IncreMacro: Incremental Macro Placement Refinement

Yuan Pu, Tinghuan Chen, *Member, IEEE*, Zhuolun He, Jiajun Qin, *Student Member, IEEE*, Chen Bai, *Member, IEEE*, Haisheng Zheng, Yibo Lin, *Member, IEEE*, and Bei Yu, *Senior Member, IEEE*

***Abstract*—This article proposes IncreMacro, a novel approach for macro placement refinement in the context of integrated circuit (IC) design. The suggested approach iteratively and incrementally optimizes the placement of macros in order to enhance IC layout routability and timing performance. To achieve this, IncreMacro utilizes several methods, including kd-tree-based macro diagnosis, gradient-based macro shifting, constraint-graph-based LP for macro legalization, and diffusion-based cell migration. By employing these techniques iteratively, IncreMacro meets two critical solution requirements of macro placement: 1) pushing macros toward the chip boundary and 2) preserving the original macro relative positional relationship. The proposed approach has been incorporated into AutoDMP and DREAMPlace 4.0, and is evaluated on seven RISC-V benchmark circuits and four TILOS macro placement circuit designs at the 7-nm technology node. Experimental results show that, compared with the macro placement solution provided by AutoDMP (DREAMPlace 4.0), our approach reduces routed wirelength by 15.1% (14.9%), improves the routed worst negative slack (WNS) and total negative slack (TNS) by 99.9 (82.6%) and 99.9% (81.3%), and reduces the total power consumption by 4.4% (4.3%). Meanwhile, compared with IncreMacro [1], our approach augmented with the cell migration algorithm improves the routed WNS and TNS by 24.7% and 23.1%, and remains the average routed wirelength and total power consumption almost unchanged.**

***Index Terms*—Legalisation, macro placement, physical synthesis, timing optimization, VLSI placement.**

I. Introduction

EFFECTIVE macro placement is a pivotal aspect of the physical design process in very large-scale integration (VLSI) circuits due to the substantial portion of the chip area that predesigned macro blocks, especially embedded memories, occupy. Inadequate placement of these macros can lead to significant blockage at the core center of the chip, detrimentally affecting overall performance by inducing routing congestion, suboptimal wirelength, and timing issues. Moreover, macro placement influences the placement of standard cells, and poor macro placement might make it challenging to place these cells optimally, leading to an unsatisfactory chip performance. As predesigned macro blocks become increasingly prevalent in modern VLSI designs, ensuring effective macro placement is essential for enhancing routability and timing performance.

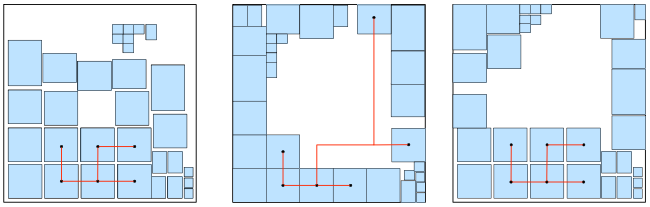


Fig. 1. (a) Macro placement by an analytical placer. (b) Macro placement refined by data structure-centric metaheuristics. (c) Macro placement refined by IncreMacro. The blue block represents macro. The white region represents the remaining space for standard cell placement and routing. A six-pin net is denoted in red color.

Modern analytical placers [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] support mixed-size placement of standard cells and macros with multiobjective optimization. Previous research [13], [14], [15] demonstrates that the major advantage of mixed-size placement is that macros with large amount of standard-cell connections can be placed in the chip center for wirelength optimization. This methodology works fine for designs with low density and small number of macros. However, for modern VLSI designs with high placement density and large amount of various-size macros, mixed-size placement often leads to serious macro blockage in the center of the chip. Such blockages can adversely affect the placement of standard cells and obstruct the subsequent routing stages, overwhelming the wirelength improvement brought by placing macros to chip center, and potentially compromising overall system performance. In Fig. 1(a), we illustrate an instance of mixed-size macro placement performed by an analytical placer. The blue blocks represent macros, while the white area signifies the available space for standard cell placement. In this example, macros are placed within the core center, causing macro blockage. Such macro blockage divides the remaining space for standard cell placement into discontinuous subregions. Consequently, standard cells associated with the same net may be dispersed across separate placement subregions,

Received 17 June 2024; revised 23 September 2024 and 16 December 2024; accepted 14 January 2025. Date of publication 20 January 2025; date of current version 21 July 2025. This work was supported in part by the Research Grants Council of Hong Kong, SAR, under Grant RFS2425-4S02, Grant CUHK14211824, and Grant CUHK14210723; and in part by the MIND Project under Grant MINDXZ202404. This article was recommended by Associate Editor I. H. R. Jiang. *(Corresponding author: Bei Yu.)*

Yuan Pu, Zhuolun He, Chen Bai, and Bei Yu are with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, SAR (e-mail: pyxch11081210@gmail.com).

Tinghuan Chen is with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shenzhen 518172, China.

Jiajun Qin is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310058, China.

Haisheng Zheng is with the Shanghai AI Laboratory, Shanghai 200003, China.

Yibo Lin is with the School of Integrated Circuits, Peking University, Beijing 100871, China.

Digital Object Identifier 10.1109/TCAD.2025.3531776

© 2025 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

necessitating routing detours in the VLSI design process. Routing detours result in longer signal routing paths, reflected in increased routed wirelength, along with elevated resistance and capacitance. These factors significantly degrade timing performance and escalate power consumption. Moreover, a common strategy employed by seasoned engineers in macro placement involves positioning macros toward the peripheral areas of the chip to avoid central blockages. To achieve superior quality of results (QoRs) in the final physical design, meticulous attention to macro placement to eliminate macro blockage is imperative.

Numerous studies have investigated macro placement using reinforcement learning (RL) and metaheuristics. RL approaches [16], [17], [18], [19], [20], [21] treat macro placement as a sequential Markov decision process (MDP), where macro assignments on the chip canvas are actions, followed by standard cell placement via an analytical placer. Unlike methods refining existing placements, RL strategies develop macro placements from scratch. Additionally, advanced data structure-centric metaheuristics [22], [23], [24], [25], [26], [27], [28], [29], [30], [31] for macro legalization have been introduced, utilizing structures like MP-tree and CP-tree to organize layouts and employing the simulated annealing (SA) algorithm to optimize macro displacement, wirelength, and routability. The newly introduced AutoDMP [32] framework uses DREAMPlace and multiobjective Bayesian optimization to effectively balance power, performance, and area (PPA) in macro placement.

Existing academic studies for macro placement have three limitations as follows.

1) The RL-based approach and AutoDMP require significant computational resources, often needing multiple iterations with varied settings to meet specific PPA targets.
2) On refining the macro placement of the existing placement prototype, the data structure-centric metaheuristics may alter the macro relative positional relationship of the placement prototype.
3) The data structure-centric metaheuristics approaches usually pack macros to the chip boundary and eliminate the macro halos (spaces) in placement prototype. For macros with large amount of standard-cell connections, the halo elimination may lead to no enough space for standard cell placement and routing.

The above three limitations of data structure-centric metaheuristics approaches can potentially lead to increased wirelength and subsequent degradation in timing performance. An example of applying the data structure-centric metaheuristic approach to legalize an analytical placement prototype is shown in Fig. 1(b). Although the approach successfully eliminates macro overlaps, it perturbs the original macro relative positional relationship of the placement prototype, resulting in poor wirelength performance for the six-pin net. Also, by packing macros to the chip periphery, no enough continuous space is left for standard cell placement and routing for the boundary macros.

To overcome the above limitations and achieve a better tradeoff between computational time and solution quality for macro placement, there is a need for a refinement algorithm that both eliminates macro blockage and preserves the relative positional relationships established by the placement prototype. Regrettably, according to our knowledge, no existing work fulfills these criteria. We thus propose IncreMacro, a novel incremental macro placement framework that leverages an analytical placer and satisfies two key requirements. Specifically, the refined solution is supposed to: 1) *push macros toward the chip boundary*, and 2) *preserve the macro relative positional relationship established by the placement prototype*. Moreover, for macros with a large number of standard-cell connections in the design, IncreMacro improves their wirelength and timing optimization in two perspectives as follows.

1) For the placement prototype provided by the analytical placer, halos are often appended to the macros. Meanwhile, IncreMacro will maintain a remarkable ratio of boundary macros and regularly placed macros as unmoved during refinement, such that the spaces (halos) among these macros are maintained after IncreMacro refinement for standard cell placement and signal routing, thus improving the wirelength and timing optimization for these macros. Besides, due to the preservation of these spaces, there will be available metals in the M1/M2 layers for standard cells to access the external signals.
2) During the IncreMacro refinement process, the diffusion-based cell migration step iteratively adjusts the standard cells connected to nonboundary macros to reasonable positions, facilitating the wirelength-optimization of poorly placed macros in the following gradient-based macro shift stages.

Fig. 1(c) presents an ideal macro placement solution achieved through the application of IncreMacro to an academic analytical placer. This solution contrasts with Fig. 1(a), by eliminating macro blockage at the core center, thus facilitating continuous open space for subsequent standard cell placement and routing. Compared to Fig. 1(b), this placement maintains the wirelength optimization achieved by the analytical placer's initial placement prototype. This is evidenced by the preserved relative positional relationships of the macros connected in the six-pin net, resulting in a reduction in net wirelength. Meanwhile, the halos of boundary macros and regularly placed macros are maintained by IncreMacro, providing extra space for standard cell placement, and thus improving the wirelength and timing for macros connected to large amount of standard cells.

Our major contributions are summarized as follows.

1) We propose IncreMacro which incrementally enhances macro placement by state-of-the-art analytical placers.
2) We propose a novel algorithmic workflow consisting of KD-tree-based macro diagnosis, gradient-based macro shift, constraint-graph-based linear programming for macro legalization, and diffusion-based cell migration, to eliminate macro blockage in the chip center while preserving the wirelength optimization by the analytical placement prototype.
3) Compared with IncreMacro [1], the incorporation of the diffusion-based cell migration algorithm provides reasonable position prediction for standard cells,

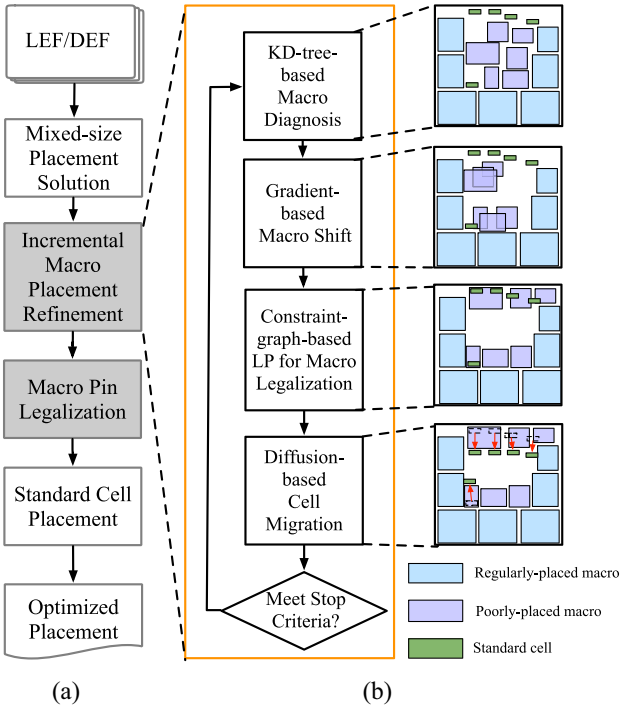


Fig. 2. (a) The mixed-size placement flow and (b) our proposed incremental macro placement refinement tool IncreMacro.

