A Machine Learning-Powered Tier Partitioning Methodology for Monolithic 3-D ICs

Yi-Chen Lu, *Student Member, IEEE*, Sai Pentapati, Lingjun Zhu, *Graduate Student Member, IEEE*, Gauthaman Murali, Kambiz Samadi, *Senior Member, IEEE*, and Sung Kyu Lim, *Senior Member, IEEE*

***Abstract*—Tier partitioning is one of the most critical stages in monolithic 3-D (M3D) integrated circuits (ICs) implementation flows. It transforms 2-D netlists into 3-D by performing tier assignment for each design instance, which directly impacts the power, performance, and area (PPA) metrics of final 3-D full-chip designs. However, the current state-of-the-art tier partitioning approach named bin-based min-cut algorithm has fundamental flaws that lead to severe drawbacks, such as timing degradation, 3-D routing overhead, and redundant monolithic intertier vias (MIVs) insertion. To overcome these issues, in this article, we propose TP-GNN, an unsupervised graph learning-based tier partitioning framework that utilizes graph neural networks (GNNs) and advanced machine learning (ML) techniques to perform tier partitioning. The proposed framework comprehends design- and technology-related parameters properly so that it is generalizable to various netlists and technologies. In addition, it can be integrated with any style of M3D design flows that require tier assignments of standard cells. In the experiments, we validate the proposed framework on seven industrial designs with two different fashions of M3D implementation flows: 1) partitioning-first (Snap3D) and 2) partitioning-last (Shrunk2D and Compact2D) flows. We demonstrate that our framework, TP-GNN, significantly improves the 3-D quality of results (QoR) across most testing designs in a large margin compared with the bin-based min-cut tier partitioning algorithm. Specifically, in OpenPiton, an RISC-V-based multicore system, we observe 27.4%, 7.7%, and 20.3% improvements in performance, wirelength, and energy-per-cycle, respectively. Finally, we perform a case study by applying the proposed framework to a heterogeneous M3D design flow, Pin3D, on a commercial CPU design and observe that TP-GNN reaches better partitioning solutions than the existing partitioning approaches for heterogeneous 3-D ICs.**

***Index Terms*—3D integrated circuits (ICs), circuit partitioning, physical design.**

Manuscript received 26 April 2021; revised 26 July 2021 and 16 November 2021; accepted 15 December 2021. Date of publication 29 December 2021; date of current version 24 October 2022. This work was supported by the National Science Foundation under Grant CNS 16-24731. This article was recommended by Associate Editor C. Zhuo. *(Corresponding author: Yi-Chen Lu.)*

Yi-Chen Lu, Sai Pentapati, Lingjun Zhu, and Gauthaman Murali are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: yclu@gatech.edu).

Kambiz Samadi is with Research and Development Department, Qualcomm Technologies, Inc., San Diego, CA 92121 USA.

Sung Kyu Lim is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA.

Digital Object Identifier 10.1109/TCAD.2021.3139310

I. Introduction

THREE-DIMENSIONAL integration technology is a promising solution that provides technology scaling beyond Moore's law. It improves the power, performance, and area (PPA) metrics of 2-D integrated circuits (ICs) by stacking dies one on top of each other using intertier vias. Based on different types of stacking approaches, 3-D ICs can be categorized into three streams: 1) through-silicon via (TSV) based; 2) monolithic intertier via (MIV) based; and 3) face-to-face (F2F) bonded [17]. TSV-based 3-D ICs were developed earlier than the others; however, due to the large pitch in micron scale (μm) and high parasitics of TSVs, TSV-based 3-D ICs often result in low 3-D integration densities and thus, fail to truly benefit from the 3-D integration [8]. Recently, monolithic 3-D (M3D) integration has become the most promising approach to build 3-D ICs. Thanks to the nanoscale (nm) size of MIVs, M3D stacking enables cheaper intertier connections and a more fine-grained physical design, leading to a much higher device density compared with the TSV-based 3-D designs [1], [33]. Therefore, in this work, we will focus on improving M3D implementation flows [21].

3-D placement is recognized as the grandest challenge to build high-quality 3-D ICs [6], which is mainly due to the fact that currently there does not exist any commercial electronic design automation (EDA) tools that can perform 3-D placement directly from 2-D netlists such as the commercial approach in [22]. To obtain commercial-grade 3-D ICs using commercial EDA tools, state-of-the-art M3D implementation flows Shrunk2D [27], Compact2D [18], and Snap3D [32] leverage 2-D commercial placers to mimic 3-D placements by performing tier partitioning on "projected 2-D designs" (detailed explanations in Section II) as an alternative of performing "true 3-D placement" (i.e., obtain final 3-D placement solutions directly from 2-D netlists).

Tier partitioning refers to the algorithmic process of assigning each design instance to a specific tier. This process is critical as it determines the locations and of standard cells and intertier vias (e.g., MIVs), which directly impacts the quality of results (QoR) of full-chip designs. Currently, all of the state-of-the-art M3D flows adopt a partitioning method named bin-based min-cut tier partitioning algorithm that partitions netlists by minimizing cutsize. The algorithm first divides the entire 2-D design into multiple "bins" (rectangular regions) on the x-y plane. Then, it adopts an area-balanced min-cut partitioning algorithm to partition the cells inside each bin into

different tiers (z-direction) by minimizing the cutsize of the partial netlist within a bin. The idea behind this approach is to minimize the intertier connections across tiers while balancing the standard cell area of the overall layout. However, there are several significant drawbacks of this approach that lead to suboptimal 3-D full-chip designs, namely, as follows.

1) *Timing Degradation:* The bin-based partitioning algorithm fails to consider the global connections among bins. It only iteratively partitions the subnetlist within a single bin, which inevitably leads to a severe timing degradation.
2) *Low 3-D Integration Density:* Min-cut partition is not necessarily good for 3-D integration as it might not realize the full potential of the high integration density that monolithic 3-D (M3D) integration provides.
3) *Placement Quality Degradation:* Hierarchy information from RTL is completely ignored in the existing bin-based algorithm. Therefore, extra cutsize will be introduced and intertier vias will be inserted in suboptimal locations, which results in a placement quality degradation.

In this article, we address all the drawbacks raised above. We present TP-GNN, an unsupervised graph-learning-based framework that performs tier partitioning using graph neural networks (GNNs) and the weighted k-means clustering algorithm [20]. Unlike previous works that neglect design-related and technology-related parameters, we consider timing, hierarchy, and library information in our algorithm. The goal of this work is to present a novel tier partitioning framework that advances the state-of-the-art M3D implementation flows in terms of the full-chip PPA metrics.

Throughout the years, active research in M3D ICs has been conducted extensively and numerous styles of M3D implementation flows have been developed. Given the fact that all M3D approaches must separate logics onto different tiers during tier partitioning, and this partitioning step can happen either early (partitioning-first) or late (partitioning-last) with respect to the placement stage, M3D design flows can be categorized into two streams: 1) partitioning-first and 2) partitioning-last flows as shown in Figs. 1 and 2, respectively. In this article, we use the proposed TP-GNN framework to explore both options. In addition, since different technologies can be leveraged in different tiers of M3D ICs (i.e., heterogeneous 3-D designs), we further validate the proposed framework with a heterogeneous M3D design flow named Pin3D [29] based on a commercial multicore CPU design. We demonstrate that the proposed framework has the ability of comprehending the technology features so as to perform effective tier partitioning.

The remainder of this article is organized as follows. Section II gives an introduction of the targeted M3D design flows and the graph representation learning. Section III revisits related works on tier partitioning methods for 3-D ICs. Section IV describes the TP-GNN algorithms and implementation details. The experimental results are presented in Section V, and critical discussions are delivered in Section VI. Finally, we conclude our work in Section VII.

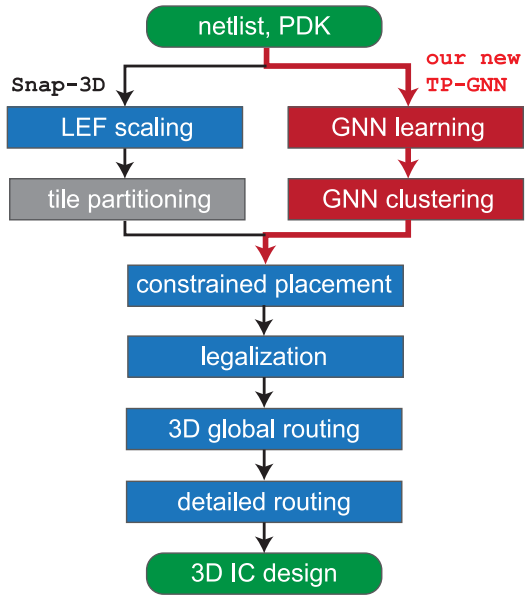


Fig. 1. Partitioning-first M3D design flow Snap-3D [32] versus our GNN-based partitioning flow.

