TP-GNN: A Graph Neural Network Framework for Tier Partitioning in Monolithic 3D ICs

Yi-Chen Lu[1], Sai Surya Kiran Pentapati[1], Lingjun Zhu[1], Kambiz Samadi[2], and Sung Kyu Lim[1]
[1]School of ECE, Georgia Institute of Technology, Atlanta, GA
[2]Qualcomm Technologies, Inc., San Diego, CA
yclu@gatech.edu

***Abstract*—3D integration technology is one of the few options that can keep Moore's Law trajectory beyond conventional scaling. Existing 3D physical design flows fail to benefit from the full advantage that 3D integration provides. Particularly, current 3D partitioning algorithms do not comprehend technology and design-related parameters properly, which results in sub-optimal partitioning solutions. In this paper, we propose TP-GNN, an unsupervised graph-learning-based tier partitioning framework, to overcome this issue. Experimental results on 7 industrial designs demonstrate that our framework significantly improves the QoR of the state-of-the-art 3D implementation flows. Specifically, in OpenPiton, a RISC-V-based multi-core system, we observe 27.4%, 7.7% and 20.3% improvements in performance, wirelength, and energy-per-cycle respectively.**

I. Introduction

3D IC placement is recognized as a grand challenge to build high-quality 3D ICs [4]. Since currently there is no existing commercial 3D placer, state-of-the-art 3D implementation flows Shrunk2D [12] and Compact2D [8] leverage 2D commercial placers to mimic 3D placements by performing tier partitioning on projected 2D designs. Both [8, 12] adopt the partitioning method named bin-based partitioning. The method first divides the entire 2D design into multiple bins on the x-y plane. Then, cells inside each bin are partitioned into two tiers (z-direction) using an area-balanced min-cut algorithm, which minimizes the inter-tier connections while balancing the cell area in both tiers.

However, there are drawbacks in this partitioning approach that limit the performance of the 3D full-chip designs, namely:

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

In this paper, 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 [9]. 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 advance the state-of-the-art 3D implementation flows in order to build commercial sign-off quality M3D ICs with much better power, performance, and area (PPA) metrics compared with their couterpart 2D ICs.

Recently, GNNs have gained great traction across various research areas [6]. 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 [5] to perform area-balanced partitioning based on the learned representation for each cell.

Furthermore, 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 3D implementation flows to significantly improve the quality of the final full-chip design. Note that in this paper we assume a 2-tier M3D design for fair comparisons with the state-of-the-art flows [8, 12], however, our framework can be easily extended to construct multi-tier designs.

II. Background

A. State-Of-The-Art 3D Implementation Flows

Shrunk2D [12] and Compact2D [8] are the state-of-the-art RTL-to-GDSII 3D implementation flows. They both leverage commercial tools for physical design implementations, where

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

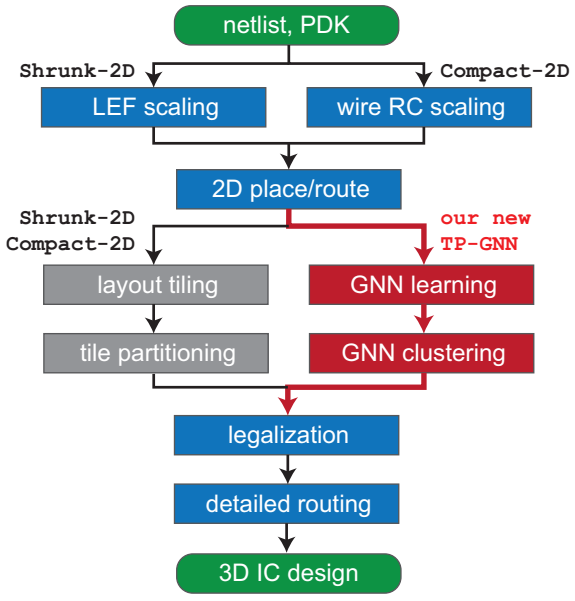

Fig. 1: State-of-the-art pseudo-3D design flow for monolithic 3D ICs including Shrunk-2D [12] and Compact-2D [8] vs. our new graph neural network (GNN) flow that replaces their tier partitioning step.