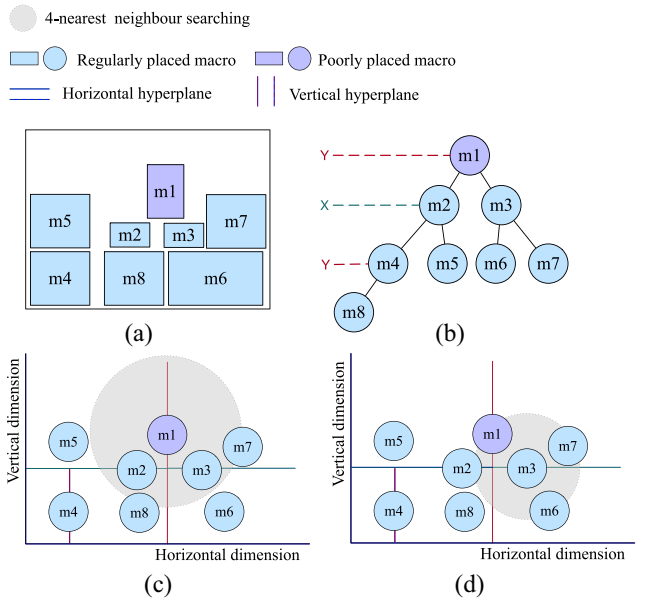


Fig. 3. Customization of KD-tree for macro diagnosis. (a) and (b) are the layout and the corresponding kd-tree of a macro placement layout. (c) and (d) visualises the nearest-neighbour search process of kd-tree.

enabling more accurate wirelength estimation and better wirelength optimization in the subsequent IncreMacro iterations.

4) We design a macro pin legalization algorithm to automatically and efficiently align the macros to manufacturing grids, guaranteeing legalized macro placement results for the following VLSI flows.
5) We conduct experiments on seven RISC-V circuits and four TILOS macro placement benchmarks at 7-nm technology node. Experimental results show that incorporating IncreMacro with cell migration into AutoDMP (DREAMPlace 4.0) improves the post-route worst negative slack (WNS) and total negative slack (TNS) by 99.9% (82.6%) and 99.9% (81.3%), and reduces the routed wirelength and total power consumption by 15.1% (14.9%) and 4.4% (4.3%). Besides, compared with IncreMacro [1], IncreMacro augmented with the diffusion-based cell migration algorithm improves the routed WNS and total TNS by 24.7% and 23.1%, and remains the routed wirelength and total power consumption almost unchanged.

II. Problem Formulation

In this section, we first introduce the mixed-size analytical placement flow into which IncreMacro is incorporated. We then give the definitions of macro regularity (including the definition of *regularly placed macro*, *poorly placed macro*, and *macro diagnosis*). Finally, the definition of the macro placement refinement problem is given.

Fig. 2(a) illustrates the integration of IncreMacro into a mixed-size placement flow. The process begins with inputting LEF and DEF files into an analytical placer, which creates a mixed-size placement prototype with legalized macros. IncreMacro is then applied iteratively to refine the macro placement. Once refinement is complete, macro locations are finalized using our macro pin legalization algorithm to correct any misalignments with the manufacturing grid. Following this, an analytical placer performs the standard cell placement, culminating in an optimized placement solution.

Before defining macro regularity, it is important to understand its impact on VLSI processes like standard cell placement and signal/clock routing. As outlined in Section I, macros positioned at the core center can block and split the placement area, scattering standard cells connected to the same net across disjoint areas. This increases wirelength and can lead to routing issues that degrade timing performance. Macros causing such blockages are termed "poorly placed macros." On the other hand, "regularly placed macros" are either placed adjacent to the chip boundary (e.g., m_4–m_8 in Fig. 3) or well-packed and aligned with other macros (e.g., m_2 and m_3 in Fig. 3). With this understanding, we proceed to formally define macro regularity.

Definition 1 (Regularly Placed Macro): Given a macro m_i and its four Euclidean-distance-nearest neighbor macros $N_i = \{m_1^i, m_2^i, m_3^i, m_4^i\}$ in a placement prototype, if N_i can surround m_i, that is, if macros in N_i are spatially and, respectively, to the north, south, west, and east direction of m_i, or if m_i clings to the chip boundary (the distance between the center of m_i and its nearest chip boundary is less than a threshold α), then m_i is a regularly placed macro.

Definition 2 (Poorly Placed Macro): If a macro m_i is not regularly placed, it is poorly placed.

Definition 3 (Macro Diagnosis): Given a macro placement solution consists of macros $M = \{m_1, m_2, \ldots, m_n\}$, divide M into two sets M_{regular} and M_{poor} such that M_{regular} contains only regularly placed macros, M_{poor} contains only poorly placed macros, and $|M| = |M_{\text{regular}}| + |M_{\text{poor}}|$.

The macro placement refinement problem is as follows.

Problem 1 (Macro Placement Refinement): A mixed-size placement prototype by an analytical placer contains 1) a set of macros $\mathcal{M} = \{m_1, m_2, \ldots, m_m\}$; 2) a set of standard cells $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$ which are in the same nets with macros in $\mathcal{M}$; and 3) their original positions $\mathcal{CO}_{\mathcal{M}} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ and $\mathcal{CO}_{\mathcal{C}} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. Our target is to find the legalized positions of macros in $\mathcal{M}$ to optimize the objective of wirelength.

III. Algorithm

Fig. 2(b) illustrates the flow of IncreMacro. Given the mixed-size placement prototype by the analytical placer, a KD-tree-based macro diagnosis method is applied to differentiate poorly placed macros (those causing blockage) from regularly placed ones Then, by making an analogy between macro shift and the neural network training process of deep learning, a deep learning toolkit is then employed to iteratively adjust the positions of poorly placed macros via a gradient-based method, aiming to minimize wirelength and periphery costs. Subsequently, based on the macro placement from the mixed-size placement prototype, two constraint graphs (CGs) are constructed to maintain the relative positional relationships of the macros both horizontally and vertically. A constraint-graph-based linear programming approach is designed to resolve overlaps among macros, minimize macro displacement, and reposition poorly placed macros toward the chip periphery while preserving the macro relative positional relationship of the initial placement prototype. Finally, after the macro placement is legalized, a diffusion-based cell migration algorithm is applied on standard cells in $\mathcal{C}$ (standard cells in the same nets of macros in $\mathcal{M}$) to rearrange their positions, thus providing a more accurate wirelength estimation for the gradient-based method in the next iteration. The KD-tree-based macro diagnosis method, the gradient-based method, constraint-graph-based linear programming, and the diffusion-based cell migration algorithm are sequentially and iteratively performed until the macro blockage is eliminated. For the setting of the stop criteria, IncreMacro refinement process stops when one of the following condition is satisfied: 1) the rate of change of the average macro displacement is less than a threshold (10% in our experiment) or 2) the total number of IncreMacro iterations is larger than a predefined threshold (10 in our experiment). After IncreMacro, we apply the macro pin legalization algorithm to align the pins of the refined macros to the manufacturing grids. In the following sections, we detail the techniques used in each stage of IncreMacro and the macro pin legalization algorithm.

A. KD-Tree-Based Macro Diagnosis

In the domain of VLSI circuits, analytical placers proficiently position macros of uniform sizes in a structured manner. However, the escalating complexity of VLSI designs often necessitates functional modules with varying memory sizes, leading to a diversity in SRAM (macro) sizes. This variation can challenge analytical placers, potentially resulting in poorly placed macros and macro blockages at the chip's center, which obstructs subsequent VLSI processes like routing. An effective and efficient methodology for *macro diagnosis* is required to help refine the placement of *poorly placed* macros and eliminate macro blockage in the analytical placement prototype.

Algorithm 1 Determine Spatial Relationship of Two Macros

Input: Two macros m_i and m_j, and their center positions (x_i, y_i) and (x_j, y_j).

```
1:  Initialize left_cnt, right_cnt, top_cnt, bottom_cnt = 0;
2:  Δx = |x_i − x_j|, Δy = |y_i − y_j|;
3:  if Δx > Δy then
4:      if x_j < x_i then
5:          left_cnt+ = 1;
6:      else
7:          right_cnt+ = 1;
8:  else
9:      if y_j < y_i then
10:         bottom_cnt+ = 1;
11:     else
12:         top_cnt+ = 1;
```

The KD-tree, an advanced form of the binary search tree, is adept at managing tasks, such as sorting, range searching, and neighbor searching within k-dimensional spaces. Each node in a KD-tree represents a point in k dimensions, and the nonleaf nodes partition the space using split hyperplanes. The worst-case time complexity of building a KD-tree with N nodes is $O(N \log^2 N)$. The derivation of this time complexity is as follows: a KD-tree can be regarded as a balanced binary tree, whose depth is $O(\log N)$. During constructing a KD-tree, we have to search for the median(s) at each layer of the tree, and the time complexity of the classical median-search algorithm, such as Heapsort or Mergesort is $O(N \log N)$. Therefore, the time complexity of building KD-tree is $O(\log N) \cdot O(N \log N) = O(N \log^2 N)$. Besides, KD-tree also supports nearest neighbor searching (NNS) with an expected time complexity of $O(\log N)$, where N is the number of nodes [33]. Given its capabilities, the KD-tree serves as an effective data structure for modeling the relative positional relationships among macros in a placement prototype, making it highly suitable for applications, such as macro diagnosis through NNS.

