The 2023 MLCAD FPGA Macro Placement Benchmark Design Suite and Contest Results

Ismail Bustany, Grigor Gasparyan, Amit Gupta, Andrew B. Kahng*, Meghraj Kalase, Wuxi Li, Bodhisatta Pramanik*
AMD Inc., San Jose, CA, USA
*University of California at San Diego, La Jolla, CA, USA
Email: {ismail.bustany, grigor.gasparyan, a.gupta, meghraj.kalase, wuxi.li}@amd.com, *{abk, bopramanik}@ucsd.edu

***Abstract*—The 2023 IEEE/ACM MLCAD Workshop program included the inaugural MLCAD Contest, a multi-month research and development competition intended to focus attention on pressing challenges at the nexus of electronic design automation (EDA) and machine learning. The 2023 contest aimed to spur research into machine learning-driven solutions that can supplant state-of-the-art algorithms for FPGA macro placement. Several factors make this problem more challenging than its ASIC counterpart. These include: (i) discrete and SITE-typed columnated nature of the FPGA device layout; (ii) cascaded macros or carry chains that can greatly impact routability and timing closure because of their high pin count and net connectivities, (iii) prevalence of resource-bound multi-clock domains; (iv) larger number of macros (typically 100s-1000s) than in ASICs; and (v) fragmentation of placement across multiple dies. Two benchmark suites containing 140 public and 198 hidden designs with varying complexities (based on Rent exponent, LUT/FF/BRAM/DSP utilization, carry chains, cascaded macro shapes and number of clock domains) were created for this competition. 19 teams globally participated in the contest. We share the results from the 8 finalists, 6 of which had competitive solutions. Our goal was for the contest to draw international participation and catalyze industry-academic collaborations, leading to contributions in premier conferences and journals. Several contestants used electrostatic-based frameworks (e.g., [13], DreamPlaceFPGA [29] and OpenPARF [24]) for global placement solution evaluations, alongside core macro placement algorithms based on classical optimization approaches (e.g., min-cost flow, bipartite graph matching, and simulated annealing). One team used an RL-parameterized simulated annealing algorithm for their macro placement solution. The contributed solutions are insightful, and we remain optimistic about the contest's potential lasting influence for developing better algorithms for FPGA macro placement.**

***Index Terms*—CAD Contest, FPGA, macro placement, physical design, electronic design automation, computer-aided design, integrated circuits, MLCAD**

I. Introduction

VLSI physical design encompasses a suite of NP-complete challenges which place-and-route (P&R) tools strive to address efficiently, often in near-linear time. As we navigate advanced process nodes coupled with intricate signoff requisites, new physical and timing constraints are introduced into the workflow. This complexity amplifies the challenge for physical design algorithms to achieve top-tier power, performance and area (PPA) metrics without compromising design turnaround time. The unceasing demand for energy-efficient yet high-performance designs highlights the urgency for improved predictive models and advanced analytics to drive implementation flows. With advances in supervised and reinforcement learning, combined with the availability of extensive computational resources, Machine Learning (ML) has the potential to introduce a paradigm change for EDA tools. The inaugural 2023 MLCAD contest [37] was organized as part of the 2023 MLCAD Workshop [39] with the aim to invigorate research in this direction by providing a platform where industrial companies can share diverse design challenges and design cases. This enables academia to study the state-of-the-art IC design challenges and advance problem-solving techniques in the realm of ML-enabled EDA.

The 2023 MLCAD contest marked the inaugural contest for the MLCAD workshop, drawing participation from 19 teams all over the world. This year saw representation from Canada, China, Hong Kong, Korea, India, Taiwan and the U.S. Table III lists the participating teams, their university affiliations, and regions. The contest commenced in April and ended in August; Figure 1 shows the detailed timeline. After final submissions and evaluations, winners were announced at a 2023 MLCAD special session.

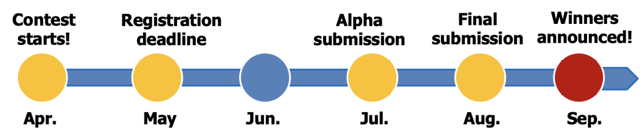

Fig. 1: Timeline of the 2023 MLCAD contest schedule

As one of the world's prominent semiconductor companies, Advanced Micro Devices (AMD) offered to co-organize the 2023 MLCAD contest. The topic of this year's contest was FPGA macro placement. The 2023 MLCAD contest witnessed an impressive level of participation, and the high quality of macro placers produced by contestants bodes well for the contest's future success in advancing ML-driven EDA research.