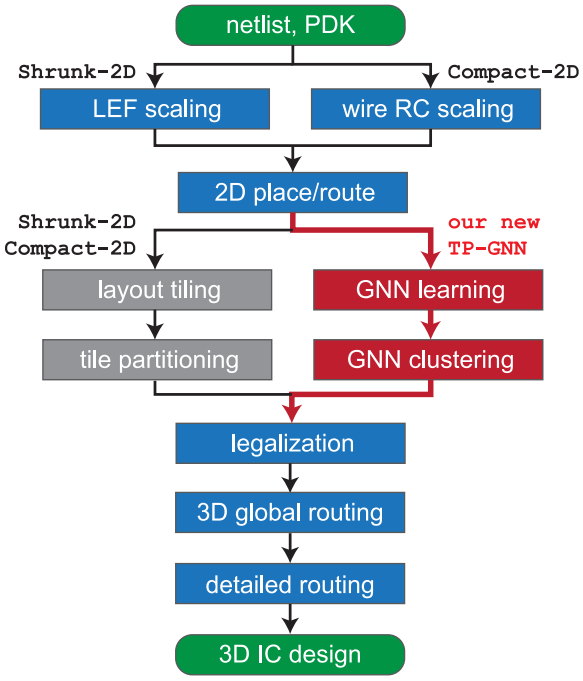


Fig. 2. Partitioning-last M3D design flows Shrunk2D [27] and Compact2D [18] versus our GNN-based partitioning flow.

II. Background

A. State-of-the-Art 3-D Implementation Flows

Based on the granularity of 3-D integration, 3-D designs can be categorized into three levels: 1) transistor level; 2) gate level; and 3) block level. Since the gate-level design methodology provides much more implementation freedom than the others, in this work, we will focus on the 3-D ICs implementation flows in this level. Furthermore, as aforementioned, depending on the timing of the tier partitioning stage with respect to the placement stage, M3D design flows can be categorized into two streams: 1) *partitioning-first* and

2) *partitioning-last* [28] flows. In this article, we apply the proposed tier partitioning framework TP-GNN on both kinds of gate-level M3D design flows and demonstrate that our framework outperforms the bin-based min-cut tier partitioning algorithm that currently most flows adopt in terms of full-chip QoR. In the following, we introduce the two kinds of M3D flows in more detail.

1) Partitioning First: Chang *et al.* [9] proposed the first partitioning-first M3D implementation flow named Cascade-2D, where they focus on the 3-D implementation of improving memory-heavy commercial CPU designs. Nonetheless, since Cascade-2D does not generalize to a wider range of design styles, in this article, we take a recent flow named Snap-3D [32] that achieves state-of-the-art results over popular benchmarks as our baseline partitioning-first M3D flow. Fig. 1 shows the design steps of the Snap-3D flow. The key idea of Snap-3D lies in the observation that the row structure of a 2-D placement can be divided into even and odd sites (rows) that naturally represent two different 3-D dies (tiers). Therefore, by carefully setting the placement constraints, one can easily place cells in different dies simultaneously (i.e., co-optimization) through any 2-D placer. In the original Snap-3D work, they utilize tiling methods, which resemble the min-cut partitioning method that Shrunk2D and Compact2D adopt to generate such placement constraints as partitioning solutions. In this work, we leverage the proposed tier partitioning framework to generate the constraints and demonstrate that our design-aware partitioning approach can lead to better PPA results.

2) Partitioning Last: Shrunk2D [27] and Compact2D [18] are the two state-of-the-art partitioning-last M3D implementation flows. They both leverage commercial tools for physical design implementations, where the main difference lies in the approach of mimicking 3-D designs in the 2-D stage. In Shrunk2D, standard cells are shrunk into half of the original sizes for placement and routing (P&R), whereas Compact2D scales the RC parasitics by a factor of $1/\sqrt{2}$ instead of shrinking the cells. After 2-D P&R, cells are expanded (Shrunk2D) or projected (Compact2D) onto a 2-D die with half of the original footprint. In the subsequent partitioning stage, both flows adopt the bin-based partitioning method as described in Section I to perform tier partitioning, which highly degrades the quality of the final full-chip 3-D design. The remaining stages are similar, starting from the legalization for both tiers to the timing closure for tape-out. In this work, we significantly improve these state-of-the-art flows by introducing a novel tier partitioning framework, TP-GNN, which overcomes the severe degradation occurred in [18] and [27] as aforementioned. The detailed algorithms of our framework are described in Section IV.

3) MIV Planning (Insertion) Details: For both partitioning-first and partitioning-last design flows, MIV insertions are performed after obtaining the partitioning results at the 3-D global routing stage as shown in Figs. 1 and 2. Specifically, in this stage, we stack the metal layers from top die and bottom die together, and advise the router to perform global routing on the stacked metal layers, where MIVs are the vias that the router inserts between the top metal layer of the bottom die and the bottom metal layer of the top die. For example, assume a two-tier M3D design and each die has six metal layers (M1–M6), which results in 12 metal layers (M1–M12) when stacked together during the 3-D global routing phase, the MIVs are the vias that the router inserts between the M6 and M7 layers. Note that the main purpose of MIVs is to connect the cells located in different dies. In the above example, MIVs are leveraged to connect the pins of the bottom die cells that are located in M1 with the pins of the top die cells located in M7.

B. Heterogeneous 3-D ICs Design Flow

Heterogeneous 3-D ICs refer to the 3-D chips that adopt more than one technologies within. A common practice is to use different technologies for various dies (tiers). The main benefit is that by using heterogeneous 3-D stacking, 3-D ICs in old technologies can reap the performance gain of the 2-D chips equipped with new technologies, which are prohibitively expensive to be developed. In other words, heterogeneous 3-D integration may produce competitive products as 2-D technology scaling but at a lower cost. However, the design flows (Shrunk2D, Compact2D, and Snap3D) introduced in the previous section do not support building heterogeneous 3-D designs. To validate the proposed tier partitioning framework in a broader scale, in this article, we take Pin3D [29], a novel heterogeneous 3-D design flow, as our reference flow, and demonstrate that the proposed partitioning strategy can achieve better full-chip PPA than the original partitioning strategy in Pin3D [29].

C. Graph Neural Networks

Recently, GNNs have gained great traction across various research areas [13]. In general, GNNs are based on a message passing scheme, where the objective is to learn a representation vector for each node by recursively aggregating and transforming the features of its neighboring nodes. After k iterations, a node will be represented by a vector, which captures the structural information and the attributes within its k-hop neighborhood. VLSI circuits can be naturally modeled as graphs. In this work, we first devise a hierarchy-aware graph transformation algorithm to convert the original netlist (hypergraph) into an edge-contracted clique-based graph. Then, we leverage GNNs to perform graph representation learning, where the goal is to construct a node representation that captures the design characteristics related to tier partitioning for each node. After the graph learning, we utilize the weighted k-means algorithm [11] to perform area-balanced partitioning based on the learned representation for each cell.

III. Related Works

Many research groups have proposed tier partitioning methods for 3-D ICs. Previous work [19] employs a dynamic programming algorithm and a flow-based min-cost algorithm to minimize TSV count and wirelength. Banerjee *et al.* [4] and Ghosal and Chatterjee [12] leveraged breadth-first search to further perform area-balanced partitioning. Another work [23] proposes a force-directed algorithm with cost-based heuristics to break long wires. However, these studies fail to consider the

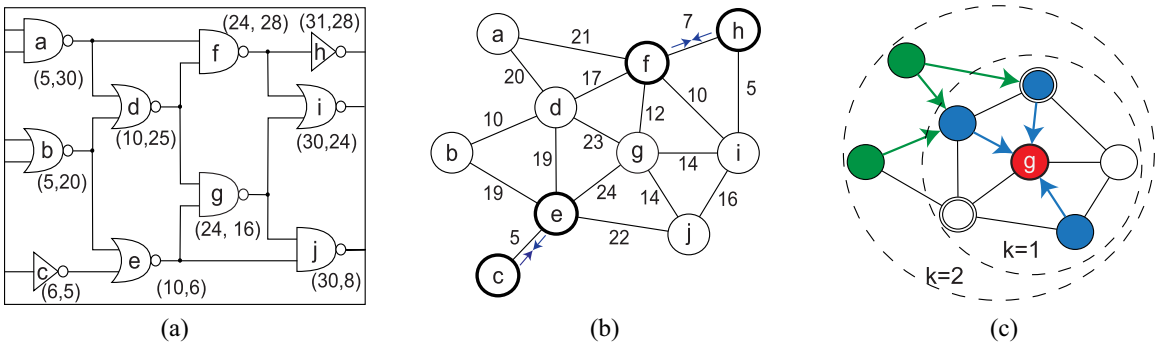


Fig. 3. TP-GNN visualization. (a) Input netlist with two design hierarchies: $\{a, b, d, f, h\}$ and $\{c, e, g, i, j\}$. Numbers represent cell locations. (b) Hierarchy-aware edge contractions on the transformed clique-based graph. Edge weights represent the Manhattan distance. (c) For target node g, sampling and aggregating features from its k-hop neighbors.

timing and power degradation when performing the partition. Furthermore, the keep-out-zone of TSVs and the requirement of die alignment limit the 3-D device integration density in these studies [7].

