Reinforcement Learning-based 3D Floorplanner with Die Swapping Mechanism

Qin Luo[1], Hailiang Li[2], Evangeline F.Y. Young[1]
[1]The Chinese University of Hong Kong, [2]ASTRI
qluo22@cse.cuhk.edu.hk, HarleyLi@astri.org, fyyoung@cse.cuhk.edu.hk

***Abstract*—3DICs offer significant advantages in performance, power and area over their 2D designs but introduce floorplanning challenges requiring the co-optimization of block positions and die assignments. While reinforcement learning (RL) has shown promising performance in 2D floorplanning, existing RL-based 3D methods rely on static and pre-assigned die placements, preventing true simultaneous optimization. To overcome this, we propose a novel RL-based 3D floorplanner that features a die-swapping mechanism, enabling the dynamic co-optimization of block placement and die assignment in RL-based approach for the first time. We develop an efficient spatial checking algorithm for available space checks in large layouts to support the die-swapping mechanism. Evaluation on the GSRC benchmark and a new benchmark generated by the OpenROAD suite demonstrates that our approach outperforms existing methods by significantly reducing cross-die wirelength, showcasing the critical importance of joint optimization and strong potential of RL for 3D physical design.**

***Index Terms*—3D Floorplan, Reinforcement Learning, Die Swapping**

I. Introduction

3D Integrated Circuits (3DICs) mark a fundamental shift from planar to vertical architectures, offering significant improvements in power, performance and area for large-scale designs. By interconnecting multiple silicon dies through Hybrid Bondings (HB) [1], 3DICs can drastically reduce global interconnect lengths. Traditional EDA tools that are designed for 2D circuits are ill-equipped to handle 3DIC, necessitating the development of specialized 3D physical design methodologies. Floorplanning is a critical stage in the physical design, determining the positions of major functional blocks and guiding subsequent placement and routing [2]. In the 3D context, floorplanning becomes significantly more complex, as it must also optimize the assignment of blocks across multiple dies.

Recent works on 3D floorplanning can be broadly categorized into two paradigms. The first category [3]–[6] employs a sequential approach: the circuit is initially partitioned into two dies using algorithms such as Fiduccia-Mattheyses (FM) [7], [8] or spectral partitioning [9], [10], after which each sub-netlist is floorplanned independently with a 2D floorplanner. A key limitation of this approach is that it decouples the die assignment from the floorplanning optimization, which often leads to suboptimal performance. The second category [11], [12] extends 2D floorplan representations to inherently support 3D circumstances by embedding the die assignment directly into the data structure. For instance, 3D slicing trees [11] enforce hierarchical layer partitioning and grouped sequence pairs [12] encode cross-layer block relationships. These representations are typically optimized using metaheuristics like simulated annealing. This category of methods is generally superior, as it enables the simultaneous optimization of block positions and their die assignments.

Reinforcement learning (RL) has recently emerged as a promising paradigm for 2D floorplanning and macro placement [13], [14]. Compared to simulated annealing [15], [16], RL offers a larger exploration of the solution space, and unlike analytical methods [17], it can effectively optimize non-differentiable objectives such as wirelength and congestion. FlexPlanner [18] was the first work to extend an RL-based methodology to the 3D floorplanning domain. However, despite utilizing 3D representations and optimizing between dies, it pre-assigns the blocks with die indexes according to the heuristics and the die indexes of blocks do not change in the overall RL training process. Consequently, the RL agent is restricted to optimizing block positions within these static die assignments, which fundamentally limits its ability to co-optimize block placement and layer assignment for superior performance.

To enable the co-optimization of block position and die assignment, we propose an RL-based 3D floorplanner featuring a novel die-swapping mechanism. This mechanism dynamically determines whether a block should be placed on its initially assigned die or swapped to the other die at each step. The optimal policy for this decision is learned by the agent through interactions with the environment. The main contributions of this paper are summarized as follows:

1) We introduce a novel die-swapping mechanism within an RL-based 3D floorplanner, enabling the simultaneous optimization of block positions and their die assignments for the first time.
2) We design an efficient spatial checking algorithm to rapidly identify feasible placement locations for large-scale layouts, which is critical for the die-swapping mechanism.
3) We construct and will release a new benchmark suite derived from the OpenROAD flow using LEF/DEF netlists, providing a standardized and accessible testbed for future research in floorplanning.

II. Preliminaries

A. 3D floorplanning

The input to the 3D floorplanner is a hypergraph, constructed by first clustering the standard cells of the original netlist into a set of macro blocks, denoted by $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$. Each block b_i is abstracted as a rectangle with an area a_i, calculated as the total area of its constituent cells ($a_i = \sum_{j \in b_i} \text{area}_j$).