II. Background

Modern digital chips incorporate a significant number of macros, including components such as SRAMs and clock generators. While standard cells are more abundant and serve as the primary building blocks of digital designs, macros are considerably larger. Their size means they have greater influence on a chip's floorplan, affecting crucial design parameters such as wirelength, power, and area. Conventionally, high-quality macro placements have been achieved during floorplanning, which involves an initial (manual or algorithmic) placement of macros followed by an independent placement of the standard cells [11] [8] [14] [1]. The commonly-studied two-stage approach begins from an initial wirelength-driven placement. In the first stage, a macro placer places only the macros without fine-grain consideration of the standard cells. After this, the macros are fixed and the standard cells are placed in the remaining whitespaces. Recent advancements in mixed-size placement, which concurrently places macros and standard cells (cf. [2] and Cadence CMP), have demonstrated notable advantages over the traditional two-stage method. By exploiting a global view of all elements to be placed, mixed-size placement can identify macro locations consistent with high-quality locations. Examples of such placers include the annealing-based Dragon [32], partitioning-based Capo [30], and analytical placers [34] [18] [31] [5] [10] [13] [22]. However, macro legalization is

979-8-3503-0955-3/23/$31.00 ©2023 IEEE

challenging for mixed-size placement, especially when many macros are tightly packed.

Related work. Past research on macro placement can generally be categorized into packing-based, analytical, and – more recently – ML and reinforcement learning (RL) methods. Packing-based techniques focus on physical relationships among modules in a floorplan, and optimize module locations through iterative perturbation techniques. Analytical approaches leverage numerical methods to directly optimize floorplan layouts. Both strategies optimize a cost function that captures area, congestion and timing. Packing-based methods typically combine floorplan representations with heuristics such as Simulated Annealing (SA) to solve the floorplanning problem. Some prominent floorplan representations proposed in the literature include Sequence Pair [27], Corner-Stitching [28], B*-tree [6] [7], MP-tree [12], CP-tree [9] and MDPtree [23]. [16] and [17] are two recent packing-based macro placers that generate high-quality macro placements. Analytical methods model the objectives to be optimized as terms in the objective cost function or as constraints, then solve the constrained optimization problem mathematically. [36] is a work in this direction. ML-based approaches utilize machine learning techniques, such as expert systems [3] [33] and reinforcement learning [15] [25] [26] for macro placement.

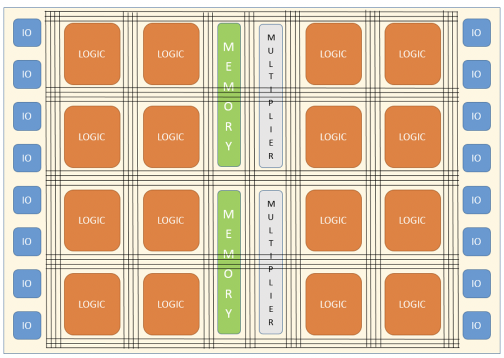

Fig. 2: Example of AMD FPGA architecture [35].

FPGA architecture. AMD FPGAs, an example of which is shown in Figure 2, consist of an array of programmable blocks of different types, including general logic (configuration logic block or *CLB*), memory (BRAM) and multiplier (DSP) blocks [35]. Each location of a programmable block is referred to as a *SITE*. This array of programmable blocks is surrounded by a programmable routing fabric (interconnect) facilitating connections between the blocks through horizontal and vertical channels. Surrounding the array are input/output (IO) blocks that link the chip to external interfaces. Beneath the array lies a configuration memory (SRAM). When loaded with configuration bits, the SRAM instructs the blocks and interconnects to operate in specific ways. To implement a user's design on the FPGA, AMD's dedicated tool flow, Vivado, translates the design into a set of configuration bits, referred to as the *bitstream*. This bitstream, once loaded onto the SRAM, programs the FPGA to emulate the intended design. The central component for implementing designs is the CLB. Each CLB comprises of logic elements that are grouped together as a *SLICE*. These logic elements can be either lookup tables (LUTs) or sequential elements (FFs). Each CLB contains one SLICE. Each SLICE provides sixteen LUTs and sixteen flip-flops. The SLICEs and their CLBs are arranged in columns throughout the device. There are, however, certain rules regarding the usage of the LUTs and FFs within each SLICE [35]. The target FPGA for the contest is a 16nm single-die Ultrascale+ xcvu3p device. Details about the architecture of this device can be obtained from [41].

III. Problem Overview

The topic for the 2023 MLCAD contest was non-timing driven FPGA macro placement. The input is a netlist comprised of macros and standard cell (LUTs and FFs) instances along with region constraints. The objective is to find legal macro block SITE locations such that the region constraints are honored and routing congestion is minimized. While there is an extensive corpus of existing ASIC macro placers, as reviewed above, these are not directly applicable to FPGAs due to the unique challenges presented by the latter. Some of these challenges are as follows.

