IncreMacro: Incremental Macro Placement Refinement

Yuan Pu
The Chinese University of Hong Kong
Hong Kong SAR
Shanghai AI Laboratory, China

Tinghuan Chen
The Chinese University of Hong Kong
Hong Kong SAR

Zhuolun He
The Chinese University of Hong Kong
Hong Kong SAR
Shanghai AI Laboratory, China

Chen Bai
The Chinese University of Hong Kong
Hong Kong SAR

Haisheng Zheng
Shanghai AI Laboratory
China

Yibo Lin
Peking University, Beijing, China
Institue of EDA, Peking University,
Wuxi, China

Bei Yu
The Chinese University of Hong Kong
Hong Kong SAR

ABSTRACT

This paper 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 and constraint-graph-based LP for macro legalization. By employing these techniques iteratively, IncreMacro meets two critical solution requirements of macro placement: (1) pushing macros to the chip boundary; and (2) preserving the original macro relative positional relationship. The proposed approach has been incorporated into DREAMPlace and AutoDMP, and is evaluated on several RISC-V benchmark circuits at the 7-nm technology node. Experimental results show that, compared with the macro placement solution provided by DREAMPlace (AutoDMP), IncreMacro reduces routed wirelength by 6.5% (16.8%), improves the routed worst negative slack (WNS) and total negative slack (TNS) by 59.9% (99.6%) and 63.9% (99.9%), and reduces the total power consumption by 3.3% (4.9%).

CCS CONCEPTS

• **Hardware → Placement**.

KEYWORDS

Macro Placement, Incremental Placement

ACM Reference Format:
Yuan Pu, Tinghuan Chen, Zhuolun He, Chen Bai, Haisheng Zheng, Yibo Lin, and Bei Yu. 2024. IncreMacro: Incremental Macro Placement Refinement . In *Proceedings of the 2024 International Symposium on Physical Design (ISPD '24), March 12–15, 2024, Taipei, Taiwan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3626184.3633321

ISPD '24, March 12–15, 2024, Taipei, Taiwan

ACM ISBN 979-8-4007-0417-8/24/03…$15.00
https://doi.org/10.1145/3626184.3633321

1 INTRODUCTION

Effective macro placement is a critical procedure of the physical design flow for Very Large-Scale Integration (VLSI) circuits, owing to the sizable fraction of chip area occupied by pre-designed macro blocks, particularly embedded memories. Inappropriate macro placement can result in macro blockage in the core center, which harms the overall chip performance by causing unwanted effects such as routing congestion, inferior wirelength, and timing performance 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 modern VLSI design increasingly includes pre-designed macro blocks, effective macro placement is crucial to improve routability and timing performance.

Analytical placers such as DREAMPlace [1], RePlace [2], ePlace [3] and NTUplace3 [4] support mixed-size placement of standard cells and macros. In macro placement, a well-established practice among experienced engineers is to place macros towards the peripheral regions of the chip to prevent macro blockage. However, existing analytical placers prioritize wirelength optimization during placement, which diverges from the aforementioned practice and can result in macro blockage in the core center. Such macro blockage may negatively impact the placement of standard cells, and hamper the subsequent routing stage. In Figure 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 sub-regions. Consequently, standard cells associated with the same net may be dispersed across separate placement sub-regions, introducing detours into the VLSI routing 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

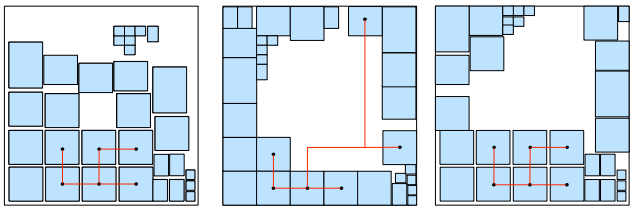


Figure 1: (a). Macro placement by an analytical placer. (b). Macro placement refined by data structure-centric metaheuristics. (c). Macro placement refined by IncreMacro**. The blue block represents macro. The white region represents the remaining space for standard cell placement and routing. A 6-pin net is denoted in red color.**

and escalate power consumption. To achieve superior quality-of-results (QoR) in the final physical design, meticulous attention to macro placement to eliminate macro blockage is imperative.

Numerous academic studies have explored the subject of macro placement, where reinforcement learning (RL) driven approaches and other metaheuristics are frequently utilized. The RL-based technique [5–11] frames macro placement as a sequential Markov Decision Process (MDP) and employs reinforcement learning to solve the problem. Within the context of the sequential MDP, intermediate macro placement solutions are represented as states and the assignment of a macro to a legalized position on the chip canvas is considered an action. Following macro placement, an analytical placer is utilized for sequential standard cell placement. Instead of refining existing macro placement, the RL-based approach generates macro placement from scratch. State-of-the-art (SOTA) data structure-centric metaheuristics have been proposed for macro legalization. The proposed approaches utilize efficient data structures, such as MP-tree, CP-tree, and modified corner sequence [12–21], to represent the macro layout. Then, Simulated Annealing (SA) algorithm is leveraged to remove overlaps among macros while optimizing objectives such as macro displacement, wirelength, and routability. Recently, AutoDMP [22], a DREAMPlace-based macro placement framework is proposed. AutoDMP explores the parameter space of DREAMPlace by multi-objective Bayesian optimization, and generates macro placement solutions with a good trade-off among power, performance and area (PPA).

