Snake-3D: Differentiable Learning for Cross-Tier Logic Path Snaking Optimization in 3D ICs

Yen-Hsiang Huang
School of ECE
Georgia Institute of Technology
Atlanta, Georgia, USA
yhhuang@gatech.edu

Sung Kyu Lim
School of ECE
Georgia Institute of Technology
Atlanta, Georgia, USA
limsk@ece.gatech.edu

***Abstract*—In the post-Moore era, 3D ICs offer superior timing performance and lower power compared to traditional 2D designs. However, current state-of-the-art (SOTA) flows suffer from excessive snaking along critical paths and require many bond pads. This not only complicates production ,but also degrades timing, as each inter-die connection demands a bond pad and 2× BEOL wires. The challenge arises from relying on bin-based FM partitioning, which overlooks inter-bin connections and scales poorly. We introduce Snake-3D, a snaking and timing-aware gradient-based partitioner. It eliminates excessive snaking and concurrently reduces bond pad usage during the partitioning stage, the earliest point these optimizations can be applied. Leveraging GPU acceleration and PyTorch, Snake-3D matches the cutsize quality of advanced gradient methods while running faster on modest hardware. Its integrated snaking- and timing-aware cost, combined with local density control, significantly reduces critical path snaking and achieves superior final PPA compared to GNN-based methods and the default method in SOTA 3D IC flows. On six benchmarks at 7 nm and 28 nm technology nodes, Snake-3D outperforms the default method in SOTA 3D IC flows. It reduces final bond pad by 29%, lowers average and maximum snaking by 34% and 20%, and improves WNS and TNS by 17% and 42%. Compared to GNN-based methods, Snake-3D achieves a 47% reduction in bond pads, reduces snaking by 39% (average) and 30% (maximum), and improves WNS and TNS by 48% and 62%. Snake-3D runs 7.4× faster than GNN-based and similar speed to the default method in SOTA 3D IC flows.**

I. Introduction

As CMOS technology nears its physical limits, 3D IC technology, exemplified by AMD's Ryzen V-Cache [1], is emerging as a promising solution to sustain Moore's Law. Numerous studies have explored 3D Place and Route (PnR) using wire-length estimations and density cost functions [2], [3], [4], [5], adapting 2D tools for 3D [6], [7], and extending commercial methods [8], [9], [10], [11]. Recent heterogeneous 3D placement research [12], [13] has even outperformed ICCAD contest winners of 2022 and 2023.

However, current SOTA design flows predominantly rely on 2D PnR tools, such as commercial solutions like Synopsys ICC2 and Cadence Innovus. These tools, originally developed for 2D designs, lack optimization for 3D architectures.A key challenge in these flows is the excessive snaking that results along critical paths. The inter-die connections that accompany snaking cause timing degradation, and lead to further issues, such as overlapping, due to the size disparity between BEOL vias and F2F bond pads, according to [14]. This ultimately increases the cost and effort required to correct these problems in later PnR stages or may even render them uncorrectable.

Figure 1 shows a side view of a path snaking between dies in a face-to-face (F2F) design. Each inter-die connection requires at least 2X BEOL layers plus a bond-pad . This overhead complicates timing closure in conventional 2D flows and can render critical paths unrecoverable. Eliminating snaking on a critical path can reduces wire

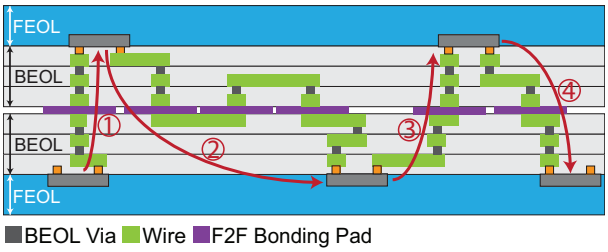

Fig. 1. Example of a path snaking 4 times between dies. Each pair of cells requiring inter-die connections incurs at least 2x BEOL of wires and 1 F2F bond pad, significantly impacting timing performance.

TABLE I
Snaking Reduction Impact: For the most critical path in case Arms' Cortex A7, eliminating snaking improved slack by 19%.

	#Snaking	Slack (ps)	Delay (ps)
Before snaking reduction	10	-190	495
After snaking reduction	0 (-100%)	-154 (-19%)	458

delay and improves slack by up to 19%; Table I demonstrates this gain and underscores the importance of snaking-aware optimization.

Reducing snaking on a single critical path can yield significant timing gains, but focusing on a few paths may exacerbate snaking and timing issues on others due to conflicting routing choices and tangled critical paths. Table II demonstrates a case where we moved the top 100 critical paths to the same die to completely avoid snaking in these paths; however, paths outside this top 100 experienced increased delays, resulting in worse slack than the original worst case, suggesting that a more global-view approach is needed.

In this paper, we propose Snake-3D, a novel global-view approach designed to mitigate snaking during the initial partitioning stage—the earliest point at which snaking can be optimized. To accomplish this, we employ a gradient-based partitioner, leveraging its inherent global-view nature, and meticulously design a loss function that is both snaking- and timing-aware, while also ensuring 3D-IC applicability by incorporating local density constraints. This integrated approach balances conflicting routing decisions across all timing-violated critical paths, effectively minimizing overall snaking and thereby substantially improving timing performance.

In summary, our contributions are:

- We propose Snake-3D, a gradient-based, fast, lightweight, snaking- and timing-aware 3D-IC partitioner.
- Generality: Snake-3D works on any node or design, including fixed macros and I/O pins, without modification.

TABLE II
RELOCATING TOP 100 PATHS TO ONE DIE: WORST PATH IMPROVES, BUT A NEW CRITICAL PATH EMERGES.

	Slack (ps)
Original Worst	-209
Relocated Worst	-193
New Worst (Outside Original Top 100)	-215

- Scalability: On commercial and academic circuit benchmarks with up to 500K cells/nets and 60K+ paths, Snake-3D achieves 30% lower TNS than bin-based methods and X% improvement over advanced GNN approaches such as [15].
- Snake-3D runs on PyTorch with minimal GPU requirements. On our largest 500K cells/nets benchmark it finishes in 15 minutes on an entry-level T4 GPU, 2.4× faster than bin-based FM and 14.8× faster than GNN-based methods, while delivering superior results across all metrics.

II. PREVIOUS WORKS

SOTA 3D IC flows such as [16], [8] and [10], [11] employ 2D PnR tools to generate a 2D pseudo chip, which is then partitioned into two dies for 3D PnR. A downside of these approaches is the introduction of excessive snaking on timing-critical paths that span both dies and require vertical traversal (Figure 1). Furthermore, because 2D tools optimize timing only within each die and offer minimal inter-die optimization, resolving snaking after partitioning becomes challenging. These flows also rely on bin-based partitioning [17] methods such as FM min-cut[18], which ignore inter-bin connections and thus exacerbate snaking.

Circuits are naturally represented as hypergraphs, and hypergraph partitioning is a well-studied field. Traditional CPU-driven partitioners (hMETIS [19], KaHyPar [20], PaToH [21], Zoltan [22], Scotch [23]) use multilevel methods but may converge to local optima due to limited global perspective, as noted in SpecPart [24], which refines solutions using spectral information. In recent years, GPU driven methods have emerged. TP-GNN [15] for 3D ICs uses GNN to partition tiers and optimize timing and hierarchy. MedPart [25] uses GPUs to optimize a differentiable cost function via gradient descent (GD) within an evolutionary multilevel framework, reducing dependence on the initial solution. Both methods incur longer runtimes and higher GPU requirements. Both support cold-start partitioning; MedPart also refines initial solutions.

Only bin-based methods [17] and TP-GNN [15] target 3D ICs with timing or density constraints. General partitioners such as KaHyPar, PaToH, Scotch, SpecPart and MedPart perform generic graph or hypergraph partitioning and cannot be applied directly to 3D-IC designs. Figure 2 shows that a generic partitioner without local density constraints creates large cell clusters, complicating legalization and harming timing, whereas a 3D-IC-aware partitioner with density constraints yields a uniform distribution and preserves better timing. Table VI summarizes each method's approach, hardware requirements, and 3D-IC applicability.