- Discrete and SITE-typed columnated nature of the FPGA device layout.
- Cascaded macros (i.e., macros formed by stacking BRAMs, DSPs or URAMs together) or carry chains (macro shapes composed of a large number of LUTs and FFs). These can significantly impact routability and timing closure because they inherently have large numbers of pins and connected nets.
- Presence of resource-limited, multiple clock domains (typically 1 – 100). The FPGA device layout is partitioned into clock regions, with a limited number of clock resources. This clock resource limitation constrains the placement of logic driven by different clocks in these regions. This can be challenging as the number of clock domains in the design grows.
- Higher number of macros (typically 100s-1000s) compared to ASIC (typically 10s-100s) counterparts.
- Fragmentation of the placement across multiple dies.

The above challenges render state-of-the-art ASIC macro placement approaches inadequate for FPGA designs. Our aim is to spur academic research that addresses these unique challenges and pushes the boundaries of current FPGA macro placement methodologies. To simplify the problem, in this year's contest, we chose a single-die device, removed timing closure as an objective, and focused solely on runtime and netlist routability metrics. We plan to introduce a more complete version of the FPGA macro placement problem in future competitions.

IV. Benchmarks

The contest benchmark suite consists of netlists of various sizes and complexity that reflect modern high-end FPGA designs. These designs were generated using an AMD internal netlist generation tool with varying degrees of complexity such as the following.

- **Design utilization.** We created netlists that utilize 70% to 84% of LUT capacity, 38% to 47% of FF capacity, and 80% to 90% of BRAM and DSP capacities. Each of the 140 netlists in the public benchmark suite contains between 561,632 and 720,227 instances, and between 3,721,746 and 4,730,679 nets.
- **Cascaded macro blocks.** We created netlists that contain large macro blocks composed of stacked DSPs, BRAMs or URAMs.
- **Rent's exponent.** Interconnection complexity, as measured by Rent exponent, was varied by creating netlists with different Rent exponents ranging from 0.65 to 0.72. This is important for testing routability of the placement solutions.
- **Number of clocks.** To vary design complexity, we generated netlists with different number of clocks (1, 16, 24, 30, 34 and 38 clocks). The macro placement complexity grows with the number of clocks since there are limited clock sources available in any given region of the device. The clock diversity also reflects the complexity of realistic modern day designs.

Detailed specifications of the public benchmark suite are available in the GitHub repo [37], and the testcases are available at the Kaggle site [38]. Specifications and testcases of the hidden benchmark suite will also be uploaded soon.

Design	LUT%	FF%	BRAM%	DSP%	Rent	#Clks	Runtime(s)
Design_409	70	38	80	80	0.67	1	2034
Design_412	70	38	80	80	0.72	1	3269
Design_424	76	42	84	84	0.72	1	3398
Design_433	82	45	88	88	0.67	1	3541
Design_442	84	47	90	90	0.72	1	2105
Design_373	70	38	80	80	0.67	38	4681
Design_376	70	38	80	80	0.72	38	7635
Design_385	76	42	84	84	0.72	38	5126
Design_397	82	45	88	88	0.67	38	3496
Design_406	84	47	90	90	0.72	38	4941

TABLE I: Sample statistics for some of the benchmark designs.

Benchmark statistics. The public benchmark suite consists of 140 publicly released designs [38], and the hidden benchmark suite consists of 198 designs to be released after the conclusion of the MLCAD workshop. Sample benchmark statistics for the public designs can be found in Table I, where the "Runtime" column denotes the Vivado P&R flow runtime for each design. Figure 3 presents more detailed runtime statistics for sample designs with 38 clocks, taken from the public benchmark suite. Note that some designs require ∼5 hours to complete P&R, reflecting their inherent complexity.

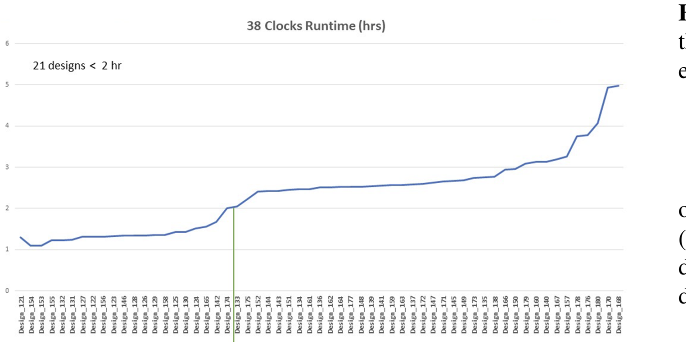

Fig. 3: Sample Vivado P&R flow runtimes on some of the public 2023 MLCAD FPGA macro placement contest benchmarks with 38 clocks.