Existing academic studies for macro placement have two major drawbacks: 1). The RL-based approach and AutoDMP are *computationally prohibitive*. To satisfy the desired PPA requirements, the macro placement generation process of the RL-based approach and AutoDMP may be iterated for many times on different parameter settings. 2). On refining the macro placement of the existing placement prototype, the data structure-centric metaheuristics may disturb the macro relative positional relationship of the placement prototype, potentially leading to wirelength degradation and timing performance issues. An example of applying the data structure-centric metaheuristic approach to legalize an analytical placement prototype is shown in Figure 1(b). Although the approach successfully eliminates macro overlaps and packs macros to the chip periphery, it perturbs the original macro relative positional relationship of the placement prototype, resulting in poor wirelength performance for the 6-pin net.

To overcome the above drawbacks and achieve a better trade-off between computational time and solution quality for macro placement, a macro placement refinement algorithm which eliminates the macro blockage and preserves the macro relative positional relationship by the placement prototype is desired. Unfortunately, to the best of our knowledge, there is no such prior work. 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 to the chip boundary**, and (2) **preserve the macro relative positional relationship established by the placement prototype**. Figure 1(c) presents an ideal macro placement solution achieved by applying IncreMacro to an academic analytical placer. Compared to Figure 1(a), this solution eliminates macro blockage in the core center, creating continuous open space for subsequent standard cell placement and routing. Compared to Figure 1(b), this placement preserves the wirelength optimization accomplished by the analytical placer's placement prototype, as evidenced by the preserved relative positional relationship of the macros involved in the 6-pin net, leading to a shorter net wirelength.

Our major contributions are summarized as follows:

- We propose IncreMacro which incrementally enhances macro placement by state-of-the-art analytical placers.
- We propose a novel algorithmic workflow consisting of KD-tree-based macro diagnosis, gradient-based macro shift and constraint-graph-based linear programming for macro legalization, to eliminate macro blockage in the chip center while preserving the wirelength optimization by the analytical placement prototype.
- We conduct experiments on several RISC-V circuits at 7-nm technology node. Experimental results show that incorporating IncreMacro into DREAMPlace improves the post-route WNS and TNS by 59.9% and 63.9%, and reduces the routed wirelength and total power consumption by 6.5% and 3.3%. Also, by incorporating IncreMacro into AutoDMP, the WNS and TNS are improved by 99.6% and 99.9%, and the routed wirelength and total power consumption are reduced by 16.8% and 4.9%, respectively.

2 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.

Figure 2(a) illustrates the mixed-size placement flow into which IncreMacro is incorporated. The circuit information, comprising LEF and DEF, is initially fed to the analytical placer, which generates a mixed-size placement prototype with legalized macros. Subsequently, IncreMacro is leveraged iteratively and incrementally to refine the macro placement. Upon the completion of incremental refinement, the locations of macros are fixed based on the refined macro placement solution, and an analytical placer is applied for the

remaining process of standard cell placement. Finally, the optimized placement solution is obtained.

Prior to delving into the definitions of macro regularity, it is imperative to first elucidate the effect of macro regularity on subsequent VLSI processes, including standard cell placement and signal/clock routing. As mentioned in Section 1, macros positioned in the center of the core have been observed to engender macro blockages, leading to the partitioning of available placement areas into non-contiguous sub-regions. Consequently, standard cells associated with the same net may be dispersed across different placement sub-regions, leading to increased wirelength and the potential emergence of routing challenges such as detours, ultimately degrading the timing performance. Macros with the potential to induce blockages are designated as "poorly-placed macros". Conversely, their counterparts, referred to as "regularly-placed macros", adhere to one of the following two criteria: 1) adjacency to the peripheral boundaries of the chip (e.g., m_4, m_5, m_6, m_7, m_8 in Figure 3), or 2) regularly packing and alignment with other macros (e.g., m_2, m_3 in Figure 3). Based on the observations above, we give the formal definitions of macro regularity as follows.

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 $M = \{m_1, m_2, ..., m_n\}$, 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 $|M| = |M_{regular}| + |M_{poor}|$.

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 $\mathcal{M} = \{m_1, m_2, \cdots, m_m\}$, 2) a set of standard cells $\mathcal{C} = \{c_1, c_2, \cdots, c_n\}$ which are in the same nets with macros in $\mathcal{M}$, and 3) their original positions $\mathcal{CO}_{\mathcal{M}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$ and $\mathcal{CO}_{\mathcal{C}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$. Our target is to find the legalized positions of macros in $\mathcal{M}$ to optimize the objective of wirelength.