III. SNAKE-3D ALGORITHM

A. Overview of the Algorithm

Snake-3D resolves prior limitations while retaining key strengths. Like MedPart [25], it uses a gradient-based method that supports cold-start partitioning and leverages GPU parallelism. As a 3D-IC-ready partitioner, Snake-3D incorporates timing, snaking, and local

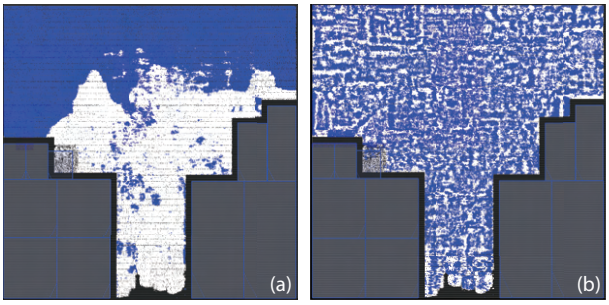

Fig. 2. 3D-IC partitioning: (a) w/o local density constraints, cells (blue/white: top/bottom die) cluster unevenly, increasing legalization effort and degrading timing; (b) with local density constraints, cells are distributed evenly.

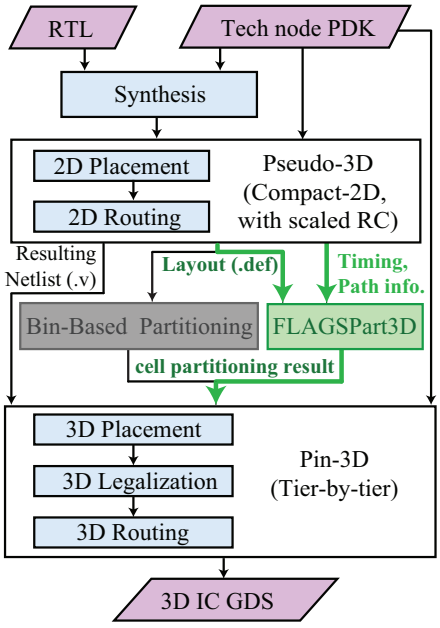

Fig. 3. Our Snake-3D flow integrated into the state-of-the-art 3D-IC flow, Pin-3D [10]. While Pin-3D originally employs bin-based partitioning, our advanced Snake-3D seamlessly replaces that component.

density constraints beyond just cutsize. Gradient descent naturally offers a global view of the design, and Snake-3D is designed to be lightweight and fast, requiring minimal GPU resources to ensure broad applicability.

The workflow of Snake-3D and its integration into the SOTA 3D IC design flow are shown in Figure 3. SOTA flows, such as those in [16], [8], and [10], consist of three stages: (1) a pseudo-3D stage that modifies RC parasitics or library files to adapt 2D tools, producing a flattened 3D IC as a pseudo-chip; (2) partitioning based on this pseudo-3D layout; and (3) actual 3D IC PnR. Pin-3D, the latest flow, uses Compact-2D as its default pseudo-3D stage and bin-based partitioning as its default partitioner. Throughout our experiments, we adopt the Pin-3D flow [10] with its default bin-based partitioning [17] as our baseline and conduct evaluations within the Pin-3D framework.

A detailed description of the Snake-3D algorithm is provided in Algorithm 1, with notations listed in Table IV. The total differentiable loss used in GD is:

$$L_{total} = W_c L_c + W_s L_s + W_d L_d,$$

where L_c, L_s, and L_d represent the losses for cutsize (#3D-nets),

Algorithm 1 Snake-3D: GD-based 3D-IC Partitioning

Input: $\mathcal{N}, \mathcal{V}^0, \mathcal{V}_f^0, \mathcal{P}, \mathcal{G}_0, \mathcal{G}_1$ **Output:** $\mathbf{z}^0$: Discrete cell Z-coordinate (0 or 1) at level 0

function PREPROCESS($\mathcal{N}, \mathcal{V}, \mathcal{V}_f, \mathcal{P}, \mathcal{G}_0, \mathcal{G}_1$)
  $\mathbf{w}_\mathcal{V} \leftarrow \text{InitWeight}(\mathcal{V})$ ▷ Vertex weight: typically the area of cells.
  $\mathbf{w}_\mathcal{N} \leftarrow \text{InitWeight}(\mathcal{N})$ ▷ Net weight: uniform, as timing info. is handled by path snaking loss.
  $\mathbf{w}_\mathcal{P} \leftarrow \text{InitWeight}(\mathcal{P})$ ▷ Path weight: determined by timing criticality.
  $\mathcal{A} \leftarrow \text{GetTimingArcs}(\mathcal{P})$
  $\mathbf{w}_\mathcal{A} \leftarrow \text{InitWeight}(\mathcal{A}, \mathbf{w}_\mathcal{P})$ ▷ Arc weight: determined by the weight of its associated path.
  $\mathbf{C^0} \leftarrow \text{InitConnectivity}(\mathcal{V}, \mathcal{N}, \mathcal{A})$ ▷ $\mathbf{C^0}$: connectivity matrix ($|\mathcal{V}^l| \times |\mathcal{N} \cup \mathcal{A}|$) at level 0.
  $\mathbf{G^0} \leftarrow \text{InitGrid}(\mathcal{V}, \mathbf{w}_\mathcal{V}, \mathcal{G}_0, \mathcal{G}_1)$ ▷ $\mathbf{G^0}$: vertex weight to each grid cell, alreagy contains $\mathbf{w}_\mathcal{V}$ information.
  return $\mathbf{C^0}, \mathbf{G^0}, \mathbf{w}_\mathcal{N}, \mathbf{w}_\mathcal{A}$

function GD_OPTIMIZE($\mathcal{V}^l, \mathbf{C^l}, \mathbf{G^l}, \mathbf{w}_\mathcal{N}, \mathbf{w}_\mathcal{A}$)
  if $|\mathcal{V}^l| > coarse\_threshold$ **then** ▷ Number of cells at level l, derived from the number of rows of $\mathbf{C^l}$.
    $k_{target} \leftarrow |\mathcal{V}^l| \times coarse\_ratio$
    $\mathcal{V}^{l+1}, \mathbf{H^l} \leftarrow \text{Coarsen}(\mathcal{V}^l, \mathbf{C^l}, k_{target})$ ▷ $\mathbf{H^l}$: assignment matrix mapping fine nodes to super-nodes.
    $\mathbf{C^{l+1}} \leftarrow \mathbf{C^l}\,\mathbf{H^l};\ \mathbf{G^{l+1}} \leftarrow \mathbf{G^l}\,\mathbf{H^l}$ ▷ Coarsened connectivity matrix and grid assignment matrix.
    $\mathbf{t^{l+1}} \leftarrow \text{GD\_Optimize}(\mathcal{V}^{l+1}, \mathbf{C^{l+1}}, \mathbf{G^{l+1}}, \mathbf{w}_\mathcal{N}, \mathbf{w}_\mathcal{A})$
    $\mathbf{t^l} \leftarrow \mathbf{t^{l+1}}\,\mathbf{H^{l}}^T$ ▷ Interpolate coarse solution to level l.
  else
    $\mathbf{t^l} \leftarrow \text{InitT}(\mathbf{t^l}, \mathbf{G^l})$ ▷ Use fast initialization heuristics: start with a 50/50 area split for each die.
  $best\_loss \leftarrow \infty;\ \mathbf{best\_t^l} \leftarrow \mathbf{t^l}$
  $\mathbf{t^l} \leftarrow \mathbf{t^l} \times scaling\_factor$ ▷ Scale $\mathbf{t^l}$ at the beginning to prevent it from getting stuck at extreme values.
  for $step = 0$ to $num\_steps - 1$ **do**
    $\widetilde{\mathbf{z}}^\mathbf{l} \leftarrow \sigma(\mathbf{t^l})$ ▷ Relaxed, continuous z-values in [0, 1].
    $L_{total} \leftarrow W_c L_c(\widetilde{\mathbf{z}}^\mathbf{l}, \mathbf{C^l}, \mathbf{w}_\mathcal{N}) + W_s L_s(\widetilde{\mathbf{z}}^\mathbf{l}, \mathbf{C^l}, \mathbf{w}_\mathcal{A}) + W_d L_d(\widetilde{\mathbf{z}}^\mathbf{l}, \mathbf{G^l})$
    $\mathbf{t^l} \leftarrow AdamOptimizer(\eta, \mathbf{t^l}, \nabla L_{total})$ ▷ Update $\mathbf{t^l}$ using the gradient and Adam Optimizer.
    if $total\_loss < best\_loss$ **then**
      $best\_loss \leftarrow total\_loss;\ \mathbf{best\_t^l} \leftarrow \text{clone}(\mathbf{t^l})$
  return $\mathbf{best\_t^l}$