During macro diagnosis, one KD-tree is constructed on the entire placement region to capture the spatial relationship among all macros, and the NNS function of KD-tree is utilized to detect the poorly placed macros. Algorithm 1 shows the procedure of determining the spatial relationship between macro m_i and m_j. Let *left_cnt*/*right_cnt*/*top_cnt*/*bottom_cnt* denote the number of left/right/top/bottom neighbors of m_i, and they are initialized to 0. Then, we calculate the horizontal and vertical offsets between m_i and m_j (denoted as Δx and Δy). If Δx is larger than Δy, the spatial relationship between these two macros should be horizontal, and lines 4–8 determines whether m_i is to the left or right of m_i. Similarly, the procedure of determining vertical relationship is illustrated in lines 10–14. During macro diagnosis, for each nonboundary macro m_i, we check its spatial relationship with its four nearest neighbor

macros, and if the *left_cnt*, *right_cnt*, *top_cnt*, *bottom_cnt* are all 1, we get the conclusion that m_i is surrounded by its four nearest neighbors and is thus regularly placed. Fig. 3 depicts the process of customizing a KD-tree for macro diagnosis. In Fig. 3(a), a macro placement solution with eight macros is presented, based on which a KD-tree is constructed in Fig. 3(b). The space is partitioned by generating horizontal or vertical hyperplanes for each nonleaf node ($\{m_1, m_2, m_3, m_4\}$), as shown in Fig. 3(c) and (d). For macro diagnosis, the KD-tree is utilized to search for the four nearest neighbors of each macro, and the macro regularity is determined by referring to Definitions 1 and 2. The process of macro diagnosis for macro m_1 and m_3 is illustrated in Fig. 3(c) and (d). Specifically, as shown in Fig. 3(c), the four nearest neighbor macros of m_1, denoted as $N_1 = \{m_2, m_3, m_7, m_8\}$, do not surround m_1 in the north direction, and m_1 does not cling to the chip boundary, indicating that m_1 is categorized as poorly placed. Conversely, as demonstrated in Fig. 3(d), the four nearest neighbor macros of m_3, denoted as $N_3 = \{m_1, m_2, m_6, m_7\}$, can surround m_3, suggesting that m_3 is diagnosed as regularly placed. With KD-tree utilized, the time complexity of macro diagnosis is $O(N \log N)$ (since each macro is required to be diagnosed for once), where N is the total number of macros in a macro placement solution.

B. Gradient-Based Macro Shift

Following macro diagnosis, the identification of poorly placed macros necessitates their relocation to the chip's periphery, while maintaining the wirelength optimization achieved by the analytical placer. Drawing inspiration from DREAMPlace [10], we treat macro placement as an analytical problem akin to deep neural network training, introducing a gradient-based macro shift algorithm. In neural network training, the model $M(w)$, where w are the weights, processes input x_i to predict $O(x_i, w)$, with a loss function $\mathcal{L}$ assessing the prediction accuracy. The goal is to adjust w to minimize $\mathcal{L}$. In macro placement, the shift offsets $(\Delta x_i, \Delta y_i)$ for each macro m_i serve a similar role to w, indicating the necessary adjustments to optimize macro positioning by minimizing placement loss. One thing deserves attention is that besides using $(\Delta x_i, \Delta y_i)$, we can also directly using the macro position (x_i, y_i) as variables for wirelength cost calculation, and these two approaches are mathematically equivalent. Each data instance (x_i, y_i) is replaced by a macro-involved net instance with its feature vector and a label of zero. The adapted loss function $\mathcal{L}$ includes convex wirelength and periphery costs, aimed at optimizing half-perimeter wirelength (HPWL) and pushing macros to the chip boundary, leaving more continuous space for routing and implicitly optimizing routability and timing performance. Using neural network training techniques, we apply backward propagation to calculate the gradient for each poorly placed macro. To update the positions, we add these gradients $(\partial\mathcal{L}/\partial\Delta x_i, \partial\mathcal{L}/\partial\Delta y_i)$ to the original positions (x_i, y_i), determining the new positions for the macros. The flow is shown in Fig. 4.

We denote any net in the netlist as e, containing instances like macros and standard cells $\{v_1, v_2, \ldots, v_m\}$. The traditional

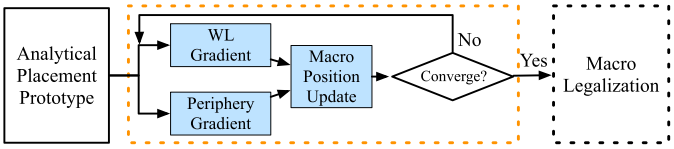


Fig. 4. Macro shifting gradient descent process.

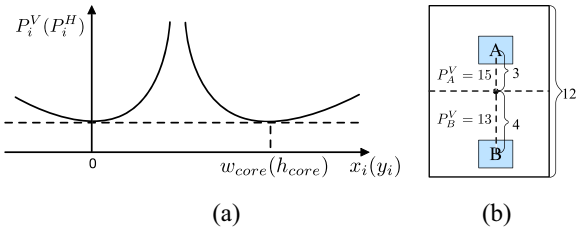


Fig. 5. Peripheral cost illustration. (a) Visualizes the mathematical expression of the peripheral cost, while (b) instantiates the physical meaning of the vertical peripheral cost: denote the vertical position of macro m_A (m_B) by y_A (y_B), and assume $h_{core} = 12$, $|h_{core}/2 - y_A| = 3$, and $|h_{core}/2 - y_B| = 4$. Since macro m_B is closer to the chip periphery compared with macro m_A, the vertical peripheral cost of macro m_B ($P_B^V = 13$) is smaller than that of macro m_A ($P_A^V = 15$).

computation of HPWL for e described in (2), is neither smooth nor convex. To improve this, we use the weighted-average wirelength (WA) method, in (2) and proposed by [34], where the parameter γ balances the accuracy and smoothness of the approximation to HPWL

$$\text{HPWL}_e = \max_{v_i \in e}\{x_i\} - \min_{v_i \in e}\{x_i\} + \max_{v_i \in e}\{y_i\} - \min_{v_i \in e}\{y_i\}. \tag{1}$$

$$\text{WA}_e = \frac{\sum_{v_i \in e} x_i e^{\frac{x_i}{\gamma}}}{\sum_{v_i \in e} e^{\frac{x_i}{\gamma}}} - \frac{\sum_{v_i \in e} x_i e^{-\frac{x_i}{\gamma}}}{\sum_{v_i \in e} e^{-\frac{x_i}{\gamma}}} + \frac{\sum_{v_i \in e} y_i e^{\frac{y_i}{\gamma}}}{\sum_{v_i \in e} e^{\frac{y_i}{\gamma}}} - \frac{\sum_{v_i \in e} y_i e^{-\frac{y_i}{\gamma}}}{\sum_{v_i \in e} e^{-\frac{y_i}{\gamma}}}. \tag{2}$$

We replace the position (x_i, y_i) of each poorly placed macro m_i by $(x_i + \Delta x_i, y_i + \Delta y_i)$ in (2), while maintaining the positions of standard cells in $\mathcal{C}$ and regularly placed macros as constants. By doing so, we obtain the wirelength cost item of each net e, denoted by WL_e.

The peripheral cost P_i for a macro m_i is also defined in (4), consisting of the horizontal cost P_i^H and vertical cost P_i^V. w_{core} (h_{core}) denotes the width (height) of the chip core. As illustrated by Fig. 5, the peripheral cost is convex, and the closer m_i is to the chip periphery, the smaller the cost is. For a macro m_i, if x_i (y_i) equals 0 or w_{core} (h_{core}), m_i is at the chip periphery, and the corresponding periphery cost P_i^H (P_i^V) is minimized. One thing deserves attention is that when putting the values of x_i/y_i into (4) for periphery cost calculation, the value relationship between $[w_{core}/2]/[h_{core}/2]$ and x_i/y_i is already known, and thus the absolute notions in (4) can be eliminated during calculation and thus becomes differentiable. Besides, when the value of x_i/y_i equals to $[w_{core}/2]/[h_{core}/2]$, we will shift m_i to either left or right / top or bottom direction by 500 units to avoid the infinity error of

$$P_i^H = \left|\frac{w_{core}}{2} - x_i\right| + \frac{(\frac{w_{core}}{2})^2}{\left|\frac{w_{core}}{2} - x_i\right|}$$
$$P_i^V = \left|\frac{h_{core}}{2} - y_i\right| + \frac{(\frac{h_{core}}{2})^2}{\left|\frac{h_{core}}{2} - y_i\right|} \quad (3)$$
$$P_i = P_i^H + P_i^V.$$

The objective function of the gradient-based method can be formulated as

$$\min \quad \mathcal{L} = \sum_{e \in Net_{macro}} WL_e + \alpha \sum_{m_i \in \mathcal{M}_{\text{poor}}} P_i \quad (4)$$

where Net_{macro} is a set of nets such that each net e in Net_{macro} contains at least one macro, and α is a parameter to tradeoff the effect between wirelength cost and peripheral cost, we set α to be 1 in this work.

At the beginning of each iteration in the gradient-descent process, the macro offsets are set to zero. For each poorly placed macro, designated as m_i, the gradient $(\partial\mathcal{L}/\partial\Delta x_i, \partial\mathcal{L}/\partial\Delta y_i)$ is directly applied as the macro offset for updating its position. The new position of m_i is calculated by adding this macro offset to the original position (x_i, y_i). If this positional update results in m_i exceeding the boundaries of the chip or overlapping with other regularly placed macros, the update is discarded, and m_i is reverted to its original position.

The gradient-based macro shift method clusters macros connected by the same net and pushes them toward the nearest chip boundary. However, this approach may lead to overlaps among poorly placed macros, as depicted in Fig. 2(b).

C. Macro Legalization

A linear programming method, based on CGs, is developed to address the issue of overlaps among poorly placed macros while maintaining the relative positional relationships established by the analytical placer, and facilitating the movement of macros toward the chip boundary. Within the context of macro placement, a CG is defined as a directed acyclic graph (DAG) where each vertex represents a macro within the placement layout, and each edge dictates the relative positional relationship between two macros. Two specific CGs are initialized for managing horizontal and vertical positions: G_h for horizontal constraints and G_v for vertical constraints. In G_h (or G_v), a directed edge from macro m_u to macro m_v implies that m_u is positioned entirely to the left of (or below) m_v within the layout. It is essential that for each pair of macros in the circuit, a directed edge connecting them should be present in either G_h or G_v, but not in both to avoid redundancy and conflicts. Fig. 6 illustrates a sample macro placement layout and its corresponding CGs, providing a visual representation of these relationships.

XDP [35] employs horizontal and vertical CGs based on a global placement prototype, utilizing linear programming to resolve overlaps among macros. However, overlaps in the placement prototype may necessitate adjusting the direction of some edges in the graphs (transferring an edge from the horizontal to the vertical graph or vice versa) to prevent macro expansion beyond the chip's boundaries. Such adjustments can alter the relative positional relationships of the macros and potentially compromise the wirelength optimization achieved by the placement prototype. Different from [35], we construct G_h and G_v on the macro-legalized placement prototype by the analytical placer. For a macro pair (m_u, m_v), we denote their center positions by (x_u, y_u) and (x_v, y_v). There are three situations for the spatial relationship between m_u and m_v, and we propose a three-stage procedure to determine the edge direction of uv: 1) if m_u and m_v are overlapped in vertical direction, but not overlapped horizontally, then edge uv will be appended to the horizontal CG and 2) if m_u and m_v are horizontally overlapped but vertically separated, then edge uv will be appended to the vertical CG.