3 ALGORITHM

Figure 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 rendered to distinguish poorly-placed macros (macros causing blockage) from regularly-placed ones. Then, by making an analogy between macro shift and the neural network training process of deep learning, we utilize a deep learning toolkit to iteratively shift the poorly-placed macros by a gradient-based method to minimize the cost of wirelength and periphery. Finally, based on the macro placement of the mixed-size placement prototype by the analytical placer, we construct two constraints graphs

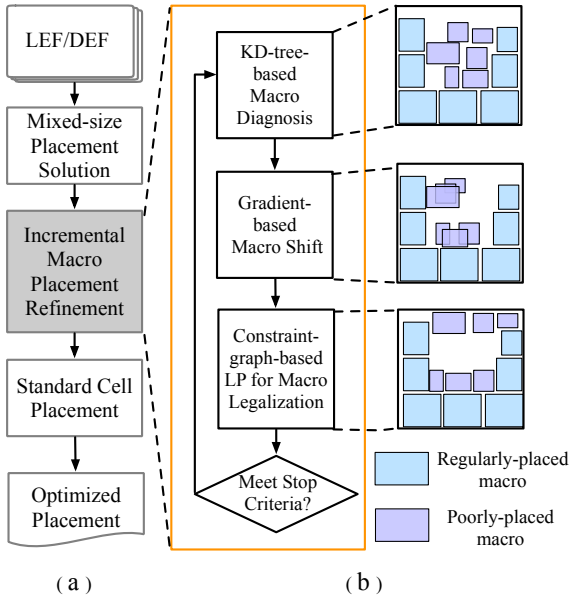


Figure 2: The mixed-size placement flow and our proposed incremental macro placement refinement tool IncreMacro**.**

to constrain the relative positional relationships of macros horizontally and vertically. A constraint-graph-based linear programming method is then designed to eliminate the overlaps among macros, minimize the macro displacement and further push poorly-placed macros to the chip periphery, with the macro relative positional relationship of the initial placement prototype preserved. The KD-tree-based macro diagnosis method, gradient-based method and constraint-graph-based linear programming are sequentially and iteratively performed until the macro blockage is eliminated (which satisfies the stop criteria).

Notice that through IncreMacro, the area and boundary of the chip remain the same as the initial placement prototype. In the following subsections, we detail the techniques used in each stage.

3.1 KD-Tree-based Macro Diagnosis

For VLSI circuits, analytical placers are effective in placing macros with similar sizes in a regular manner. However, as VLSI designs become more complex, functional modules may require varying memory sizes, resulting in diverse SRAM (macro) sizes. Placing varying-size macros with analytical placers can lead to *poorly-placed* macros and even macro blockage in the center of the chip. This can hinder subsequent VLSI processes such as 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 (the definitions of *regularly-placed macro*, *poorly-placed macro* and *macro diagnosis* are given in Definition 1, 2 and 3).

KD-tree is a versatile extension of the binary search tree, facilitating sorting, range searching, and neighbor searching within k-dimensional spaces. Each node in the KD-tree represents a k-dimensional point, and non-leaf nodes uniformly divide the space by means of generating split hyperplanes. KD-tree supports nearest neighbor searching (NNS) with an expected time complexity of

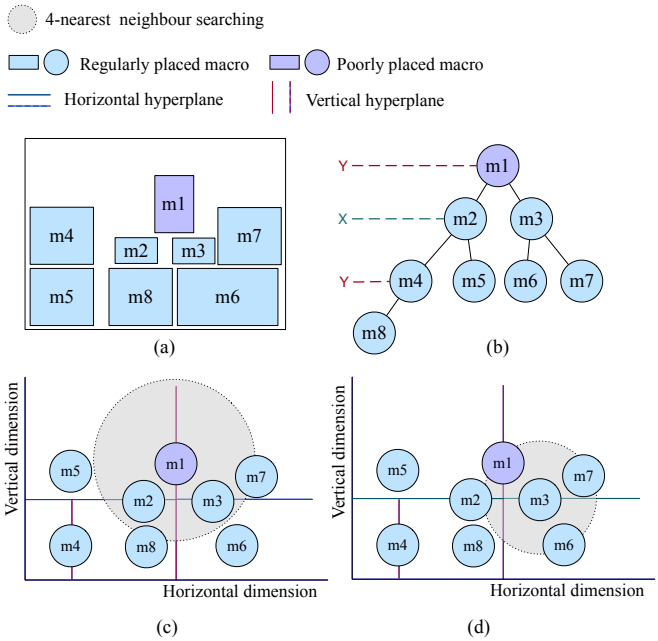


Figure 3: Customization of KD-tree for macro diagnosis.

$O(\log N)$, where N represents the number of nodes [23]. KD-tree can be represented as an efficient data structure for modeling the relative positional relationship among macros in a placement prototype, and its application in nearest neighbor searching (NNS) can be utilized for macro diagnosis.

