OpenPARF: An Open-Source Placement and Routing Framework for Large-Scale Heterogeneous FPGAs with Deep Learning Toolkit

(Invited Paper)

Jing Mai[1,2†], Jiarui Wang[1,2†], Zhixiong Di[3], Guojie Luo[1,4], Yun Liang[2,4,5], Yibo Lin[2,4,5*]
[1]School of Computer Science, Peking University [2]School of Integrated Circuits, Peking University
[3]School of Information Science and Technology, Southwest Jiaotong University
[4]Institute of Electronic Design Automation, Peking University, Wuxi, China
[5]Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China
Email: {jingmai,jiaruiwang,gluo,ericlyun,yibolin}@pku.edu.cn, zxdi@home.swjtu.edu.cn

***Abstract*—This paper proposes `OpenPARF`, an open-source placement and routing framework for large-scale FPGA designs[1]. `OpenPARF` is implemented with the deep learning toolkit `PyTorch` and supports massive parallelization on GPU. The framework proposes a novel asymmetric multi-electrostatic field system to solve FPGA placement. It considers fine-grained routing resources inside configurable logic blocks (CLBs) for FPGA routing and supports large-scale irregular routing resource graphs. Experimental results on ISPD 2016 and ISPD 2017 FPGA contest benchmarks and industrial benchmarks demonstrate that `OpenPARF` can achieve 0.4-12.7% improvement in routed wirelength and more than 2× speedup in placement. We believe that `OpenPARF` can pave the road for developing FPGA physical design engines and stimulate further research on related topics.**

I. Introduction

Computer-Aided Design (CAD) of Field-Programmable Gate Arrays (FPGAs) has been a hot topic in the rapid advancement and adoption of FPGA technology over the past decades [1]. FPGA CAD flow consists of four major steps: logic synthesis, placement, routing, and bitstream generation [2]. Among these steps, placement and routing (PAR) are two critical steps that affect the overall timing and power performance of the FPGA design [3]. It is reported that routing alone accounts for 41-86% runtime of the entire FPGA CAD flow [4]. Therefore, the quality and efficiency of PAR algorithms highly impact FPGA design closure.

Due to the high heterogeneity of FPGA architectures, FPGA PAR has several unique characteristics that are different from ASIC. 1) Instance packing. A collective packing is necessitated in placement to ensconce instances within the designated site. Packing instances must be subject to sophisticated constraints that vary across FPGA architectures [5]. 2) Resource heterogeneity. Sites, categorized by diverse functions such as CLB, DSP, BRAM, IO, etc., exhibit a non-uniform distribution across the FPGA layout [6]. 3) Large routing scale. The inclusion of fine-grained routing resources with CLBs gives rise to routing resource graphs with billions of nodes, resulting in substantial routing complexity and runtime overhead [7].

To achieve high quality and efficiency, the literature has extensively studied PAR algorithms for FPGAs in the past decades [8]–[20]. However, most of the existing works highly rely on FPGA vendors' CAD tools to obtain indirect feedback and tightly bind to vendors' architectures, limiting the flexibility of algorithms and the ability to adapt to new FPGA architectures.

In order to alleviate the burden of reinventing the wheel and facilitate research on FPGA physical design, in this paper, we propose `OpenPARF`, an open-source placement and routing framework for large-scale heterogeneous FPGAs with deep learning toolkits. The main contributions are summarized as follows.

- `OpenPARF` is an open-source academic FPGA PAR engine that supports complex industrial FPGA architectures with *state-of-the-art* (SOTA) algorithms. It is implemented with the deep learning toolkit `PyTorch`, running on both CPU and GPU platforms. It is highly flexible and extensible for new FPGA architectures and PAR algorithms.
- `OpenPARF` implements the SOTA nonlinear FPGA placement algorithms based on an asymmetrical multi-electrostatic field system. It is capable of achieving superior placement results under various constraints such as routability, clock feasibility, and SLICEL-SLICEM heterogeneity from advanced FPGA architectures [5].
- `OpenPARF` implements a two-stage FPGA routing algorithm. It supports fine-grained CLB-level routing models and flexible scenarios such as logic pin inequivalence. It is capable of alleviating routing congestion on advanced FPGA architectures effectively [7].