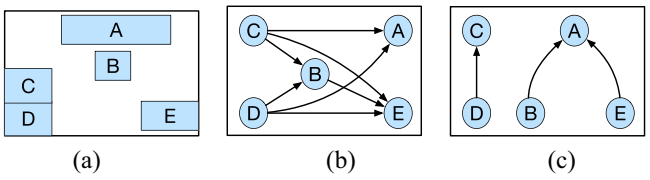


Fig. 6. Example of CGs in macro placement. (a) Sample of macro placement layout. (b) Horizontal CG. (c) Vertical CG.

3) If m_u and m_v are simultaneously separated in horizontal and vertical directions, which means they are not overlapped horizontally or vertically, then the direction of uv is determined due to the value relationship between $|x_u - x_v|$ and $|y_u - y_v|$: if $|x_u - x_v| > |y_u - y_v|$, the direction of the edge uv is horizontal; otherwise vertical. By the above three-stage edge-determination procedure, we can guarantee that on the longest path of the constructed horizontal/vertical CG, each adjacent pair of macros are not overlapped horizontally/vertically. Since the placement prototype is legalized for macro placement, the longest path of the horizontal/vertical CG will not exceed the width/height of the die. Therefore, the generated macro-legalization solution is always feasible and legal. This three-stage edge-determination procedure eliminates the ambiguity in the relative positional relationship of macro pairs and preserves wirelength optimization in the analytical placer.

We introduce some related notions before the detailed linear programming formulation. For any macro m_i in poorly placed macro set $\mathcal{M}_{\text{poor}} = \{m_1, m_2, \ldots, m_n\}$, its center positions *before* and *after* macro legalization are denoted by (x_i, y_i) and (x_i', y_i'), and its width and height are denoted by w_i and h_i, respectively. For the chip placement region R, the positions of the left-bottom corner and the right-top corner are $(0, 0)$ and (W, H). Denote any edge in G_h or G_v by e_{ij}, the macro legalization can be in

$$\begin{aligned}
\min \quad & \sum_{i=1}^{|M_{\text{poor}}|} |x_i' - x_i| + |y_i' - y_i| \\
& + \min(x_i, W - x_i) + \min(y_i, H - y_i) \\
\text{s.t.} \quad & x_j' - x_i' \geq \frac{w_i + w_j}{2} \ \forall e_{ij} \in G_h \\
& y_j' - y_i' \geq \frac{h_i + h_j}{2} \ \forall e_{ij} \in G_v \\
& \frac{w_i}{2} \leq x_i' \leq W - \frac{w_i}{2}, \frac{h_i}{2} \leq y_i' \leq H - \frac{h_i}{2} \quad \forall m_i \in \mathcal{M}_{\text{poor}}.
\end{aligned} \quad (5)$$

In (5), the primary objective term $(|x_i' - x_i| + |y_i' - y_i|)$ is designed to minimize the displacement of each poorly placed macro and preserve the wirelength-optimization by the placement prototype. The secondary objective term $(\min(x_i, W - x_i) + \min(y_i, H - y_i))$ is aimed at reducing the distances between each macro m_i and its nearest horizontal and vertical chip boundaries, thereby encouraging m_i to be positioned closer to the chip's periphery. This, in turn, creates more contiguous space in the chip center for standard cell placement and routing. The initial two inequalities in (5), are derived from G_h and G_v to eliminate any overlaps between macros while maintaining their relative positional relationships based on the placement prototype. The third inequality ensures that the legalized macros do not exceed the physical chip boundary

$$\begin{aligned}
\min \quad & \sum_{i=1}^{|M_{\text{poor}}|} x_i^p + x_i^q + y_i^p + y_i^q \\
& + \min(x_i, W - x_i) + \min(y_i, H - y_i) \\
\text{s.t.} \quad & x_i^p - x_i^q = x_i' - x_i,\ y_i^p - y_i^q = y_i' - y_i\ \forall m_i \in \mathcal{M}_{\text{poor}} \\
& x_i^p \geq 0,\ x_i^q \geq 0,\ y_i^p \geq 0,\ y_i^q \geq 0\ \forall m_i \in \mathcal{M}_{\text{poor}} \\
& x_j' - x_i' \geq \frac{w_i + w_j}{2}\ \forall e_{ij} \in G_h \\
& y_j' - y_i' \geq \frac{h_i + h_j}{2}\ \forall e_{ij} \in G_v \\
& \frac{w_i}{2} \leq x_i' \leq W - \frac{w_i}{2},\ \frac{h_i}{2} \leq y_i' \leq H - \frac{h_i}{2}\quad \forall m_i \in \mathcal{M}_{\text{poor}}.
\end{aligned} \tag{6}$$

In (5), the macro displacement cost item is expressed in the format of absolute value, which cannot be directly solved in linear programming. Therefore, we employ an equivalent transformation method, as previously discussed in [24], on (5) to obtain the revised (6). This transformation involves introducing a set of auxiliary variables $\{x_i^p, x_i^q, y_i^p, y_i^q\}$ for each poorly placed macro m_i, while adhering to the initial two constraints outlined in (6). Consequently, the macro displacement cost of m_i can be reformulated as $x_i^p + x_i^q + y_i^p + y_i^q$.

D. Diffusion-Based Cell Migration

After the iteration of macro shift (Section III-B) and macro legalization (Section III-C), the positions of poorly placed macros are updated. However, the positions of standard cells in $\mathcal{C}$ (standard cells in the same nets of macros) remain unaltered. Consequently, overlaps between the standard cells and macros occur, resulting in an imprecise wirelength estimation during the next gradient-based macro shift iteration. Therefore, a practical approach to rearranging the standard cells into appropriate positions is necessary for more precise wirelength estimation in the subsequent IncreMacro iteration.

Inspired by [36], we propose a diffusion-based cell migration algorithm for the standard cell rearrangement. In physics, diffusion depicts the net movement of any object from a high-concentration region to a low-concentration region until the equilibrium of concentration is achieved. We can make the analogy between diffusion and standard cell rearrangement by dividing the placement region uniformly into subregions, and using the cell/macro density of each subregion as the measure of concentration. Usually, subregions with high density are prone to overlaps among macros and standard cells, and standard cells in a high-density subregion are expected to migrate to a low-density one, simulating the behavior of an analytical placer for standard cell placement.

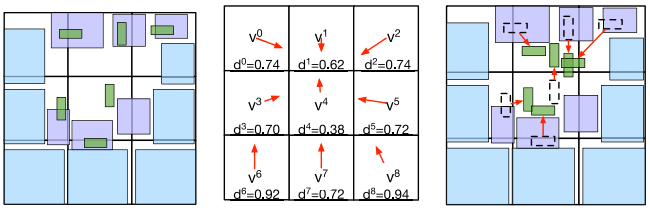


Fig. 7. Visualization of diffusion-based cell migration. Blue and purple blocks represent the regularly and poorly placed macros, and green rectangles represent standard cells. The red arrow in (b) denotes the velocity direction of each bin, while the red arrow in (c) represents the standard cell migration.

We now elaborate on the diffusion-based cell migration algorithm. Assuming the cell migration is finished within a given time t, we first decompose the cell movements into n time steps $\{t_i, t_2, \ldots, t_n\}$ such that $t = n\Delta t$. At each time step t_m, the total movement of a standard cell c_i is calculated as $v_i^m \Delta t$, where v_i^m denotes the velocity of c_i at the timestamp t_m. To calculate v_i^m, we evenly divide the chip into squared bins and for each bin b_k, its density at timestamp t_m, denoted by $d^k(t_m)$, is calculated beforehand. Assume b_k contains a set of macros $\mathcal{M}_k$ and a set of standard cells $\mathcal{C}_k$, the function area() returns the area of either an instance (a macro or cell) or a squared bin, and $d^k(t_m)$ is calculated in

$$d^k(t_m) = \frac{\sum_{c \in \mathcal{C}_k} \text{area}(c) + \sum_{m \in \mathcal{M}_k} \text{area}(m)}{\text{area}(b_k)}. \tag{7}$$

Next, we calculate the velocity of standard cells at the timestamp t_m. For simplicity, we assume all cells in the same bin share the same velocity, and denote the velocity of bin b_k by $V^k = \{V_H^k, V_V^k\}$, where V_H^k and V_V^k are for horizontal and vertical velocity, respectively. For any bin b_k with its four orthogonal-direction neighbors $\{b_k^{\text{left}}, b_k^{\text{right}}, b_k^{\text{upper}}, b_k^{\text{bottom}}\}$ and their corresponding densities $\{d_{\text{left}}^k, d_{\text{right}}^k, d_{\text{upper}}^k, d_{\text{bottom}}^k\}$, we calculate $\{V_k^H, V_k^V\}$ in (8). For the bin b_k, the vector sum of V_k^H and V_k^V represents the net velocity of all cells in b_k, and the net velocity is visualized as the red arrow in Fig. 7(b)

$$\begin{aligned}
V_k^H(t_m) &= \frac{d_{\text{right}}^k(t_m) - d_{\text{left}}^k(t_m)}{d^k(t_m)} \\
V_k^V(t_m) &= \frac{d_{\text{upper}}^k(t_m) - d_{\text{bottom}}^k(t_m)}{d^k(t_m)}.
\end{aligned} \tag{8}$$

For any bin $b_{k'}$ on the chip boundary, one or two of the orthogonal-direction neighbor bin(s) may be missing. For this scenario, the density of $b_{k'}$, namely $d^{k'}(t_m)$, is directly used to replace that of the missing bin(s). For example, for bin b_0 in Fig. 7(b), the densities of b_0^{upper} and b_0^{left} are replaced by the density of b_0 at timestamp t_m (denoted by $d^0(t_m)$), whose value is 0.74.

Finally, after the calculation of the velocity of each bin, the migrated positions of standard cells in any bin b_k at the next timestamp can be calculated as follows. For each cell $c \in C_k$ at timestamp t_m, given its current position $(x_c(t_m), y_c(t_m))$, its position at timestamp $t_m + \Delta t$ $(x_c(t_m + \Delta t), y_c(t_m + \Delta t))$ can be calculated in (9), indicating that standard cells in bin b_k will be shifted horizontally/vertically toward its orthogonally horizontal/vertical neighbor bin with smaller density

$$\begin{aligned} x_c(t_m + \Delta t) &= x_c(t_m) + V_k^H(t_m)\Delta t \\ y_c(t_m + \Delta t) &= y_c(t_m) + V_k^V(t_m)\Delta t. \end{aligned} \tag{9}$$

Fig. 7 visualizes the process of diffusion-based cell migration. Fig. 7(a) and (b) demonstrates one circuit layout and its corresponding density/velocity calculation at timestamp t_m. Fig. 7(c) shows the standard cell migration at timestamp $t_m + \Delta t$. Note that for standard cell rearrangement, overlaps between standard cells are permitted as long as they are rearranged to reasonable positions, since coarse-grained migration is enough for relatively accurate wirelength estimation in the next iteration of IncreMacro. After the cell migration process, standard cells are rearranged to more reasonable locations of the chip, for more reliable and accurate wirelength estimation in the next iteration.