Figure 3 depicts the process of customizing a KD-tree for macro diagnosis. In Figure 3(a), a macro placement solution with 8 macros is presented, based on which a KD-tree is constructed in Figure 3(b). The space is partitioned by generating horizontal or vertical hyperplanes for each non-leaf node ($\{m_1, m_2, m_3, m_4\}$), as shown in Figure 3(c) and Figure 3(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 Figure 3(c) and Figure 3(d). Specifically, as shown in Figure 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 Figure 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 expected time complexity of macro diagnosis is $O(N \log N)$, where N is the total number of macros in a macro placement solution.

3.2 Gradient-based Macro Shift

After macro diagnosis, poorly-placed macros are detected, and we need to shift them to the chip periphery while preserving the wirelength optimization by the analytical placer. Inspired by DREAMPlace [1], we formulate macro placement as an analytical placement problem, make the analogy between macro placement and deep neural

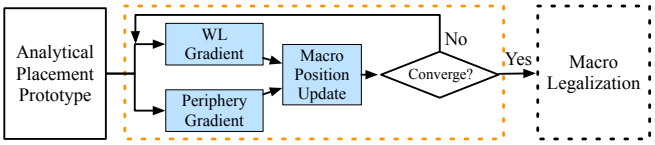


Figure 4: The Macro shifting gradient descent process.

network training, and propose a gradient-based macro shift algorithm. In the neural network training process, the neural network model is denoted as $M(w)$, where w is the model weight. For a data instance of feature vector x_i and label y_i, x_i can be fed into $M(w)$, and the corresponding label is predicted as $O(x_i, w)$. A loss function $\mathcal{L}$ is formulated to quantify the discrepancy between the true labels and the prediction results by $M(w)$, and the purpose of neural network training is to adjust the values of w such that $\mathcal{L}$ is minimized. In the analogy, we use the macro shift offset $(\Delta x_i, \Delta y_i)$ of each poorly-placed macro m_i to replace the model weight w, where the macro shift offset denotes the horizontal and vertical movement of the macro. Each data instance (x_i, y_i) is replaced with a macro-involved net instance with a feature vector and a label zero, and the loss function $\mathcal{L}$ is replaced by the convex forms of a wirelength cost (Equation (1)) and a periphery cost (Equation (3)). Minimizing the wirelength cost explicitly optimizes HPWL, while minimizing the periphery cost pushes macros to the chip boundary, leaving more continuous space for routing and implicitly optimizing routability and timing performance. Finally, by elaborating the mechanism of neural network training, we use backward propagation to calculate the gradient of each poorly-placed macro. To update the position of any poorly-placed macro m_i, we sum up the calculated gradient $(\partial\mathcal{L}/\partial\Delta x_i, \partial\mathcal{L}/\partial\Delta y_i)$ and the original position (x_i, y_i) as the new position of m_i. The flow is shown in Figure 4.

Denote any net in the design netlist by e, and e contains a set of instances (macros and standard cells) $\{v_1, v_2, \ldots, v_m\}$. The original formulation of the half-perimeter wirelength (HPWL) of e, as shown in Equation (1), is neither smooth nor convex; We thus use the weighted-average wirelength (WA) proposed by [24] to model the wirelength cost. The parameter γ governs the trade-off between the accuracy and smoothness of the approximation to half-perimeter wirelength (HPWL). Decreasing γ enhances the accuracy of the HPWL approximation while reducing its smoothness.

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

$$\begin{aligned} WA_e = &\frac{\sum_{v_i \in e} x_i e^{\frac{x_i}{\gamma}}}{\sum_{v_i \in e} e^{\frac{x_i}{\gamma}}} - \frac{\sum_{v_i \in e} x_i e^{-\frac{x_i}{\gamma}}}{\sum_{v_i \in e} e^{-\frac{x_i}{\gamma}}} \\ &+ \frac{\sum_{v_i \in e} y_i e^{\frac{y_i}{\gamma}}}{\sum_{v_i \in e} e^{\frac{y_i}{\gamma}}} - \frac{\sum_{v_i \in e} y_i e^{-\frac{y_i}{\gamma}}}{\sum_{v_i \in e} e^{-\frac{y_i}{\gamma}}}. \end{aligned} \tag{2}$$

To use WA as the wirelength cost, in Equation (2), we replace the position (x_i, y_i) of each poorly-placed macro m_i by $(x_i + \Delta x_i, y_i + \Delta y_i)$ 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 Equation (3), consisting of the horizontal cost P_i^H and vertical cost P_i^V.