Compared with other SOTA academic PAR engines, `OpenPARF` can reduce 0.4%-12.7% routed wirelength as well as more than 2× speedup in placement efficiency. We believe that `OpenPARF` will stimulate the development of FPGA PAR algorithms and foster a surge of research in the future.

II. The OpenPARF Framework

A. Overall Flow

The overall flow of `OpenPARF` is depicted in Fig. 1. `OpenPARF` consists of two major components: placement and routing. Firstly, placement information files (bookshelf format) and routing architecture files (XML format like `VTR` [16]) are loaded and constructed into the `OpenPARF` internal data structure. Second, based on the internal data structure, `OpenPARF` employs multi-electrostatic-based FPGA global placement algorithms and incorporates SLICEL-SLICEM heterogeneity, routability, and clock

†Equal contribution, ordered by last names. *Corresponding author.
[1]`OpenPARF` is available at https://github.com/PKU-IDEA/OpenPARF.

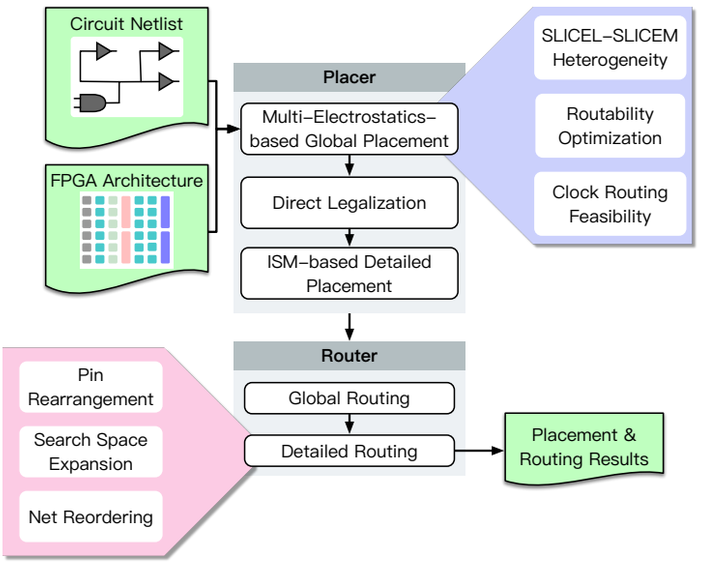

Fig. 1: The overall flow of `OpenPARF`.

feasibility into a unified optimization framework [5]. `OpenPARF` subsequently employs direct legalization algorithm to generate feasible placement results and further improves the placement quality by applying Independent Set Matching (ISM) based detailed placement algorithm [21]. Subsequently, the placement results are fed into the two-stage FPGA router in `OpenPARF`. Finally, `OpenPARF` generates feasible routing results through global and detailed routing with three strategies to resolve routing congestion and improve routing quality, i.e., *pin rearrangement*, *search space expansion*, and *net reordering*. In the following subsections, we will elaborate on the core features of placement and routing.

B. Placement Features

The objective of placement in `OpenPARF` is to minimize wirelength while considering SLICEL-SLICEM heterogeneity, routability optimization, and clock routing feasibility.

1) SLICEL-SLICEM Heterogeneity: The CLBs on FPGA can be further categorized into SLICEL and SLICEM. The logic resources on SLICEL can be configured as LUTs. Besides LUTs, SLICEM has additional logic resources and can hence be configured as either a distributed RAM or a SHIFT with storage capabilities. However, if a SLICEM is configured as LUTs, it cannot be utilized as SHIFTs or distributed RAMs, and vice versa.

To support SLICEL-SLICEM heterogeneity, the placer constructs two sets of electric fields: *LUTL* and *LUTM-AL* [5]. LUTL models the LUT resources supplied in SLICEL and SLICEM, while LUTM-AL models the additional logic resources supplied in SLICEM, but not in SLICEL. A LUT instance only utilizes resources in the LUTL field, whereas a distributed RAM or SHIFT instance utilizes resources in both the LUTL and LUTM-AL fields. This field setup prevents a distributed RAM or SHIFT instance from being placed in SLICEL sites, but allows a LUT instance to be placed in both SLICEL and SLICEM sites.

2) Routability Optimization: Routability reveals whether the current placement is difficult or impossible to route given available target device routing resources. Our placer adopts the area inflation-based routability optimization strategy [22] to improve the routability of placement results. Instances in congested regions are inflated by a greater factor. Therefore, the placer can effectively reduce the routing congestion and improve the routability of the placement result.

