, C. Liu, and Y. Guo, "Accurate timing prediction at placement stage with look-ahead rc network," in *Proc. DAC*, 2022, pp. 1213–1218.

[36] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Fast and Accurate Wire Timing Estimation Based on Graph Learning," in *Proc. DATE*. Antwerp, Belgium: IEEE, Apr. 2023, pp. 1–6.

[37] ——, "Graph-learning-driven path-based timing analysis results predictor from graph-based timing analysis," in *Proc. ASPDAC*, 2023, pp. 547–552.

[38] Y. Zhou, H. Ren, Y. Zhang, B. Keller, B. Khailany, and Z. Zhang, "Primal: Power inference using machine learning," in *Proc. DAC*, 2019, pp. 1–6.

[39] Z. Xie, X. Xu, M. Walker, J. Knebel, K. Palaniswamy, N. Hebert, J. Hu, H. Yang, Y. Chen, and S. Das, "Apollo: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors," in *Proc. MICRO*, 2021, pp. 1–14.

[40] Z. Xie, "Efficient runtime power modeling with on-chip power meters," in *Proc. ISPD*, 2023, pp. 168–174.

[41] Z. Chai, Y. Zhao, Y. Lin, W. Liu, R. Wang, and R. Huang, "Circuitnet: An open-source dataset for machine learning applications in electronic design automation (eda)," *SCIENCE CHINA Information Sciences*, September 2022.

[42] Z. Chai, Y. Zhao, W. Liu, Y. Lin, R. Wang, and R. Huang, "Circuitnet: An open-source dataset for machine learning in vlsi cad applications with improved domain-specific evaluation metric and learning strategies," *IEEE TCAD*, 2023.

[43] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "Replace: Advancing solution quality and routability validation in global placement," *IEEE TCAD*, 2018.

[44] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE TCAD*, vol. 27, no. 7, pp. 1228–1240, 2008.

[45] G. Guo, T.-W. Huang, Y. Lin, Z. Guo, S. Yellapragada, and M. D. F. Wong, "A gpu-accelerated framework for path-based timing analysis," *IEEE TCAD*, pp. 1–1, 2023.

[46] Y. Zhao, Z. Chai, Y. Lin, R. Wang, and R. Huang, "Hybridnet: Dual-branch fusion of geometrical and topological views for vlsi congestion prediction." Nanjing, China: IEEE/ACM, May 2023.

[47] X. He, T. Huang, W.-K. Chow, J. Kuang, K.-C. Lam, W. Cai, and E. F. Young, "Ripple 2.0: High quality routability-driven placement via global router integration," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–6.

[48] C.-C. Huang, H.-Y. Lee, B.-Q. Lin, S.-W. Yang, C.-H. Chang, S.-T. Chen, Y.-W. Chang, T.-C. Chen, and I. Bustany, "NTUplace4dr: A Detailed-Routing-Driven Placer for Mixed-Size Circuit Designs With Technology and Region Constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (ICCAD)*, vol. 37, no. 3, pp. 669–681, 2018.

[49] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "RePlAce: Advancing Solution Quality and Routability Validation in Global Placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (ICCAD)*, vol. 38, no. 9, pp. 1717–1730, 2019.

[50] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *Automation Test in Europe Conference Exhibition 2007 Design*, 2021.

[51] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proc. ISPD*, 2015, pp. 157–164.

[52] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," in *Proc. DATE*, Virtual Conference, February 2021.

[53] C.-C. Huang, H.-Y. Lee, B.-Q. Lin, S.-W. Yang, C.-H. Chang, S.-T. Chen, Y.-W. Chang, T.-C. Chen, and I. Bustany, "Ntuplace4dr: A detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 3, pp. 669–681, 2017.

[54] M. Kim, J. Hu, J. Li, and N. Viswanathan, "ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite," in *Proc. ICCAD*, 2015, pp. 921–926.

[55] 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.

[56] 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.

[57] 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.

[58] K. Chang, J. Ahn, H. Park, K.-M. Choi, and T. Kim, "DTOC: integrating Deep-learning driven Timing Optimization into the state-of-the-art Commercial EDA tool," in *Proc. DATE*. Antwerp, Belgium: IEEE, Apr. 2023, pp. 1–6.

[59] M. Burstein and M. N. Youssef, "Timing influenced layout design," in *Proc. DAC*. IEEE, 1985, pp. 124–130.

[60] 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.

[61] 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.

[62] T. Kong, "A novel net weighting algorithm for timing-driven placement," in *Proc. ICCAD*, 2002, pp. 172–176.

[63] B. Halpin, C. R. Chen, and N. Sehgal, "A sensitivity based placer for standard cells," in *Proc. GLSVLSI*, 2000, pp. 193–196.

[64] 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.

[65] Z. Xiu and R. A. Rutenbar, "Timing-driven placement by grid-warping," in *Proc. DAC*, 2005, pp. 585–591.

[66] 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 TCAD*, pp. 1–1, 2023.

[67] 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 *Proc. DAC*, 2019, pp. 1–4.

[68] 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.