Benchmark format. All contest benchmark designs were provided to the contestants in an *extended Bookshelf format*, i.e., an extension of the standard Bookshelf format [4] as shown in Table II. The *design.nodes* and *design.nets* adhere to the standard Bookshelf format. The formats of *design.lib* and *design.scl* files are similar to the ACM ISPD 2016 placement contest [35]. Notably, the *design.scl* file comprises two sections: (i) the SITE definition, detailing available resources (LUT/FF/RAMB/DSP) for a SITE, and (ii) the SITE map, outlining a two-dimensional SITE array for the entire device. The *design.pl* file contains the locations of the placeable macro instances. The location of an instance has three fields: (i) column number, (ii) SITE number and (iii) *BEL* number. The BEL number is the index within the SITE (the BEL number for macro instances is always 0). For *design.regions*, all region constraints are mutually non-overlapping boxes specified by $x_{low} \leq x < x_{high}$ and $y_{low} \leq y < y_{high}$, where (x_{low}, y_{low}) are the lower-left coordinates and (x_{high}, y_{high}) are the upper-right coordinates of a given region constraint box.

V. Evaluation Methodology

The macro placement solutions produced by participating placers were evaluated using the Vivado physical design compiler. Contestant teams were provided with a Vivado license and a P&R flow that read an input macro placement in the extended Bookshelf format, checked macro placement legality, and performed standard cell placement and routing. The P&R flow was chosen to be non-timing driven for this contest, with focus on routability and runtime. Accordingly, the solutions were evaluated based on the following factors:

File	Description
design.nodes	Placeable instances in Bookshelf format.
design.nets	Nets in Bookshelf format.
design.lib	Cell library for placeable object types.
design.pl	SITE locations of fixed IO blocks.
design.scl	Target device layout + permissible SITE locations for placeable object types.
design.cascade_shape	Types of placeable cascaded macro shapes.
design.cascade_shape_instances	Cascaded macro shape instances.
design.regions	Box region constraints for placeable objects
design.dcp	Synthesized Vivado checkpoint database.

TABLE II: Extended Bookshelf format used for the 2023 MLCAD FPGA macro placement contest. See [37] for further details.

- Column SITE legality of the macro placement solution.
- Total routing congestion.
- Macro placement runtime.
- Total Vivado place-and-route flow runtime (5 hours limit).

Evaluation metrics. For each $design_j$, $j = 1, ..., \#designs$, in the benchmark suite, the evaluation metric is a scoring function encompassing the above factors, as shown in Equation 1.

$$score_j = (time_{mpl_j} + time_{P\&R_j}) * routability_j \quad (1)$$

The **first** score factor in Equation 1 is the runtime score comprising of the macro placement runtime ($time_{mpl_j}$) and the P&R runtime ($time_{P\&R_j}$) for a design. The specific calculation for $time_{mpl_j}$ is detailed in Equation 2, where $time_{mpl_j}$ is measured in minutes and denotes the macro placement runtime for $design_j$.

$$time_{mpl_j} = 1 + max(0, time_{mpl_j} - 10) \quad (2)$$

Note that no penalty is incurred if the macro placement runtime is under 10 minutes. The $time_{P\&R_j}$ is expressed as the overall P&R runtime in hours. The **second** score factor in Equation 1 is $routability_j$, which consists of two parts: (i) initial routing score and (ii) final routing score as shown in Equation 3.

$$routability_j = routing\_score_{initial_j} + routing\_score_{final_j} \quad (3)$$

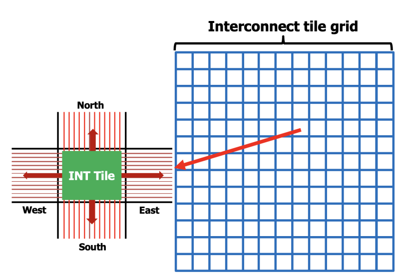

Fig. 4: An example of an AMD FPGA interconnect tile grid.

The $routing\_score_{initial_j}$ is derived from the routing congestion level determined by the initial router. The Vivado router reports routing congestion metrics from which $routing\_score_{initial_j}$ is calculated. In AMD FPGAs, routing congestion is assessed on the interconnect tile grid (Figure 4), which defines the device's programmable routing fabric.[1] Routing congestion arises when there is an overutilization of these routing resources. Routing congestion is further categorized based on interconnect wire length (short or long segments) and direction (North, South, East, or West).

- **Short congestion:** Resulting from closely grouped cell placements which can induce potential routability challenges.
- **Long congestion:** Resulting from dispersed (e.g., poor wirelength) cell placements which can induce both timing and routability challenges.
- **Global congestion:** Representing a combined measure of both short and long congestion types.