the main difference lies in the approach of mimicking 3D designs in the 2D 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 2D P&R, cells are expanded (Shrunk2D) or projected (Compact2D) onto a 2D 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 3D design. The remaining stages are similar, starting from the legalization for both tiers to the timing closure for tape-out. In the experiments, we take the Shrunk2D and the Compact2D flows as baselines and demostrate the significant improvements achieved by our TP-GNN framework.

B. Related 3D ICs Partitioning Works

A study of unbalanced area partitioning for M3D designs is presented in [13], where a crucial conclusion is drawn that the minimization of inter-tier via count is no longer critical to obtain high-quality 3D ICs as in TSV-based designs. In [3], 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 inter-tier vias. In [4], a folding-based method is proposed to transform 2D layouts into 3D designs. Nonetheless, mainly due to the insufficient utilization of commercial tools and the deficiency in the partitioning methods, these works only show marginal 3D savings over the 2D counterparts.

III. TP-GNN Algorithms

Figure 1 demonstrates the integration of our proposed tier partitioning framework TP-GNN with the state-of-the-art 3D

Algorithm 1 Hierarchy-aware edge contraction algorithm.

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

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

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

Figure 2 shows the visualization of our framework. Given a projected 2D design as shown in Figure 2(a), we transform the netlist hypergraph into an edge-contracted clique-based graph as shown in Figure 2(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 Figure 2(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. The detailed algorithms of our framework are described in the following sub-sections.

A. Hierarchy-Aware Edge Contraction

Starting from a projected 2D 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 2D 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).

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, inter-tier vias will be inserted in sub-optimal locations that introduce redundant cuts. Second, previous works fail to consider the actual cell distance in the 2D 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 3D routing overhead. Finally, Algorithm 1

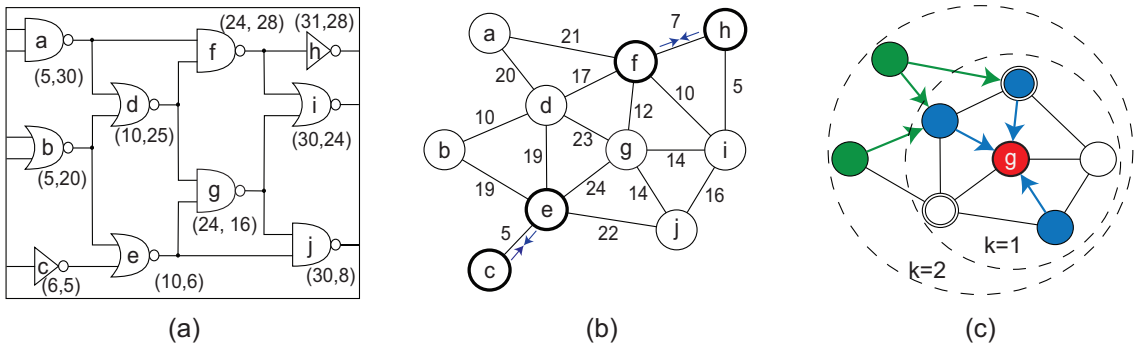

Fig. 2: TP-GNN visualization. (a) Input netlist with two design hierarchies: $\{a, b, d, f, h\}$ and $\{c, e, g, i, j\}$. Numbers represent cell locations. (b) Hierarchy-aware edge contractions on the transformed clique-based graph. Edge weights represent the Manhattan distance. (c) For target node g, sampling and aggregating features from its k-hop neighbors.

TABLE I: Initial features of a node in the 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'

can be performed recursively to condense the graph and to benefit from the run time and memory requirement of the later graph learning. However, a denser graph does not always achieve better PPA. In the experiments, we perform 2 runs of Algorithm 1 for each design implemented by our framework.