3) Clock Routing Feasibility: Advanced FPGAs have dedicated clock routing resources to route clock signals. Take Xilinx *UltraScale VU095* as an example of FPGA devices. The target FPGA device is divided into rectangular-shaped clock regions (CRs) in a grid manner, and each CR can be further subdivided into pairs of lower and upper half columns (HCs). The clock architecture imposes clock routing constraints on placement, the clock region constraint, and the half-column constraint [23]. Our placer adopts the clock network planning algorithm in [5], and the clock feasibility is incorporated into the placement objective function as a penalty term.

Nested Lagrangian Relaxation: `OpenPARF` adopts the nested Lagrangian relaxation algorithm [5] to solve the placement problem with SLICEL-SLICEM heterogeneity, routability optimization, and clock routing feasibility. SLICEL-SLICEM heterogeneity is modeled by the *LUTL* and *LUTM-AL* electric field systems, which are incorporated into the electrostatic density function. Routing congestion is gradually resolved by inflating the area of instances in congested regions. The clock routing feasibility is ensured by the clock network planning algorithm. The placement problem is solved by iteratively solving the relaxed placement problem and updating the Lagrangian multipliers until the placement result is feasible.

C. Routing Features

The router of our `OpenPARF` framework consists of two stages, coarse-grained level global routing and fine-grained level detailed routing [7]. In the global routing phase, we generate an inter-site level routing result to guide the behavior of the detailed router. In the detailed routing phase, we generate the logic element routing result at both inter-site and intra-site levels.

1) Global Routing: The target of the global router is to generate inter-site level coarse-grained routing results to guide the follow-up detailed routing. global router follows the `Pathfinder` algorithm by abstracting the FPGA layout as a grid graph. logic site is represented as a vertex, and the routing channels are represented as edges in the grid graph. Two vertices are connected if there are routing channels connecting them. The capacity of an edge is the width of the routing channel between its two logic sites, and the edge capacity is set to the routing channel width.

2) Detailed Routing: The target of the detailed router is to generate the inter- and intra-site routing paths under the guidance of the global routing results. Detailed router follows a similar negotiation-based algorithm to the global router, but abstracts the FPGA layout to a more fine-grained routing resource graph (RRG). In the detailed router's RRG, the vertex represents the logic pin inside the FPGA, and the directed edge represents the logic connection between two logic pins.

Detailed router applies three strategies to efficiently and effectively generate detailed routing results. 1) To deal with routing congestion at the input of each LUT, the detailed router applies *pin rearrangement* technique. Detailed router regards the input logic

TABLE I: Placement runtime (PRT in seconds), routing runtime (RRT in minutes), and routed wirelength ($\times 10^4$ RWL) comparison on ISPD 2016 benchmarks [24].

Design	#LUT/#FF/#RAM/#DSP	RippleFPGA [13]			DREAMPlaceFPGA [20]			OpenPARF		
		PRT	RRT	RWL	PRT	RRT	RWL	PRT	RRT	RWL
FPGA01	50K/55K/0/0	41	3	36.44	32	3	31.78	39	3	31.72
FPGA02	100K/66K/100/100	64	5	75.29	57	5	68.17	59	5	67.73
FPGA03	250K/170K/600/500	245	17	346.91	108	15	299.56	120	15	294.75
FPGA04	250K/172K/600/500	337	22	632.96	97	22	569.85	112	22	577.30
FPGA05	250K/174K/600/500	391	57	1222.46	91	56	1167.60	123	54	1148.72
FPGA06	350K/352K/1000/600	593	25	652.41	183	29	571.11	218	27	573.54
FPGA07	350K/355K/1000/600	782	46	1106.96	159	51	964.44	209	45	965.24
FPGA08	500K/216K/600/500	490	40	958.26	146	36	911.67	184	38	896.91
FPGA09	500K/366K/1000/600	738	58	1327.34	191	54	1203.49	260	50	1198.22
FPGA10	350K/600K/1000/600	1180	28	711.48	180	28	544.52	258	35	542.02
FPGA11	480K/363K/1000/400	721	58	1281.65	148	56	1250.49	221	59	1253.78
FPGA12	500K/602K/600/500	883	40	761.37	184	37	674.21	290	38	670.94
Ratio		2.771	1.015	1.127	0.786	1.000	1.004	1.000	1.000	1.000

pins of a LUT as a vertex with large capacity, and rewrites the truth table of the LUT after detailed routing. 2) Detailed router dynamically applies *search space expansion* technique to restrict the search space. 3) To effectively resolve the routing congestion between different logic nets, the detailed router adopts the *net reordering* strategy at the beginning of each rip-up and reroute iteration.