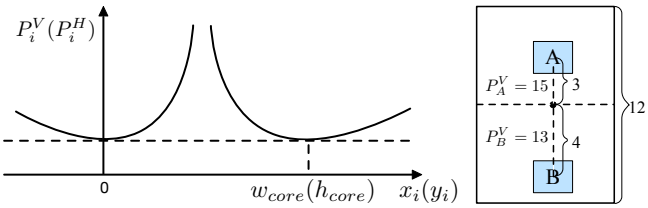


Figure 5: Peripheral cost illustration. (a) visualizes the mathematical expression of the peripheral cost, while (b) instantiates the physical meaning of the vertical peripheral cost: Denote the vertical position of macro m_A (m_B) by y_A (y_B), and assume $h_{core} = 12$, $|h_{core}/2 - y_A| = 3$ and $|h_{core}/2 - y_B| = 4$. 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 ($P_A^V = 15$).

w_{core} (h_{core}) denotes the width (height) of the chip core. As illustrated by Figure 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.

$$\begin{aligned} P_i^H &= \left|\frac{w_{core}}{2} - x_i\right| + \frac{(\frac{w_{core}}{2})^2}{\left|\frac{w_{core}}{2} - x_i\right|}, \\ P_i^V &= \left|\frac{h_{core}}{2} - y_i\right| + \frac{(\frac{h_{core}}{2})^2}{\left|\frac{h_{core}}{2} - y_i\right|}, \\ P_i &= P_i^H + P_i^V. \end{aligned} \tag{3}$$

The objective function of the gradient-based method can be formulated as Formulation (4):

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

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 trade off the effect between wirelength cost and peripheral cost, we set α to be 1 in this work.

The macro offsets are initialized as zero at the beginning of each gradient-descent iteration. For each poorly-placed macro m_i, the gradient ($\partial\mathcal{L}/\partial\Delta x_i$,$\partial\mathcal{L}/\partial\Delta y_i$) is directly used as the macro offset for position update: We sum up the original position (x_i,y_i) with the macro offset as the new position of m_i. If the update causes m_i to exceed the chip boundary or overlap with other regularly-placed macros, we discard the update and restore the original position of m_i.

By gradient-based macro shift, the macros in the same net are clustered together and pushed to the nearest chip boundary. However, overlaps may be generated among poorly-placed macros, as shown in Figure 2.

3.3 Macro Legalization

A linear programming method based on constraint graphs is formulated to remove the overlaps among poorly-placed macros, preserve the macro relative positional relationship by the analytical placer

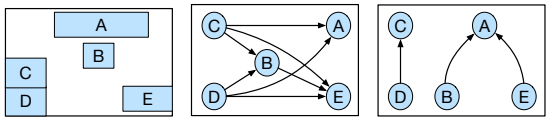


Figure 6: An example of constraint graphs in macro placement: (a) A sample of macro placement layout; (b) Horizontal constraint graph; (c) Vertical constraint graph.

and further push macros to the chip boundary. In the context of macro placement, a constraint graph is a directed acyclic graph (DAG) with the vertex set being all macros in a macro placement layout, and each edge in the graph constrains the relative positional relationship between a pair of macros. We initialize two constraint graphs, namely, G_h and G_v, for horizontal and vertical constraints, respectively. Assume in $G_h(G_v)$, a directed edge uv is between the macro m_u and m_v, then m_u is completely to the left/upon m_v in the layout. For each macro pair of the circuit, a directed edge connecting them must appear in either G_h or G_v, but not both. Figure 6 shows an example of a macro placement layout and its corresponding constraint graphs.

XDP [25] constructs horizontal and vertical constraint graphs on a global placement prototype and uses linear programming to eliminate the overlaps among macros. However, due to the existing overlaps among macros in the placement prototype, the direction of some edges in the graphs may require adjustment (move the edge from the horizontal(vertical) graph to the vertical(horizontal) graph) to avoid the macro expansion exceeding the chip periphery. This adjustment compromises the relative positional relationship of the macros and may degrade the wirelength-optimization by the placement prototype. Different from [25], 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). If $|x_u - x_v|$ is larger than $|y_u - y_v|$, we set the direction of edge uv to be horizontal; otherwise vertical. This approach 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 $\mathcal{M}_{poor} = \{m_1, m_2, \cdots, m_n\}$, 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 Formula (5).

Originally the expression of macro displacement is in the format of absolute value ($|x_i - x_i'| + |y_i - y_i'|$), which cannot be solved by linear programming. Adopting the equivalent transformation mentioned in [26], we introduce a set of auxiliary variables ($\{x_i^p, x_i^q, y_i^p, y_i^q\}$, $\forall m_i \in \mathcal{M}_{poor}$) subjected to the first two constraints listed in Formula (5), and convert the objective of macro displacement into $x_i^p + x_i^q + y_i^p + y_i^q$. By minimizing macro displacement, the displacement of each poorly-placed macro is minimized and the wirelength-optimization by the placement prototype is preserved. 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 third and fourth inequalities in Formula (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 last inequality ensures that the legalized macros do not exceed the physical chip boundary.