In the 3D floorplanner, each die $l \in \mathcal{L}$ is a rectangular region with width W and height H. The bottom-left coordinate of block b_i is denoted as (x_i, y_i), and the die where b_i is located is denoted as z_i. Blocks are categorized as either *hard block* or *soft block*. For a *hard block*, its aspect ratio $AR_i = w_i/h_i$ (where w_i and h_i are the fixed width and height of the block) is fixed. For a *soft block*, its aspect ratio can vary within a range $[AR_{min}, AR_{max}]$, subject to the area constraint $w_i \cdot h_i = a_i$. Additionally, we denote the I/O port list as $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ with m ports. Each port t_j is modeled as a point with a predefined position (x_j, y_j, z_j). Given the block list $\mathcal{B}$, the I/O port list $\mathcal{T}$, and the hypergraph defining their connections, the 3D floorplanning problem aims to determine the following:

- The assigned die z for each block.
- The position (x, y) for each block on its assigned die.
- The aspect ratio AR_i for each *soft block*.

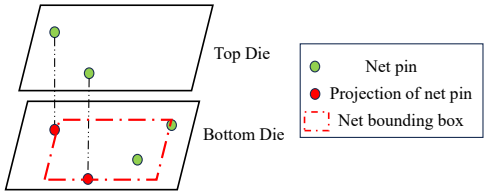


Fig. 1: The calculation of the cross-die wirelength. Cross-die wirelength is calculated as the half perimeter of the net bounding box.

B. Problem Formulation

The primary objective of our 3D floorplanner is to minimize the cross-die wirelength. The total cross-die wirelength (WL) is calculated as the sum of the half-perimeter wirelength (HPWL) over all nets in the hypergraph:

$$WL = \sum_{n \in N} [(\max_{i \in P(n)} x_i^c - \min_{i \in P(n)} x_i^c) + (\max_{i \in P(n)} y_i^c - \min_{i \in P(n)} y_i^c)] \quad (1)$$

where $P(n)$ is the set of all pins in net n. The center coordinates (x_p^c, y_p^c) for a pin are defined as follows:

- For a pin on a block b_i: $x_i^c = x_i + w_i/2$, $y_i^c = y_i + h_i/2$.
- For a pin on an I/O port t_j: $x_j^c = x_j$, $y_j^c = y_j$.

The calculation of cross-die wirelength is illustrated in Figure 1. For a net spanning multiple dies, the locations of all its pins are first projected onto a single, common plane. The bounding box of these projected pin locations (shown in red) is then constructed. Cross-die wirelength is defined as the half-perimeter of this bounding box. In addition, a valid floorplan solution must satisfy three constraints:

- Overlap: To ensure physical feasibility, blocks must not overlap beyond a predefined tolerance $\epsilon_{overlap}$.
- Aspect ratio: *Soft blocks* must maintain specified aspect ratio bounds denoted by AR_{min} and AR_{max}.
- Fixed-outline: All blocks should be placed within the die region $[W, H]$

III. Methodology

A. Markov Process Modeling

The 3D floorplanning problem can be modeled as an episodic Markov Decision Process (MDP), where an RL agent sequentially places blocks onto a multi-die canvas. The overall MDP modelling is illustrated in Figure 2. We now define the core elements in it as follows:

- State s_t: the state encodes the partial floorplan solution, the topological information of the netlist and the characteristics of the blocks to be placed.
- Action a_t: the action at step t is a compound decision. It finalizes the placement of the current block b_t by selecting its X-Y coordinates (x_t, y_t), and it pre-assigns the next block b_{t+1} to a specific die z_{t+1} with the determined aspect ratio AR_{t+1}. This design serves a key purpose: it decomposes the challenging problem of jointly optimizing a block's die, shape, and location. The agent first makes the strategic decision of the die to be placed and the shape for the block and then focuses on the feasible placement for the block with known die and shape.
- State transition $T(s_{t+1}|s_t, a_t)$: the probability distribution over the next state s_{t+1} given the current state s_t and action a_t.
- Reward R_t: the reward function provides immediate feedback to guide the agent's policy. At each step placing block b_t, the rewards is defined as :

$$R_t = -\Delta WL_t - \lambda OV_t \quad (2)$$

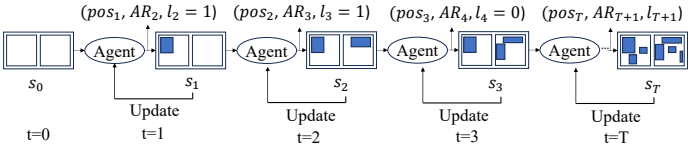


Fig. 2: The MDP modeling of the RL-based 3D floorplan.

where ΔWL_t is the change in total cross-die wirelength (calculated per Equation (1)) caused by the latest placement. The term OV_t is the overlap ratio that penalizes any over-utilization of grid cells. λ is a weighting coefficient. OV_t is calculated as:

$$OV = \frac{\sum_{z,x,y} \max\{0, f_c(z,x,y) - 1\}}{WH}$$

where W and H represent die width and height and $f_c(z,x,y)$ is the view mask that is initialized as zeros and added by 1 at grid (x_i, y_i) when it is occupied by the cell.