A study of unbalanced area partitioning for M3D designs is conducted in [31], where a crucial conclusion is drawn that the minimization of intertier via count is no longer critical to obtain high-quality 3-D ICs as in TSV-based designs (this conclusion also holds for F2F stacking fashion). In [5], an iterative partitioning tool for M3D designs is presented, where a simulated annealing algorithm is introduced to optimize a wire-cost function without limiting the usage of intertier (3-D stacking) vias. Another work [14] presents a folding-based method to transform 2-D layouts into 3-D. Also, a bin-connection graph is proposed in [14] to remove cell overlaps. However, these studies [5], [14] are shaded by Shrunk2D and Compact2D due to the absence of optimizations from commercial tools.

To benefit from commercial tools, previous studies [6], [26] have proposed 3-D design flows similar to Fig. 2, but with different tier partitioning strategies. Billoint *et al.* [6] introduced a folding-based partitioning technique similar to [14]. However, even with the aid of commercial tools, this approach [6] only shows marginal 3-D savings. Panth *et al.* [26] proposed a routability-driven tier partitioning model, which leverages min-overflow routing heuristics to perform bin-based partitioning. Nonetheless, this approach only achieves minor improvement on few designs, since it retains the significant drawbacks of the bin-based partitioning method as described in Section I.

IV. TP-GNN Algorithms

A. Overview

Figs. 1 and 2 demonstrate the integration of our proposed tier partitioning framework TP-GNN with the state-of-the-art 3-D design flows. As shown in the figure, the input to the TP-GNN framework is a projected 2-D design, where all the cells are placed, routed, and projected onto a 2-D die with half of the 2-D counterpart's footprint. The output of the framework is a partitioned design, where each cell is assigned to a unique tier.

Fig. 3 shows the visualization of our framework. Given a projected 2-D design as shown in Fig. 3(a), we transform the netlist hypergraph into an edge-contracted clique-based graph as shown in Fig. 3(b) by devising a hierarchy-aware edge contraction algorithm. After the contraction, we leverage GNNs to perform instance-based graph representation learning as shown in Fig. 3(c), where features within K-hop neighbors ($K = 2$) of the target node are sampled and aggregated to learn accurate representations for the downstream clustering stage.

Finally, our tier partitioning framework TP-GNN is generalizable to *every* design, since it learns the feature representations by optimizing an unsupervised loss function (unsupervised learning). Also, it does not assume anything regarding the netlist structure or design characteristics. Instead, it learns and adapts to various netlists using graph embedding techniques. Finally, TP-GNN can be easily integrated with existing 3-D implementation flows to significantly improve the quality of the final full-chip design. Note that ideally, our method can be extended to support multitier partitioning by clustering the nodes into $k > 2$ groups. However, the transition will not be that smooth because it will depend on the ways that pseudoplacements are generated and MIVs are inserted into multitiers. Furthermore, currently state-of-the-art M3D design flows (Shrunk2D [27], Compact2D [18], Snap3D [32], and Pin3D [29]) only support two-tier M3D designs, and in this article, we focus on improving the full-chip PPA metrics of two-tier 3-D designs. The detailed algorithms of our framework are described in the following sections.

B. Hierarchy-Aware Edge Contraction

Starting from a projected 2-D design, we first transform the original netlist (a directed hypergraph) into an undirected clique-based graph G, where a net that originally contains k cells forms a k-clique in G, and each edge $e = (u, v)$ is assigned a weight representing the Manhattan distance between cell u and cell v in the projected 2-D placement. Then, we apply a hierarchy-aware edge contraction algorithm (Algorithm 1) on this graph G, where pairs of nodes within the same hierarchy are contracted into supernodes based on the ascending order of edge weights (lines 1–4). When a supernode v' is obtained, we update the edge weights between its neighbors and its center of gravity (lines 5–7). Note that the term "hierarchy" refers to the "module" defined in the synthesized netlist (RTL).

Algorithm 1 Hierarchy-Aware Edge Contraction Algorithm

Input: $G(V, E)$: original clique-based graph.
Output: $G'(V', E')$: edge-contracted clique-based graph.

$E \leftarrow$ **SortEdgesByWeight**(E) (in ascending order)
for $e = (u, v) \in E$ **do**
  if u, v not contracted **and** u, v in the same hierarchy **then**
    contract (u, v) to form a new vertex v' ▷ in-place
    $v'_x \leftarrow \frac{u_x+v_x}{2}$, $v'_y \leftarrow \frac{u_y+v_y}{2}$ ▷ update location
    for $n \in \{\text{neighbors}(u) \cup \text{neighbors}(v)\}$ **do**
      $\text{edgeWeight}(v', n) = |v'_x - n_x| + |v'_y - n_y|$
$G'(V', E') \leftarrow G(V, E)$

The goal of Algorithm 1 is to prevent the severe placement quality degradation occurred in Shrunk2D and Compact2D, which can be accounted by two reasons. First, cells within the same hierarchy are highly connected with each other. If the hierarchy information is ignored in the partitioning algorithm, intertier vias will be inserted in suboptimal locations that introduce redundant cuts. Second, previous works fail to consider the actual cell distance in the 2-D placement while performing partitioning. Cells that are nearby and connected should have a higher chance to remain in the same tier compared with other distant cells; otherwise, designs will suffer from severe 3-D routing overhead. Finally, Algorithm 1 can be performed recursively to condense the graph and to benefit from the runtime and memory requirement of the later graph learning. However, a denser graph does not always achieve better PPA. In the experiments, we perform two runs of Algorithm 1 for each design implemented by our framework.

C. GNN Feature Aggregator

After obtaining the edge-contracted clique-based graph G' from Algorithm 1, we leverage GNN to perform graph learning. The goal of this stage is to learn accurate node representations that capture the characteristics of the design regarding tier partitioning. These learned representations are further utilized to determine the tier assignment in the later clustering stage.

Before the actual learning process, we determine an initial feature vector for each node as shown in Table I. Note that features in Table I are designed for partitioning-last M3D flows where the tier partitioning stage happens after the 2-D physical implementation stage as shown in Fig. 2. For the partitioning-first design flow, the features are extracted right after the synthesis stage. Due to the lack of physical implementations (e.g., placement and routing), we drop the slack and slew features presented in Table I while remaining others.

The features in the table span from a node's structural information and its design attributes. Unlike previous works that ignore timing information during tier partitioning, we prevent the severe timing degradation by considering four timing related features as shown in Table I. Note that these initial node representations are insufficient to perform tier partitioning. To learn better representations, we train GNNs to sample and aggregate the neighboring features for each node. The GNN model will capture the local structural information as well as the node attributes that are related to tier partitioning.

TABLE I
INITIAL NODE FEATURES FOR PARTITIONING LAST DESIGN FLOWS IN EDGE-CONTRACTED GRAPH G'. NOTE THAT A NODE MAY REPRESENT MULTIPLE CELLS IN THE DESIGN

features	descriptions
hierarchy	"module" defined in the synthesized netlist
sum_slack	sum of worst slacks of all cells
sum_slew	sum of maximum pin slew of all cells
sum_delay	sum of worst delay of all cells
dist2source	length of shortest path to clock source on G'
1-hop degree	number of 1-hop neighbors on G'
2-hop degree	number of 2-hop neighbors on G'

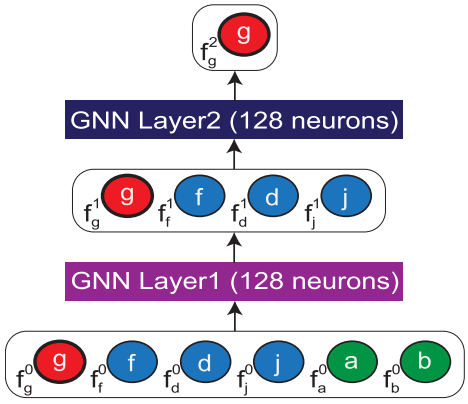


Fig. 4. Graph learning for target node g. Following from Fig. 3(c), we demonstrate the detailed learning process, where $\{f^0\}$ represent the initial features and f_g^2 represents the learned representations.