B. 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. The features 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. Inspired by [6], 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 aggregation the information within the K-hop neighborhood of v. The aggregator (Equation 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 neural network (θ_1, θ_2) has an output dimension of 128.

C. 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 pre-training 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:

$$\begin{aligned}\mathcal{L}(y_v) = &- \sum_{u \in N(v)} \log(\sigma(y_v^\top y_u)) \\ &- \sum_{i=1}^{M} \mathbb{E}_{n_i \sim Neg(v)} \log(\sigma(-y_v^\top y_{n_i})),\end{aligned} \tag{2}$$

where $Neg(v)$ 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

Algorithm 2 TP-GNN training methodology.
We use default values of $\alpha = 0.001, K = 2, NRW = 5, LRW = 5, M = 30, s_1 = 30, s_2 = 20, \beta_1 = 0.9, \beta_2 = 0.999$.

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 $\{V' - N(v)\} \backslash v$
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: $\{\theta_k\} \leftarrow Adam(\alpha, \{\theta_k\}, g_v, \beta_1, \beta_2)$

to generate $N(v)$ that represents the passed by nodes. Also, in Equation 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 $Neg(v)$ (nodes that are not occurred in the random walk).

D. GNN Training Methodology

To update the parameters of our framework, we introduce a gradient descent optimizer Adam [7] to minimize $\mathcal{L}$ (Equation 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 Equation 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 (Equation 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.

E. Weighted k-means Clustering

The final stage of the proposed framework is the clustering process, where we leverage the weighted k-means clustering algorithm [5] 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 $\{C_1, ..., C_k\}$, the loss function is derived as

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

where $c_i = \frac{\sum_{v \in C_i} y_v w(v)}{\sum_{v \in C_i} w(v)}$ denotes the weighted centroid of cluster C_i. To update Equation 3, we adopt an iterative minimization technique as illustrated in Algorithm 3. Strating from an initial centroids $\{c_1, ..., c_k\}$, for each iteration, we determine the clusters $\{C_1, ..., C_k\}$ 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. Note that our method can easily support multi-tier partitioning by clustering the nodes into multiple groups. In the implementation, we set $k = 2$ to perform fair comparison with the state-of-the-art flows: Shrunk2D [12] and Compact2D [8].

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: $\{C_1, ..., C_k\} = \underset{C}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{v \in C_i} w(v) \|y_v - c_i\|^2$
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

IV. Experimental Results

In this section, we perform thorough experiments to demonstrate the achievements of TP-GNN framework. We validate our framework on 7 industrial designs, including two RISC-V-based multi-core systems OpenPiton [2] and RocketCore [1], NOVA, LDPC, TATE, JPEG from *OpenCores.org*, and NETCARD from *ISPD 2012 benchmark* [11]. All the 7 benchmarks are synthesized under *TSMC 28nm* technology node using *Synopsys Design Compiler 2015*. We leverage *Cadence Innovus Implementation System v18.1* to perform P&R, and utilize *Synopsys PrimeTime 2018* for signoff analysis. Finally, the TP-GNN framework is implemented in *Python3* with *Tensorflow* library, and the training time is measured on a machine with 2.40 GHz CPU and a NVIDIA RTX 2070 graphics card.

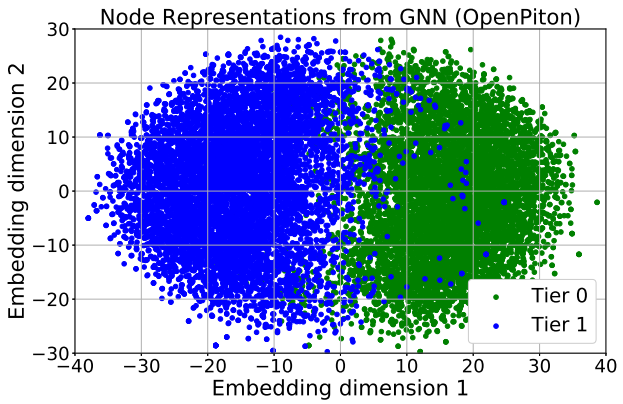

Fig. 3: t-SNE visualizations of the learned node representations from GNN. Each dot represents a cell in the design and is colored by its final tier assignment from Algorithm 3.