B. Position and Die Determination Network

To achieve this, we adopt an actor-critic framework in Figure 3. The policy network serves as the agent to determine the position and die for the blocks, while the value network estimates long-term rewards to stabilize and guide the training process. Blocks are first sorted by area in descending order and distributed into two FIFO queues $\{q_i\}_{i=1}^2$ for the top and bottom die, to initialize a balanced area distribution. At each step t, the state is represented by a combination of spatial, structural, and block-specific features. The policy network processes the state to produce probability distributions guiding two key decisions: the placement of the current block b_t, and the die assignment and aspect ratio for the subsequent block b_{t+1}. The same feature vector is also fed into the value network to produce the expected long-term reward value. Here, we use the Proximal Policy Optimization (PPO) algorithm [19] to train this actor-critic system.

1) Network inputs: The network inputs integrate three distinct modalities: the vision masks for the current partial floorplan, a graph representation of the netlist, and the embedding of the block placement order.

- **Vision Modality**
  It encodes the current floorplanning through images with three distinct vision masks.
  View Mask. The view mask $f_c \in \mathbb{N}^{|\mathcal{L}|xWxH}$ provides a global, multi-die representation of the current chip layout, with initialization to all zeros. After placing a block b at each step, we modify the view mask with $f_c[z, x : x + w, y : y + h] += 1$.
  Wire mask. The wire mask $f_w \in \mathbb{N}^{WxH}$ depicts how WL will increase when placing a new block on each position.
  Position mask. The wire mask $f_p \in \{0,1\}^{W \times H}$ identifies all feasible placement locations for the current block, where a value of 0 indicates a valid, non-overlapping position within the die boundaries. Beyond its role as an input to the network, it also actively constrains the agent's action space by filtering out invalid positions during action sampling.
- **Netlist Graph Modality**
  Given a netlist, we convert it to a graph $G(V, E)$ by expanding each net into a clique. Each vertex $v_i \in V$ corresponding to a block b_i is associated with a feature vector $(bid, x\ y, z, w, h, a, p)$, where bid is the block index, (x, y) is the X-Y coordinates, z is the die index, (w, h) is the block width and height and p indicates block placement status. Then we employ a Graph

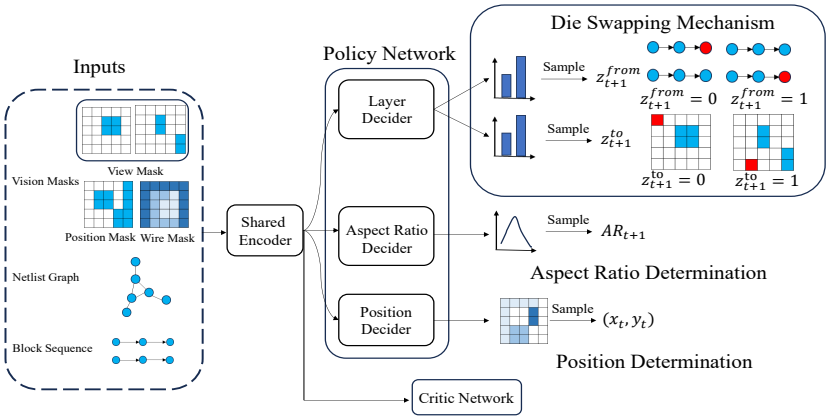


Fig. 3: Pipeline of the RL-based 3D floorplanner with die swapping mechanism. The layer decider predicts the source queue and target die for the next block b_{t+1}.

Attention Network [20] to produce the embeddings of nodes, capturing the logical connection among blocks.

- **Block Placing Sequence Modality**
  The block placing order for each die is pre-determined by its respective FIFO queue. We construct a structured sequence representation $\mathbf{S} \in \mathbb{R}^{|\mathcal{L}| x N x C}$, where N is the maximum sequence length and C is the number of features per block. Each block in these sequences is represented by the same feature vector $(bid, x, y, z, w, h, a, p)$ used in the graph modality.