$(\mathbf{C^0}, \mathbf{G^0}, \mathbf{w}_\mathcal{N}, \mathbf{w}_\mathcal{A}) \leftarrow \text{Preprocess}(\mathcal{N}, \mathcal{V}^0, \mathcal{V}_f^0, \mathcal{P}, \mathcal{G}_0, \mathcal{G}_1)$
$\mathbf{t^0} \leftarrow \text{GD\_Optimize}(\mathcal{V}^0, \mathbf{C^0}, \mathbf{G^0}, \mathbf{w}_\mathcal{N}, \mathbf{w}_\mathcal{A})$
$\widetilde{\mathbf{z}}^\mathbf{0} \leftarrow \sigma(\mathbf{t^0})$ ▷ σ is the Sigmoid function.
$\mathbf{z^0} \leftarrow \text{Snap}(\widetilde{\mathbf{z}}^\mathbf{0})$ ▷ Discrete z-values: 0 if $\widetilde{z} < 0.5$, 1 if $\widetilde{z} \geq 0.5$.
return $\mathbf{z^0}$

TABLE III
IMPACT OF LOSS COMPONENTS ON DIFFERENT INSTANCE TYPES.

Loss	Main Target	Affected Instance
L_c	Net cutsize (3D net count) minimization	gates, FFs, clock instances
L_s	Critical logic path snaking minimization	gates, FFs
L_d	Local density violation minimization	gates, FFs, clock instances

snaking of critical paths, and local density, respectively. The impact of each loss component on different instance types is summarized in Table III. The weights W_c, W_s, and W_d are user-configurable and should be chosen based on the input net and path counts to balance the terms and prevent any one of them from dominating. Each term is discussed in the following subsections.

B. Differentiable Cutsize Loss: the Definition

The 3D IC partitioning problem seeks to minimize the cutsize net cost. A net is considered to have a cut if it connects cells on both the top and bottom dies, thereby increasing the cutsize cost. For a net $n \in \mathcal{N}$, the cost is defined as:

$$\text{cutsize}(n) = \begin{cases} 1 & \text{if } \exists (v_i, v_j) \in n \text{ s.t. } z(v_j) \neq z(v_{j+1}), \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

TABLE IV
NOTATIONS USED IN SNAKE-3D ALGORITHM. VARIABLES WITH SUPERSCRIPT l ARE LEVEL-SPECIFIC AND AFFECTED BY COARSENING.

Symbol	Description and Role
$\mathcal{V}^l, \mathcal{V}_f^l$	Set of vertex and, fixed vertex at level l. $\mathcal{V}^0$ mean all cells. $\mathcal{V}_f$ includes fixed I/O and memory macros.
$\mathcal{N}, \mathcal{P}, \mathcal{A}$	Set of nets, timing paths and their arcs; timing arcs can be derived from the paths.
$\mathcal{G}_0, \mathcal{G}_1$	Grid cells on bottom/top dies with user-defined dimensions.
$\mathbf{z}^l, \widetilde{\mathbf{z}}^l$	Discrete (0/1) and relaxed ($[0, 1]$) z-coordinates at level l for each cell $v \in \mathcal{V}^l$, for die assignments and relaxations.
$\mathbf{t^l}$	Tensor at level l; primary variable operated on by GD solver, with $\sigma(\mathbf{t^l}) = \widetilde{\mathbf{z}}^\mathbf{l}$ (1-to-1 mapping), where σ is sigmoid.
$\mathbf{w}_\mathcal{V}, \mathbf{w}_\mathcal{N}$	Weights for vertices (e.g., cell area) and nets. we use a uniform net weight; timing information is inserted by the snaking loss.
$\mathbf{w}_\mathcal{P}, \mathbf{w}_\mathcal{A}$	Weights for paths and arcs. $\mathbf{w}_\mathcal{P}$ is defined by Equation 17, and $\mathbf{w}_\mathcal{A}$ is derived from $\mathbf{w}_\mathcal{P}$.
$\mathbf{C^l}$	Connectivity matrix ($|\mathcal{V}^l| \times |\mathcal{N} \cup \mathcal{A}|$) at level l; defines net-vertex connections at level l.
$\mathbf{G^l}$	Grid assignment matrix at level l; maps vertex weight to corresponding grid cells.
$\mathbf{H^l}$	Assignment matrix at level l; maps fine vertices ($\mathcal{V}^l$) to coarsened vertices ($\mathcal{V}^{l+1}$) during coarsening.

where $z(v_i) \in [0,1]$ is the z-coordinate of vertex v_i, The total weighted cutsize cost is given by

$$C_{\text{total-cut}} = \sum_{n \in \mathcal{N}} w(n) \cdot \text{cutsize}(n), \tag{2}$$

where the weight $w(n)$ is user-defined and can be based on timing, power, or other metrics. In this work, we use unit weights throughout, as timing information will be incorporated in the subsequent snaking-aware cost function section subsection III-D.

Since $z(v)$, the z-coordinate of a vertex (cell or coarsened super-cell), is discrete (0 or 1) and unsuitable for differentiation, we introduce a relaxed variable $\widetilde{z}(v)$ to enable a differentiable cost function. We select the sigmoid function, which is continuous and bounded in $[0,1]$, defining: $\sigma(\mathbf{t}) = \widetilde{\mathbf{z}}$, where $\mathbf{t}$ is the tensor optimized by GD-based Snake-3D, and $\widetilde{\mathbf{z}}$ has a one-to-one correspondence with $\mathbf{t}$. Beyond its continuity and bounded range, the sigmoid's vanishing gradient, typically a drawback in deep learning when $\mathbf{t}$ is extremely small or large, proves advantageous here. As $\mathbf{t}$ approaches extreme values, $\widetilde{\mathbf{z}}$ nears 0 or 1, and the diminished gradient stabilizes these near-binary outcomes, benefiting our optimization. In the following section, we focus on $\widetilde{\mathbf{z}}$ rather than $\mathbf{t}$ for simplicity and convenience.

A relaxed, differentiable cost function $\widetilde{\text{cutsize}}(n)$ aims to achieve two objectives: (1) binarization: drive all $\widetilde{\mathbf{z}}$ values toward binary outcomes (0 or 1); the first and second proxies in [25] are designed for this. $-\left(\prod_{v_i \in n} \widetilde{z}(v_i) + \prod_{v_i \in n}(1-\widetilde{z}(v_i))\right)$, and the second as Entropy $\left(\text{Mean}_{v_i \in n} \widetilde{z}(v_i)\right)$. (2) convergence: encourage all $\widetilde{\mathbf{z}}$ values in a net to align. The first and second proxies from [25] support this implicitly, while the third and fourth are explicitly designed for it. The third proxy is given by MSE $\left(\text{Mean}_{v_i \in n} \widetilde{z}(v_i)\right)$, and the fourth by $\max_{v_i \in n} \widetilde{z}(v_i) + \max_{v_i \in n}(1-\widetilde{z}(v_i))$. Although the approach of augmenting 4 proxies in [25] gives users flexibility to tune their weights, achieving an optimal balance remains challenging—even with a Bayesian tuner. Moreover, the added complexity increases the computational cost of backpropagation. To address these issues, we derive a single unified equation that meets our requirements.