E. Macro Pin Legalization

After IncreMacro, the refined macro placement result is obtained. However, the macro pins of the refined result may not align with the manufacturing grids defined in the PDK, which is not legal and causes issues in the following VLSI flow. For example, commercial tools, such as Cadence Innovus report errors and terminate when detecting any disalignment between macro pins and the manufacturing grids [37]. Although it is possible that the size of the industrial IP macro is not the integer multiply of the manufacturing grid unit, the offsets among the macro pins are usually set as integer multiplies of the manufacturing grid size for manufacturing consideration.

Macro pin legalization targets at slightly adjusting the positions of the macros to align the macro pins to the manufacturing grid. However, manually shifting macros for manufacturing grid alignment is labor-intensive and time-consuming. Meanwhile, the greedy approach which greedily and sequentially moves each macro for pin legalization is not robust and may cause overlaps among macros. Therefore, an automatic, efficient, and robust algorithm for macro pin legalization is in the genuine demand.

We propose a macro pin legalization algorithm using a CG method and topological traversal, with pin offsets of vendor macros assumed to be integer multiples of the manufacturing grid size. This allows for simplifying the alignment of macro pins by aligning the entire macro to the grid. Given a refined macro placement, the first step of our algorithm is to construct the horizontal and vertical CGs on the refined macro placement layout. For any pair of macros m_i and m_j, if they are overlapped in the horizontal (vertical) direction and the vertical/horizontal position of m_i is less than that of m_j, an edge ij will be added to the vertical (horizontal) CG. We assign a weight w_{ij} to the edge ij, where the value of the weight is the vertical (horizontal) boundary L1 distance between m_i and m_j. The pseudo source (SO) and sink (SI) nodes are finally connected to the nodes without fanins and fanouts. Fig. 8(b) shows one vertical CG constructed on macro placement in Fig. 8(a). For example, macro B and D have overlap in the horizontal direction and the boundary L1 distance between them is 5 (assume the width and height of each grid is 10), then, an edge BD with weight 5 is added to the vertical CG.

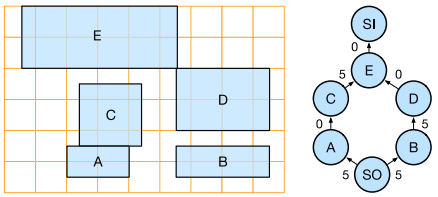


Fig. 8. (a) Refined macro placement on the manufacturing grid. The orange grids represent the manufacturing grids. Assume the width and height of each grid is 10. (b) Vertical CG constructed on the refined macro placement.

In the second step of our algorithm, the movement margins for each macro are calculated to determine how far a macro can move horizontally or vertically without causing overlaps with other macros. For vertical movement margins, the lower bound is calculated by taking the negative value of the smallest incoming edge weight to macro m_i. The upper bound is established by summing up the weights on the longest path from m_i to the pseudo sink node SI. For instance, as illustrated in Fig. 8(b), the vertical movement margin for macro B is $[-5, 5]$, meaning B can move up or down by up to five units without overlapping other macros or exceeding the chip's boundaries. The process to determine horizontal movement margins is analogous but utilizes the horizontal CG.

After calculating the horizontal and vertical movement margins for each macro, the final step of our algorithm involves moving each macro in the topological order. During this step, the horizontal and vertical movements are independent and are separately implemented sequentially. The topological movement algorithm is almost the same for the horizontal and vertical directions. For briefness, only the vertical movement algorithm is illustrated in the remainder of this paragraph. Following the topological order of the vertical CG, for any macro m_i with macro pin violation and its movement margin $[l_i, h_i]$, we first try to move m_i upward within the movement h_i to align to the manufacturing grid. If the movement fails, that is, within the upward distance h_i the macro can not be aligned to the grid, we try to move the macro downward within movement $|l_i|$ for pin-legalized placement. Especially when l_i is a positive number, the macro can be only shifted upward. After the movement of each macro, the movement margin(s) of its fanout macro(s) is(are) updated accordingly. Fig. 9 illustrates the topological macro movement algorithm step by step.

IV. Experiments

We develop IncreMacro in Python with Pytorch for its optimizer and autograd engine, and Gurobi for linear programming

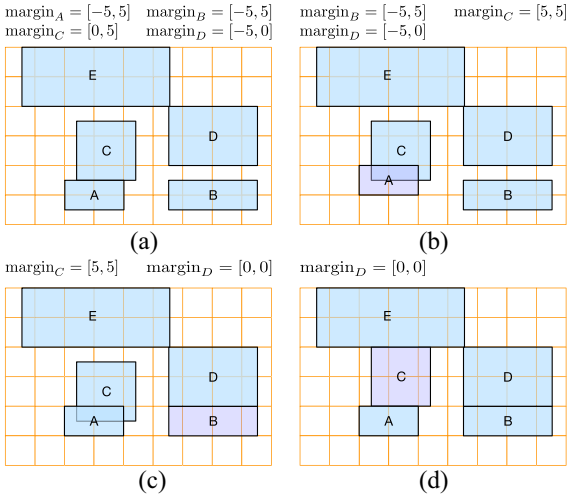


Fig. 9. Illustration of the topological macro movement algorithm for macro pin legalization. (a) Initial state of the macro placement. (b) A is shifted upward by five units, and the movement margin of C becomes [5, 5]. (c) B is shifted upward by five units, the movement margin of D becomes [0, 0]. (d) C is shifted upward by five units. After the topological movement, all macros are aligned with the manufacturing grids.

TABLE I
DESIGN INFORMATION AFTER LOGIC SYNTHESIS

Benchmark	# Macros	# Std cells	# Nets	freq. (MHz)
Rocket	121	203633	208595	500.0
GemminiRocket	737	1176141	1189717	333.3
Sha3Rocket	121	235944	240907	666.7
LargeBoomAndRocket	636	856576	874017	333.3
FFTRocket	121	211543	216507	1000.0
SmallBoom	314	4657975	372623	500.0
HwachaRocket	292	679667	687204	250.0
ariane133	133	107724	110632	1111.1
ariane136	136	102388	105362	1111.1
mempool_tile	20	131738	136380	250.0
nvdla	128	152872	162595	1111.1

solver. We run the program of IncreMacro on a Linux server with 80 Intel Xeon 1.90 GHz CPU cores and 1 NVIDIA GeForce RTX 3090 GPU. We use RISC-V SoC RTL designs from chipyard [38], an open-source framework for SoC agile development, as benchmarks. All the designs are the variants of RISC-V SoC with the core(s) being Rocket [39] and/or Boom [40]. To convert the RTL designs into gate-level netlist, we first apply Barstools [38] to map the RTL memory modules to vendor SRAMs. Then, adopting an academic 7-nm technology node (ASAP7 [41]), we leverage Cadence Genus for logic synthesis to get the gate-level netlists. Besides, for the TILOS macro placement testcases [42], the ASAP7 physical design flow is only supported for four designs, namely ariane133, ariane136, mempool_tile, and nvdla, and they are all selected as our evaluation benchmarks. In conclusion, we obtain 11 benchmark designs in total, and the placement density of all these designs is set to 0.8. More detailed information of each benchmark is listed in Table I.

A. Experiment Setting

To evaluate the performance of IncreMacro, we integrate it into the placement workflows of AutoDMP and DREAMPlace 4.0. The process begins with AutoDMP\DREAMPlace 4.0 generating a macro-legalized placement prototype. Subsequently, IncreMacro is employed to refine the macro placement on this prototype. After refining the macros, their positions are fixed, DREAMPlace is used for standard cell placement for AutoDMP, while DREAMPlace 4.0 is applied to place the standard cells for DREAMPlace 4.0. Upon completion of the standard cell placement stage, the optimized placement solution is processed by Cadence Innovus for post-placement steps, including post-placement optimization, clock tree synthesis (CTS), post-CTS optimization, signal routing, post-route optimization, and PPA evaluation.

The vanilla macro placement solutions of AutoDMP and DREAMPlace 4.0, along with IncreMacro [1] and MP-tree [22], are used as the comparative baselines. The mixed-size placement flow of AutoDMP\DREAMPlace 4.0 is segmented into two distinct stages to facilitate a consistent evaluation. In the first stage, AutoDMP\DREAMPlace 4.0 is utilized to generate a macro-legalized placement prototype. During this phase, all macros are positioned and fixed, excluding standard cells, thereby establishing the baseline macro placement. In the second stage, the corresponding analytical placer (DREAMPlace or DREAMPlace 4.0), is employed to handle the placement of the remaining standard cells. This baseline setting ensures consistency in the VLSI flows related to standard cell placement and routing across both the baseline and the refined placements facilitated by IncreMacro. Consequently, this approach allows for an isolated assessment of the impact that different macro placement solutions have on the post-route PPA metrics. Regarding the implementation of AutoDMP, we use the default parameter space and PPA configuration provided in the official Github repository [1], setting the iterations to 20 and the number of parallel workers to 2. When generating multiple Pareto-optimal placement solutions for a circuit by AutoDMP, rather than applying IncreMacro to each candidate, we select the placement solution with the maximal *congestion* value from the Pareto-optimal candidates and apply IncreMacro solely to the chosen candidate.

Moreover, we incorporate MP-tree [22] into DREAMPlace 4.0 as a baseline. After obtaining the placement prototype of DREAMPlace 4.0, MP-tree is applied for macro placement, after which DREAMPlace 4.0 is called again for the remaining standard cell placement. For the implementation of MP-tree, we set the weighting parameters of area, displacement, overlap, and wirelength objective items to be 10^{-12}, 10^{-14}, 10^{-6}, and 10^{-8}, respectively. The weighting parameter of the overlap cost item is set to a large value to guarantee the legal macro placement solution by MP-tree for the high-density designs.