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{|M_{poor}|} x_i^p + x_i^q + y_i^p + y_i^q \\
& + \min(x_i, W - x_i) + \min(y_i, H - y_i) \\
\text{s.t.} \quad & x_i^p - x_i^q = x_i' - x_i,\ y_i^p - y_i^q = y_i' - y_i, && \forall m_i \in \mathcal{M}_{poor} \\
& x_i^p \geq 0,\ x_i^q \geq 0,\ y_i^p \geq 0,\ y_i^q \geq 0, && \forall m_i \in \mathcal{M}_{poor} \\
& x_j' - x_i' \geq \frac{w_i + w_j}{2}, && \forall e_{ij} \in G_h \\
& y_j' - y_i' \geq \frac{h_i + h_j}{2}, && \forall e_{ij} \in G_v \\
& \frac{w_i}{2} \leq x_i' \leq W - \frac{w_i}{2}, \frac{h_i}{2} \leq y_i' \leq H - \frac{h_i}{2}. && \forall m_i \in \mathcal{M}_{poor}
\end{aligned} \tag{5}
$$

4 EXPERIMENTS

We implement IncreMacro in Python with Pytorch [27] for its optimizer and autograd engine, and Gurobi [28] for linear programming 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 [29], 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 [30] and/or Boom [31]. To convert the RTL designs into gate-level netlist, we first apply Barstools [29] to map the RTL memory modules to vendor SRAMs. Then, adopting an academic 7-nm technology node (ASAP7 [32]), we leverage Cadence Genus for logic synthesis to get the gate-level netlists. We obtain 7 benchmarks in total and the detailed information of each benchmark is listed in Table 1.

4.1 Experiment Setting

To assess the efficacy of IncreMacro, we incorporate IncreMacro into the placement flow of DREAMPlace and AutoDMP: Firstly, DREAMPlace\AutoDMP is applied to generate a macro-legalized placement prototype. Then, IncreMacro is leveraged on the placement prototype for macro placement refinement. Once the macro refinement is completed, the positions of refined macros are fixed, and an analytical placer (DREAMPlace) is applied for standard cell placement. Finally, the optimized placement solution is fed into CadenceInnovus for clock tree synthesis (CTS), signal routing and PPA evaluation.

The vanilla macro placement solutions of DREAMPlace and AutoDMP are used as the comparative baselines. We divide the mixed-size placement flow of DREAMPlace\AutoDMP into two stages: at the first placement stage, we use DREAMPlace\AutoDMP to generate a macro-legalized placement prototype. By fixing the positions of all macros and ignoring the standard cells, the baseline macro placement is obtained. For the second stage, an analytical placer (DREAMPlace) is applied for the remaining standard cell placement. With such baseline setting, the subsequent VLSI flows of standard cell placement and routing between baselines and the IncreMacro refinement are consistent, and we solely evaluate the influence of different macro placement solutions on the post-route PPA metrics.

Table 1: 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	362479	372623	500.0
HwachaRocket	292	679667	687204	250.0

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.

4.2 Overall Comparisons

To demonstrate the improvement in routability and timing achieved by IncreMacro, we report the post-route PPA metrics, including routed wirelength (WL), timing (setup WNS and TNS) and power.

We integrate IncreMacro into the placement flow of DREAMPlace and AutoDMP, with Table 2 and Table 3 presenting the respective PPA results and *runtime*. The *runtime* of baselines (DREAMPlace and AutoDMP) is the total runtime of the two-stage placement, while for IncreMacro, the *runtime* accounts for mixed-size placement prototyping (first placement stage), IncreMacro macro refinement and standard cell placement (second placement stage). Note that due to the fixed-outline nature of IncreMacro, the core area of each design remains consistent between the baseline approach and the IncreMacro refinement. Following the official code's parameter settings, AutoDMP fails to complete the placement flow for several benchmarks, including *Sha3Rocket*, *LargeBoomAndRocket* and *SmallBoom*. Consequently, we only present the PPA results comparison for the remaining four benchmarks in Table 3.

Table 2 and Table 3 demonstrate enormous post-route timing improvement by incorporating IncreMacro into the placement flow of DREAMPlace and AutoDMP. The routed wirelength is reduced by 6.5% and 16.8%, respectively. The setup WNS and TNS is improved by 59.9% and 63.9% for DREAMPlace, 99.6% and 99.9% for AutoDMP. Besides, the incorporation of IncreMacro also reduces the post-route power consumption by 3.3% and 4.9% for DREAMPlace and AutoDMP, respectively.