2) Network outputs: The policy network generates four distinct outputs that define the action:

- **Spatial placement for the block** b_t
  The placement coordinates $(x_t,\ y_t)$ for the current block b_t are sampled from a two-dimensional probability distribution over the canvas.
- **Source queue for the next block** b_{t+1}
  The policy selects the queue from which the next block will be retrieved by sampling from a categorical distribution. Let $z_{t+1}^{from} \in \{0, 1\}$ denote this choice:
  - If $z_{t+1}^{from} = 0$, the first block is popped from queue q_0.
  - If $z_{t+1}^{from} = 1$, the first block is popped from queue q_1.

  The popped block becomes b_{t+1} in the subsequent step.
- **Target die for the next block** b_{t+1}
  The target die for placing b_{t+1} is selected by sampling from another categorical distribution. Let $z_{t+1}^{to} \in \{0, 1\}$ denote this assignment:
  - If $z_{t+1}^{to} = 0$, block b_{t+1} is assigned to the top die.
  - If $z_{t+1}^{to} = 1$, block b_{t+1} is assigned to the bottom die.
- **The aspect ratio for the next block** b_{t+1}
  The aspect ratio AR_{t+1} for the next block is sampled from a Gaussian distribution, whose mean and standard deviation are output by the policy network. The sampled value is clamped to the permissible range $[AR_{min}, AR_{max}]$ for block b_{t+1}.

C. Die Swapping Mechanism

The policy network produces two distinct die-related predictions for the next block b_{t+1}: the source queue z_{t+1}^{from} from which it is retrieved, and the target die z_{t+1}^{to} where it will be placed. The die swapping mechanism operates as follows:

- When $z_{t+1}^{from} \neq z_{t+1}^{to}$, a block is effectively moved from its source die to the opposite target die before placement.

(a) Single Block sequence (b) Dual Block sequences

Fig. 4: Illustration of the diverse placing orders with dual block sequences.

- If $z_{t+1}^{from} = z_{t+1}^{to}$, the block remains on its originally assigned die for placement.

We now discuss the insights behind the die swapping mechanism by two central ideas:

1) **Maintaining dual block sequences for top and bottom die**
   Traditional RL-based floorplanners [13], [14] typically process blocks from a single, fixed sequence. This constraint inherently limits the diversity of placement orders and can lead to suboptimal results. In contrast, our die-swapping mechanism maintains two separate sequences and dynamically selects the next block from either one. By interleaving blocks from the two queues in an order determined by the learned policy, the mechanism can explore a larger space of placement trajectories.
   The advantage of the mechanism is illustrated in Figure 4. Figure 8a depicts a conventional single sequence (e.g., ordered by descending area) with 4 blocks, which enforces a single, rigid placing order. In contrast, Figure 8b shows two separate, area-balanced sequences. While the blocks within each sequence are ordered, the policy dynamically selects between the heads of the two queues at each step and enables six possible block placing orders. This flexibility allows the RL agent to discover more effective placement strategies that are simply unreachable under a single-sequence constraint.
2) **Die swapping**
   While maintaining dual sequences expands the possible placing orders, it does not solve the fundamental problem of die assignment. Fixing a block's target die based solely on its source queue [18] remains a rigid constraint, and the optimal inter-die block distribution is hard to achieve. Our die-swapping mechanism gives more flexibility by decoupling the block's source queue from its target die. This allows the policy network to dynamically assign a block to either die, regardless of

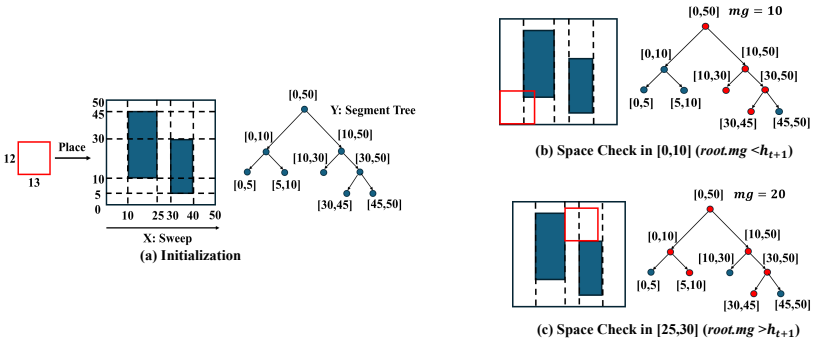


Fig. 5: Illustrate of *checkAvailableSpace*. (Red marks denote the nodes with attribute updates in the segment tree)