Short, long, and global congestion levels vary from 1 to 8 and are reported by Vivado at the end of the initial routing stage. For example, congestion level 5 for short (resp. long) wires means that the design contains at least one contiguous 5×5 short (resp. long) interconnect tile region that is overutilized. Other congestion levels are defined analogously. We enumerate the North/South/East/West directions as $i = 1$ to 4 respectively, and define $routing\_score_{initial_j}$ as in Equation 4. Note that only congestion levels 4 and above are penalized.

$$routing\_score_{initial_j} = 1 + \sum_{i=1}^{4} [max(0, short\_level_i - 3)^2 + max(0, global\_level_i - 3)^2] \quad (4)$$

We define $routing\_score_{final_j}$ as the number of outer iterations of the detailed router as reported in Vivado at the end of final routing. If a solution fails to place-and-route then we assign a penalty of 500 to $score_j$. Using the aforementioned scoring metrics, we evaluated all team solutions on the 140 designs in the public benchmark suite, and a subset of 38 out of 198 designs in the the hidden benchmark suite. The final team score was then computed as a weighted geometric mean of the design scores, ensuring equal contributions from the public and hidden design scores to the overall score.[2]

VI. Results

As mentioned in Section I and detailed in Table III, the 2023 MLCAD contest saw participation from 19 teams. 8 teams progressed to submit final solutions, with 6 solutions standing out as particularly competitive. To evaluate the submitted macro placement solution executables, we followed the evaluation methodology in Section V: (i) we used all 140 designs in the public benchmark suite; and (ii) from the hidden benchmark suite of 198 designs, we selectively evaluated 38 representative designs to reduce the overall evaluation runtime. The majority of teams submitted their solution executables using Docker containers. Our evaluation platform consisted of standalone 3.885GHz, 512GB RAM, 64-processor AMD EPYC 7F52 servers. Each team was evaluated according to the following three steps.

- We built a Docker container when provided with the submission.
- We executed the submitted macro placer on each design using 16 cores, recording the runtime.
- We ran the contest-specified non-timing-driven P&R flow [40] after pre-placing macros based on the macro placement solution.

[1]The interconnect tile grid is where the device's interconnect resources (programmable routing fabric) are pre-allocated.

[2] The final team scores are computed as follows:

$$\text{final team score} = \frac{\sum_{j=1}^{\#designs} w_j \cdot score_j^2}{\sum_{j=1}^{\#designs} w_j}$$

where $w_j = 1$ for public designs and $w_j = 140/38$ for hidden designs.

Team	University	Region
TAMU	University of Texas A&M	USA
MPKU	Peking University	China
CUMPLE	Chinese University of Hong Kong	Hong Kong
Pomelo	Nanjing University of Posts/Telecommunications	China
DAG-MP	Shanghai Jiao Tong University	China
MacroM	Nirma University	India
MacroD	Pohang University	Korea
IMR	Not provided	Not provided
SEU	Southeast University	China
GoFish	Georgia Institute of Technology	USA
EFM	University of Calgary	Canada
BO	University of Illinois-Urbana Champaign	USA
Duke	Duke University	USA
UBCP	University of British Columbia	Canada
GD	Dalian University of Technology	China
MacroW	Simon Fraser University	Canada
CUMP	Chinese University of Hong Kong	Hong Kong
UTDA	University of Texas at Austin	USA
NTHU	National Tsing Hua University	Taiwan

TABLE III: List of participating teams in the 2023 MLCAD FPGA macro placement contest.

	Runtime			Score		
Team	**Avg.**	**Geo. Mean**	**StdDev**	**Avg.**	**Geo. Mean**	**StdDev**
MPKU	0.55	**0.53**	0.16	5.72	3.02	9.18
SEU	0.55	0.54	0.13	3.7	2.23	6.50
UTDA	0.58	0.56	0.25	4.2	**2.12**	16.24
TAMU	0.62	0.56	0.39	10.78	3.59	26.86
CUMPLE	0.59	0.58	0.16	6.19	3.03	10.65
CUMP	0.76	0.68	0.46	30.60	6.65	63.61

TABLE IV: Runtime and score comparisons on the public benchmark suite.

Then, we harvested relevant metrics to compute the initial/final routing congestion and overall P&R flow runtime.

Team	Avg.	Geo. Mean	StdDev
MPKU	5.86	**2.57**	11.03
SEU	11.10	2.85	40.17
UTDA	10.93	2.93	38.69
TAMU	34.45	5.37	82.06
CUMPLE	15.38	3.59	35.96
CUMP	71.53	10.63	128.08

TABLE V: Score comparisons on the hidden benchmark suite.

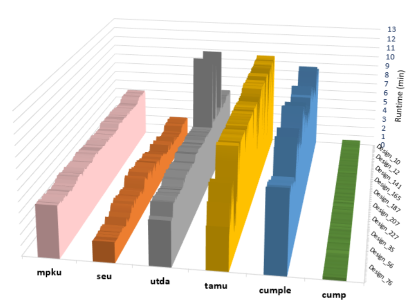

Fig. 5: Macro placement runtime comparison on the public benchmark suite.