III. Experimental Results

OpenPARF is implmented in C++ and Python along with the open-source machine learning toolkit Pytorch for fast gradient computation [25]. We conduct experiments on a Linux server that consists of an Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz (40 cores), one NVIDIA RTX 2080Ti GPU, and 512GB memory. We demonstrate the effectiveness and efficiency of our proposed algorithm on both ISPD 2016 [24] and ISPD 2017 [23] academic benchmarks and industrial benchmarks. The industrial benchmarks are provided by our industrial collaborator. We also complete the routing architecture of the academic benchmarks under their guidance. TABLE I, TABLE II and TABLE III provide the overview of evaluation benchmarks. The benchmarks encompass various instance types and netlist sizes ranging from 21K to 1100K.

A. Evaluation on Academic Benchmarks

TABLE I and TABLE II elucidate the comparative results between OpenPARF and two SOTA FPGA placers, RippleFPGA [13] and DREAMPlaceFPGA [20] [2]. It is noteworthy that RippleFPGA and DREAMPlaceFPGA embody the epitome of analytical placement algorithms, one based on quadratic programming, and the other on non-convex optimization models. By incorporating the placement results of the aforementioned placers into OpenPARF's router, We proceed to access placement runtime (PRT), routing runtime (RRT), and routed wirelength (RWL) between them, with routing time serving as an indicator of routability.

[2]DREAMPlaceFPGA includes two works, namely [19] and [20]. In this paper, the placement runtime of DREAMPlaceFPGA is extracted from the latest work [20].

1) Evaluation on ISPD 2016 Benchmarks: The experimental results on the ISPD 2016 experiment unveil OpenPARF's advantages over other placers, showcasing a reduction in wirelength by 12.7% and 0.4%, respectively, coupled with up to 2.77× placement speedup. Notably, DREAMPlaceFPGA harnesses GPU acceleration for legalization. OpenPARF exhibits a 21.4% decrease in runtime when compared with DREAMPlaceFPGA. We believe that OpenPARF has vast potential for further performance enhancement by delving into the realms of heterogeneous parallelization, accelerating both legalization and detailed placement.

2) Evaluation on ISPD 2017 Benchmarks: We further conducted a comparative analysis between RippleFPGA and OpenPARF on ISPD2017 benchmarks. As DREAMPlaceFPGA does not support clock routing constraints in ISPD2017 benchmarks, we do not include it in the comparison. The experimental results demonstrate that OpenPARF outperforms RippleFPGA by achieving a 12.8% reduction in wirelength and with 2.251× speedup. It is noteworthy that OpenPARF exhibits a reduction of 3.7% in routing runtime compared to RippleFPGA, indicating that OpenPARF is capable of generating routing results with favorable routability even under intricate constraints.

B. Evaluation on Industrial Benchmarks

TABLE III presents the placement runtime, routing runtime, and routed wirelength of OpenPARF on industrial benchmarks, which encompass distributed RAMs and SHIFTs, exhibiting the heterogeneous nature of SLICEL-SLICEM constraints. The experimental results show the performance and efficiency of OpenPARF when dealing with SLICEL-SLICEM constraints. OpenPARF has demonstrated commendable efficacy by leveraging an asymmetric multi-electrostatic system to address the SLICEL-SLICEM heterogeneity.

IV. Conclusion and Future Work

This paper introduces OpenPARF, an open-source placement and routing framework for large-scale FPGAs. Built upon the deep learning toolkit PyTorch, OpenPARF supports GPU acceleration with agile and flexible programming interfaces. In placement, OpenPARF embodies the SOTA asymmetrical multi-electrostatic FPGA placement algorithms and harnesses the nested Lagrangian

TABLE II: Placement runtime (PRT in seconds), routing runtime (RRT in minutes), and routed wirelength ($\times 10^4$ RWL) comparison on ISPD 2017 benchmarks [23].