Algorithm 1: *checkAvailableSpace*

Input: the target die z_{t+1}^{to} with the outline (W, H), the next block b_{t+1} with the shape (w_{t+1}, h_{t+1}), the set of blocks that are placed on the target die $\mathcal{B}^{to}$
Output: whether there exists space to place the next block b_{t+1} on the target die

1 $y_s \leftarrow \{0, H\}$, $ev_a \leftarrow \{\}$, $ev_s \leftarrow \{\}$, $ev_e \leftarrow \{\}$, $cnt \leftarrow 0$
2 **for** every block blk in $\mathcal{B}^{to}$ **do**
3 $x_1 \leftarrow blk.x$, $x_2 \leftarrow blk.x + blk.w$
4 $y_1 \leftarrow blk.y$, $y_2 \leftarrow blk.y + blk.h$
5 Add y_1, y_2 to y_s
6 Add event $(x_1, 1, y_1, y_2)$ and $(x_2, -1, y_1, y_2)$ to event set ev_a
7 Add $(x_1{:}\{1, y_1, y_2\})$ to starting event map ev_s and $(x_2{:}\{-1, y_1, y_2\})$ to ending event map ev_e
8 **end**
9 Sort and remove the duplicates in y_s and ev_a
10 Construct the segment tree ST based on the y_s
11 **for** *every event* $(x_1^i, l^i, y_1^i, y_2^i)$ *in* ev_a **do**
12 $x_2^i \leftarrow x_1^i + w_{t+1}$
13 **if** x_2^i ¿ W **then**
14 **return False**
15 **end**
16 **for** *every* $\{l^j, y_1^j, y_2^j\}$ *in* $ev_e[x_1^i]$ **do**
17 $ST.update(y_1^j, y_2^j, -1)$ ▷ Remove $[y_1^j, y_2^j]$ from ST
18 **end**
19 **while** cnt ¡ $|ev_a|$ *and* $ev_a[cnt].x_1 < x_2^i$ **do**
20 $\{x_1^j, l^j, y_1^j, y_2^j\} = ev_a[cnt]$
21 **if** $l^j == 1$ **then**
22 $ST.update(y_1^j, y_2^j, 1)$ ▷ Add $[y_1^j, y_2^j]$ to ST
23 **end**
24 $cnt++$
25 **end**
26 **if** $ST.root.mg > h_{t+1}$ **then**
27 **return True**
28 **end**
29 **end**
30 **return False**

which queue it was drawn from. This capability is essential for optimizing performance metrics that are highly sensitive to the die distribution, such as cross-die wirelength.

A critical challenge in the die-swapping mechanism is ensuring that the target die has sufficient contiguous free space to accommodate a new block without causing overlaps. An efficient feasibility check is therefore essential in the die-swapping mechanism. A simple solution that checks all potential grid locations for overlap has a time complexity of $\mathcal{O}(W \times H \times N)$, where W and H are the die dimensions and N is the number of blocks. This becomes computationally prohibitive for large-scale layouts. To overcome this bottleneck, we propose an efficient $\mathcal{O}(N \log N)$ solution based on a sweep-line algorithm and a segment tree data structure [21]. The core idea is to represent the vertical (y-axis) space of the die using a segment tree. As a vertical sweep line moves horizontally (along the x-axis) across the boundaries of placed blocks, we dynamically maintain the segment tree to track the largest contiguous free space. Algorithm 1 details the overall process. Lines 1-10 initialize the segment tree and the event sets for the sweep-line. Lines 11-27 describe the sweeping procedure, which checks for the largest available space in the y-direction and updates the segment tree accordingly.

Each node in the segment tree stores the following attributes for its corresponding y-interval:

- Lazy flag f: An integer indicating if the entire interval is covered ($f > 0$) or uncovered ($f = 0$).
- Max gap mg: The length of the largest contiguous free space within the interval.
- Left gap lg: The length of the contiguous free space starting from the interval's left boundary.
- Right gap rg: The length of the contiguous free space starting from the interval's right boundary.

These interval properties (mg, lg, rg) are maintained efficiently in $\mathcal{O}(\log N)$ time per update through combining the values from a node's left and right children. A die is deemed capable of accommodating a block with height h_{t+1} if the root's mg value is greater than h_{t+1}. Figure 5 provides a visual example of this method. Since the overall algorithm performs $2N$ segment tree operations (one for each block boundary during the sweep), this yields a total time complexity of $\mathcal{O}(N \log N)$.

IV. Experiment

This section presents an experimental evaluation of our proposed RL-based 3D floorplanner with die-swapping mechanism. Our method is validated using benchmarks from two sources: the widely-adopted GSRC floorplanning suite [22] and the OpenROAD open-source flow [23]. All experiments are performed on a Linux workstation equipped with dual Intel Xeon Gold 6426Y processors and an NVIDIA A100-80GB GPU. The floorplanning environment and RL agent are implemented using the PyTorch framework and the Tianshou reinforcement learning library [24].