We begin with the observation Equation 1 can be expressed as

$$\text{cutsize}(n) = \left(1 - \min_{v_i \in n} z(v_i)\right) \cdot \max_{v_i \in n} z(v_i), \tag{3}$$

which equals 0 only when $\max_{v_i \in n} z(v_i) = 0$ or $\min_{v_i \in n} z(v_i) = 1$, implying that either $\forall v_i \in n, z(v_i) = 0$ or $\forall v_i \in n, z(v_i) = 1$, respectively. However, this formulation is only suited to the discrete domain. For a function $\widetilde{z}(v_i)$, the $\min$ and $\max$ operations are not differentiable. To address this, we adopt the smooth maximum function, LogSumExp (LSE), which provides a close, differentiable approximation of the $\max$ and $\min$ function and integrates seamlessly into tensor operations. The LSE max and min functions are defined as

$$\text{LSE-max}\{z_1, z_2, \ldots, z_n\} = \frac{1}{\alpha} \log\left(\sum_i e^{\alpha z_i}\right), \tag{4}$$

$$\text{LSE-min}\{z_1, z_2, \ldots, z_n\} = -\frac{1}{\alpha} \log\left(\sum_i e^{-\alpha z_i}\right), \tag{5}$$

where $\alpha > 0$ controls the smoothness of the approximation. Thus, Equation 3 can be reformulated into a differentiable closed form:

$$\widetilde{\text{cutsize}}(n) = \left(1 - \underset{v_i \in n}{\text{LSE-min}}\{\widetilde{z}(v_i)\}\right) \cdot \underset{v_i \in n}{\text{LSE-max}}\{\widetilde{z}(v_i)\} \tag{6}$$

and the corresponding differentiable loss function of Equation 2 is:

$$L_{\text{c}} = \sum_{n \in \mathcal{N}} w(n) \cdot \widetilde{\text{cutsize}}(n), \tag{7}$$

C. Differentiable Cutsize Loss: Behavior Analysis

In this section, we explain why our cost function $\widetilde{\text{cutsize}}(n)$ can both encourage all $\widetilde{z}(v_i)$ in a network to converge to a common value and drive them toward the binary values 0 and 1. The gradient of Equation 6 is complex due to the intricate interplay of the LSE function with other terms. However, insight emerges by computing the partial derivatives with respect to LSE-max and LSE-min and applying the chain rule, given by:

$$\frac{\partial \text{LSE-max}\{\widetilde{z}(v_i)\}}{\partial \widetilde{z}(v_j)} = \frac{e^{\alpha \widetilde{z}(v_j)}}{\sum e^{\alpha \widetilde{z}(v_i)}}, \tag{8}$$

$$\frac{\partial \text{LSE-min}\{\widetilde{z}(v_i)\}}{\partial \widetilde{z}(v_j)} = \frac{e^{-\alpha \widetilde{z}(v_j)}}{\sum e^{-\alpha \widetilde{z}(v_i)}}, \tag{9}$$

When $\widetilde{z}(v_i)$ values are spread out, only the maximum $\widetilde{z}(v_i)$ significantly affects LSE-max, and only the minimum affects LSE-min:

$$\frac{\partial \text{LSE-max}\{\widetilde{z}(v_i)\}}{\partial \widetilde{z}(v_j)} \approx \begin{cases} 1 & \text{if } v_j = \arg\max_{v_i \in n} \widetilde{z}(v_i), \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

$$\frac{\partial \text{LSE-min}\{\widetilde{z}(v_i)\}}{\partial \widetilde{z}(v_j)} \approx \begin{cases} 1 & \text{if } v_j = \arg\min_{v_i \in n} \widetilde{z}(v_i), \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

Given the low correlation between LSE-max and LSE-min when $\widetilde{z}(v_i)$ are spread out, the partial derivatives of $\widetilde{\text{cutsize}}(n)$ are:

$$\frac{\partial \widetilde{\text{cutsize}}(n)}{\partial \text{LSE-max}} \approx 1 - \text{LSE-min}, \quad \frac{\partial \widetilde{\text{cutsize}}(n)}{\partial \text{LSE-min}} \approx -\text{LSE-max}. \tag{12}$$

Combining Equation 10, 11, and 12, we get:

$$\frac{\partial \widetilde{\text{cutsize}}(n)}{\partial \widetilde{z}(v_j)} \approx \begin{cases} 1 - \text{LSE-min} & \text{if } v_j = \arg\max_{v_i \in n} \widetilde{z}(v_i), \\ -\text{LSE-max} & \text{if } v_j = \arg\min_{v_i \in n} \widetilde{z}(v_i), \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

Thus, when $\widetilde{z}(v_i)$ values are spread out, $\nabla\widetilde{\text{cutsize}}(n)$ pushes the maximum $\widetilde{z}(v_j)$ toward 0 in proportion to the distance of the minimum from 1, and pushes the minimum toward 1 in proportion to the distance of the maximum from 0, gradually aligning all $\widetilde{z}(v_i)$ to a common value z as the spread narrows. After several iterations, $\forall v_i \in n$, $\widetilde{z}(v_i) \approx \bar{z}$. Then, Equation 6 simplifies to:

$$\widetilde{\text{cutsize}}(n) \approx \left(1 - \left(\bar{z} - \frac{\log n}{\alpha}\right)\right) \cdot \left(\bar{z} + \frac{\log n}{\alpha}\right). \tag{14}$$

For all $v_i \in n$, the partial derivative becomes:

$$\frac{\partial \widetilde{\text{cutsize}}(n)}{\partial \widetilde{z}(v_j)} \approx \frac{\partial \widetilde{\text{cutsize}}(n)}{\partial z} \approx 1 - 2\bar{z}. \tag{15}$$

This derivative changes sign at $z = 0.5$, which aligns with our goal: when $z > 0.5$, it drives $\widetilde{z}(v_i)$ toward 1, and when $z < 0.5$, toward 0, facilitating binary convergence.

In summary, Equation 6 quickly aligns all $\widetilde{z}(v_i)$ for $v_i \in n$ to a common value $\bar{z}$, then shifts it toward 0 or 1 depending on its magnitude. This achieves both convergence and binarization, which is the primary objective of the cutsize cost function discussed in subsection III-B, without auxiliary terms. Equation III-C demonstrates this behavior on high degree nets with eight cells, compared with the first and second proxy cost functions from [25]. While the first proxy eventually achieves convergence, it only does so after binarization. The second proxy has even more limited impact on convergence.

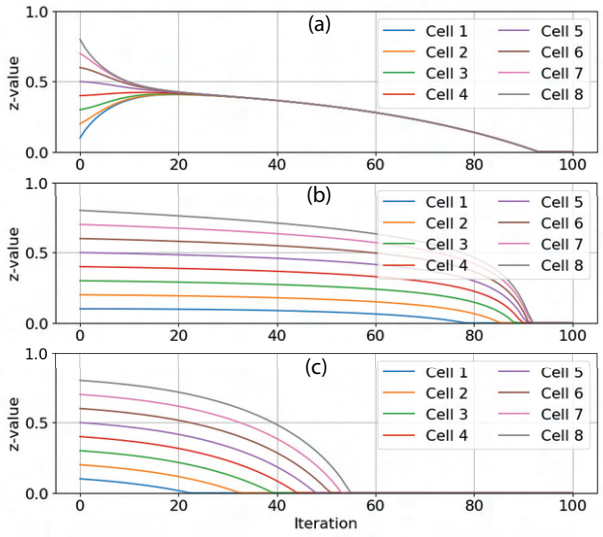

Fig. 4. Convergence of $\widetilde{z}(v_i)$ over iterations for an 8-cell net: (a) Our cost function rapidly aligns $\widetilde{z}(v_i)$ to a uniform value and then shifts it to 0 or 1; (b) first proxy cost function; (c) second proxy cost function—both from [25].

D. Differentiable Snaking Loss

To prevent path snaking, we extract a specified number of violated paths set $\mathcal{P}$. The k-th path, denoted $p_k = (v_{k_1}, v_{k_2}, \ldots, v_{k_n})$, is an ordered sequence of vertices, where each v_{k_i} represents a cell with global index k_i, and a_{k_i} denotes the timing arc between v_{k_i} and $v_{k_{i+1}}$. These arcs form a sequence of two-cell connections. The snaking of a single path is then defined as:

$$\text{snaking}(p_k) = \sum_{i=1}^{|p_k|-1} \text{cutsize}(a_{k_i}), \tag{16}$$

where $\text{cutsize}(a_{k_i}) = |z(v_{k_i}) - z(v_{k_{i+1}})|$.