Design	#LUT/#FF/#BRAM/#DSP	#Clock	RippleFPGA [13]			OpenPARF		
			PRT	RRT	RWL	PRT	RRT	RWL
CLK-FPGA01	211K/324K/164/75	32	278	10	238.54	131	10	205.44
CLK-FPGA02	230K/280K/236/112	35	250	15	261.85	127	14	246.65
CLK-FPGA03	410K/481K/850/395	57	537	24	648.69	206	24	594.00
CLK-FPGA04	309K/372K/467/224	44	346	18	440.09	157	19	420.30
CLK-FPGA05	393K/469K/798/150	56	501	25	560.18	201	23	510.62
CLK-FPGA06	425K/511K/872/420	58	545	28	678.43	218	28	617.28
CLK-FPGA07	254K/309K/313/149	38	288	13	276.29	136	13	256.62
CLK-FPGA08	212K/257K/161/75	32	235	10	213.06	119	10	196.67
CLK-FPGA09	231K/358K/236/112	35	312	13	297.02	148	14	250.98
CLK-FPGA10	327K/506K/542/255	47	465	23	544.07	195	14	451.28
CLK-FPGA11	300K/468K/454/224	44	421	24	516.67	182	30	421.52
CLK-FPGA12	277K/430K/389/187	41	378	18	403.59	167	20	336.03
CLK-FPGA13	339K/405K/570/262	47	393	21	464.78	178	19	428.41
Ratio			2.251	1.037	1.128	1.000	1.000	1.000

TABLE III: Placement runtime (PRT in seconds), routing runtime (RRT in minutes), and routed wirelength ($\times 10^3$ RWL) on industrial benchmarks.

Design	#LUT/#FF/ #BRAM/#DSP	#Distributed RAM + #Shift	#Net	OpenPARF		
				PRT	RRT	RWL
IND01	17K/11K/0/13	9	52492	72.36	10	90
IND02	11K/10K/0/24	6	26678	77.82	15	100
IND03	109K/12K/0/0	0	121554	109.54	108	1021
IND04	29K/17K/0/16	218	60968	69.39	19	283
IND05	64K/191K/64/928	29K	371808	126.38	109	2360
IND06	112K/65K/21/0	0	221182	88.28	176	1593
IND07	40K/156K/89/768	26K	294075	140.33	68	1450

relaxation methodology to resolve SLICEL-SLICEM heterogeneity, routability and clock routing feasibility. In routing, OpenPARF implements a two-stage FPGA routing algorithm, which supports fine-grained CLB-level routing models and flexible scenarios like logic pin inequivalence. We have a belief that OpenPARF can serve as a crucial platform for exploring next-generation high-performance FPGA placement and routing algorithms, and stimulates the development of future FPGA CAD algorithms.

ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation of China (Grant No. T2293700 and T2293701) and the 111 Project (B18001).

REFERENCES

[1] D. Chen, J. Cong, and P. Pan, "FPGA design automation: A survey," *Found. Trends Electron. Des. Autom.*, pp. 195–330, 2006.

[2] B. Ghavami and L. Shannon, "Unraveling the Integration of Deep Machine Learning in FPGA CAD Flow: A Concise Survey and Future Insights," *arXiv preprint*, 2023.

[3] S.-C. Chen and Y.-W. Chang, "FPGA placement and routing," in *Proc. ICCAD*, 2017, pp. 914–921.

[4] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-Driven Titan: Enabling Large Benchmarks and Exploring the Gap between Academic and Commercial CAD," *ACM TRETS*, pp. 1–18, 2015.

[5] J. Mai, Y. Meng, Z. Di, and Y. Lin, "Multi-electrostatic FPGA placement considering SLICEL-SLICEM heterogeneity and clock feasibility," in *Proc. DAC*, 2022, pp. 649–654.

[6] N. Zhang, X. Chen, and N. Kapre, "RapidLayout: Fast Hard Block Placement of FPGA-optimized Systolic Arrays Using Evolutionary Algorithm," *ACM TRETS*, pp. 1–23, 2022.

[7] J. Wang, J. Mai, Z. Di, and Y. Lin, "A Robust FPGA Router with Concurrent Intra-CLB Rerouting," in *Proc. ASPDAC*, 2023, pp. 529–534.