Note that for all 3D 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 t-distributed stochastic neighboring embedding [10] (t-SNE) technique to visualize the learned node representations $\{y\} \in R^{128}$ from Algorithm 2 in R^2 with OpenPiton [2]. The visualization result is shown in Figure 3, where we observe that the learned representations form two observable linear separable clusters. Based on the embedded locations in R^2, we further color each dot (cell) by its tier assignment from the weighted k-means algorithm (Algorithm 3) and demonstrate that the algorithm efficiently identify 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 2D, Shrunk2D, and TP-GNN flows on two RISC-V-based designs: OpenPiton [2] (# macros: 28) and RocketCore [1] (# 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 consistently large wirelength saving. Figure 4 further shows the GDS layout comparison, where we observe that TP-GNN introduces fewer cross-macro wires than Shrunk2D.

TABLE II: Performance, area, and energy comparison of Shrunk-2D (S2D) [12] 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 [2]			
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 [1]			
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

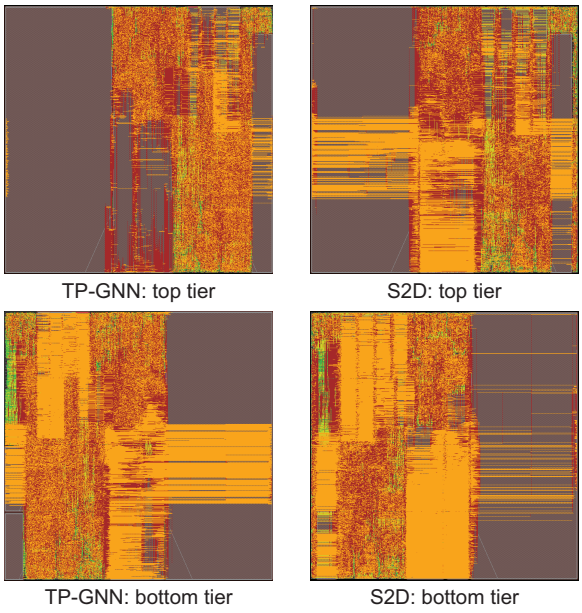

Fig. 4: GDS layouts of OpenPiton [2] using TP-GNN vs. Shrunk-2D [12] flow. TP-GNN flow achieves 7.7% better wirelength.

C. Iso-Performance Comparison

In this experiment, we validate TP-GNN on 6 memory-free designs, where we perform iso-performance comparison between TP-GNN flow and the two state-of-the-art flows. The results are shown in Table III, where we observe that TP-GNN consistently outperforms Shrunk2D and Compact2D in QoR with only a little runtime overhead in the tier partitioning.

D. Why Does GNN Work Better?

Across the two experiments (Table II, Table III), we observe that TP-GNN framework significantly improves the performance (timing) from the state-of-the-art flows in consistent. The main reason is that the original bin-based partitioning

TABLE III: Iso-performance comparison of Shrunk-2D (S2D), Compact-2D (C2D), and TP-GNN flows. Δ_S and Δ_C respectively denotes the percentage difference between TP-GNN vs. S2D and C2D. We report the time spend on tier partitioning in minutes.