Inspired by [13], our feature aggregator aggregates the k-hop neighborhood features of a node v as follows:

$$f_v^k = \sigma\left(f_v^{k-1} + \theta_k \cdot \frac{1}{s_k} \sum_{u \in SN_k(v)} f_u^{k-1}\right) \tag{1}$$

where σ is the sigmoid function, f_v^k denotes the representation vector of node v at level k, $SN_k(v)$ denotes the neighbors sampled at k-hop, s_k denotes the corresponding sampling size, and θ_k represents the parameters of the neural network (NN) at k-hop (each hop has its own NN). Note that the concept of "level" is corresponding to the concept of "hop," where f_v^0 is the initial features defined in Table I for node v, and $f_v^{k=K}$ is the final representation after aggregating the information within the K-hop neighborhood of v. The aggregator (1) can be considered as a "graph filter," since it performs instance-based learning that aggregates a node's neighboring information iteratively. In the experiments, we set $K = 2$ and each NN (θ_1, θ_2) has an output dimension of 128. Finally, Fig. 4 further demonstrates the feature aggregation process based on Fig. 3(c), where our goal is to construct the node representation for the target node g. The learning process happens as follows. First, we sample a fixed amount of neighbors from its 1-hop (denoted in blue) and 2-hop (denoted in green) neighbors. Then, starting from the initial features $\{f^0\}$, we leverage a two-layer GNN to perform iterative feature aggregation in order to construct the final representation f_g^2.

D. Unsupervised GNN Learning

In this work, we leverage unsupervised learning to train the TP-GNN framework. Therefore, our framework is generalizable, since it does not require any pretraining before using. Here, we introduce an unsupervised instance-based loss function $\mathcal{L}(y_v)$, which takes $y_v = f_v^K$, the final representation vector of node v, as the input and calculates the cross-entropy between v and its neighboring nodes $N(v)$ (not necessary in K-hop) as

$$\mathcal{L}(y_v) = -\sum_{u\in N(v)} \log\left(\sigma\left(y_v^\top y_u\right)\right) - \sum_{i=1}^{M} \mathbb{E}_{n_i\sim \text{Neg}(v)} \log\left(\sigma\left(-y_v^\top y_{n_i}\right)\right) \quad (2)$$

where $\text{Neg}(v)$ denotes the negative sampling distribution of node v, and M denotes the negative sampling size. In practice, rather than taking $N(v)$ as the full k-hop neighborhood of node v, which causes overfitting and damages computational efficiency, we perform a random walk starting from node v to generate $N(v)$ that represents the passed by nodes. Also, in (2), the negative sampling technique improves the efficiency of GNN learning, where an underlying idea is that the GNN model should not only improve the similarity between a node v and its true contexts $N(v)$ but also enhance the disparity of v to the false samples $\text{Neg}(v)$ (nodes that are not occurred in the random walk).

E. GNN Training Methodology

To update the parameters of our framework, we introduce a gradient descent optimizer Adam [16] to minimize $\mathcal{L}$ (2). The detailed training methodology is described in Algorithm 2. In lines 1–9, we perform random walks on every node $v \in V'$ to generate the neighborhood structures. Then, starting from the initial features (Table I), we aggregate the neighborhood features for each node through (1) (lines 11–17). Finally, in lines 18–23, we leverage Adam to update the parameters of the GNNs through the introduced unsupervised loss function (2). After the learning process, the learned node representations $\{y\} \in R^{128}$ are fed to the later clustering stage to determine the tier assignment for each cell.

F. Weighted K-Means Clustering

The final stage of the proposed framework is the clustering process, where we leverage the weighted k-means clustering algorithm [11] to partition the edge-contracted clique-based graph $G' = (V', E')$. The goal at this stage is to determine the tier assignment for each node $v \in V'$ based on its learned representation y_v from Algorithm 2. In this work, we introduce a weight to each node $v \in V'$, which denotes the total area of the gates that it represents. Note that a node may represent multiple gates in the actual netlist, and gates corresponding to the same node will be assigned to the same tier. Given the learned node representations $\{y\}$ and the weights $\{w\}$, the algorithm clusters the nodes V' into k weight-balanced groups based on the similarity of $\{y\}$. Assume V' is classified into k clusters $\{C_1, \ldots, C_k\}$, the loss function is derived as

Algorithm 2 TP-GNN Training Methodology. We Use Default Values of $\alpha = 0.001, K = 2, NRW = 5, LRW = 5, M = 30, s_1 = 30, s_2 = 20, \beta_1 = 0.9$, and $\beta_2 = 0.999$

Input: $G'(V', E')$: edge-contracted clique-based graph. $\{f^0\}$: initial features. α: learning rate, K: depth of neighborhood, NRW: # random walks starting from a node, LRW: length of a walk, M: negative sampling size, $\{s_k, \forall k \in \{1, ..., K\}\}$: k-hop neighborhood sampling size, σ: sigmoid function, $\{\theta_k, \forall k \in \{1, ..., K\}\}$: parameters of NN at hop k, $\{\beta_1, \beta_2\}$: Adam parameters.

Output: $\{y\}$: learned node representations.

1: **for** $v \in V'$ **do** ▷ random walks on each node
2: $N(v) \leftarrow \{\}$ ▷ initialization of neighboring nodes
3: **for** $n \leftarrow 1$ to NRW **do**
4: $cur\_v \leftarrow v$
5: **for** $l \leftarrow 1$ to LRW **do**
6: $next\_v \leftarrow$ Sample a 1-hop neighbor of cur_v
7: **if** $next\_v$ is not v **then**
8: add $next\_v$ to $N(v)$
9: $cur\_v \leftarrow next\_v$
10: **while** $\{\theta_k\}$ do not converge **do** ▷ train to converge
11: $f_v^0 \leftarrow \frac{f_v^0}{\|f_v^0\|_2}, \forall v \in V'$
12: **for** $k \leftarrow 1$ to K **do** ▷ aggregate neighborhood features
13: **for** $v \in V'$ **do**
14: $S_k \leftarrow$ Sample s_k neighbors at k-hop neighborhood
15: $f_v^k = \sigma\left(f_v^{k-1} + \theta_k \cdot \frac{1}{s_k} \sum_{u\in S_k} f_u^{k-1}\right)$
16: $f_v^k \leftarrow \frac{f_v^k}{\|f_v^k\|_2}, \forall v \in V'$
17: $y_v \leftarrow f_v^K, \forall v \in V'$
18: **for** $v \in V'$ **do** ▷ minimize unsupervised loss
19: **for** $u \in N(v)$ **do**
20: $Neg(v) \leftarrow$ Sample M samples from $\{V' - N(v)\} \setminus v$
21: $neg\_loss \leftarrow \sum_{n_i\in Neg(v)} \log(\sigma(-y_v^\top y_{n_i}))$
22: $g_v \leftarrow \nabla_\theta[\log(\sigma(y_v^\top y_u)) + neg\_loss]$
23: $\{\theta_k\} \leftarrow Adam(\alpha, \{\theta_k\}, g_v, \beta_1, \beta_2)$

$$\mathcal{L}_{k\text{mean}} = \sum_{i=1}^{k} \sum_{v\in C_i} w(v) \cdot \|y_v - c_i\|^2 \quad (3)$$

where $c_i = ([\sum_{v\in C_i} y_v w(v)]/[\sum_{v\in C_i} w(v)])$ denotes the weighted centroid of cluster C_i. To update (3), we adopt an iterative minimization technique as illustrated in Algorithm 3. Starting from an initial centroids $\{c_1, \ldots, c_k\}$, for each iteration, we determine the clusters $\{C_1, \ldots, C_k\}$ by assigning each node to the centroid that has the minimum weighted distance (line 3). After the assignments, we update the centroids based on the newly obtained clusters (line 4). The clustering process is complete when the assignments no longer change.

G. Postpartitioning Optimization

The clustering results of the weighted K-means algorithm (Algorithm 3) can already be taken as valid tier partitioning

Algorithm 3 Weighted k-Means Clustering. We Use Default Value of $k = 2$

Input: $G'(V', E')$: edge-contracted clique-based graph, $\{w\}$: node weights, $\{y\}$: node representations, k: number of clusters.

Output: $\{C_1, ..., C_k\}$: k clusters.

1: Select k initial centroids $\{c_1, ..., c_k\}$ randomly
2: **repeat**
3: $\{C_1, ..., C_k\} = \underset{C}{\text{argmin}} \sum_{i=1}^{k} \sum_{v \in C_i} w(v) \|y_v - c_i\|^2$
4: $c_i = \frac{\sum_{v \in C_i} y_v w(v)}{\sum_{v \in C_i} w(v)}, \forall i = 1, ..., k$
5: **until** $\{C_1, ..., C_k\}$ no longer change

Algorithm 4 Postpartitioning Optimization. We Assume a Two-Tier 3-D Design and Top Die Is Faster

Input: $G(V, E)$: original 2D design, C_{top}: instances in top tier, C_{bot}: instances in bottom tier.

Output: C'_{top} and C'_{bot}: updated partitioning results.