To aggregate snaking across all paths, each path p_k is assigned a timing-based weight, given by:

$$w(p_k) = \max\left(1, \frac{t_{\text{clk}} - \text{slack}(p_k)}{t_{\text{clk}}}\right), \tag{17}$$

where t_{clk} is the clock period. This ensures a minimum weight of 1, with weights increasing linearly for paths with greater negative slack relative to t_{clk}. The total discrete snaking cost is then:

$$C_{\text{total-snaking}} = \sum_{p\in\mathcal{P}} w(p) \cdot \text{snaking}(p) \tag{18}$$

$$= \sum_{p\in\mathcal{P}} (w(p) \sum_{i=1}^{|p|-1} \text{cutsize}(a_{p_i})) = \sum_{a\in\mathcal{A}} w(a) \cdot \text{cutsize}(a), \tag{19}$$

where $w(a) = \sum_{k:a\in p_k} w(p_k)$, and $\mathcal{A}$ is the set of all unique timing arcs across all paths. This formulation aligns with the framework used for the net cutsize cost in Equation 2, and can be interpreted as a netlist comprising all two-degree edges, making it easily integrable into our GD framework. The resulting differentiable snaking loss function is then:

$$L_{\text{s}} = \sum_{a\in\mathcal{A}} w(a) \cdot \widetilde{\text{cutsize}}(a), \tag{20}$$

E. Differentiable Local Density Loss

For 3D IC partitioning, since the XY coordinates are determined after the pseudo-3D stage, we must constrain the maximum density of each grid cell to ensure balanced cell distribution across dies, preventing overcrowding and maintaining design feasibility. As stated in line 7 of Algorithm 1, we precompute each cell's grid assignment, denoted as the matrix G, a vertex-to-grid weight mapping (e.g., cell area to grid cell). For each grid, we penalize only overflow; the differentiable local density loss is:

$$L_{\text{d}} = \sum_{g\in\mathcal{G}_0} \text{ReLU}\left(\sum_{v_i\in\mathcal{V}^{\updownarrow}} (1 - \widetilde{z}(v_i)) \cdot G[v_i, g] - d_g\right) + \sum_{g\in\mathcal{G}_1} \text{ReLU}\left(\sum_{v_i\in\mathcal{V}^{\updownarrow}} \widetilde{z}(v_i) \cdot G[v_i, g] - d_g\right), \tag{21}$$

where L_d is the loss of local density, $\mathcal{G}_0$ and $\mathcal{G}_1$ are the sets of grid-cells on the die 0 and 1, respectively, d_g is the local density constraint of all grids.

F. Multi-Level Support

Snake-3D employs a multi-level approach. Lines 11–17 in Algorithm 1 recursively call the coarsening part of our algorithm. Given an assignment matrix H^l mapping fine vertices to coarsened vertices, fast matrix operations enable recursive coarsening of the GD function and tensor expansion using H^l (Lines 14–16). The coarsened higher level $\mathbf{t}^{l+1}$ initializes the states of the lower level $\mathbf{t}^l$, consistent with traditional partitioners such as [19], [20], [21], [22], [23] and MedPart [25]. Due to the global-view nature of the GD method across all levels, we do not need to coarsen to extremely small sizes in our experiments, unlike [19]; a size of 10,000 is sufficient. Our implementation adopts the Heavy-Edge Matching method from [19], although other coarsening methods can be seamlessly integrated.

G. Computational Load Analysis

The most memory-demanding element in the algorithm is the connectivity matrix $\mathbf{C}$, which stores net-vertex connections with a dense space complexity of $|\mathcal{N}|\times|\mathcal{V}|$, where $|\mathcal{N}|$ is the number of nets and $|\mathcal{V}|$ is the number of cells. In VLSI, $\mathbf{C}$ is sparse, as most cells are not connected to most nets. Using a sparse matrix representation, the space complexity reduces to $\sum_{n\in\mathcal{N}} \deg(n) = \sum_{v\in\mathcal{V}} \deg(v)$, where $\deg(v)$ is the average fan-in/fan-out per cell (typically under 5, per the technology node library), making the space complexity $O(|\mathcal{V}|)$. This implies the linear scalability of Snake-3D.

IV. Experimental Results

A. Experiment Settings

Experiments were run on a server with an Intel Xeon Gold 6454S CPU using Cadence Innovus v21.14 with 16 threads. GPU tasks were executed on an entry-level NVIDIA T4 (16GB GDDR6) via Google Colab. We used the ADAM optimizer [26] for GD-based optimization. The following circuits are designed for logic-on-logic (LoL) partitioning:

- **Commercial Processors**: ARM Cortex-A7 (200K gates) and A35 (500K gates), both without L2 cache.
- **Academic Research Processor**: OpenPiton [27], 1-core, 64KB L3 cache, 300K gates.

Since the 7nm node lacks memory support, memories are scaled from a commercial 16nm node based on contacted poly pitch (CPP) and standard cell height. Bond pad pitch is critical to this experiment. The current smallest pitch for wafer-to-wafer (W2W) hybrid bonding is 0.4 µm, as reported by IMEC in [28], while other works, such as [29], suggest that a 0.2 µm pitch will be available within two years. Thus, for the 28nm design, we assume a bond pad pitch of 1 µm,

TABLE V

CASE-BY-CASE STATISTICS OF INTER- VS. SINGLE-DIE CONNECTION DELAY DIFFERENCES FOR ARM CORTEX A7, A35, AND OPENPITON[27] (64KB L3, 1 CORE). INCLUDES AVERAGE CELL DELAY FOR REFERENCE.

Tech. Node / Bond-Pad Pitch	28nm / 1.0µm			7nm / 0.2µm		
Design	A7	A35	OP64	A7	A35	OP64
#Single-Die Connection (K)	223	587	794	277	701	921
#Inter-Die Connection (K)	199	509	750	203	471	921
Single-Die Conn. Delay (ps)	0.6	0.4	0.8	1.1	1.0	2.1
Inter-Die Conn. Delay (ps)	2.9	3.3	3.5	5.4	5.4	8.7
Inter- vs. Single-Die Conn. (ps)	2.3	2.9	2.7	4.3	4.4	6.6
Avg. Cell Delay (ps)	23.0	25.1	29.1	16.8	20.4	23.9

and for the 7nm design, 0.2 µm. In both cases, the bond pad width is set to half the pitch. For bond pad parasitics, according to extraction results, the 1µm pitch bond pads in the 28nm design have a resistance of 0.045Ω and a mean capacitance of 0.22fF. A bond pad with a 0.2µm pitch exhibits a resistance of 0.9Ω and capacitance of 0.21fF.

B. State-of-the-Art Used for PPA Comparison

Table VII and Table VIII present the PPA results of our Snake-3D optimizer compared to the SOTA Pin-3D default bin-based FM partitioner and the GNN-based unsupervised TPGNN [15], using 2D as a reference. As noted in Table VI, other partitioners are not applicable to 3D ICs because they do not consider timing nor local density, and are therefore excluded from PPA comparisons. For PPA comparison, we report metrics at both post-3D-placement and post-3D-routing stages, along with additional metrics indicating design scale. For partitioners not applicable to PPA, we present pure cutsize comparisons in Table XI.

C. Snaking Overhead Analysis

As mentioned in Figure 1, snaking incurs a cost of at least two BEOL layers and one bond pad. In addition to this apparent RC cost, implicit costs may arise from bond pad legalization or routing detours. This issue worsens as the ratio of bond pad size to wire size increases in more advanced nodes. When bond pads are relatively large, nearby nets may require rerouting to accommodate them, leading to more severe detours and increased snaking overhead. To quantify this overhead, we conducted a case-by-case analysis, as shown in Table V. Our results reveal that the delay difference between inter-die and single-die connections typically ranges from 2–7 ps, reflecting the overhead of a single snaking instance. The example in Table I, implemented on 28nm, further supports this: removing 10 snaking instances reduces delay by 37 ps (approximately 3.7 ps per snaking), consistent with our observation in Table V.

D. PPA Results: Post Placement

Snake-3D outperforms Pin-3D [10] across all metrics in 28 nm. Compared to TPGNN [15], it shows a slight drop in TNS and violated paths (1% and 3%) but significantly improves all other metrics. In 7 nm, Snake-3D consistently surpasses both baselines with even greater gains. Since routing is incomplete at the placement stage, timing metrics (including violated paths) are based on early global estimates, and bond-pad counts are unavailable. Instead, we report 3D-net count, defined as nets spanning both dies. Snake-3D reduces snaking by 37% on average and 26% on the worst path compared to Pin-3D, and by 18% and 36% versus TPGNN. We also observed that TPGNN performs better than the bin-based baseline in this stage, as it incorporates timing information into its model, whereas bin-based methods do not. Overall, Snake-3D delivers consistent improvements in timing, snaking, and bond-pad usage over both baselines.