[8] C. Pui, G. Chen, W. Chow, K. Lam, J. Kuang, P. Tu, H. Zhang, E. F. Y. Young, and B. Yu, "RippleFPGA: a routability-driven placement for large-scale heterogeneous FPGAs," in *Proc. ICCAD*, 2016, p. 67.

[9] W. Li, S. Dhar, and D. Z. Pan, "Utplacef: A routability-driven FPGA placer with physical and congestion aware packing," *IEEE TCAD*, vol. 37, no. 4, pp. 869–882, 2018.

[10] R. Pattison, Z. Abuowaimer, S. Areibi, G. Grewal, and A. Vannelli, "Gplace: a congestion-aware placement tool for ultrascale FPGAs," in *Proc. ICCAD*, 2016, p. 68.

[11] W. Li, Y. Lin, and D. Z. Pan, "elfPlace: Electrostatics-based placement for large-scale heterogeneous FPGAs," in *Proc. ICCAD*, 2019, pp. 1–8.

[12] Z. Abuowaimer, D. Maarouf, T. Martin, J. Foxcroft, G. Grewal, S. Areibi, and A. Vannelli, "GPlace3.0: Routability-driven analytic placer for UltraScale FPGA architectures," *ACM TODAES*, pp. 66:1–66:33, 2018.

[13] G. Chen, C.-W. Pui, W.-K. Chow, K.-C. Lam, J. Kuang, E. F. Young, and B. Yu, "RippleFPGA: Routability-driven simultaneous packing and placement for modern FPGAs," *IEEE TCAD*, pp. 2022–2035, 2018.

[14] W. Li, Y. Lin, M. Li, S. Dhar, and D. Z. Pan, "Utplacef 2.0: A high-performance clock-aware FPGA placement engine," *ACM TODAES*, vol. 23, no. 4, pp. 42:1–42:23, 2018.

[15] W. Li, M. E. Dehkordi, S. Yang, and D. Z. Pan, "Simultaneous Placement and Clock Tree Construction for Modern FPGAs," in *Proc. FPGA*, 2019, pp. 132–141.

[16] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: High-performance CAD and Customizable FPGA Architecture Modelling," *ACM TRETS*, vol. 13, no. 2, pp. 1–55, Jun. 2020.

[17] J. Chen, Z. Lin, Y. Kuo, C. Huang, Y. Chang, S. Chen, C. Chiang, and S. Kuo, "Clock-Aware Placement for Large-Scale Heterogeneous FPGAs," *IEEE TCAD*, vol. 39, no. 12, pp. 5042–5055, 2020.

[18] T. Liang, G. Chen, J. Zhao, L. Feng, S. Sinha, and W. Zhang, "AMF-Placer: High-Performance Analytical Mixed-size Placer for FPGA," in *Proc. ICCAD*, 2021, pp. 1–6.

[19] R. S. Rajarathnam, M. B. Alawieh, Z. Jiang, M. Iyer, and D. Z. Pan, "DREAMPlaceFPGA: An Open-Source Analytical Placer for Large Scale Heterogeneous FPGAs using Deep-Learning Toolkit," in *Proc. ASPDAC*. Taipei, Taiwan: IEEE, Jan. 2022, pp. 300–306.

[20] R. S. Rajarathnam, Z. Jiang, M. A. Iyer, and D. Z. Pan, "DREAMPlaceFPGA-PL: An Open-Source GPU-Accelerated Packer-Legalizer for Heterogeneous FPGAs," in *Proc. ISPD*, 2023, pp. 175–184.

[21] W. Li and D. Z. Pan, "A new paradigm for FPGA placement without explicit packing," *IEEE TCAD*, vol. 38, no. 11, pp. 2113–2126, 2019.

[22] Y. Meng, W. Li, Y. Lin, and D. Z. Pan, "elfPlace: Electrostatics-based Placement for Large-Scale Heterogeneous FPGAs," *IEEE TCAD*, 2021.

[23] S. Yang, C. Mulpuri, S. Reddy, M. Kalase, S. Dasasathyan, M. E. Dehkordi, M. Tom, and R. Aggarwal, "Clock-aware FPGA placement contest," in *Proc. ISPD*, 2017, pp. 159–164.

[24] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven FPGA placement contest," in *Proc. ISPD*, 2016, pp. 139–143.

[25] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement," in *Proc. DAC*, 2019, pp. 1–6.