1: *critiCells* $\leftarrow$ **FindCellsOnCriticalPaths**(G) ▷ in hash map
2: **for** *net* $\in$ *Nets* **do**
3: *critiCount* $\leftarrow 0$
4: **for** *cell* $\in$ *net* **do**
5: **if** *cell* in *critiCells* **then** ▷ O(1) look-up
6: *critiCount*++
7: **if** *critiCount* $\geq 0.5 *$ *countCell*(*net*) **then**
8: Fix all cells on *net* in top tier. ▷ in-place

solutions. However, for certain design flows such as the heterogeneous 3-D design flow, extra handling on timing degradation during tier partitioning phase is needed. The reason is that such design flow leverages different technologies in various tiers, where the performance (timing) between BEOLs can vary by as much as 30% [30]. Therefore, the 3-D designs can easily result in worse performance if cells on critical paths in the original 2-D design are partitioned randomly as occurred in Shrunk2D and Compact2D.

To solve the above issue, in this work, we further propose a postpartitioning optimization algorithm to mitigate the performance degradation of tier partitioning in heterogeneous 3-D design flows. The proposed algorithm is shown in Algorithm 4. Given a tier partitioning result that denotes the cell locations in top tier and bottom tier, we first build a hash map to identify the critical cells in the original 2-D design (line 1). Then, for each net in the design, if greater or equal to half of the cells are in the critical cell map, then we fix the entire cells on the net in top tier (lines 2–8). Note that the algorithm is based on the assumption that top die is faster than bottom die (as in the Pin-3D [29] design flow).

H. Implementation Details

In this article, we apply the proposed TP-GNN framework to a variety styles of M3D design flows, which include partitioning-first, partitioning-last, and heterogeneous 3-D design flows. In the partitioning-first design flow, since the tier partitioning happened before the pseudoplacement, the timing related features in Table I are taken from a synthesis tool (*Synopsys Design Compiler*), where in the partitioning-last design flow, the features are found in *Cadence Innovus*. As for the feature "dist2source," we take the hop count as the representation in partitioning-first design flows, and take the actual physical distance on layout as the denotation in partitioning-last design flows. Finally, in the heterogeneous design flow Pin-3D, since it accepts a partitioned design as inputs and continues the design flow through 3-D legalization to tape-out, the feature extraction process is same as the partitioning-last design flow.

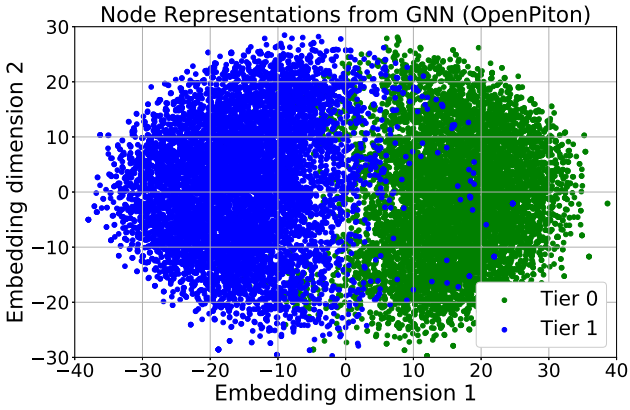


Fig. 5. t-SNE visualizations of the learned node representations from GNN. Each dot represents a cell in the design and is colored by its final tier assignment from Algorithm 3.

V. Experimental Results

In this section, we perform thorough experiments to demonstrate the achievements of the TP-GNN framework. We validate our framework on seven industrial designs, including two RISC-V-based multicore systems OpenPiton [3] and RocketCore [2], NOVA, LDPC, TATE, JPEG from *OpenCores.org*, and NETCARD from *ISPD 2012 benchmark* [25]. All the seven benchmarks are synthesized under TSMC 28-nm technology node using *Synopsys Design Compiler 2015*. We leverage the *Cadence Innovus Implementation System v18.1* to perform placement and routing, and utilize *Synopsys PrimeTime 2018* for signoff analysis. Finally, the TP-GNN framework is implemented in *Python3* with the *Tensorflow* library, and the training time is measured on a machine with 2.40 GHz CPU, 16-GB RAM, and a NVIDIA RTX 2070 graphics card. Note that for all 3-D designs implemented by Shrunk2D and Compact2D, we have performed bin sweeping to find the optimal bin size for fair comparisons.

A. GNN-Related Results

First, to evaluate the graph learning, we leverage the t-distributed stochastic neighboring embedding [24] (t-SNE) technique to visualize the learned node representations $\{y\} \in R^{128}$ from Algorithm 2 in R^2 with OpenPiton [3]. The visualization result is shown in Fig. 5, where we observe that the learned representations form two observable linear separable clusters. Based on the embedded locations in R^2, we

TABLE II
PERFORMANCE, AREA, AND ENERGY COMPARISON OF SHRUNK2D (S2D) [27] AND TP-GNN FLOWS ON RISC-V-BASED DESIGNS USING F2F STACKING. Δ DENOTES THE PERCENTAGE DIFFERENCE BETWEEN TP-GNN AND S2D

Metrics	2D	S2D	TP-GNN (Δ)
OpenPiton [3]			
eff. freq. (MHz)	289	270	344 (27.4%)
WL (m)	6.33	4.91	4.56 (-7.7%)
energy/cycle (pJ)	343.94	339.73	270.52 (-20.3%)
footprint (mm^2)	1.22	0.61	0.61
# MIVs	0	76,083	99,423 (30.7%)
critical path WL (um)	542.6	579.3	291.7 (-49.6%)
partitioning time (min)	-	9	26
RocketCore [2]			
eff. freq. (MHz)	832	921	964 (4.6%)
WL (m)	1.78	1.62	1.51 (-6.8%)
energy/cycle (pJ)	125.67	107.20	101.37 (-5.4%)
footprint (mm^2)	0.28	0.14	0.14
# MIVs	0	38,627	22,738 (-41.1%)
critical path WL (um)	314.2	289.4	128.9 (-55.5%)
partitioning time (min)	-	5	22

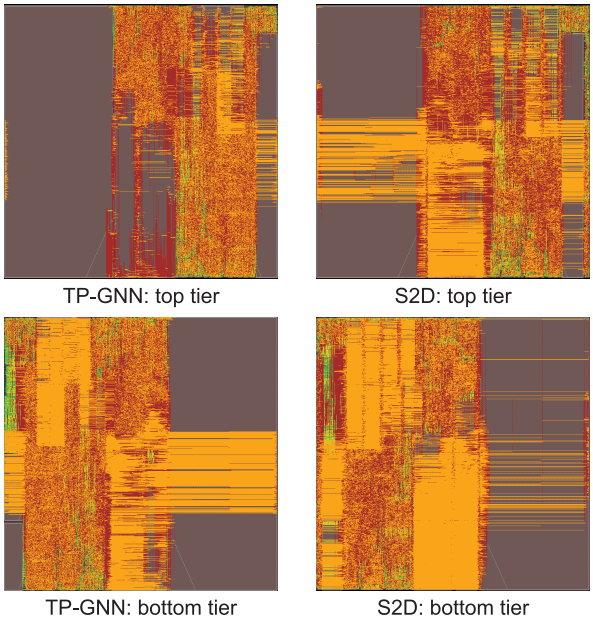


Fig. 6. GDS layouts of OpenPiton [3] using TP-GNN versus Shrunk2D [27] flow. TP-GNN flow achieves 7.7% better wirelength.

further color each dot (cell) by its tier assignment from the weighted k-means algorithm (Algorithm 3) and demonstrate that the algorithm efficiently identifies the two observable clusters. Now, we conclude that our TP-GNN framework is capable of transforming the initial features into meaningful high-dimension representations. In the later experiments, we demonstrate the superior achievements of TP-GNN in a complete design flow.

B. Maximum Performance Comparison

In this experiment, we perform maximum performance comparison between 2-D, Shrunk2D, and TP-GNN flows on two RISC-V-based designs: 1) OpenPiton [3] (# macros: 28) and 2) RocketCore [2] (# macros: 6). Note that for designs with extensive memory macros, such as OpenPiton and RocketCore, Shrunk2D significantly outperforms Compact2D. Therefore, we have taken the best-case scenario (Shrunk2D) of the existing state-of-the-art flows to perform the comparison. The results are shown in Table II, where we observe that our TP-GNN flow significantly outperforms the Shrunk2D flow across the two designs. The savings in timing-related metrics are noteworthy, where the critical path wirelength saving is 52% in average and the effective frequency is 27.4% better in OpenPiton. Also, even with a higher target frequency, TP-GNN has consistently large wirelength saving. Fig. 6 further shows the GDS layout comparison, where we observe that TP-GNN introduces fewer cross-macrowires than Shrunk2D. Note that the partitioning time of the proposed framework TP-GNN includes the runtime of both graph representation learning and the weighted k-means clustering algorithm. Since the graph learning is conducted in an unsupervised manner (i.e., we do not need to pretrain the model), there is no runtime overhead to apply the proposed framework.