TABLE VI

COMPARISON OF PARTITIONING APPROACHES AND KEY ATTRIBUTES. NOTE THAT ONLY BIN-BASED [17], TP-GNN [15], AND OUR SNAKE-3D ARE READY FOR 3D-IC PARTITIONING OUT OF THE BOX.

Method	Key Approach	HW. Demand	Init. Sol.	Metric Used	3D-IC Ready
hMETIS [19]	Multi-level, Lightweight, Fast	Low (CPU)	No	cutsize	No
Bin-Based [17]	Bin based FM, placement-driven	Low (CPU)	No	PPA& cutsize	Yes
TP-GNN [15]	GNN based, unsupervised	Mid (GPU)	No	PPA& cutsize	Yes
SpecPart [24]	GNN based, supervised	Mid (CPU)	Yes	cutsize	No
MedPart [25]	Evolutionary, Gradient-based	High (GPU)	N/Y	cutsize	No
Snake-3D	Lightweight, Fast, Gradient-based	Low (GPU)	No	PPA& cutsize	Yes

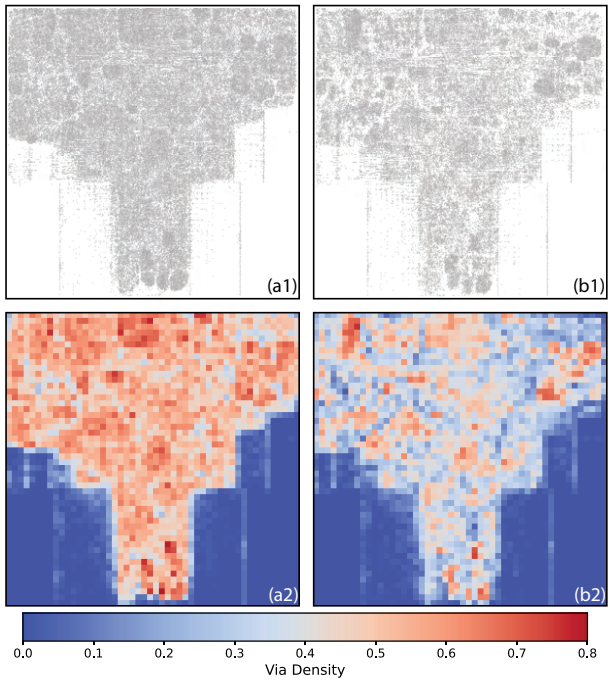

Fig. 5. Post-route bond-pad comparison for ARM Cortex-A7. (a) Bin-based: (a1) bond-pad locations, (a2) density map. (b) Snake-3D: (b1) bond-pad locations, (b2) density map. Snake-3D reduces bond-pad usage by 35%.

E. PPA Results: Post Routing

Table VII and Table VIII show post-route results, where bond pads are finalized. While bond-pad gains slightly decrease compared to the 3D-net count in post-placement, Snake-3D still achieves 31% and 26% fewer bond pads than Pin-3D, and 60% and 34% fewer than TPGNN, in 28 nm and 7 nm, respectively. Average snaking also drops by 25% and 33% in 28 nm, and 43% and 44% in 7 nm, compared to Pin-3D and TPGNN. Figure 5 shows bond pad locations and the corresponding local density map. As noted in [14], high 3D-VIA usage complicates legalization. Snake-3D keeps most grid densities below 50%, while the bin-based method often exceeds 50–60%, potentially hindering legalization and routing.

Compared to post-placement, Snake-3D's timing advantage over Pin-3D [10] further increases, as post-route optimization amplifies

TABLE VII

28 NM PPA METRICS: PIN-3D [10] VS. TPGNN [15] VS. SNAKE-3D (OURS) ON THREE BENCHMARKS AT 28 NM NODE WITH 1.0 µm BOND PAD PITCH. 2D RESULTS ALSO PROVIDED AS REFERENCE. UNDERLINED VALUES INDICATE BEST AMONG THREE PARTITIONING METHODS (EXCLUDING 2D).

Design		ARM Cortex-A7				ARM Cortex-A35				OpenPiton 64KB L3				Δ	Δ
Algorithms		2D	[10]	[15]	Ours	2D	[10]	[15]	Ours	2D	[10]	[15]	Ours	[10]	[15]
Tech. Node / Bond Pad Pitch		28nm / 1.0 µm												-	
Target Freq.		3GHz				2GHz				1.5GHz				-	
#Gate (after synthesis)		209K				538K				317K				-	-
Footprint (mm^2)		0.47	0.23	0.23	0.23	1.16	0.48	0.48	0.48	1.50	0.47	0.47	0.47	0%	0%
Runtime (min)		-	5.0	46.2	8.0	-	36.8	151.5	12.5	-	15.8	97.8	12.5	-9%	-87%
Post Place	WNS (ps)	-301	-161	-153	-155	-195	-114	-111	-105	-250	-309	-283	-281	-7%	-2%
	TNS (ns)	-1181	-948	-838	-831	-209.3	-589	-365	-385	-1041	-683	-572	-563	-22%	+1%
	#Vio. Path (K)	15	20	18	19	4	23	19	19	11	18	14	15	-13%	+3%
	#3D-Nets (K)	-	45	56	28	-	102	161	59	-	52	112	36	-36%	-60%
	Avg #Snk.	-	2.8	2.4	2.0	-	3.2	3.0	2.0	-	3.8	2.2	2.1	-37%	-18%
	Max #Snk.	-	12	11	s9	-	11	14	8	-	19	26	14	-26%	-36%
Legalization	Displacement (µm)	-	3.7	8.9	3.3	-	3.2	5.7	2.8	-	2.8	21.5	2.9	-7%	-67%
Post Route	WNS (ps)	-184	-173	-312	-147	-121	-116	-237	-107	-334	-337	-2310	-303	-11%	-65%
	TNS (ns)	-966	-1287	-1905	-888	-152	-226	-1782	-195	-578	-1021	-17874	-850	-20%	-79%
	#Vio. Path (K)	14	16	17	13	7	8	26	7	7	10	42	8	-20%	-60%
	#Bond-Pad (K)	-	50	66	33	-	118	183	81	-	61	160	45	-31%	-60%
	Avg #Snk.	-	3.6	3.3	2.6	-	4.3	5.1	3.2	-	5.2	6.6	4.0	-25%	-33%
	Max #Snk.	-	13	14	11	-	15	17	13	-	21	36	16	-18%	-34%
	Power (W)	0.79	0.79	0.91	0.78	1.10	1.08	1.21	1.05	0.87	0.87	1.31	0.85	-2%	-21%

TABLE VIII

7NM PPA METRICS: PIN-3D [10] VS. TPGNN [15] VS. SNAKE-3D (OURS) ON THREE BENCHMARKS AT 7 NM NODE WITH 0.2 µm BOND PAD PITCH.