A high-level summary of submitted solution methods can be extracted from the video snapshots presented by each team. Most teams used either quadratic or electrostatic formulations for their mixed-sized global placers and min-cost flow or bipartite graph matching for macro placement legalization. Teams UTDA, MPKU and SEU used implementation variants of the electrostatic global placement formulation ([24], [29]). By contrast, teams CUMPLE, TAMU and CUMP used SimPL-based [19] quadratic placement formulations. Team UBCPlacer was unique in adopting an RL-parameterized simulated annealing macro placer along with netlist hypergraph clustering. We do not report team UBCPlacer's results since their late-entry executable ran into errors on all the benchmark designs.

Figure 5 compares the macro placement runtimes for the public benchmark suite. Runtime penalties were mostly avoided as only 4 macro placement runs exceeded the specified 10-minute threshold. The majority of teams leveraged multi-threading across 16 cores to speed up their solutions. Team CUMP's macro placer outperformed others, running 5-20X faster.

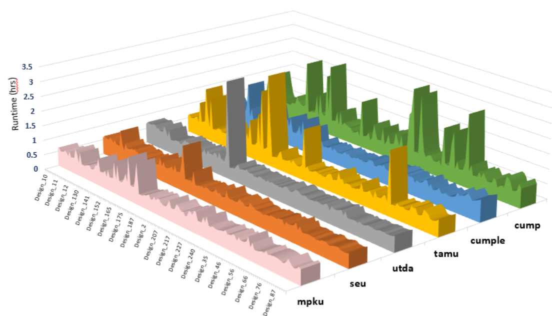

Fig. 6: P&R runtime comparison on the public benchmark suite.

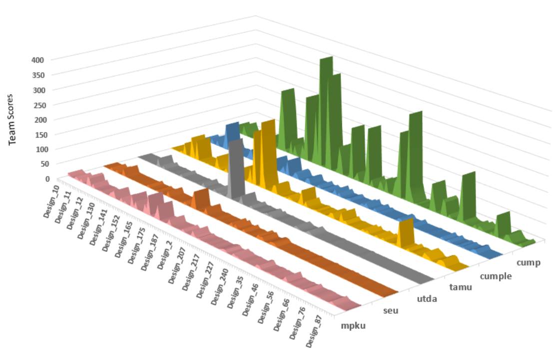

Fig. 7: Team scores comparison on the public benchmark suite.

Figures 6 and 7 show the P&R flow runtimes and scores for the public benchmark suite. Team MPKU achieved the best runtime score with a geometric mean score of 0.53, closely followed by teams SEU and UTDA scoring 0.54 and 0.56 respectively as detailed in Table IV. The team scores on the public and hidden benchmark suites are listed in Tables IV and V, respectively.[3] On the hidden benchmark suite, Team MPKU achieved the best scores (Table V), followed closely by team SEU and UTDA. Table VI lists consolidated scores, corresponding to the weighted geometric mean of scores on hidden and public benchmarks. Teams UTDA and SEU were judged to tie for first place, with team MPKU in second place, team CUMPLE in third place, and teams TAMU and CUMP in fourth and fifth places, respectively. Team MPKU's updated executable achieved a 2.216 weighted geometric mean score[3], notably better than the top two contest scores of 2.513 and 2.516 (listed in Table VI).

[3]Team MPKU submitted an updated executable for their macro placer after the contest deadline, after being requested by contest organizers to incorporate a fix for benchmark file parsing errors. While the updated executable was not factored into the official contest rankings, it yielded superior scores across both the public and hidden benchmark suites, surpassing all competing macro placers. For transparency and to acknowledge their work, we have included comparisons in Figures 8 and 9 using team MPKU's post-contest deadline executable. We believe it is important to share these results with all participants and the broader MLCAD community to accurately establish a baseline of best-in-class results.

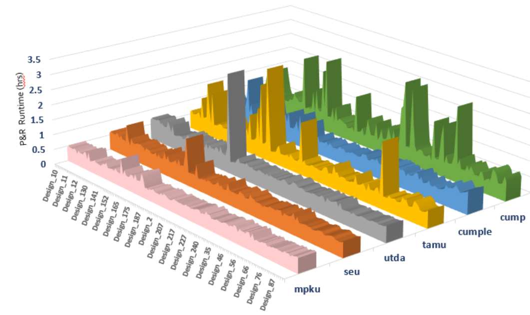

Fig. 8: P&R runtime comparison on the public benchmark suite with team MPKU's updated executable.[3]

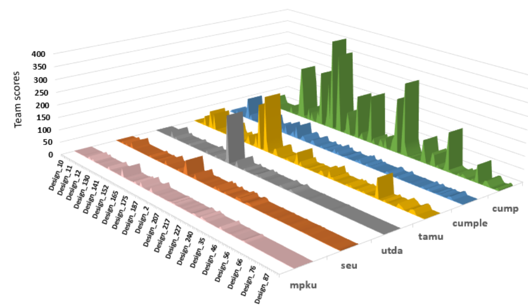

Fig. 9: Team scores comparison on the public benchmark suite with team MPKU's updated executable.[3]

Team	Weighted Geo. Mean	Rank
UTDA	2.513	1*
SEU	2.516	1*
MPKU	2.751	2
CUMPLE	3.605	3
TAMU	4.399	4
CUMP	8.433	5

TABLE VI: Final team scores for the 2023 MLCAD FPGA macro placement contest.