A. Comparison on GSRC benchmark

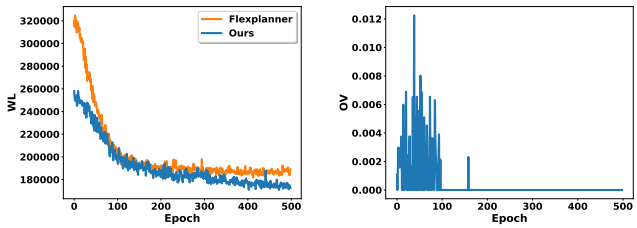
We evaluate performance by comparing the cross-die wirelength (WL) and runtime (RT) on GSRC netlists. The number following 'n' in each netlist name indicates its block count and all blocks are soft blocks with given aspect ratio ranges. Our method is compared against seven baseline approaches: (1) FM+AR: FM partitioning

TABLE I: Comparison on GSRC benchmark (WL: wirelength, RT: runtime(s))

netlist	FM+AR		FM+Li		Spectral + AR		Spectral + Li		GSP		TA3D		FlexPlanner		Ours	
	WL	RT	WL	RT	WL	RT	WL	RT	WL	RT	WL	RT	WL	RT	WL	RT
n10	43223	6.72	48881	2.16	**24571**	5.04	28020	2.08	36444	7.62	41166	41.1	35709	1.75	29779	1.83
n30	84456	13.84	88703	5.68	**67193**	15.68	72048	10	100818	41.86	79429	55.76	98153	3.69	83116	4.25
n50	217731	21.44	219508	18.32	144133	32.56	147835	18.82	174368	109.74	203847	159.06	114154	9.28	**104383**	11.71
n200	880486	130.16	852036	1074.32	446443	115.04	408192	1105.04	397594	1360.86	832867	772.84	346616	31.97	**331361**	45.64
Avg	306474	43.04	302282	275.12	170585	42.08	164024	283.99	177306	380.02	289327	257.19	148658	11.67	**137160**	15.86
Ratio	2.234	2.71	2.204	17.35	1.244	2.65	1.196	17.91	1.293	23.96	2.109	16.22	1.084	0.74	**1**	1

TABLE II: Comparison on OpenROAD benchmark (WL: wirelength, RT: runtime(s))

Netlist	# Blocks		PaOR		FlexPlanner		Ours	
	# HB	#SB	WL	RT	WL	RT	WL	RT
ariane133	127	26	Used for Training					
ariane136	125	27	10887809	2214.22	9154637	69.73	**8796905**	87.14
bp_parrot	9	20	10794121	105.55	8999052	20.99	**8547209**	25.87
bp_multi_top	10	9	5540351	93.16	4809298	17.32	**4723881**	21.04
bp_quad	183	65	206102882	3617.41	175388559	130.15	**169217131**	157.92
Avg			58331291	1507.59	49587887	59.55	**47821282**	72.99
Ratio			1.22	20.65	1.037	0.82	**1**	1



(a) WL (b) OV

Fig. 6: The training curves on netlist $n100$.

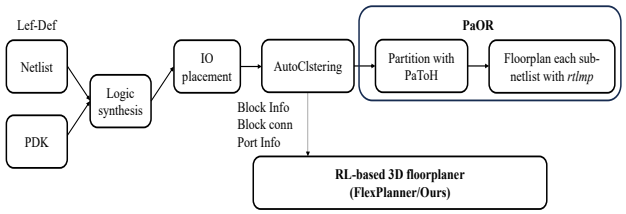


Fig. 7: Construction of the new benchmark and the baseline PaOR in the OpenROAD flow.

[25] combined with the AR 2D floorplanner [26]; (2) FM+Li: FM partitioning with Li's 2D optimizer [27]; (3) Spectral+AR: Spectral partitioning with the AR floorplanner; (4) Spectral+Li: Spectral partitioning with Li's floorplanner; (5) GSP: a Grouped Sequence Pair-based 3D floorplanner [12]; and (6) TA3D: a thermal-aware 3D planner using smoothed HPWL and FM-style optimization [5]. (7) FlexPlanner: the state-of-the-art RL-based 3D floorplanner that uses heuristics for static die assignment [18].

The netlist $n100$ is used to train both the baseline FlexPlanner and our proposed RL-based 3D floorplanner. Figure 6 presents the training curves for wirelength (WL) and overlap (OV) over 500 epochs. As shown in Figure 6a, our method achieves a smaller wirelength than FlexPlanner after 500 epochs (170828 vs. 181794). Furthermore, Figure 6b demonstrates that our agent successfully learns to minimize overlap throughout the training process, effectively converging to a policy that avoids illegal placements.