Design		ARM Cortex-A7				ARM Cortex-A35				OpenPiton 64KB L3				Δ	Δ
Algorithms		2D	[10]	[15]	Ours	2D	[10]	[15]	Ours	2D	[10]	[15]	Ours	[10]	[15]
Tech. Node / Bond Pad Pitch		7nm / 0.2 µm												-	
Target Freq.		3GHz				2GHz				1.5GHz				-	
#Gate (after synthesis)		198K				489K				280K				-	-
Footprint (mm^2)		0.069	0.041	0.041	0.041	0.12	0.060	0.060	0.060	0.17	0.082	0.082	0.082	0%	0%
Runtime (min)		-	4.9	50.9	9.9	-	25.2	137.0	13.6	-	11.4	89.9	12.5	+22%	-86%
Post Place	WNS (ps)	-102	-147	-156	-137	-79	-354	-338	-291	-64	-269	-231	-228	-13%	-9%
	TNS (ns)	-138	-405	-373	-314	-30	-1006	-1090	-764	-28	-1726	-1101	-1049	-29%	-17%
	#Vio. Path (K)	6	17	15	14	0.7	31	32	27	1.8	34	24	23	-20%	-9%
	#3D-Nets (K)	-	34	93	16	-	75	74	53	-	61	93	25	-47%	-62%
	Avg #Snk.	-	2.5	4.5	1.1	-	3.3	3.4	1.5	-	4.3	2.1	1.6	-57%	-51%
	Max #Snk.	-	10	15	5	-	14	11	5	-	22	30	9	-58%	-64%
Legalization	Displacement (µm)	-	1.0	1.9	0.9	-	0.6	0.6	0.6	-	0.9	1.3	0.8	-11%	-32%
Post Route	WNS (ps)	-286	-146	-218	-106	-196	-201	-213	-138	-477	-360	-337	-318	-23%	-31%
	TNS (ns)	-1137	-603	-1071	-402	-390	-496	-449	-434	-368	-1269	-1445	-481	-36%	-44%
	#Vio. Path (K)	17	15	18	10	5	15	14	13	3	17	15	8	-34%	-34%
	#Bond-Pad (K)	-	102	150	74	-	251	244	228	-	178	190	102	-26%	-34%
	Avg #Snk.	-	3.8	5.1	2.1	-	3.9	3.9	2.8	-	5.9	4.6	2.5	-43%	-44%
	Max #Snk.	-	14	15	12	-	14	14	13	-	26	28	15	-21%	-25%
	Power (W)	0.22	0.21	0.22	0.21	0.26	0.25	0.26	0.25	0.28	0.28	0.29	0.28	-0.5%	-5%

the effects of reduced bond-pad count and snaking. WNS improves by 11% and 23%, and TNS by 20% and 36% in 28 nm and 7 nm, respectively. Snake-3D also extends its lead over TPGNN [15], with WNS gains of 65% and 31%, and TNS gains of 79% and 44% in 28 nm and 7 nm. Notably, TPGNN now underperforms Pin-3D in most benchmarks and metrics, especially in 28 nm. This is primarily due to legalization, which follows the 3D placement stage. While TPGNN incorporates timing, it neglects local density constraints, causing severe cell overlap after 3D placement, as illustrated in Figure 2. This leads to excessive legalization and degraded timing. The effect is reflected in the legalization displacement metric in Table VII and Table VIII, which measures average cell movement after 3D placement and strongly correlates with final timing. For example, in the OpenPiton benchmark at 28 nm, TPGNN shows significantly higher displacement than the other two methods, resulting in much worse WNS and TNS. As discussed in subsection IV-D, timing awareness provides a strong starting point for optimizing WNS and TNS during 3D PnR. Separately, local density constraints, shown earlier, are critical for preventing timing degradation during legalization. Bin-based methods and TPGNN each lack one of these elements, while Snake-3D is the first to integrate both effectively.

F. Snaking Awareness and Critical Path Analysis

Our key contribution is integrating snaking awareness into partitioning. As shown in Table VII and Table VIII, Snake-3D shows 25% and 33% less snaking than [10] and [15] in 28 nm, and 43% and 44% less in 7 nm, respectively. Figure 6 shows that in the 28 nm A7 benchmark, the worst path in the bin-based method is deeply

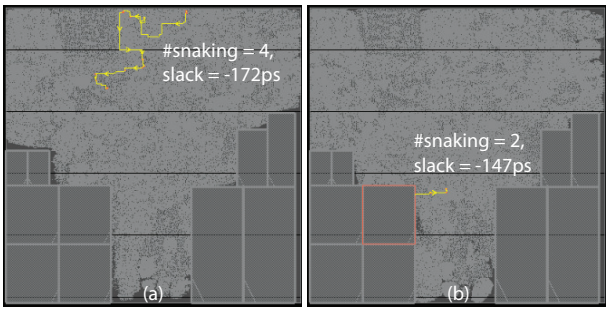

Fig. 6. Worst path in the 28 nm Cortex-A7: (a) the bin-based method results in deep snaking (#snaking=4, slack=−172 ps); (b) Snake-3D shifts the worst path to a route with reduced snaking (#snaking=2, slack=−147 ps).

TABLE IX
SNAKE-3D WITH/WITHOUT SNAKING AWARENESS VS. BIN-BASED FOR BENCHMARK A35 ON 28NM NODE. WITHOUT SNAKING AWARENESS, SNAKE-3D MAY WORSEN WNS AND TNS, EMPHASIZING ITS NECESSITY.

	Pin3D (Bin-Based)	Snake-3D W/O snaking aware	Snake-3D
WNS (ps)	-213	-215	-183
TNS (ns)	-610	-455	-426
#Vio. Paths (K)	80	67	64
#Bond-Pad (K)	118	78	80
Avg / Max #Snk.	4.3 / 15	3.6 / 15	3.2 / 13

snaking. With Snake-3D, it shifts to a memory path, suggesting timing bottlenecks move to less controllable factors. We also compare Snake-3D with and without snaking awareness on A35 in Table IX. Snake-3D reduces bond-pad count by 32% with snaking awareness and slightly more, 35%, without it. However, bond-pad count alone does not determine snaking. Without snaking awareness, average snaking is 3.6 (16% better than bin-based), while the aware version achieves 3.2 (26% better). TNS improves by 25.4% without and 30% with snaking awareness. WNS slightly worsens without awareness (1% worse than bin-based) but improves by 14% with it. These results underscore the importance of snaking-aware partitioning.

G. Impact of individual Loss Component

We compare our cost function (Equation 6) with Proxy 1 and Proxy 2 from [25], also discussed in subsection III-B and shown in Table X. We exclude the other two proxies, as they only encourage $\widetilde{\mathbf{z}}$ convergence without binarization and cannot function independently. Due to its superior convergence behavior (Equation III-C), our function yields 25% and 72% fewer 3D nets compared to Proxy 1 and Proxy 2, respectively. This leads to better snaking reduction (subsection III-B) and delivers the best PPA.

H. Cutsize Comparison with State-of-the-Art

We evaluate Snake-3D on ISPD98 benchmarks [31], disabling local density and path-awareness to focus on cutsize with more aggressive coarsening. Table XI reports cutsize for Snake-3D and other partitioners. Snake-3D shows only a 6.8% gap from the best-known results, close to MedPart's 5.0%. Yet, it already runs efficiently on entry-level GPUs and offers room for improvement on more powerful GPU and cutsize-specific tuning. MedPart [25] Figure 4 reports a 100-minute runtime for 500K edges on an A100, while our A35 case completes in 15 minutes on a T4 GPU, as shown in Table VII and VIII. The T4 has 5× less memory (16 vs. 80 GB) and 2.4× lower computational power (8.1 vs.19.5 TFLOPS).

TABLE X
COMPARISON OF SINGLE-NET COST FUNCTIONS (PROXY 1, PROXY2 FROM [25], AND EQUATION 6) ON CUTSIZE (#3D-NETS) AND PPA POST-PLACEMENT, ROUTING. BEST VALUES PER ROW ARE UNDERLINED.

		Proxy 1	Proxy 2	Ours
Post Place	WNS (ps)	-160	-163	-155
	TNS (ns)	-869	-1004	-831
	#Vio. Path (K)	19	21	19
	#3D-Nets (K)	37	99	28
	Avg / Max #Snk.	2.5 / 8	5.8 / 16	2.0 / 9
Post Route	WNS (ps)	-154	-318	-147
	TNS (ns)	-972	-1658	-888
	#Vio. Path (K)	13	16	13
	#Bond-Pad (K)	67	116	33
	Avg / Max #Snk.	2.9 / 11	6.0 / 19	2.6 / 11

TABLE XI
CUTSIZE COMPARISON ACROSS BENCHMARKS FOR PARTITIONING METHODS (SPECPART [24], HMETIS [19], MEDPART [25], AND OUR SNAKE-3D) AT $\epsilon = 2\%$. SOTA DENOTES THE BEST RESULT EVER REPORTED FOR EACH TEST CASE SUMMARIZED IN [30].