C. Isoperformance Comparison

In the this experiment, we perform isoperformance validation of TP-GNN in the state-of-the-art partitioning-first (Snap-3D [32]) and partitioning-last (Shrunk2D [27] and Compact2D [18]) M3D design flows across six real-world designs. Furthermore, due to the fact that active research has been conducted extensively on solving the problem of 3-D placement [10], [15], [22], in this article, we take a recent MIV-compatible 3-D placement work [22] as our reference "true" 3-D placement flow termed True3D. The reason we use the term true here is to show the difference between the (3-D) placement results obtained by 3-D analytical placers [10], [15], [22] and 2-D commercial tools (Shrunk2D and Compact2D). Note that [22] does not propose a complete 3-D design flow, and to benchmark it against other flows in full-chip design, we further route the placements results achieved by the 3-D analytical placer using Cadence Innovus.

Note that in order to reasonably benchmark the analytical approach [22] that is originally developed for TSV-based 3-D ICs with other M3D flows that we focus on in this article, we relax the penalty of inserting intertier vias in the objective function. The reason is because M3D technology provides much cheaper 3-D stacking cost than the TSV-based 3-D technology.

As aforementioned, in this article, we validate the proposed tier partitioning framework on two different styles of M3D design flows. The results for partitioning-first design flow, Snap3D [32], are shown in Table III, and the results for the partitioning-last design flows, Shrunk2D [27] and Compact2D [18], are shown in Table IV. In the partitioning-last design flows, we observe that TP-GNN consistently outperforms Shrunk2D and Compact2D in QoR across all designs with only a little runtime overhead in tier partitioning. As for the comparison between pseudo-3D (Shrunk2D and Compact2D) and true 3-D (T3D) flow, we observe that the pseudo-3D flows consistently achieve much better PPA in terms of wirelength and power, where the T3D flow does not always obtain better QoR metrics compared with the original 2-D designs.

TABLE III
PARTITIONING-FIRST ISOPERFORMANCE COMPARISON OF SNAP-3D [32] AND TP-GNN FLOWS. Δ_{Snap} DENOTES THE PERCENTAGE DIFFERENCE BETWEEN TP-GNN AND THE SNAP-3D FLOW. WE REPORT THE TIME SPEND ON TIER PARTITIONING IN MINUTES

Metrics	2D	Snap-3D	TP-GNN (Δ_{Snap})
AES ($2.8GHz$) (# cells: 133,051)			
WL (m)	1.78	1.35	1.34 (-0.8%)
power (mW)	229.7	213.26	219.09 (+2.7%)
# MIV	0	51,138	44,533 (-12.9%)
WNS (ps)	69	48	36 (-25.0%)
partition-time	-	3 hr	20 min
ECG ($1.35GHz$) (# cells: 96,416)			
WL (m)	1.17	1.02	1.01 (-0.9%)
power (mW)	307.2	286.52	284.02 (-0.9%)
# MIV	0	26,764	32,105 (+19.8%)
WNS (ps)	49	51	22 (-56.8%)
partition-time	-	3 hr	18 min
JPEG ($1.53GHz$) (# cells: 219,534)			
WL (m)	2.43	2.07	2.06 (-0.4%)
power (mW)	704.5	668.5	665.4 (-0.4%)
# MIV	0	31,824	32,701 (+2.7%)
WNS (ps)	68	39	28 (-28.2%)
partition-time	-	6 hr	24 min
LDPC ($1.5GHz$) (# cells: 41,817)			
WL (m)	1.71	1.15	1.14 (-0.8%)
power (mW)	192.2	134.9	133.6 (-0.9%)
# MIV	0	17,182	20,184 (+17.4%)
WNS (ps)	25	40	20 (-50.0%)
partition-time	-	2 hr	14 min
NETCARD ($1.0GHz$) (# cells: 316,137)			
WL (m)	7.82	6.77	6.78 (-0.1%)
power (mW)	651.7	632.3	635.9 (-0.5%)
# MIV	0	50,341	47,366 (+5.9%)
WNS (ps)	56	37	40 (+8.1%)
partition-time	-	8 hr	42 min
NOVA ($1.0GHz$) (# cells: 131,737)			
WL (m)	2.33	2.25	2.26 (-0.4%)
power (mW)	479.0	218.9	217.4 (-0.6%)
# MIV	0	18,423	17,238 (-6.5%)
WNS (ps)	47	31	29 (-6.4%)
partition-time	-	8 hr	20 min
TATE ($1.37GHz$) (# cells: 211,911)			
WL (m)	1.99	1.94	1.93 (-0.5%)
power (mW)	395.7	397.3	396.4 (-0.2%)
# MIV	0	54,698	60,703 (+10.9%)
WNS (ps)	36	45	42 (-6.6%)
partition-time	-	5 hr	22 min

TABLE IV
PARTITIONING-LAST ISOPERFORMANCE COMPARISON OF TRUE3D (T3D) [22], SHRUNK2D (S2D), COMPACT2D (C2D), AND TP-GNN FLOWS. Δ_S AND Δ_C, RESPECTIVELY, DENOTE THE PERCENTAGE DIFFERENCE BETWEEN TP-GNN VERSUS S2D AND C2D. WE REPORT THE TIME SPEND ON TIER PARTITIONING IN MINUTES

Metrics	2D	T3D	S2D	C2D	TP-GNN (Δ_S, Δ_C)
AES ($2.8GHz$) (# cells: 133,015)					
WL (m)	1.78	1.86	1.45	1.42	1.46 (-0.6%, -2.8%)
power (mW)	229.7	233.1	217.8	215.2	218.3 (-0.2%, -1.4%)
# MIV	0	43,895	41,262	39,403	44,921 (+8.8%, -14.0%)
WNS (ps)	69	51	33	38	41 (+24.2%, +7.8%)
partition-time	-	-	5	5	20
ECG ($1.35GHz$) (# cells: 96,416)					
WL (m)	1.17	1.55	1.06	1.07	1.05 (-0.9%, -1.8%)
power (mW)	307.2	311.4	290.2	291.3	288.9 (-0.4%, -0.8%)
# MIV	0	19,843	13,681	14,525	14,028 (+2.5%, -3.4%)
WNS (ps)	49	18	80	62	35 (-56.2%, -43.5%)
partition-time	-	-	3	3	18
JPEG ($1.53GHz$) (# cells: 219,534)					
WL (m)	2.43	4.93	2.09	2.12	1.94 (-7.2%, -8.5%)
power (mW)	704.5	727.9	674.2	675.9	665.1 (-1.3%, -1.6%)
# MIV	0	34,190	27,839	28,231	27,154 (-2.5%, -3.8%)
WNS (ps)	68	20	49	41	23 (-53.1%, -43.9%)
partition-time	-	-	8	8	24
LDPC ($1.8GHz$) (# cells: 43,381)					
WL (m)	1.78	1.97	1.61	1.57	1.42 (-11.8.%, -9.6%)
power (mW)	362.5	311.1	301.4	292.5	271.4 (-10.0%, -7.2%)
# MIV	0	12,509	8,955	9,237	7,454 (-25.6%, -27.9%)
WNS (ps)	34	29	26	20	16 (-38.5%, -20.0%)
partition-time	-	-	2	2	14
NETCARD ($1.0GHz$) (# cells: 316,137)					
WL (m)	7.82	8.66	6.83	6.87	6.11 (-10.5%, -11.1%)
power (mW)	651.7	702.4	639.8	639.2	598.9 (-6.4%, -6.3%)
# MIV	0	54,403	43,823	43,754	39,987 (-8.8%, -8.6%)
WNS (ps)	56	11	51	49	26 (-49.0%, -46.9%)
partition-time	-	-	14	14	42
NOVA ($1.08GHz$) (# cells: 131,737)					
WL (m)	2.33	2.45	2.30	2.28	2.17 (-5.7%, -4.8%)
power (mW)	479	396.5	220.2	216.9	211.0 (-4.6%, -2.7%)
# MIV	0	19,922	16,672	16,935	15,813 (-5.2%, -6.6%)
WNS (ps)	47	0	28	25	19 (-32.1%, -24.0%)
partition-time	-	-	5	5	20
TATE ($1.37GHz$) (# cells: 211,911)					
WL (m)	1.99	2.41	1.97	1.95	1.92 (-2.5%, -1.5%)
power (mW)	395.7	439.2	398.4	398.6	396.5 (-0.4%, -0.5%)
# MIV	0	70,457	56,467	56,820	59,727 (5.8%, 5.2%)
WNS (ps)	36	0	87	76	31 (-64.4%, -59.2%)
partition-time	-	-	8	8	22

In the partitioning-first design flow comparison, we observe that the final PPA results of the original Snap-3D and the proposed TP-GNN enhanced flow are similar. This is mainly because in such design flows, the tier partitioning stage occurs before any physical implementation (e.g., placement, routing, etc.). Therefore, the impact of partitioning solutions to the QoR of the 3-D full-chip design is not as direct as the case of that in the partitioning-last design flows, where tier partitioning directly determines the design quality degradation occurred by 2-D–3-D transformation. Furthermore, we want to emphasize that the reason Snap-3D implementation flow takes hours to perform tier partitioning is because it still relies on the partitioning solutions from Shrunk2D to take them as placement constraints. Hence, we include the time to build the 2-D implementation of Shrunk2D in the partitioning time.