Table I compares the wirelength and runtime of various 3D floorplanning methods on the GSRC benchmark. For our method and FlexPlanner, we use the agent that are trained on the $n100$ to test on the remaining test cases. Compared with the best non-RL baseline, our proposed RL-based floorplanner can achieve an average reduction of 19.6% in WL. This can be attributed to two aspects: 1) better cooptimization across both dies for our RL-based floorplanner against the floorplanner that partitions the netlist and floorplans each sub-netlist respectively. 2) larger exploration space against the non-RL floorplanner that relies on specific sequence expression like B* tree or sequence pair. Compared with the RL baseline FlexPlanner, our floorplanner can achieve 8.4% WL reductions. This is due to our unique die swapping mechanism that does not rely on the initial die assignment. Both FlexPlanner and our method have a significant speedup over the non-RL approach because of the fast GPU inference. In addition, our approach introduces 26% time overhead compared with the FlexPlanner, which is the computational cost of enabling the die-swapping mechanism.

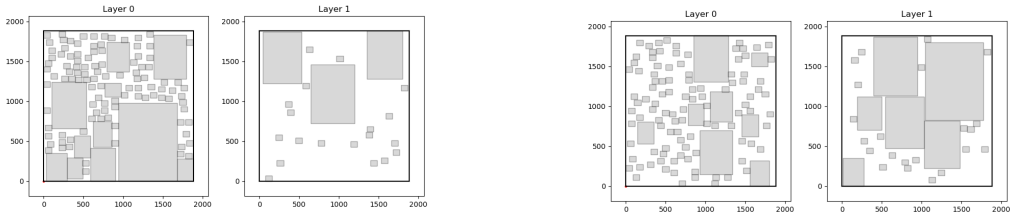
B. Comparison on OpenROAD benchmark

Previous research heavily uses the existing GSRC benchmark, and here we contribute another new floorplanning benchmark[1] generated by the LEF-DEF formatted netlists and OpenROAD flow scripts. The generation process, illustrated in Figure 7, begins with synthesizing netlists using Yosys. OpenROAD is then used for IO placement and autoclustering. The blocks that simply contain hard macros are regarded as *hard blocks*, while the other blocks are regarded as *soft blocks*. The final output includes files detailing block information, inter-block connectivity, and IO port locations. Table II provides 5 representative netlists synthesized with Nangate45 PDK and processed with the OpenROAD flow, where #HB and #SB indicate the number of the hard blocks and soft blocks.

We construct a non-RL baseline termed PaOR. In this approach, the auto-clustered netlist is bipartitioned with PaToH [7], and each sub-netlist is floorplanned respectively by $rtlmp$, the floorplanner in OpenROAD [28]. Using the $ariane133$ for training and the remaining netlists for validation, our floorplanner achieves an average wirelength reduction of 22.0% compared to the PaOR baseline and 3.7% compared to FlexPlanner. Our method introduces 18% time overhead, compared with FlexPlanner.

Figure 8 visualizes the 3D floorplan dynamics for *ariane133* at different training epochs, to demonstrate our agent's improving capability for die distribution optimization. In the early stages of training, the agent places a disproportionate number of blocks on a single die, due to an underdeveloped die swapping policy. As training progresses, the agent successfully learns to leverage the

[1]https://drive.google.com/drive/folders/1uyz2zquupWmWXOxJYuMvDn52754ad_Bf



(a) Epoch 15 (WL=9791091)

(b) Epoch 40 (WL=8176310)

Fig. 8: 3D floorplan dynamics on $ariane133$ at different epochs trained with our RL-based floorplanner with die swapping mechanism.

die-swapping mechanism, resulting in a more balanced and logical block distribution across the two dies. This learned behavior directly correlates with the significant wirelength reduction in the floorplan.

V. Conclusion

This paper introduced a reinforcement learning-based 3D floorplanner with a novel die-swapping mechanism that enables the dynamic co-optimization of block placement and die assignment. Our method overcomes a key limitation of prior approaches by integrating die reassignment directly into the RL action space, supported by an efficient $\mathcal{O}(N \log N)$ spatial checking algorithm for scalability. Experimental results demonstrate that our framework outperforms state-of-the-art floorplanners on the crossdie wirelength, validating the effectiveness of the die-swapping mechanism. This work provides a robust new paradigm for 3D physical design optimization.