VII. Conclusion

The published industrial benchmarks augmented with the hidden benchmarks will remain available to academic researchers. We encourage participants to continually refine their solutions, anticipating further innovations in next year's *timing-driven* follow-on macro placement contest. As noted in section VI, all but one of the submitted solutions for this year's contest were based on classical optimization approaches. This could be attributable to the contest timeline and the lack of scalable ML-based macro placement approaches. There are many promising nascent RL and decision transformer-based works

[15] [25] [26] [20]. However, these approaches entail solving the macro placement problem sequentially in a "coordinate descent" fashion. That is, the objective function is minimized by updating one variable at a time. In this context, the RL agent places one macro, while querying a standard-cell placement oracle (e.g. force-directed or electrostatic-based methods [25], [29], [24]) to compute the objective function value for updating the reward and state. However, such an approach can be challenging to scale for inference if it is not *zero-shot* or *few-shot*, especially if the state update involves querying a timing-driven and congestion-aware placement. The scalability issue can be further exacerbated for the FPGA macro placement problem because of the larger number of macros and the much tighter runtime budget for P&R than in the ASIC context. Other potential approaches to mitigate the scalability issue include the use of ML for parameter optimization of classical optimization approaches [2] and, more generally, emerging learning-to-optimize paradigms [21]. We remain optimistic that scalable ML-boosted solutions can be found for this challenging problem.

VIII. Acknowledgments

We thank The OpenROAD Project and AMD Inc. for sponsoring the contest, and the TILOS AI Institute for hosting the contest GitHub repository. We also thank Zhiang Wang, Yuji Kukimoto, Sreevidya Maguluri, Ravishankar Menon, Nima Karimpour-Darav, Mehrdad Eslami, Chaithanya Dudha, Lin Chai, Kai Zhu, Kristin Perry, Cathal McCabe, Mark O Brien, and Vishal Suthar for their assistance.

References

[1] S. N. Adya and I. Markov, "Combinatorial techniques for mixed-size placement", *ACM TODAES* 10(1) (2005), pp. 58-90.

[2] A. Agnesina, P. Rajvanshi, T. Yang, G. Pradipta, A. Jiao, B. Keller, B. Khailany and H. Ren, "AutoDMP: automated DREAMPlace-based macro placement", *Proc. ISPD*, 2023, pp. 149–157.

[3] R. Bruck, K.-H. Temme and H. Wronn, "FLAIR-a knowledge-based approach to integrated circuit floorplanning", *Proc. Intl. Workshop on Artificial Intelligence for Industrial Applications*, 1988, pp. 194-199.

[4] A.E. Caldwell, A. B. Kahng and I. L. Markov, "Toward CAD-IP reuse: the MARCO GSRC bookshelf of fundamental CAD algorithms", *IEEE Design and Test* 19(3) (2002), pp. 70-79. http://vlsicad.eecs.umich.edu/BK/Slots/

[5] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze and M. Xie, "mPL6: enhanced multilevel mixed-size placement", *Proc. ISPD*, 2006, pp. 212-214.

[6] Y.-C. Chang, Y.-W. Chang, G.-M. Wu and S.-W. Wu, "B*-trees: a new representation for non-slicing floorplans", *Proc. DAC*, 2000, pp. 458-463.

[7] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on B∗-tree and fast simulated annealing", *IEEE Trans. CAD* 25(4) (2006), pp. 637-650.

[8] H.-C Chen, Y.-L. Chuang and Y.-C. Chang, "Constraint graph-based macro placement for modern mixed-size circuit designs", *Proc. ICCAD*, 2008, pp. 218-223.

[9] Y. Chen, C. Huang, C. Chiou, Y. Chang and C. Wang, "Routability driven blockage-aware macro placement", *Proc. DAC*, 2014, pp. 1-6.

[10] 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 Trans. CAD*, 27(7) (2008), pp. 1228-1240.

[11] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang and D. Liu, "MP-tree: a packing-based macro placement algorithm for mixed-size designs", *Proc. DAC*, 2007, pp. 447-452.

[12] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang and T.-Y. Liu, "MPtrees: a packing-based macro placement algorithm for modern mixed size designs", *IEEE Trans. CAD* 27(9) (2008), pp. 1621-1634.