Metrics	2D	S2D	C2D	TP-GNN (Δ_S, Δ_C)
LDPC (1.8GHz) (# cells: 43,381)				
WL (m)	1.78	1.61	1.57	1.42 (-11.8%, -9.6%)
power (mW)	362.5	301.4	292.5	271.4 (-10.0%, -7.2%)
# MIV	0	8,955	9,237	7,454 (-25.6%, -27.9%)
WNS (ps)	34	26	20	16 (-38.5%, -20.0%)
partition-time	-	2	2	14
NOVA (1.08GHz) (# cells: 131,737)				
WL (m)	2.33	2.30	2.28	2.17 (-5.7%, -4.8%)
power (mW)	479	220.2	216.9	211.0 (-4.6%, -2.7%)
# MIV	0	16,672	16,935	15,813 (-5.2%, -6.6%)
WNS (ps)	47	28	25	19 (-32.1%, -24.0%)
partition-time	-	5	5	20
TATE (1.37GHz) (# cells: 211,911)				
WL (m)	1.99	1.97	1.95	1.92 (-2.5%, -1.5%)
power (mW)	395.7	398.4	398.6	396.5 (-0.4%, -0.5%)
# MIV	0	56,467	56,820	59,727 (5.8%, 5.2%)
WNS (ps)	36	87	76	31 (-64.4%, -59.2%)
partition-time	-	8	8	22
JPEG (1.53GHz) (# cells: 219,534)				
WL (m)	2.43	2.09	2.12	1.94 (-7.2%, -8.5%)
power (mW)	704.5	674.2	675.9	665.1 (-1.3%, -1.6%)
# MIV	0	27,839	28,231	27,154 (-2.5%, -3.8%)
WNS (ps)	68	49	41	23 (-53.1%, -43.9%)
partition-time	-	8	8	24
NETCARD (1.0GHz) (# cells: 316,137)				
WL (m)	7.82	6.83	6.87	6.11 (-10.5%, -11.1%)
power (mW)	651.7	639.8	639.2	598.9 (-6.4%, -6.3%)
# MIV	0	43,823	43,754	39,987 (-8.8%, -8.6%)
WNS (ps)	56	51	49	26 (-49.0%, -46.9%)
partition-time	-	14	14	42

method ignores the global connections among bins. It only partitions the sub-netlist within a bin. Therefore, critical nets in the projected 2D designs are partitioned randomly. In 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 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 3D 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.

Note 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 vs. 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.

V. Conclusion

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

VI. Acknowledgments

This work is supported by the National Science Foundation under Grant No. CNS 16-24731 and the industry members of the Center for Advanced Electronics in Machine Learning.

References

[1] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, et al. The rocket chip generator. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, 2016.

[2] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, et al. Openpiton: An open source manycore research framework. In ACM SIGARCH Computer Architecture News. ACM, 2016.

[3] G. Berhault, M. Brocard, S. Thuries, F. Galea, and L. Zaourar. 3dip: An iterative partitioning tool for monolithic 3d ic. In 2016 IEEE International 3D Systems Integration Conference (3DIC). IEEE, 2016.

[4] O. Billoint, H. Sarhan, I. Rayane, M. Vinet, P. Batude, C. Fenouillet-Beranger, O. Rozeau, G. Cibrario, F. Deprat, A. Fustier, et al. A comprehensive study of monolithic 3d cell on cell design using commercial 2d tool. In 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2015.

[5] R. C. De Amorim and B. Mirkin. Minkowski metric, feature weighting and anomalous cluster initializing in k-means clustering. Pattern Recognition, 2012.

[6] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems, 2017.

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

[8] 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 Proceedings of the 2018 International Symposium on Physical Design. ACM, 2018.

[9] S. Lloyd. Least squares quantization in pcm. IEEE transactions on information theory, 28(2), 1982.

[10] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. Journal of machine learning research, 9(Nov):2579–2605, 2008.

[11] 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 Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design, pages 161–164. ACM, 2012.

[12] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems, 36(10), 2017.

[13] H. Sarhan, S. Thuries, O. Billoint, and F. Clermidy. An unbalanced area ratio study for high performance monolithic 3d integrated circuits. In 2015 IEEE Computer Society Annual Symposium on VLSI.