Bench.	|V|	|E|	SOTA	[24]	[19]	[25]	Ours
IBM01	12752	14111	200	202	213	202	203
IBM02	19601	19584	307	336	339	352	358
IBM03	23136	27401	951	959	972	955	980
IBM04	27507	31970	573	593	617	583	587
IBM05	29347	28446	1706	1720	1744	1748	1746
IBM06	32498	34826	962	963	1037	1000	1023
IBM07	45926	48117	878	935	975	913	988
IBM08	51309	50513	1140	1146	1146	1158	1140
IBM09	53395	60902	620	620	637	625	640
IBM10	69429	75196	1253	1318	1313	1327	1343
IBM11	70558	81454	1051	1062	1114	1069	1097
IBM12	71076	77240	1919	1920	1982	1955	2187
IBM13	84199	99666	831	848	871	850	865
IBM14	147605	152772	1842	1859	1967	1876	1871
IBM15	161570	186608	2730	2741	2886	2896	2870
IBM16	183484	190048	1827	1915	2095	1972	2246
IBM17	185495	189581	2270	2354	2520	2336	2346
IBM18	210613	201920	1521	1535	1587	1955	1750
AVG gap to SOTA			0%	2.30%	6.20%	5.00%	6.75%

V. CONCLUSION

We introduce Snake-3D, the first 3D IC partitioner that addresses snaking early in the partitioning stage while also considering local density. Results show that Snake-3D consistently outperforms both 3D IC partitioning baselines, including the bin-based FM method and the GNN based approach, across all benchmarks, technology nodes, and metrics. Snake-3D adopts gradient descent for its natural global view and optimizes bond-pad count, snaking, and local density simultaneously across all cells and nets using a carefully designed, novel differentiable cost function. Unlike other gradient-based methods that require tuning multiple weighted loss terms, our design ensures both convergence and binarization using a single cutsize cost function, maintaining simplicity and efficiency. Already fast on entry-level GPUs with moderate memory, Snake-3D is widely accessible and demonstrates strong scalability and generality.

VI. ACKNOWLEDGMENT

This work was supported by the SRC Logic and Memory Devices program (Task 3142.001) and JUMP 2.0 CHIMES Center (Task 3136.002) , the Ministry of Trade, Industry & Energy of South Korea (1415187652, RS-2023-00234159), and Samsung Electronics.

REFERENCES

[1] "AMD 3D V-Cache™ Technology," https://www.amd.com/en/products/processors/technologies/3d-v-cache.html.

[2] J. Lu et al., "EPlace-3D: Electrostatics Based Placement for 3D-ICs," in Proceedings of the 2016 on International Symposium on Physical Design, 2016.

[3] X. Zhao, S. Chen, Y. Qiu, J. Li, Z. Huang, B. Xie, X. Li, and Y. Bao, "ipl-3d: A novel bilevel programming model for die-to-die placement," in 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), 2023, pp. 1–9.

[4] J. Cong and G. Luo, "A multilevel analytical placement for 3d ics," in 2009 Asia and South Pacific Design Automation Conference, 2009, pp. 361–366.

[5] B. Goplen and S. Sapatnekar, "Efficient thermal placement of standard cells in 3D ICs using a force directed approach," in International Conference on Computer Aided Design, 2003.

[6] M. Hsu, V. Balabanov, and Y. Chang, "TSV-Aware Analytical Placement for 3-D IC Designs Based on a Novel Weighted-Average Wirelength Model," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2013.

[7] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 7, pp. 1228–1240, 2008.

[8] B. W. Ku et al., "Compact-2D: A Physical Design Methodology to Build Commercial-Quality Face-to-Face-Bonded 3D ICs," in Proceedings of the 2018 International Symposium on Physical Design, 2018.

[9] P. Vanna-Iampikul, C. Shao, Y.-C. Lu, S. Pentapati, and S. K. Lim, "Snap-3d: A constrained placement-driven physical design methodology for face-to-face-bonded 3d ics," in Proceedings of the 2021 International Symposium on Physical Design, ser. ISPD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 39–46. [Online]. Available: https://doi.org/10.1145/3439706.3447049

[10] S. Pentapati et al., "Pin-3D: A Physical Synthesis and Post-Layout Optimization Flow for Heterogeneous Monolithic 3D ICs," in 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2020.

[11] S. S. K. Pentapati and S. K. Lim, "Heterogeneous monolithic 3d ics: Eda solutions, and power, performance, cost tradeoffs," in 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021, pp. 925–930.

[12] Y.-J. Chen, Y.-S. Chen, W.-C. Tseng, C.-Y. Chiang, Y.-H. Lo, and Y.-W. Chang, "Late breaking results: Analytical placement for 3d ics with multiple manufacturing technologies," in 2023 60th ACM/IEEE Design Automation Conference (DAC), 2023, pp. 1–2.

[13] Y. Zhao, P. Liao, S. Liu, J. Jiang, Y. Lin, and B. Yu, "Analytical heterogeneous die-to-die 3d placement with macros," 2024.

[14] Y.-H. Huang, S. Pentapati, A. Agnesina, M. Brunion, and S. K. Lim, "On legalization of die bonding bumps and pads for 3-d ics," Trans. Comp.-Aided Des. Integ. Cir. Sys., vol. 43, no. 9, p. 2741–2754, Sep. 2024. [Online]. Available: https://doi.org/10.1109/TCAD.2024.3382835

[15] Y.-C. Lu, S. S. Kiran Pentapati, L. Zhu, K. Samadi, and S. K. Lim, "Tp-gnn: A graph neural network framework for tier partitioning in monolithic 3d ics," in 2020 57th ACM/IEEE Design Automation Conference (DAC), 2020, pp. 1–6.

[16] Panth, Shreepad and Samadi, Kambiz and Du, Yang and Lim, Sung Kyu, "Shrunk-2-d: A physical design methodology to build commercial-quality monolithic 3-d ics," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 36, no. 10, pp. 1716–1724, 2017.

[17] S. Panth, K. Samadi, Y. Du, and S. K. Lim, "Placement-driven partitioning for congestion mitigation in monolithic 3d ic designs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 4, pp. 540–553, 2015.

[18] C. Fiduccia and R. Mattheyses, "A linear-time heuristic for improving network partitions," in 19th Design Automation Conference, 1982, pp. 175–181.

[19] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: application in vlsi domain," in Proceedings of the 34th Annual Design Automation Conference, ser. DAC '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 526–529. [Online]. Available: https://doi.org/10.1145/266021.266273

[20] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz, ¡italic¿k¡/italic¿-way Hypergraph Partitioning via ¡italic¿n¡/italic¿-Level Recursive Bisection, pp. 53–67. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611974317.5

[21] U. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," IEEE Trans. Parallel Distrib. Syst., vol. 10, no. 7, p. 673–693, Jul. 1999. [Online]. Available: https://doi.org/10.1109/71.780863

[22] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine, "The zoltan and isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering and coloring," Scientific Programming, vol. 20, no. 2, p. 713587, 2012.

[23] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in High-Performance Computing and Networking, H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 493–498.

[24] 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, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3508352.3549390

[25] R. Liang, A. Agnesina, and H. Ren, "Medpart: A multi-level evolutionary differentiable hypergraph partitioner," in Proceedings of the 2024 International Symposium on Physical Design, ser. ISPD '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 3–11. [Online]. Available: https://doi.org/10.1145/3626184.3633319

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980v9, 2014. [Online]. Available: https://arxiv.org/abs/1412.6980v9

[27] J. Balkind et al., "OpenPiton: An Open Source Manycore Research Framework," in Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, 2016.

[28] S.-A. Chew, B. Zhang, K. Vanstreels, E. Chery, J. De Messemaeker, L. Witters, K. Van Sever, S. Iacovo, S. Dewilde, M. Stucchi, J. De Vos, G. Beyer, A. Miller, and E. Beyne, "The challenges and solutions of cu/sicn wafer-to-wafer hybrid bonding scaling down to 400nm pitch," in 2023 International Electron Devices Meeting (IEDM), 2023, pp. 1–4.

[29] S. K. Lee, "Hybrid bonding plays starring role in 3d chips," IEEE Spectrum, August 2024. [Online]. Available: https://spectrum.ieee.org/hybrid-bonding

[30] I. Bustany, A. Kahng, Y. Koutis, B. Pramanik, and Z. Wang, "Partition solutions, scripts and SpecPart," https://github.com/TILOSAI-Institute/HypergraphPartitioning, 2023, accessed: 2023.

[31] C. J. Alpert, "The ispd98 circuit benchmark suite," in Proceedings of the 1998 International Symposium on Physical Design, ser. ISPD '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 80–85. [Online]. Available: https://doi.org/10.1145/274535.274546