[13] C.-K. Cheng, A. B. Kahng, I. Kang and L. Wang, "RePlAce: advancing solution quality and routability validation in global placement", *IEEE Trans. CAD* 38(9) (2019), pp. 1717-1730.

[14] J. Cong and M. Xie, "A robust detailed placement for mixed-size IC designs", *Proc. ASP-DAC*, 2006, pp. 188-194.

[15] Z. He, Y. Ma, L. Zhang, P. Liao, N. Wong, B. Yu and M. D.-F. Wong, "Learn to floorplan through acquisition of effective local search heuristics", *Proc. ICCAD*, 2020, pp. 324-331.

[16] A. B. Kahng, R. Varadarajan and Z. Wang, "RTL-MP: toward practical, human-quality chip planning and macro placement", *Proc. ISPD*, 2022, pp. 3-11.

[17] A. B. Kahng, R. Varadarajan and Z. Wang, "Hier-RTLMP: a hierarchical automatic macro placer for large-scale complex IP blocks", *arXiv:2304.11761*, 2023.

[18] A. B. Kahng and Q. Wang, "APlace: a faster implementation of APlace", *Proc. ISPD*, 2006, pp. 218-220.

[19] M-C. Kim, D-J. Lee and I. Markov, "SimPL: an effective placement algorithm," *Proc. ICCAD*, 2010, pp 649–656.

[20] Y. Lai, J. Liu, Z. Tang, B. Wang, J. Hao and P. Luo, "ChiPFormer: transferable chip placement via offline decision transformer", *arXiv:2306.14744*, 2023.

[21] K. Li and J. Malik, "Learn to optimize", *arXiv 1606.01885*, 2016.

[22] 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", *Proc. DAC*, 2019, pp. 1-6.

[23] Y.-C. Liu, T.-C. Chen, Y.-W. Chang and S.-Y. Kuo, "MDP-trees: multi-domain macro placement for ultra large-scale mixed-size designs", *Proc. ASP-DAC*, 2019, pp. 557-562.

[24] 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", *arXiv:2306.16665*, 2023.

[25] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang et al., "Chip placement with deep reinforcement learning", *arXiv 2004.10746*, 2020.

[26] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang et al., "A graph placement methodology for fast chip design", *Nature* 594 (2021), pp. 207-212.

[27] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair", *IEEE Trans. CAD* 15(12) (1996), pp. 1518-1524.

[28] J. K. Ousterhout, "Corner stitching: a data-structuring technique for vlsi layout tools", *IEEE Trans. CAD* 3(1) (1984), pp. 87-100.

[29] 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", *Proc. ASP-DAC*, 2022, pp. 300-306.

[30] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, A. N. Ng, J. F. Lu and I. L. Markov, "Capo: robust and scalable open-source min-cut floorplacer", *Proc. ISPD*, 2005, pp. 224-226.

[31] P. Spindler and F. M. Johannes, "Kraftwerk: a fast and robust quadratic placer using an exact linear net model", *Modern Circuit Placement*, Boston, Springer, 2007.

[32] T. Taghavi, X. Yang, B.-K. Choi, M. Wang and M. Sarrafzadeh, "Dragon2006: blockage-aware congestion-controlling mixed-size placer", *Proc. ISPD*, 2006, pp. 209-211.

[33] K.-H. Temme and R. Bruck, "Chip-architecture planning based on expert knowledge", *Proc. Intl. Workshop on Artificial Intelligence for Industrial Applications*, 1988, pp. 188-193.

[34] N. Viswanathan, M. Pan and C. Chu, "FastPlace 3.0: a fast multilevel quadratic placement algorithm with placement congestion control", *Proc. ASP-DAC*, 2007, pp. 135-140.

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

[36] Y. Zhan, Y. Feng and S. S. Sapatnekar, "A fixed-die floorplanning algorithm using an analytical approach", *Proc. ASP-DAC*, 2006, pp. 771-776.

[37] 2023 MLCAD FPGA macro placement contest repository, https://github.com/TILOS-AI-Institute/MLCAD-2023-FPGA-Macro-Placement-Contest

[38] 2023 MLCAD FPGA macro placement contest benchmark suite, https://www.kaggle.com/datasets/ismailbustany/mlcad2023-fpga-macroplacement-contest/settings?resource=download%2F

[39] 2023 MLCAD workshop, https://mlcad-workshop.org/

[40] Vivado P&R flow script for 2023 MLCAD contest, https://github.com/TILOS-AI-Institute/MLCAD-2023-FPGA-Macro-Placement-Contest/blob/main/Flow/vivado_pnr.tcl

[41] Xilinx, "UltraScale architecture and product data sheet: Overview", https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds890-ultrascale-overview.pdf