Finally, for the stop criteria setting of IncreMacro, the program terminates when either the rate of change of the average macro displacement is less than 10%, or the number of total iterations exceed 10. Denote the average macro displacement at the ith iteration by $disp_i$, the calculation of the rate of change of the average macro displacement at the

[1] https://github.com/NVlabs/AutoDMP

TABLE II
POST-PLACEMENT HPWL, OVERFLOW , THE AVERAGE PATH LENGTH FOR THE TOP-1000 CRITICAL PATHS (DENOTED AS $\overline{CPL}_{1000}$) AND RUNTIME, FOR THE INCORPORATION OF IncreMacro TO AutoDMP. FOLLOWING THE PARAMETER SETTING OF THE OFFICIAL CODE, AutoDMP FAILS TO FINISH THE PLACEMENT FOR THE BENCHMARK CIRCUITS *Sha3Rocket*, *LargeBoomAndRocket*, AND *SmallBoom*. THEREFORE, THE METRICS OF THESE CIRCUITS ARE N/A

Benchmark	AutoDMP				AutoDMP + [1]				AutoDMP + IncreMacro (ours)			
	HPWL (um)	Overflow	$\overline{CPL}_{1000}$ (um)	Runtime (s)	HPWL (um)	Overflow	$\overline{CPL}_{1000}$ (um)	Runtime (s)	HPWL (um)	Overflow	$\overline{CPL}_{1000}$ (um)	Runtime (s)
Rocket	38967760	0.127	15993	69.24	38393140	**0.119**	**8061**	82.99	**38252080**	0.121	11071	90.01
GemminiRocket	112994500	0.082	21716	205.95	113044100	0.069	**14396**	381.58	**112747600**	**0.062**	18107	410.10
Sha3Rocket	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
LargeBoomAndRocket	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
FFTRocket	29628460	0.113	14146	55.14	30315320	**0.077**	**10856**	101.73	**29148840**	0.113	14504	112.69
SmallBoom	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
HwachaRocket	**63001130**	0.098	19936	193.24	66060070	0.091	18796	321.65	65200420	**0.070**	**17960**	326.90
Normalize	1.000	1.000	1.000	1.000	1.014	**0.985**	**0.726**	1.640	**0.999**	0.986	0.859	1.757

TABLE III
POST-ROUTE PPA RESULTS FOR THE INCORPORATION OF IncreMacro TO AutoDMP

Benchmark	AutoDMP				AutoDMP + [1]				AutoDMP + IncreMacro (ours)			
	WL (um)	WNS (ns)	TNS (ns)	Power (mW)	WL (um)	WNS (ns)	TNS (ns)	Power (mW)	WL (um)	WNS (ns)	TNS (ns)	Power (mW)
Rocket	14704880	-0.41	-354.87	74.33	11186898	**0.01**	**0**	68.43	**10936192**	**0.002**	**0**	**67.34**
GemminiRocket	41031737	-0.24	-17.76	301.39	**34580828**	**-0.01**	**-0.01**	**287.51**	34626683	**-0.01**	**-0.01**	288.15
Sha3Rocket	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
LargeBoomAndRocket	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
FFTRocket	8108137	-0.001	-0.001	55.43	8176606	**0.05**	**0**	**55.31**	**8019312**	**0.04**	**0**	55.37
SmallBoom	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
HwachaRocket	31675727	-0.05	-0.14	130.37	**22684613**	**0.35**	**0**	**121.66**	25946502	**0.14**	**0**	125.30
Normalize	1.000	1.000	1.000	1.000	**0.832**	**0.01**	**0.01**	**0.951**	0.849	**0.01**	**0.01**	0.956

ith iteration (denoted as r_i) can be written as

$$r_i = \frac{disp_{i-1} - disp_i}{disp_i}. \tag{10}$$

On the one hand, whenever there exists macros which are not spread out sufficiently and congested in the chip center, the displacement objective of the macro legalization step [shown in (6)] will shift those macros to the chip periphery, causing large value of r_i. Consequently, IncreMacro will continue to shift those congested macros toward the chip boundary. On the other hand, constrained by the relative positional relationship among macros determined by the KD-tree-based macro diagnosis step, the poorly placed macros will never be shifted to exceed the regularly placed macros at the chip periphery. Therefore, these poorly placed macros will never be spread out too much, which avoids drastic wirelength increase. Additionally, for some benchmarks in our experiment, we observe that after shifting all the poorly placed macros to the chip periphery, due to the different optimization directions between the wirelength objective in (2) and periphery objective in (4), some of the macros may slightly bounce around near the chip periphery even if the macro blockage is eliminated, causing a value of r_i slightly larger than 10%. To avoid such situation, IncreMacro will also terminate when the refinement iterations number exceeds 10.

B. Overall Comparisons

To demonstrate the improvement in routability and timing achieved by IncreMacro with cell migration, we report the post-placement metrics of HPWL, overflow, runtime, and average length of the top-1000 most critical paths ($\overline{CPL}_{1000}$). The *runtime* of analytical placer baselines (AutoDMP and DREAMPlace 4.0) is the total runtime of the two-stage placement, while for IncreMacro/MP-tree, *runtime* accounts for mixed-size placement prototyping (first placement stage), IncreMacro refinement/MP-tree macro placement, and standard cell placement (second placement stage). To report $\overline{CPL}_{1000}$, we first use Cadence Innovus to get the pre-CTS timing report of the analytical placement prototype, and select the top 1000 paths with the worst setup slack for average length calculation. Besides, we also report the post-route PPA metrics, including routed WL, timing (setup WNS and TNS), and total power consumption.

We integrate IncreMacro into the placement flow of AutoDMP and DREAMPlace 4.0, with Tables II and IV presenting the respective post-placement metrics. Note that for DREAMPlace 4.0, the stop criteria of global placement is when the value of overflow is less than 0.1, the overflow values among all circuits are very similar. Therefore, the overflow column is ignored in Table IV. Experimental results demonstrate that by incorporating IncreMacro with cell migration into AutoDMP (DREAMPlace 4.0), while the HPWL is maintained almost unchanged, the average length of the top-1000 critical paths ($\overline{CPL}_{1000}$ for short) are decreased by 14.1% (14.3%). On the one hand, the improved overflow and $\overline{CPL}_{1000}$ demonstrates the benefit brought by IncreMacro for improving timing performance and routability. On the other hand, compared with IncreMacro [1], the post-placement improvement also illustrates the effectiveness of our proposed algorithm of diffusion-based cell migration for improving the wirelength and timing.

The post-route PPA metrics for AutoDMP and DREAMPlace 4.0, detailed in Tables III and V, reveal significant improvements with the integration of IncreMacro

TABLE IV
POST-PLACEMENT HPWL, THE AVERAGE PATH LENGTH FOR THE TOP-1000 CRITICAL PATHS (DENOTED AS $\overline{CPL}_{1000}$), AND RUNTIME, FOR THE INCORPORATION OF IncreMacro TO DREAMPlace 4.0. SINCE FOR DREAMPlace 4.0, THE STOP CRITERIA OF GLOBAL PLACEMENT IS WHEN THE OVERFLOW IS LESS THAN 0.1, SO THE OVERFLOW VALUES AMONG ALL CIRCUITS ARE VERY SIMILAR. THEREFORE, WE DO NOT LIST THE OVERFLOW METRIC. DREAMPlace 4.0 FAILS TO LEGALIZE THE MACRO PLACEMENT OF THE CASE *FFTRocket*, SO WE USE THE RESULTS OF DREAMPlace FOR THIS CASE

Benchmark	DREAMPlace 4.0			DREAMPlace 4.0 + MP-tree			DREAMPlace 4.0 + [1]			DREAMPlace 4.0 + IncreMacro (ours)		
	HPWL (um)	$\overline{CPL}_{1000}$ (um)	Runtime (s)	HPWL (um)	$\overline{CPL}_{1000}$ (um)	Runtime (s)	HPWL (um)	$\overline{CPL}_{1000}$ (um)	Runtime (s)	HPWL (um)	$\overline{CPL}_{1000}$ (um)	Runtime (s)
Rocket	95245570	**11541**	75.78	111537900	15327	119.56	73515470	17513	100.84	**59379830**	17996	102.46
GemminiRocket	**287269600**	34762	233.28	290363600	28962	514.29	306140000	26833	596.60	306303000	**26807**	637.61
Sha3Rocket	28781060	8975	110.55	44998320	8293	173.73	29400200	8993	154.89	**28670900**	**7382**	157.12
LargeBoomAndRocket	105730700	21572	188.46	143496200	20489	285.68	104558900	12601	352.30	**104244500**	**9764**	360.64
FFTRocket	25750380	8821	243.78	34481300	**6573**	90.68	25666840	8751	72.89	**25586860**	8982	76.41
SmallBoom	**41300900**	10649	137.75	55815420	**9329**	168.49	42683060	10903	277.29	42458020	11253	320.92
HwachaRocket	**66691150**	16135	171.16	101557700	18694	301.40	68801680	**13690**	338.30	68742620	14029	337.30
ariane133	**11080710**	371	42.40	14971350	**250**	87.06	11687480	320	63.63	11939210	323	63.05
ariane136	**10287310**	516	41.81	14631670	**323**	67.36	10756020	561	69.82	10393810	524	66.43
mempool_tile	**11581190**	481	32.02	11589730	450	117.08	12459750	**385**	72.61	12514290	440	72.72
nvdla	**23187850**	90	63.18	41428060	**48**	124.48	29895610	92	103.53	29749470	92	106.41
Normalize	**1.000**	1.000	1.000	1.352	0.955	1.530	1.034	0.884	1.644	1.016	**0.857**	1.714

TABLE V
POST-ROUTE PPA RESULTS FOR THE INCORPORATION OF IncreMacro TO DREAMPlace 4.0. DREAMPlace 4.0 FAILS TO LEGALIZE THE MACRO PLACEMENT OF THE CASE *FFTRocket*, SO WE USE THE RESULTS OF DREAMPlace FOR THIS CASE

Benchmark	DREAMPlace 4.0				DREAMPlace 4.0 + MP-Tree				DREAMPlace 4.0 + [1]				DREAMPlace 4.0 + IncreMacro (ours)			
	WL (um)	WNS (ns)	TNS (ns)	Power (mW)	WL (um)	WNS (ns)	TNS (ns)	Power (mW)	WL (um)	WNS (ns)	TNS (ns)	Power (mW)	WL (um)	WNS (ns)	TNS (ns)	Power (mW)
Rocket	11271361	-0.04	-0.13	71.73	15764988	-0.002	-0.002	78.48	8760509	-0.001	-0.001	67.88	**8308426**	**0.01**	**0**	**67.20**
GemminiRocket	74163451	-0.99	-763.85	375.18	48353466	-0.08	-0.11	294.76	28417930	**0**	**0**	269.19	**28404088**	-0.03	-0.03	**268.91**
Sha3Rocket	11079460	-0.68	-83.32	96.02	13680424	-0.30	-27.81	103.94	9641501	-0.04	-0.13	91.77	**9249697**	**-0.02**	**-0.02**	**90.65**
LargeBoomAndRocket	31158986	0.19	0	131.31	40760734	0.03	0	138.09	**30800290**	0.24	0	**130.20**	31146497	0.14	0	130.40
FFTRocket	10740986	-0.01	-0.02	70.80	27011484	-1.09	-1207.00	100.02	9716426	0.02	0	67.06	**9635488**	0.01	0	**66.78**
SmallBoom	12063350	-0.01	-0.01	95.44	16481290	-1.57	-351.30	100.83	**12617001**	0.002	0	94.74	12657212	0.001	0	**94.69**
HwachaRocket	21564284	0.19	0	122.06	32075352	0.16	0	134.01	**19776057**	0.16	0	120.46	20017944	0.18	0	**120.42**
ariane133	1422190	-2.47	-12136.9	510.79	1207504	-0.36	-935.07	511.72	**983438**	-0.21	-222.58	**508.86**	997427	**-0.14**	**-190.29**	509.46
ariane136	924777	-1.42	-4159.1	512.44	1130450	-0.16	-202.69	517.72	896911	-0.18	-346.11	515.84	**862909**	**-0.13**	**-178.16**	**515.33**
mempool_tile	**787043**	-0.01	-0.28	**32.81**	1103988	-0.67	-2991.80	34.02	819267	-0.03	-8.54	33.53	819949	**0**	**0**	33.48
nvdla	2606292	-3.08	-6036.20	564.52	3209693	0	0	580.28	2229943	0	0	562.27	**2225188**	0	0	**561.17**
Normalize	1.000	1.000	1.000	1.000	1.333	0.727	0.771	1.060	0.859	0.410	0.374	0.960	**0.851**	**0.174**	**0.187**	**0.957**