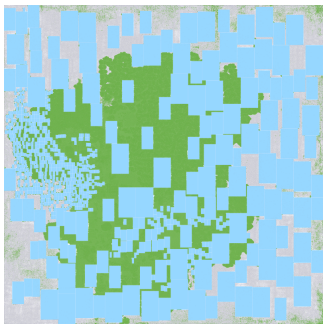
[1] https://github.com/NVlabs/AutoDMP

Table 2: Post-Route PPA results for the incorporation of IncreMacro to DREAMPlace

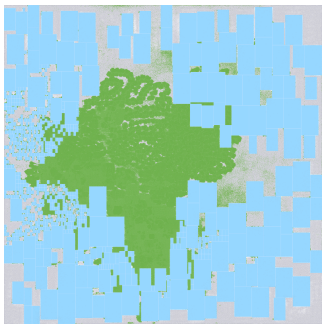
Benchmark	DREAMPlace (two stage placement)					DREAMPlace + IncreMacro				
	WL (um)	WNS (ns)	TNS (ns)	Power (mW)	Runtime (s)	WL (um)	WNS (ns)	TNS (ns)	Power (mW)	Runtime (s)
Rocket	14838552	-1.40	-67.16	72.54	64.55	**11794141**	**-0.05**	**-0.88**	**68.90**	101.77
GemminiRocket	**62420588**	-3.06	-15146.20	520.73	207.83	69708870	**-2.45**	**-4927.20**	**497.50**	418.82
Sha3Rocket	11584499	-0.22	**-40.84**	95.49	77.15	**11570282**	**-0.21**	-48.60	**95.08**	99.61
LargeBoomAndRocket	36201457	0	0	132.57	298.71	**33286377**	0.09	0	**131.46**	533.97
FFTRocket	10740986	-0.01	-0.02	70.80	63.19	**9716426**	**0.02**	**0**	**67.06**	72.89
SmallBoom	13967377	-0.13	-22.01	98.10	112.40	**13547457**	**0.01**	**0**	**96.55**	153.29
HwachaRocket	48096700	-0.10	-2.82	145.55	146.64	**40552488**	**0.03**	**0**	**137.48**	253.25
Normalize	1.065	1.599	1.639	1.033	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	1.600

Table 3: Post-Route PPA results for the incorporation of IncreMacro to AutoDMP

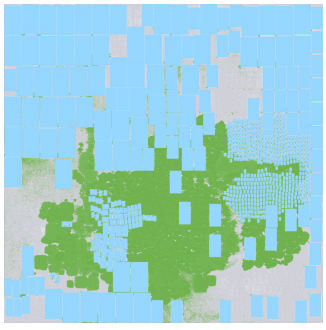
Benchmark	AutoDMP (two-stage placement)					AutoDMP + IncreMacro				
	WL (um)	WNS (ns)	TNS (ns)	Power (mW)	Runtime (s)	WL (um)	WNS (ns)	TNS (ns)	Power (mW)	Runtime (s)
Rocket	14704880	-0.41	-354.87	74.33	69.24	**11186898**	**0.01**	**0**	**68.43**	82.99
GemminiRocket	41031737	-0.24	-17.76	301.39	205.95	**34580828**	**-0.01**	**-0.01**	**287.51**	381.58
FFTRocket	**8108137**	-0.001	-0.001	55.43	55.14	8176606	**0.05**	**0**	**55.31**	101.73
HwachaRocket	31675727	-0.05	-0.14	130.37	193.24	**22684613**	**0.35**	**0**	**121.66**	321.65
Normalize	1.168	1.996	1.999	1.049	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	1.640



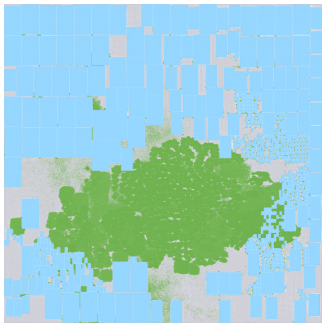
(a) GemminiRocket: DREAMPlace



(b) GemminiRocket: DREAMPlace + IncreMacro



(c) GemminiRocket: AutoDMP



(d) GemminiRocket: AutoDMP + IncreMacro

Figure 7: Visualization of *Post-Route* layout of *GemminiRocket*, with and without the incorporation of IncreMacro. Blue blocks are macros, and green blocks are standard cells.

4.3 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 to the chip boundary** and (2) **preserving original macro relative positional relationship**. Firstly, IncreMacro repositions the macros blocked in the chip center to the chip periphery and eliminates the macro blockage, leaving more continuous space for the following flow of standard cell placement and net routing, thus relieving routing detours. Secondly, after the refinement from IncreMacro, by preserving the macro relative positional relationship by the placement prototype of the analytical placer (DREAMPlace), macros in the same net(s) are placed spatially close to each other. 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 decreased, leading to a significant enhancement in timing performance. For the analysis of total power consumption reduction, reduced signal propagation infers faster signal transition, which directly reduces short-circuit power and switching power: On the one hand, short-circuit power is associated with the overlap of the ON states of NMOS and PMOS transistors in a CMOS gate during signal transitions. It occurs when both transistors are partially conducting simultaneously. Faster signal transitions lead to a decrease in short-circuit power because the duration of simultaneous conduction is shorter. On the other hand, switching power consumption is primarily related to the charging and discharging of the gate and interconnect capacitance during signal transitions. Faster signal transitions reduce the transition time, which, in turn, reduces the amount of energy required to charge or discharge the capacitance. This results in a decrease in switching power.