References

[1] K.-S. Hu, I.-J. Lin, Y.-H. Huang, H.-Y. Chi, Y.-H. Wu, and C.-F. C. Shen, "2022 iccad cad contest problem b: 3d placement with d2d vertical connections," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–5.

[2] Q. Ma and E. F. Young, "Multivoltage floorplan design," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 29, no. 4, pp. 607–617, 2010.

[3] L. Xiao, S. Sinha, J. Xu, and E. F. Young, "Fixed-outline thermal-aware 3d floorplanning," in *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2010, pp. 561–567.

[4] B. Fu, L. Liu, Y. Sun, W.-H. Lau, M. D. Wong, and E. F. Young, "Coplace: Coherent placement engine with layout-aware partitioning for 3d ics," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 65–70.

[5] J.-M. Lin, W.-Y. Chang, H.-Y. Hsieh, Y.-T. Shyu, Y.-J. Chang, and J.-M. Lu, "Thermal-aware floorplanning and tsv-planning for mixed-type modules in a fixed-outline 3-d ic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 9, pp. 1652–1664, 2021.

[6] P. Vanna-Iampikul, C. Shao, Y.-C. Lu, S. Pentapati, Y. Heo, J.-S. Choi, and S. K. Lim, "Snap-3d: A constrained placement-driven physical design methodology for high performance 3-d ics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2331–2335, 2022.

[7] Ü. Çatalyürek and C. Aykanat, "Patoh (partitioning tool for hypergraphs)," in *Encyclopedia of parallel computing*. Springer, 2011, pp. 1479–1487.

[8] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, "High-quality hypergraph partitioning," *ACM Journal of Experimental Algorithmics*, vol. 27, pp. 1–39, 2023.

[9] C. J. Alpert and S.-Z. Yao, "Spectral partitioning: The more eigenvectors, the better," in *Proceedings of the 32nd annual ACM/IEEE design automation conference*, 1995, pp. 195–200.

[10] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "Specpart: A supervised spectral framework for hypergraph partitioning solution improvement," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[11] L. Cheng, L. Deng, and M. D. Wong, "Floorplanning for 3-d vlsi design," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, 2005, pp. 405–411.

[12] R. K. Nain and M. Chrzanowska-Jeske, "Fast placement-aware 3-d floorplanning using vertical constraints on sequence pairs," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 19, no. 9, pp. 1667–1680, 2010.

[13] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nova *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[14] Y. Lai, Y. Mu, and P. Luo, "Maskplace: Fast chip placement via reinforced visual representation learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 019–24 030, 2022.

[15] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: A new representation for non-slicing floorplans," in *Proceedings of the 37th annual design automation conference*, 2000, pp. 458–463.

[16] H. Murata and E. S. Kuh, "Sequence-pair based placement method for hard/soft/pre-placed modules," in *Proceedings of the 1998 international symposium on Physical design*, 1998, pp. 167–172.

[17] 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 *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[18] R. Zhong, X. Du, S. Kai, Z. Tang, S. Xu, J. Hao, M. Yuan, and J. Yan, "Flexplanner: Flexible 3d floorplanning via deep reinforcement learning in hybrid action space with multi-modality representation," *Advances in Neural Information Processing Systems*, vol. 37, pp. 49 252–49 278, 2024.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[20] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[21] Y.-K. Chang and Y.-C. Lin, "Dynamic segment trees for ranges and prefixes," *IEEE Transactions on Computers*, vol. 56, no. 6, pp. 769–784, 2007.

[22] GSRC Benchmark Suite, http://vlsicad.eecs.umich.edu/BK/GSRCbench/, 2024.

[23] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–4.

[24] J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, Y. Su, H. Su, and J. Zhu, "Tianshou: A highly modularized deep reinforcement learning library," *Journal of Machine Learning Research*, vol. 23, no. 267, pp. 1–6, 2022.

[25] L.-T. Liu, M.-T. Kuo, S.-C. Huang, and C.-K. Cheng, "A gradient method on the initial partition of fiduccia-mattheyses algorithm," in *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*. IEEE, 1995, pp. 229–234.

[26] C. Luo, M. F. Anjos, and A. Vannelli, "Large-scale fixed-outline floorplanning design using convex optimization techniques," in *2008 Asia and South Pacific Design Automation Conference*. IEEE, 2008, pp. 198–203.

[27] W. Li, F. Wang, J. M. Moura, and R. Blanton, "Global floorplanning via semidefinite programming," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[28] A. B. Kahng, R. Varadarajan, and Z. Wang, "Rtl-mp: toward practical, human-quality chip planning and macro placement," in *Proceedings of the 2022 International Symposium on Physical Design*, 2022, pp. 3–11.