with cell migration. Notably, since IncreMacro maintains the core area fixed, area metrics are not altered and thus not listed. AutoDMP could not complete the placement for several benchmarks, such as *Sha3Rocket*, *LargeBoomAndRocket*, and *SmallBoom*. Therefore, PPA results are only provided for the benchmarks that completed successfully. Tables III and V demonstrate enormous post-route timing improvement by incorporating IncreMacro with cell migration into the placement flow of AutoDMP and DREAMPlace 4.0. The routed wirelength is reduced by 15.1% and 14.9%, respectively. The setup WNS and TNS is improved by 82.6% and 81.3% for DREAMPlace 4.0, and the timing violations are nearly eliminated for AutoDMP. Besides, the incorporation of IncreMacro with cell migration also reduces the post-route power consumption by 4.4% and 4.3% for AutoDMP and DREAMPlace 4.0, respectively. Moreover, unlike MP-tree which packs macros to the chip boundary without the preservation of the relative positional relationship among macros, IncreMacro preserves the halos (spaces) among boundary macros and regularly placed macros, and maintains the macro relative spatial relationship while pushing macros toward chip boundary, thus preserving the wirelength and timing optimization by the analytical placement prototype. Table V demonstrates that compared with the macro placement results by MP-tree, IncreMacro with cell migration reduces the routed wirelength and total power consumption by 48.2% and 10.3%, and improves the WNS and TNS by 55.3% and 58.4%, respectively. Finally, compared with IncreMacro [1], our approach augmented by cell migration improves WNS and TNS by 24.7% and 23.1%, with the routed wirelength and total power consumption remain almost unchanged.

C. Analysis, Visualization, and Profiling

We owe the timing improvement and power reduction to the two macro placement requirements satisfied by IncreMacro, namely 1) *pushing macros toward the chip boundary* and 2) *preserving original macro relative positional relationship*. First, IncreMacro strategically relocates macros that are obstructing the center of the chip to the periphery, effectively eliminating macro blockage. This repositioning opens up contiguous space, facilitating smoother standard cell placement and net routing, which helps in minimizing routing detours. Second, the refinement process by IncreMacro, while maintaining the relative positional relationships of macros as established by the analytical placer's prototype (DREAMPlace 4.0 and AutoDMP), ensures that macros connected by the same net(s) are positioned in close proximity. The fulfillment of these two requirements results in a notable reduction in the total routed wirelength. This reduction subsequently translates into decreased wire capacitance and resistance. As a consequence, signal propagation delay is

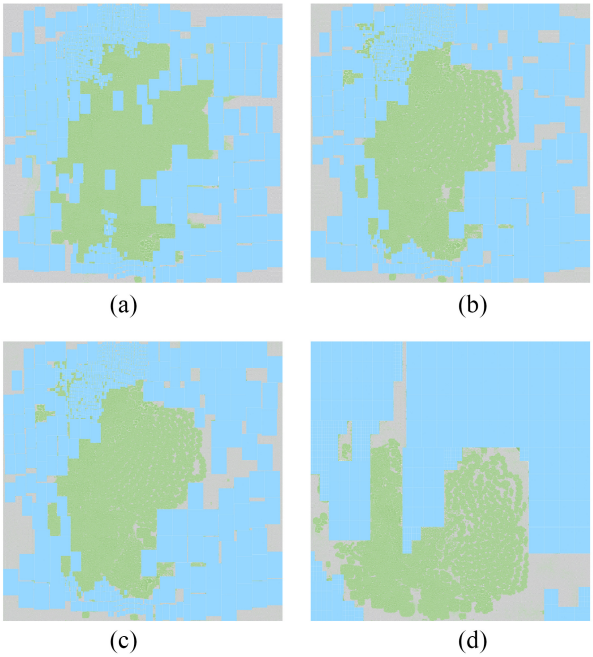


Fig. 10. Visualization of *Post-Route* layout of *GemminiRocket*, with and without the incorporation of IncreMacro and MP-tree. Blue blocks are macros and green blocks are standard cells. (a) GemminiRocket: DREAMPlace 4.0. (b) GemminiRocket: DREAMPlace 4.0 + [1]. (c) GemminiRocket: DREAMPlace 4.0 + IncreMacro (ours). (d) GemminiRocket: DREAMPlace 4.0 + MP-tree.

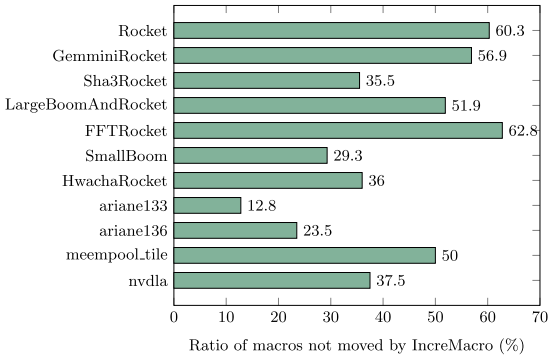


Fig. 11. Ratio of macros which are not moved after incorporating IncreMacro to DREAMPlace 4.0.

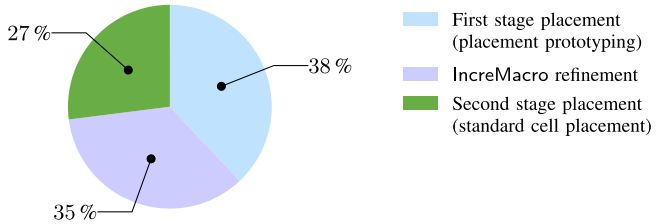


Fig. 12. Average runtime breakdown of DREAMPlace 4.0 + IncreMacro w. cell migration.

decreased, leading to a significant enhancement in timing performance. For the analysis of total power consumption reduction, the shortened wirelength leads to faster signal transitions, which minimizes short-circuit power by reducing the overlap time when both NMOS and PMOS transistors are on. It also decreases switching power by shortening the charge and discharge cycles of gate and interconnect capacitances. Additionally, reduced wire capacitance and resistance contribute to lower leakage power, overall enhancing power efficiency.

Fig. 10 visualizes the post-route layout of *GemminiRocket* for DREAMPlace 4.0, DREAMPlace 4.0 + [1], DREAMPlace 4.0 + IncreMacro (ours), and DREAMPlace 4.0 + MP-tree. Compared with the placement prototype [see Fig. 10(a)], IncreMacro preserves the relative positional relationship among macros while eliminating the macro blockage in the chip center, preserving the halos among boundary and regularly placed macros and providing continuous space in the chip center for standard cell placement and routing, as visualized in Fig. 10(b) and (c). While for MP-tree [see Fig. 10(d)], although the macros are packed to the chip boundary, the halos among macros are eliminated and the macro relative positional relationship is disturbed, leading to poor timing performance. Fig. 11 shows that the ratio of macros whose positions are not modified by IncreMacro after incorporating IncreMacro to DREAMPlace 4.0. Experimental statistics shows that by incorporating IncreMacro, the positions of 41.5% of the macros are maintained the same as the DREAMPlace 4.0 prototype on average. The halos (spaces) among the unmoved macros provide extra space for standard cell placement and signal routing, thus improving the overall timing performance. Fig. 12 shows the runtime breakdown of DREAMPlace + IncreMacro averaged on 11 benchmarks in Table V, IncreMacro with cell migration takes 35% of the total runtime.

For circuit designs with a low placement utilization rate, most of the components in their analytical placement prototype will be spread in the chip center for wirelength optimization. Consequently, directly applying IncreMacro on those low-utilization placement prototypes will inevitably lead to longer wirelength. Meanwhile, shifting macros toward the chip boundary will help eliminate the macro blockage and decrease the overflow, benefiting the following VLSI flows. To further explore the effect of different placement utilization rates on the performance of IncreMacro, we conduct experiments on the DAC2012 placement benchmark [43]. For each testcase in the benchmark, we unfix all the macros and abandon the macros with no pin, and modify the utilization rate to be 40%, 60%, and 80%, meanwhile maintaining the aspect ratio to be 1. Due to the absence of timing and PDK information in the DAC2012 benchmark, we only select AutoDMP as the baseline (DREAMPlace4.0 is timing-driven and thus not applicable), and only report the post-placement HPWL and global placement overflow as the performance metrics. During the experiments, AutoDMP is run on each testcase of three difference utilization rates, and the Pareto-Optimial solution with the maximal *congestion* value is chosen, on which IncreMacro is applied for macro placement refinement. Experimental results in Fig. 13, demonstrate that for designs with relatively low placement utilization rate (such as 40% and 60%), IncreMacro causes longer HPWL by shifting macros

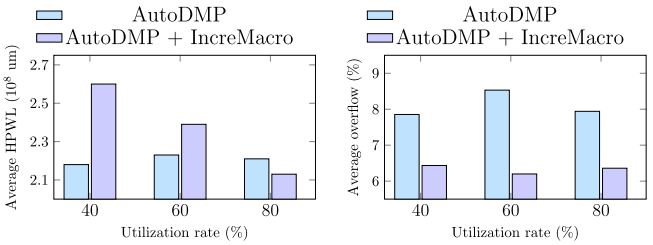


Fig. 13. Post-placement average (a) HPWL and (b) overflow for the incorporation of IncreMacro into AutoDMP, evaluated on the DAC2012 benchmark with different utilization rates. Since AutoDMP fails on the case *superblue2*, the metrics are averaged on the remaining nine testcases.

toward the chip boundary. However, when the utilization rate is high, for example, 80%, by eliminating the centric macro blockage, IncreMacro improves the wirelength. Moreover, at all utilization rates, IncreMacro prominently decreases the overflow. In conclusion, IncreMacro is the most effective in refining the high-utilization circuit placement, and for circuits with low utilization rate, although IncreMacro leads to worse wirelength, it improves the overflow, which is beneficial for the subsequent VLSI flows, such as signal routing and timing optimization.