Finally, head-to-head comparisons are available between partitioning-first and partitioning-last design flows. We observe that in general, the Snap-3D (partitioning-first) design flow gives better PPA metrics than the partitioning-last design flows. The key reason is that Snap-3D tricks the commercial tool to optimize 3-D placements during the pseudo 2-D placement stage (more description in Section II), where both Shrunk2D and Compact2D require die-by-die legalization to obtain a legal 3-D placement solution after tier partitioning, which may degrade the quality of the obtained partitioning solutions.

D. Sweeping Experiments on Contracting Edges

In this experiment, we demonstrate the PPA effect of executing different number of times of the hierarchy-aware edge contraction algorithm (Algorithm 1) on the LDPC benchmark. The results are shown in Table V. As aforementioned, the designs built by TP-GNN in this work are achieved by running the algorithm two times. The straightforward benefit of running Algorithm 1 is to prevent the short nets in the original

TABLE V
SWEEPING EXPERIMENTS ON RUNNING HIERARCHY-AWARE EDGE CONTRACTION ALGORITHM (ALGORITHM 1) MULTIPLE TIMES

LDPC	0-run	1-run	2-run	3-run	4-run
WL (m)	1.49	1.44	1.42	1.43	1.47
power (mW)	277.5	272.1	271.4	272.8	280.5
# MIV	8,130	9,269	7,454	7,526	6,012
WNS (ps)	26	14	16	22	31
partitioning-time	20	18	14	13	11

TABLE VI
ISOPERFORMANCE COMPARISON ON A HETEROGENEOUS 3-D DESIGN OF A COMMERCIAL CPU DESIGN IMPLEMENTED BY PIN3D [29]. TP-GNN$_{opt}$ DENOTES THE POSTPARTITIONING OPTIMIZATION (ALGORITHM 4) IS ENABLED

Metrics	Pin3D	TP-GNN	TP-GNN$_{opt}$
commercial CPU design in 1.2GHz # cells: 176,352, # macros: 21			
WL (m)	3.17	3.11 (-1.9%)	3.07 (-3.2%)
total power (mW)	203.3	192.5 (-5.3%)	189.0 (-6.8%)
internal power (mW)	96.3	90.9 (-5.6%)	88.4 (-8.2%)
switching power (mW)	93.9	88.5 (-5.7%)	86.5 (-7.9%)
# MIV	30,155	34,068	28,883
WNS (ns)	0.452	0.237 (-47.5%)	0.085 (-81.2%)
partition-time	7 min	18 min	21 min

2-D designs from being partitioned into two separate tiers and turned into long nets, which may cause severe QoR degradation. Nonetheless, as shown in the table, overrunning the algorithm may as well incur the QoR degradation in final full-chip design, because some optimization opportunities are lost when various nodes are forced to be merged into one node.

E. Results on Heterogeneous 3-D ICs

In this experiment, we validate the proposed framework on a heterogeneous 3-D IC of a commercial CPU-core based on the Pin3D [29] design flow, where fast corner is used for top tier and slow corner is used for bottom tier (both in foundry 28 nm). The results are shown in Table VI. First, we observe that the proposed TP-GNN framework improves many critical QoR metrics of the original Pin3D design flow. Second, we find that with the postpartitioning optimization algorithm, TP-GNN can further optimize the full-chip PPA with little runtime overhead. In particular, the performance of the 3-D full-chip design is pushed much higher.

We attribute the success of TP-GNN in heterogeneous 3-D ICs in twofold. First, due to the fact that TP-GNN comprehends the technology features, timing related information is taken into account while performing tier partitioning, where the original partitioning algorithm that Pin3D adopts only considers to minimize the cutsize of connections. Second, the postpartitioning optimization algorithm reckons the idea that cells on critical paths should be considered carefully, which prevents the severe timing degradation that happens in the original Pin3D design flow.

VI. DISCUSSION

A. Why Does GNN Work Better?

Across the experiments for various fashions of M3D design flows, we observe that TP-GNN framework significantly improves the timing from the state-of-the-art flows in consistent. The main reason is that the original bin-based partitioning method ignores the global connections among bins. It only partitions the subnetlist within a bin. Therefore, critical nets in the projected 2-D designs are partitioned randomly. In the TP-GNN framework, we solve this issue by introducing timing-related features to the graph learning, which encourages cells on critical nets to be partitioned into same tier.

Furthermore, we observe that the TP-GNN framework achieves great wirelength savings, which can be explained by two reasons. First, Algorithm 1 prevents nearby and connected cells from being partitioned into different tiers, which reduces the significant 3-D routing overhead occurred in Shrunk2D and Compact2D flows. Second, with the structural features introduced in Table I and the message passing characteristics of the graph learning, cells that are logically connected would have similar node representations. Therefore, unlike bin-based partitioning method that partitions long nets randomly, our framework partitions long nets based on the netlist structure.

Finally, we want to emphasize that TP-GNN runtime is measured from the beginning of Algorithm 1 to the end of Algorithm 3. The runtime of our GNN-based tier partitioning algorithm basically involves training our GNN using unsupervised learning. Therefore, we do not report training versus inferencing time separately as our GNN learns while traversing the nodes in netlist graphs and collecting features from their neighbors. The time complexity of our TP-GNN is linear in terms of the netlist size. This is because our GNN model visits all the nodes in the netlist graph and spends a constant amount of time collecting features from the neighbors. The total number of neighbors for a given node under consideration is constant as we limit our neighbor search within a fixed hop count.

B. MIV Impact

In the experiments, we do not refrain the router to insert MIVs during the routing stage. This is because as pointed out in [33], the intertier vias density of M3D designs can achieve up to 100 million/mm in a 14-nm technology node, which leads to the conclusion that in M3D designs, no intertier via density constraints are needed as in TSV-based designs. Furthermore, as mentioned in [31], the minimization of MIV count no longer has a major impact on the full-chip PPA of M3D designs. Therefore, in this article, we do not strive for minimizing the MIV count.

C. Timing Impact on Crossing Tiers

Due to the small pitch and low parasitic of MIVs (nano-scale), in M3D designs, the timing impact of a net crossing different tiers is not as severe as in TSV-based designs. Since MIVs possess similar RC characteristics to regular vias, the timing delay of a net in M3D designs is proportional to its time (RC) constants. Table VII quantifies the delay and wirelength between cross-tier and same-tier nets on the ECG benchmark.

TABLE VII
TIMING IMPACT BETWEEN 3-D (CROSS-TIER) AND 2-D (SAME-TIER) NETS ON ECG BENCHMARK. THE UNIT FOR NET LENGTH IS μM, AND THE UNIT FOR DELAY IS PS

net type	# path	avg. length	avg. delay	wst. delay
cross-tier	43,645	21.2	1.3	58.9
same-tier (top)	86,083	7.6	0.3	35.7
same-tier (bot.)	101,493	9.5	0.4	29.5

As shown in the table, although the average net delay of cross-tier nets is higher than same-tier nets, the reason behind is that cross-tier nets tend to have longer wirelength and therefore, higher RC timing constants.

VII. CONCLUSION

In this article, we proposed TP-GNN, a novel tier partitioning framework based on GNN. First, we proposed a hierarchy-aware edge contraction algorithm to reduce the severe 3-D routing overhead occurred in the bin-based partitioning algorithm. Then, we mapped the classical tier partitioning problem into a clustering problem and solved it with advanced machine learning (ML) techniques. The graph representation learning provides the freedom for designers to deal with various partitioning objectives, and the unsupervised learning promises the generality.

ACKNOWLEDGMENT

The authors are thankful to the industry members of the Center for Advanced Electronics in ML and the anonymous reviewers who help improve the quality of this work with their insightful reviews.

REFERENCES

[1] K. Arabi, K. Samadi, and Y. Du, "3D VLSI: A scalable integration beyond 2D," in *Proc. Symp. Int. Symp. Phys. Design*, 2015, pp. 1–7.

[2] K. Asanovic *et al.*, "The rocket chip generator," Dept. Electr. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Rep. UCB/EECS-2016-17, 2016.

[3] J. Balkind *et al.*, "OpenPiton: An open source manycore research framework," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 2, pp. 217–232, 2016.

[4] S. Banerjee, S. Majumder, and B. B. Bhattacharya, "A graph-based 3D IC partitioning technique," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2014, pp. 613–618.

[5] G. Berhault, M. Brocard, S. Thuries, F. Galea, and L. Zaourar, "3DIP: An iterative partitioning tool for monolithic 3D IC," in *Proc. IEEE Int. 3D Syst. Integr. Conf. (3DIC)*, 2016, pp. 1–5.

[6] O. Billoint *et al.*, "A comprehensive study of monolithic 3D cell on cell design using commercial 2D tool," in *Proc. Design Autom. Test Europe Conf. Exhibition (DATE)*, 2015, pp. 1192–1196.

[7] K. Chang, A. Koneru, K. Chakrabarty, and S. K. Lim, "Design automation and testing of monolithic 3D ICs: Opportunities, challenges, and solutions," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2017, pp. 805–810.

[8] K. Chang, S. Pentapati, D. E. Shim, and S. K. Lim, "Road to high-performance 3D ICs: Performance optimization methodologies for monolithic 3D ICs," in *Proc. Int. Symp. Low Power Electron. Design*, 2018, pp. 1–6.

[9] K. Chang *et al.*, "Cascade2D: A design-aware partitioning approach to monolithic 3D IC with 2D commercial tools," in *Proc. 35th Int. Conf. Comput. Aided Design*, 2016, pp. 1–8.

[10] J. Cong and G. Luo, "A multilevel analytical placement for 3D ICs," in *Proc. Asia South Pac. Design Autom. Conf.*, 2009, pp. 361–366.

[11] R. C. De Amorim and B. Mirkin, "Minkowski metric, feature weighting and anomalous cluster initializing in k-means clustering," *Pattern Recognit.*, vol. 45, no. 3, pp. 1061–1075, 2012.

[12] P. Ghosal and S. Chatterjee, "Partitioning in 3D ICs: A TSV aware strategy with area balancing," in *Proc. Int. Conf. Devices Circuits Syst. (ICDCS)*, 2012, pp. 576–580.

[13] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Assoc., 2017.

[14] X. He, Y. Wang, Y. Guo, and S. Cotofana, "A mixed-size monolithic 3D placer with 2D layout inheritance," in *Proc. Great Lakes Symp. VLSI*, 2017, pp. 29–34.

[15] 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 4, pp. 497–509, Apr. 2013.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[17] J. Knechtel and J. Lienig, "Physical design automation for 3D chip stacks: Challenges and solutions," in *Proc. Int. Symp. Phys. Design*, 2016, pp. 3–10.

[18] B. W. Ku, K. Chang, and S. K. Lim, "Compact-2D: A physical design methodology to build commercial-quality face-to-face-bonded 3D ICs," in *Proc. Int. Symp. Phys. Design*, 2018.

[19] C.-R. Li, W.-K. Mak, and T.-C. Wang, "Fast fixed-outline 3-D IC floorplanning with TSV co-placement," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 3, pp. 523–532, Mar. 2012.

[20] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 129–137, Mar. 1982.

[21] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim, "TP-GNN: A graph neural network framework for tier partitioning in monolithic 3D ICs," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[22] G. Luo, Y. Shi, and J. Cong, "An analytical placement framework for 3-D ICs and its extension on thermal awareness," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 4, pp. 510–523, Apr. 2013.

[23] L. Lyu and T. Yoshimura, "A force directed partitioning algorithm for 3D floorplanning," in *Proc. IEEE 12th Int. Conf. ASIC (ASICON)*, 2017, pp. 718–721.

[24] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.

[25] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo, "The ISPD-2012 discrete cell sizing contest and benchmark suite," in *Proc. ACM Int. symp. Int. Symp. Phys. Design*, 2012, pp. 161–164.

[26] S. Panth, K. Samadi, Y. Du, and S. K. Lim, "Placement-driven partitioning for congestion mitigation in monolithic 3D IC designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 4, pp. 540–553, Apr. 2015.

[27] S. Panth, K. Samadi, Y. Du, and S. K. Lim, "Shrunk-2-D: A physical design methodology to build commercial-quality monolithic 3-D ICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 10, pp. 1716–1724, Oct. 2017.

[28] H. Park, B. W. Ku, K. Chang, D. E. Shim, and S. K. Lim, "Pseudo-3D approaches for commercial-grade RTL-to-GDS tool flow targeting monolithic 3D ICs," in *Proc. Int. Symp. Phys. Design*, 2020, pp. 47–54.

[29] S. S. K. Pentapati, K. Chang, V. Gerousis, R. Sengupta, and S. K. Lim, "Pin-3D: A physical synthesis and post-layout optimization flow for heterogeneous monolithic 3D ICs," in *Proc. 39th Int. Conf. Comput. Aided Design*, 2020, pp. 1–9.

[30] S. K. Samal, D. Nayak, M. Ichihashi, S. Banna, and S. K. Lim, "Tier partitioning strategy to mitigate beol degradation and cost issues in monolithic 3D ICs," in *Proc. 35th Int. Conf. Comput. Aided Design*, 2016, pp. 1–7.

[31] H. Sarhan, S. Thuries, O. Billoint, and F. Clermidy, "An unbalanced area ratio study for high performance monolithic 3D integrated circuits," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2015, pp. 350–355.

[32] 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 *Proc. Int. Symp. Phys. Design*, 2021, pp. 39–46.

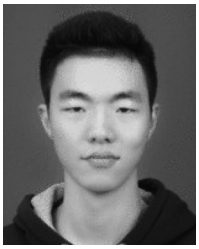
[33] M. Vinet *et al.*, "Monolithic 3D integration: A powerful alternative to classical 2D scaling," in *Proc. SOI-3D-Subthreshold Microelectron. Technol. Unified Conf. (S3S)*, 2014, pp. 1–3.

Yi-Chen Lu (Student Member, IEEE) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2017, and the M.S. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2019, where he is currently pursuing the Ph.D. degree under Prof. S. K. Lim's guidance.

His current research focuses on devising machine learning, reinforcement learning, and graph algorithms to enhance the electronic design automation flow for 2-D and 3-D integrated circuits.

Sai Pentapati received the bachelor's degree in electronics and electrical communication Engineering from the Indian Institute of Technology Kharagpur, Kharagpur, India, in 2017, and the master's degree from Georgia Tech, Atlanta, GA, USA, in 2020, where he is currently pursuing the Ph.D. degree.

His research interests include DTCO and design methodologies for 3-D ICs, exploring new physical design solutions enabled by 3-D ICs, and enabling machine learning for EDA.



Lingjun Zhu (Graduate Student Member, IEEE) received the B.S. degree in microelectronics from Fudan University, Shanghai, China, in 2018. He is currently pursuing the Ph.D. degree with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA.

His current research interests include physical design methodologies, power delivery, and thermal analysis approaches for high-performance 3-D ICs.

Gauthaman Murali received the B.E. degree in electronics and communication engineering from the PSG College of Technology, Coimbatore, India, in 2016, and the M.S. degree in electrical engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2020, where he is currently pursuing Ph.D. degree with the School of Electrical and Computer Engineering.

He was an SoC Design Engineer with Intel, Santa Clara, CA, USA, from 2016 to 2018. His current research interest includes physical design of 2.5-D systems and 3-D ICs, true 3-D placement, ML for 3-D physical design, and physical design of 3-D accelerator hardware.

Kambiz Samadi (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees from the University of California at San Diego, San Diego, CA, USA, in 2007 and 2010, respectively.

He joined Qualcomm Research, San Diego, in 2011, where he focused on 3-D IC EDA solutions and 3-D IC architecture-level design space explorations. Since 2018, he has been working on advanced timing/signoff methodology development as well as investigating machine learning-driven design methodologies for latest technology nodes. He has coauthored more than 50 publications in refereed journals and conferences and has more than 45 patents.

Sung Kyu Lim (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the Computer Science Department, University of California at Los Angeles, Los Angeles, CA, USA, in 1994, 1997, and 2000, respectively.

In 2001, he joined the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA, where he is a Professor. His research focus is on the architecture, design, test, and EDA solutions for 2.5-D and 3-D ICs. He has authored Practical Problems in *VLSI Physical Design Automation* (Springer, 2008) and *Design for High Performance, Low Power, and Reliable 3-D Integrated Circuits* (Springer, 2013). He has published more than 350 papers on 2.5-D and 3-D ICs. He has been leading two projects (CHIPS and 3DSOC) under DARPA Electronics Resurgence Initiative since 2017. His research is featured as Research Highlight in the Communication of the ACM in January 2014.

Prof. Lim received the National Science Foundation Faculty Early Career Development (CAREER) Award in 2006, the ACM SIGDA Distinguished Service Award in 2008, and the Best Paper Award from ATS'12, IITC'14, and EDAPS'17. His works have been nominated for the Best Paper Award at several top venues in EDA and circuit/packaging design. He was an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS from 2007 to 2009 and the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS from 2013 to 2018.