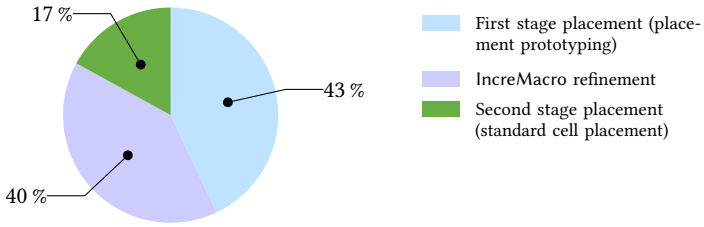


Figure 8: Average runtime breakdown of IncreMacro + DREAMPlace**.**

Figure 7 visualizes the post-route layout of *GemminiRocket* for DREAMPlace and AutoDMP, before and after the incorporation of IncreMacro: When comparing the macro placements of DREAMPlace (Figure 7(a)) and AutoDMP (Figure 7(c)), AutoDMP demonstrates superior performance with reduced macro blockage, leading to enhanced routability and timing. The difference of performance is further accentuated by IncreMacro, resulting in AutoDMP + IncreMacro (see Figure 7(d)) getting better routability and timing than DREAMPlace + IncreMacro (see Figure 7(b)). Therefore, by setting the same clock frequency (333.3 MHz) of *GemminiRocket* for both DREAMPlace and AutoDMP, the values of WNS and TNS of *GemminiRocket* in Table 2 is much larger than that in Table 3. Figure 8 shows the runtime breakdown of DREAMPlace + IncreMacro averaged on 7 benchmarks in Table 2. The first stage placement (mixed-size placement prototyping) of DREAMPlace accounts for the largest ratio of the total runtime (43%), while IncreMacro takes 40% of the total runtime.

5 CONCLUSION

We propose 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, and constraint-graph-based linear programming for macro legalization. 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.

ACKNOWLEDGEMENT

This work is partially supported by AI Chip Center for Emerging Smart Systems (ACCESS) and The Research Grants Council of Hong Kong SAR (No. CUHK14210723).

REFERENCES

[1] 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.
[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, 2018.
[3] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method," *ACM TODAES*, vol. 20, no. 2, pp. 1–34, 2015.
[4] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE TCAD*, vol. 27, no. 7, pp. 1228–1240, 2008.
[5] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.
[6] R. Cheng and J. Yan, "On joint learning for solving placement and routing in chip design," *Proc. NeurIPS*, vol. 34, pp. 16 508–16 519, 2021.
[7] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae *et al.*, "Chip placement with deep reinforcement learning," *arXiv preprint arXiv:2004.10746*, 2020.
[8] 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," *Proc. NeurIPS*, vol. 35, pp. 26 350–26 362, 2022.
[9] Y. Lai, Y. Mu, and P. Luo, "Maskplace: Fast chip placement via reinforced visual representation learning," *arXiv preprint arXiv:2211.13382*, 2022.
[10] Y. Lai, J. Liu, Z. Tang, B. Wang, J. Hao, and P. Luo, "Chipformer: Transferable chip placement via offline decision transformer," *arXiv preprint arXiv:2306.14744*, 2023.
[11] Z. Jiang, E. Songhori, S. Wang, A. Goldie, A. Mirhoseini, J. Jiang, Y.-J. Lee, and D. Z. Pan, "Delving into macro placement with reinforcement learning," in *Proc. MLCAD*. IEEE, 2021, pp. 1–3.
[12] 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.
[13] 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.
[14] 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.
[15] 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, 1996.
[16] X. Hong, "Non-slicing floorplan and placement using corner block list topological representation," *IEEE TCAS I*, vol. 51, no. 5, pp. 228–233, 2004.
[17] 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," in *Proc. ASPDAC*, 2019, pp. 557–562.
[18] 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.
[19] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on fast simulated annealing," in *Proc. ISPD*, 2005, pp. 104–112.
[20] 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.
[21] 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.
[22] A. Agnesina, P. Rajvanshi, T. Yang, G. Pradipta, A. Jiao, B. Keller, B. Khailany, and H. Ren, "AutoDMP: Automated DREAMPlace-based Macro Placement," in *Proc. ISPD*, 2023, pp. 149–157.
[23] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.
[24] 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, 2013.
[25] J. Cong and M. Xie, "A robust detailed placement for mixed-size ic designs," in *Proc. ASPDAC*, 2006, pp. 7–pp.
[26] D. F. Shanno and R. L. Weil, ""linear" programming with absolute-value functionals," *Operations Research*, vol. 19, no. 1, pp. 120–124, 1971.
[27] "Pytorch," https://pytorch.org/.
[28] "Gurobi," https://www.gurobi.com/.
[29] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.
[30] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, 2016.
[31] K. Asanovic, D. A. Patterson, and C. Celio, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," University of California at Berkeley Berkeley United States, Tech. Rep., 2015.
[32] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm FinFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.