V. Conclusion

We proposed an incremental macro placement framework incorporated with analytical placers, IncreMacro, which incrementally and iteratively refines the placement of poorly placed macros. IncreMacro integrates several techniques for macro placement, including a KD-tree-based macro diagnosis method, a gradient-based macro shift, constraint-graph-based linear programming for macro legalization, and diffusion-based cell migration. With such algorithmic workflow, IncreMacro eliminates the macro blockages in the core center while preserving the macro relative positional relationship, thus preserving the wirelength-optimization by analytical placers, and leaves more continuous centric space for routing. Experimental results have demonstrated the effectiveness of our framework, compared with state-of-the-art macro placement solutions.

References

[1] Y. Pu et al., "IncreMacro: Incremental macro placement refinement," in *Proc. ISPD*, 2024, pp. 169–176.

[2] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "Replace: Advancing solution quality and routability validation in global placement," *IEEE TCAD*, vol. 38, no. 9, pp. 1717–1730, Sep. 2019.

[3] 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, Jul. 2008.

[4] A. B. Kahng and Q. Wang, "A faster implementation of APlace," in *Proc. ISPD*, 2006, pp. 218–220.

[5] J. Lu et al., "ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method," *ACM TODAES*, vol. 20, no. 2, pp. 1–34, 2015.

[6] J. Lu et al., "ePlace-MS: Electrostatics-based placement for mixed-size circuits," *IEEE TCAD*, vol. 34, no. 5, pp. 685–698, May 2015.

[7] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "POLAR: A high performance mixed-size wirelengh-driven placer with density constraints," *IEEE TCAD*, vol. 34, no. 3, pp. 447–459, Mar. 2015.

[8] T. Lin, C. Chu, and G. Wu, "POLAR 3.0: An ultrafast global placement engine," in *Proc. ICCAD*, 2015, pp. 520–527.

[9] L. Liu, B. Fu, M. D. Wong, and E. F. Young, "Xplace: An extremely fast and extensible global placement framework," in *Proc. DAC*, 2022, pp. 1309–1314.

[10] 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.

[11] P. Liao, H. Liu, Y. Lin, B. Yu, and M. Wong, "On a Moreau envelope wirelength model for analytical global placement," in *Proc. DAC*, 2023, pp. 1–6.

[12] 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 10, pp. 3374–3387, Oct. 2023.

[13] T.-C. Chen, P.-Y. Lee, and T.-C. Chen, "Automatic floorplanning for AI SoCs," in *Proc. VLSI-DAT*, 2020, pp. 1–2.

[14] "Innovus mixed placer." Accessed: Aug. 1, 2024. [Online]. Available: https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/innovus-mixed-placer

[15] "AI-driven macro placement boosts PPA." Accessed: Aug. 1, 2024. [Online]. Available: https://semiengineering.com/ai-driven-macro-placement-boosts-ppa/

[16] R. Cheng and J. Yan, "On joint learning for solving placement and routing in chip design," in *Proc. NeurIPS*, vol. 34, 2021, pp. 16508–16519.

[17] A. Mirhoseini et al., "Chip placement with deep reinforcement learning," 2020, *arXiv:2004.10746*.

[18] R. Cheng, X. Lyu, Y. Li, J. Ye, J. Hao, and J. Yan, "The policy-gradient placement and generative routing neural networks for chip design," in *Proc. 36th Conf. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 26350–26362.

[19] Y. Lai, Y. Mu, and P. Luo, "MaskPlace: Fast chip placement via reinforced visual representation learning," 2022, *arXiv:2211.13382*.

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

[21] Z. Jiang et al., "Delving into macro placement with reinforcement learning," in *Proc. ACM/IEEE 3rd Workshop Mach. Learn. CAD (MLCAD)*, 2021, pp. 1–3.

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

[23] Y.-F. Chen, C.-C. Huang, C.-H. Chiou, Y.-W. Chang, and C.-J. Wang, "Routability-driven blockage-aware macro placement," in *Proc. DAC*, 2014, pp. 1–6.

[24] C.-H. Chiou, C.-H. Chang, S.-T. Chen, and Y.-W. Chang, "Circular-contour-based obstacle-aware macro placement," in *Proc. ASPDAC*, 2016, pp. 172–177.

[25] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE TCAD*, vol. 15, no. 12, pp. 1518–1524, Dec. 1996.

[26] X. Hong, "Nonslicing floorplan and placement using corner block list topological representation," *IEEE Trans. Circuits Syst.*, vol. 51, no. 5, pp. 228–233, Dec. 2004.

[27] Y.-C. Liu, T.-C. Chen, Y.-W. Chang, and S.-Y. Kuo, "MDP-trees: Multidomain macro placement for ultra large-scale mixed-size designs," in *Proc. ASPDAC*, 2019, pp. 557–562.

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

[29] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on fast simulated annealing," in *Proc. ISPD*, 2005, pp. 104–112.

[30] C.-H. Chang, Y.-W. Chang, and T.-C. Chen, "A novel damped-wave framework for macro placement," in *Proc. ICCAD*, 2017, pp. 504–511.

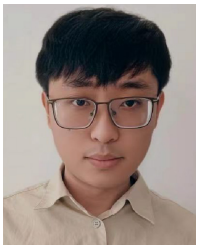
[31] H.-C. Chen, Y.-L. Chuang, Y.-W. Chang, and Y.-C. Chang, "Constraint graph-based macro placement for modern mixed-size circuit designs," in *Proc. ICCAD*, 2008, pp. 218–223.

[32] A. Agnesina et al., "AutoDMP: Automated DREAMPlace-based macro placement," in *Proc. ISPD*, 2023, pp. 149–157.

[33] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[34] M.-K. Hsu, V. Balabanov, and Y.-W. Chang, "TSV-aware analytical placement for 3-D IC designs based on a novel weighted-average wirelength model," *IEEE TCAD*, vol. 32, no. 4, pp. 497–509, Apr. 2013.

[35] J. Cong and M. Xie, "A robust detailed placement for mixed-size IC designs," in *Proc. ASPDAC*, 2006, p. 7.
[36] H. Ren, D. Z. Pan, C. J. Alpert, P. G. Villarrubia, and G.-J. Nam, "Diffusion-based placement migration with application on legalization," *IEEE TCAD*, vol. 26, no. 12, pp. 2158–2172, Dec. 2007.
[37] "encounter error while loading design." Accessed: Aug. 1, 2024. [Online]. Available: https://community.cadence.com/cadence_technology_forums/f/digital-implementation/11022/encounter--error-while-loading-design
[38] A. Amid et al., "Chipyard: Integrated design, simulation, and implementation framework for custom SoCS," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, Jul./Aug. 2020.
[39] K. Asanovic et al., "The rocket chip generator," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, Berkeley, CA, USA, Rep. UCB/EECS-2016-17, 2016.
[40] K. Asanovic, D. A. Patterson, and C. Celio, "The Berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, Berkeley, CA, USA, Rep. UCB/EECS-2015-167, 2015.
[41] L. T. Clark et al., "ASAP7: A 7-nm finFET predictive process design kit," *Microelectron. J.*, vol. 53, pp. 105–115, Jul. 2016.
[42] "TILOS-AI-institute/macroplacement." Accessed: Aug. 1, 2024. [Online]. Available: https://github.com/TILOS-AI-Institute/MacroPlacement/
[43] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei, "The DAC 2012 routability-driven placement contest and benchmark suite," in *Proc. DAC*, 2012, pp. 774–782.



Yuan Pu received the B.S. degree in computer science from the Chinese University of Hong Kong, Hong Kong, in 2022, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering supervised by Prof. Yu.

His research interest includes combinatorial algorithms/AI in EDA.



Tinghuan Chen (Member, IEEE) received the B.Eng. and M.Eng. degrees in electronics engineering from Southeast University, Nanjing, China, in 2014 and 2017, respectively, and the Ph.D. degree in computer science and engineering from the Chinese University of Hong Kong, Hong Kong, in 2021.

He is currently an Assistant Professor with the School of Science and Engineering, Chinese University of Hong Kong, Shenzhen, China. His research interests include machine learning for EDA and deep learning accelerators.



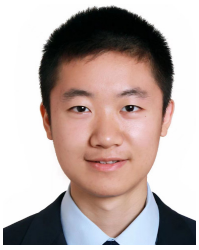
Zhuolun He received the B.S. degree in computer science and engineering from Peking University, Beijing, China, in 2017, and the Ph.D. degree from the Chinese University of Hong Kong (CUHK), Hong Kong, in 2023.

He is currently a Postdoctoral Fellow with the Department of Computer Science and Engineering, CUHK.



Jiajun Qin (Student Member, IEEE) is currently pursuing the undergraduate degree with the Department of Computer Science and Technology, Zhejiang University, Hangzhou, China.

His research interests lie at the intersection of artificial intelligence and hardware.



Chen Bai (Member, IEEE) received the B.E. degree in software engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2020, and the Ph.D. degree in computer science and engineering from the Chinese University of Hong Kong, Hong Kong, in 2024.

His research interests include computer architecture and electronic design automation.

Dr. Bai received the William J. McCalla Best Paper Award from ICCAD 2021 and the Best Paper Award Nomination from ISPD 2024.



Haisheng Zheng received the B.Eng. degree in Internet of Things engineering from Tiangong University, Tianjin, China, in 2020.

He is currently a Researcher with Shanghai AI Laboratory, Shanghai, China, since 2022. From 2020 to 2022, he was a Research and Development Engineer with SmartMore, Shanghai. His research interests include artificial intelligence, embedded systems, high performance compute, and IC design.



Yibo Lin (Member, IEEE) received the B.S. degree in microelectronics from Shanghai Jiaotong University, Shanghai, China, in 2013, and the Ph.D. degree in electrical and computer engineering from the University of Texas at Austin, Austin, TX, USA, in 2018 advised by Prof. Pan.

From 2018 to 2019, he was as a Postdoctoral Researcher with the University of Texas at Austin. He is currently an Assistant Professor with the School of Integrated Circuits, Peking University, Beijing, China. His research interests include physical design, machine learning applications, and heterogeneous computing in VLSI CAD.

Dr. Lin is a recipient of the Best Paper Awards at premier EDA/CAD journals/conferences, such as TCAD, DAC, DATE, and ISPD.



Bei Yu (Senior Member, IEEE) received the Ph.D. degree from the University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong.

Dr. Yu received the 11 Best Paper Awards from ICCAD in 2013, 2021, and 2024, IEEE TSM 2022, DATE 2022, ASPDAC 2021 and 2012, ICTAI 2019, Integration, Journal of VLSI Circuits and Systems in 2018, ISPD 2017, SPIE Advanced Lithography Conference in 2016, six ICCAD/ISPD contest awards, and many other awards, including the DAC Under-40 Innovator Award in 2024, the IEEE CEDA Ernest S. Kuh Early Career Award in 2022, and the Hong Kong RGC Research Fellowship Scheme Award in 2024. He has served as a TPC Chair for ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees.