ParSGCN: Bridging the Gap Between Emulation Partitioning and Scheduling

Ziyi Wang, Wenqian Zhao, Yuan Pu, Lei Chen, Wilson Wang Kit Thong, Weihua Sheng, Tsung-Yi Ho, *Fellow, IEEE*, and Bei Yu, *Senior Member, IEEE*

***Abstract*—Efficient functional verification is crucial in the very-large-scale integration (VLSI) design flow. Existing processor-based emulation systems suffer from low efficiency due to the gap between partitioning and scheduling during compilation. To address the above concern, we propose ParSGCN, a scheduling-friendly emulation compilation flow that considers the objective of scheduling during partitioning. To incorporate the hard-to-perceive look-ahead information about scheduling, we embed it into a net cut probability distribution, which is easier to utilize. We estimate this probability distribution using a tailored variant of graph convolutional network (GCN) that is trained through a customized loss function and a large dataset of real-world compilation solutions. Additionally, we have developed a set of novel techniques to guide the emulation partitioning process using the estimated probability distribution. The proposed method is integrated into an industrial emulator and evaluated on large-scale designs with up to over 100 million cells. Comprehensive experimental results demonstrate the effectiveness of ParSGCN, showcasing an average improvement of 16.38%, 26.04%, and 19.52% in the best, worst, and median solution quality, respectively, based on 50 runs.**

***Index Terms*—Deep learning, design automation, emulation, graph neural networks, partitioning algorithms.**

I. Introduction

FUNCTIONAL errors and bugs at the register transfer level (RTL) significantly increase design cycles of integrated circuits [1]. This underlines the necessity of comprehensive functional verification in the early stages [2]. However, the significant growth in design size has rendered traditional software-based verification techniques, e.g., formal verification or software simulation, inadequate due to their limitations in scalability and runtime. To counter these issues, hardware emulation has emerged as a prevailing choice for large-scale circuit verification, offering orders of magnitude performance improvement compared to software simulation [3].

Manuscript received 10 March 2024; revised 4 July 2024; accepted 27 August 2024. Date of publication 2 September 2024; date of current version 21 February 2025. This work was supported in part by the Research Grants Council of Hong Kong, SAR, under Grant CUHK14210723 and Grant CUHK14211824. This article was recommended by Associate Editor W. Qian. *(Corresponding author: Bei Yu.)*

Ziyi Wang, Wenqian Zhao, Yuan Pu, Tsung-Yi Ho, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: byu@cse.cuhk.edu.hk).

Lei Chen is with the Noah's Ark Lab, Huawei Technology, Hong Kong, SAR.

Wilson Wang Kit Thong and Weihua Sheng are with Design Automation Lab of Huawei HiSilicon, Hong Kong, SAR.

Digital Object Identifier 10.1109/TCAD.2024.3453199

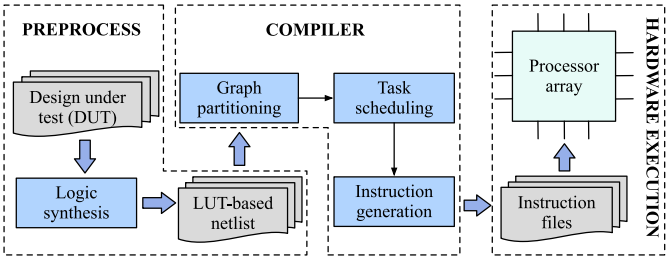


Fig. 1. Overall processor-based emulation flow, whose efficiency heavily depends on the scheduling stage.

Most existing hardware emulation systems fall into two categories: FPGA-based [4], [5], [6] or Boolean processor-based [7], [8]. FPGA-based emulation platform is suitable to support system prototyping when the design is complete, and high emulation efficiency is required, as it can offer faster emulation speed than processor-based platforms. In comparison, processor-based emulators provide superior debugging capabilities, making them more suitable for early-stage design verification where bugs and fixes are frequent [9]. In this work, we target the processor-based emulation, whose overall flow is depicted in Fig. 1. The design-under-test (DUT) is first converted into a 4-input lookup-table (4-LUT)-based netlist via logic synthesis. This is followed by a compilation process that schedules the computation of LUTs to each processor and generates corresponding instructions to be loaded onto the processor array. Once the compilation is complete, the DUT can be verified through extensive hardware emulations using a variety of test patterns.

In the above flow, the efficiency of hardware emulation heavily depends on the scheduling of computations, which determines the total number of clock cycles required to complete one round of emulation [9]. To fully leverage the benefits of parallelism, it is essential to distribute and balance workloads across processors effectively.

However, as the design size increases dramatically, direct scheduling of the entire netlist becomes excessively time-consuming and impractical. Hence, partitioning is usually performed first to confine the candidate processors for each LUT within a specific low-level processor cluster, as shown in Fig. 2. Yet, this divide-and-conquer approach often leads to performance degradation due to the gap between the objectives of the two stages. Specifically, while partitioning aims to minimize bandwidth usage, scheduling strives to minimize the maximum execution step. Unfortunately, these two objectives

1937-4151 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

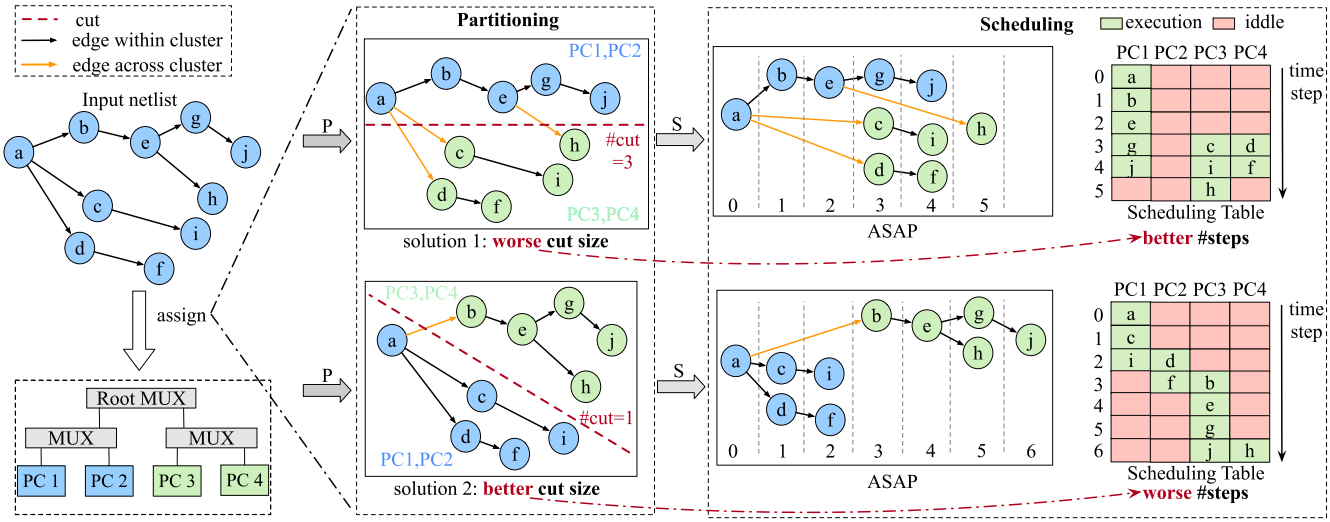


Fig. 2. Example to show that a better partitioning solution does not necessarily leads to better scheduling performance. Assume that the communication cost (delay) between processors within the same cluster is 1, e.g., from PC1 to PC2, and that across clusters is 2, e.g., from PC1 to PC3. It is important to note that the communication cost is simplified for ease of understanding, while the actual hardware topology and costs are much more complicated.

do not always coincide, as illustrated in Fig. 2. Consequently, there arises the necessity to bridge this gap to improve emulation efficiency.

However, bridging the above gap presents challenges due to the dynamic nature of the scheduling and instruction generation process, which can be seen as a closed box process that relies on the partitioning results and the specific hardware characteristics of the emulation system. Therefore, accurately modeling the instruction execution status ahead at the partitioning stage is difficult. For instance, the final post-scheduling critical paths often differ significantly from the pre-scheduling ones. This renders heuristic-based methods, such as avoiding the cutting of pre-scheduling critical paths, ineffective and lacking in generalization ability.

To the best of our knowledge, no prior research has been carried out on scheduling-driven hypergraph partitioning algorithms. This could be due to the fact that scheduling-driven partitioning is significantly more complex than conventional methods, as it necessitates forward-looking information from another NP-hard problem. Renowned traditional hypergraph partitioners, such as hMETIS [10], PaToH [11], and KaHyPar [12], are recognized for their ability to minimize cut size with balancing constraints. However, they do not take scheduling performance into account. There are several recent advancements in the field of hypergraph partitioning. For instance, Specpart [13] introduced a supervised spectral framework to address the two-way hypergraph partitioning problem. Another research effort [14] focused on enhancing the k-way cut size by creating vertex embeddings and applying cut-overlay clustering in hypergraph partitioning. Nonetheless, these recent studies do not account for the context of hardware emulation or a scheduling-driven objective.

To overcome the aforementioned challenge, we embed the hard-to-perceive look-ahead scheduling information into a cut probability distribution, which serves as the bridge between emulation partitioning and scheduling. We construct a graph convolutional network (GCN)-based probability estimator, which is driven by a customized loss function and a large training dataset gathered from an industrial emulator. With the estimated net cut probability serving as look-ahead hints, we develop a novel emulation compilation flow, ParSGCN. This new approach integrates the objective of scheduling during the partitioning phase, leading to improved scheduling performance and increased emulation efficiency.

The major contributions of this article are listed as follows.

1) To the best of our knowledge, we are the first to introduce a machine-learning-driven approach to bridge the gap between emulation partitioning and scheduling.
2) We develop a set of novel techniques to guide the emulation partitioning toward scheduling-friendly.
3) A novel heterogeneous GCN architecture is specifically designed for learning look-ahead scheduling hints.
4) We integrate our method into an industrial tool and conduct comprehensive experiments on large-scale industrial designs to validate the effectiveness of our proposed techniques.

The remainder of this article is organized as follows. Section II gives some preliminaries and the problem formulation. Section III introduces our motivation and the overview of our proposed emulation compilation flow. Section IV presents our scheduling-friendly emulation partitioning scheme. Section V introduces our net cut probability estimator based on a customized GCN variant. Section VI presents experimental evaluations of our proposed framework. Section VII concludes this article.

II. Preliminaries

A. Task Scheduling in Emulation

After logic synthesis, the netlist representation of the DUT is a directed acyclic graph (DAG) in which each cell corresponds to a 4-LUT/flip-flop (FF) with a specific function, and edges represent input/output signals of cells. Within the netlist,

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

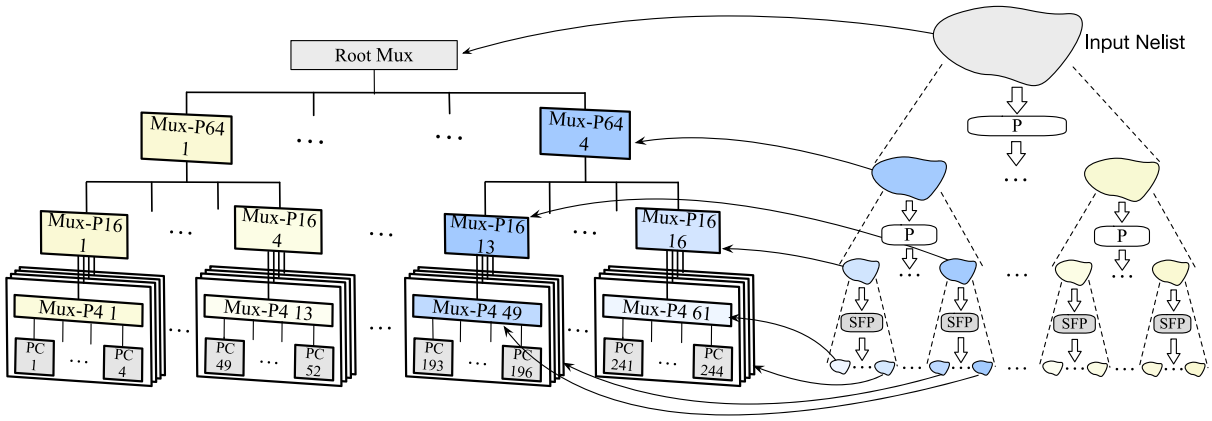


Fig. 3. Example of recursive netlist partitioning following the hardware hierarchy of a simple 256-node emulation system. Real-world emulation systems have more complex hardware hierarchies, and the partitioning may stop at a higher level, such as P64. Here, PC is for processor, mux is for multiplexer, P is for partitioning, and SFP is for scheduling-friendly partitioning.

each cell depends on the output of its preceding cells, meaning that a cell will not proceed during emulation until the output signals of all its predecessors are ready. Consequently, the emulation process is performed level by level in topological order, and the maximum length of instructions is proportional to the number of levels in the netlist.

To manage hardware resources efficiently, many modern mainstream emulation systems use a hierarchical interconnection network [15] as depicted in Fig. 3. Interprocessor data communication is enabled through signal hopping via reconfigurable multiplexer arrays. It is important to note that the more hierarchies a signal hops through, the higher the communication cost. Hence, to reduce communication delays and the number of topological levels, it is preferable to assign cells with dependencies to the same processor or nearby processors within the same low-level cluster. Given an emulation system, the goal of task scheduling is to assign all cells of the netlist to processors in a way that minimizes the number of steps needed to complete one round of emulation.

B. Hypergraph Partitioning in Emulation

Within the emulation compilation process, the partitioning stage divides the netlist into multiple subnetlists, each corresponding to a low-level processor cluster. For instance, the input netlist as shown in Fig. 3 is divided into 64 subnetlists corresponding to the 64 P4 clusters. This partitioning is typically performed recursively, following the hardware hierarchy. Usually, the goal of partitioning in emulation is to minimize intercluster connections to reduce bandwidth usage and data waiting time. Additionally, maintaining workload balance is crucial to ensure efficient utilization, which involves assigning a similar number of cells to each cluster.

Given a netlist (hypergraph), the balanced hypergraph partitioning problem aims to divide it into k approximately equal-sized blocks $P_1, P_2, \ldots P_k$. The objective is to minimize a cost function that focuses on the cut nets, i.e., the nets connecting cells from different blocks. In hardware emulation, the value of k depends on the number of assigned processors. Commonly used cost functions include net-cut and connectivity. The net-cut metric, denoted as $\sum_{e\in\text{cut}} w(e)$, counts the number of cut nets. On the other hand, the connectivity metric, denoted as $\sum_{e\in\text{cut}} w(e)(\lambda(e)-1)$, assigns a weight to each net e with a factor $\lambda(e)-1$, where $\lambda(e)$ represents the number of different blocks connected through net e.

C. Graph Convolutional Network

GCN has emerged as a popular machine learning branch for handling problems with nonstructure data [16]. For several years, GCN-based approaches have shown promising performance in extracting information from graph-structure data [17], [18], [19].

In general, the analytical tasks of GCN mainly fall into three categories: 1) *node-level task:* node regression or node classification; 2) *edge-level task:* edge classification, link prediction, or graph partition; and 3) *graph-level task:* output a graph classification or compact representation at graph level.

The essence of GCN includes node neighbor feature aggregation and message passing. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ denotes the vertex set of $\mathcal{G}$, and $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ denotes the edge set of $\mathcal{G}$ such that $\mathcal{E} \subseteq \mathcal{V}\times\mathcal{V}$. Each node v is encoded with a vector representation h_v. GCN updates each h_v with iterations of aggregating information from neighboring nodes $\mathcal{N}(v)$ and combining with existing feature vector h_v

$$h_v^k = \text{COMBINE}\left(h_v^{k-1}, \text{AGGREGATE}\left(\left\{h_u^{k-1} | u \in \mathcal{N}(v)\right\}\right)\right) \tag{1}$$

where $k = 1, \ldots, L$ denotes each layer of GCN, namely, the iteration number. Many choices on COMBINE$(\cdot)$ and AGGRAGATE$(\cdot)$ can be applied and are crucial to the performance [20]. A naive example of the AGGREGATE$(\cdot)$ function is a multilayer perceptron (MLP) embedding with following `mean` operation. COMBINE$(\cdot)$ can be any function to obtain the next layer's hidden state after aggregation. GCN will stack up L layers of aggregation to increase the depth of the network.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

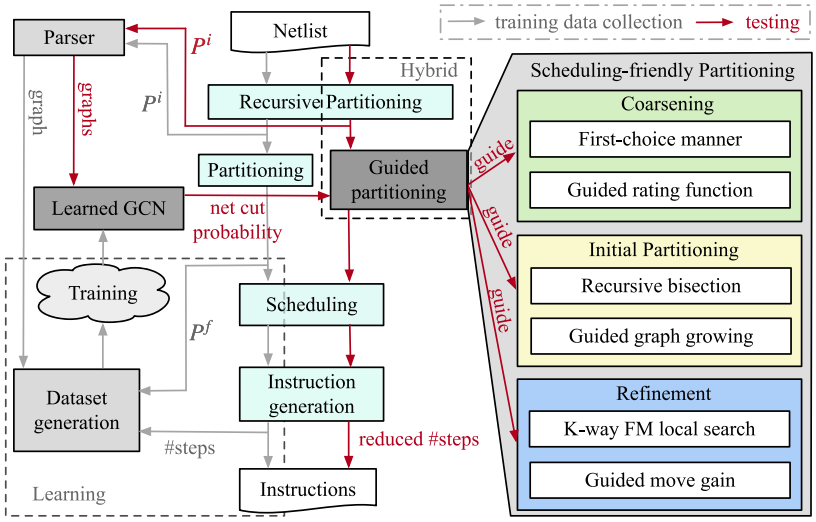


Fig. 4. Overview of our proposed emulation compilation flow with an SFP scheme. Suppose that the target hardware hierarchy for partitioning is h_k, then P_i denotes an intermediate partitioning solution at hierarchy h^{k-1}, and P_f denotes the final partitioning solution at hierarchy h^k.

D. Related Machine Learning Works

Recently, there have been notable efforts to employ machine learning techniques to graph partitioning, which can be classified into reinforcement learning (RL)-based and unsupervised learning-based approaches. RL-driven methods treat partitioning as a sequential decision process, e.g., gradually assigning vertices to different blocks [21]. However, applying RL to our task presents a significant challenge due to the sparse reward issue. The RL model can only receive a positive reward once the entire emulation compilation process is completed, leading to increased training difficulty. On the other hand, unsupervised methods typically formulate partitioning as a vertex classification problem and train the model using loss functions derived from partitioning objectives, e.g., minimizing the normalized cut [22]. This is also impractical for our scenario due to the difficulty in accurately modeling the final scheduling performance.

E. Problem Definition

Problem 1 (Scheduling-Friendly Emulation Compilation): Given an emulation system, our goal is to improve its emulation efficiency by incorporating scheduling objectives, i.e., minimizing the maximum instruction execution step, into its partitioning process.

III. Motivation and Flow Overview

As mentioned before, effectively perceiving emulation scheduling performance ahead during the netlist partitioning stage can be challenging. This difficulty arises from the substantial disparity between pre-scheduling estimations and actual scheduling performance, which can be attributed to the intricate nature of the dynamic scheduling process. For example, it is relatively straightforward to estimate node slacks and identify critical paths based on an intermediate partitioning solution. However, the critical paths estimated before scheduling may differ significantly from the final post-scheduling critical paths. Consequently, optimizing the pre-scheduling critical paths may not necessarily result in improved scheduling performance. To address these concerns, it becomes crucial to acquire general knowledge about scheduling that can be shared across different partitioning solutions and designs, and adaptable to the dynamic nature of scheduling.

In order to achieve this objective, we propose to capture underlying knowledge about scheduling from a diverse range of partitioning solutions, rather than relying solely on a single "optimal" solution that leads to minimized maximum scheduling step. This necessitates a deeper exploration of the correlation between emulation partitioning and scheduling. To this end, we conducted statistical experiments aimed at identifying common characteristics among high-quality emulation partitioning solutions. Our analysis yielded an intriguing observation: certain nets consistently appear in the cut set of good solutions, while they are seldom found in the cut set of inferior solutions, and vice versa. This suggests a strong correlation between the cut frequency of certain nets and the subsequent scheduling performance.

Based on the above observation, we introduce ParSGCN, a novel emulation compilation flow with enhanced scheduling performance, as depicted in Fig. 4. Our flow incorporates a scheduling-friendly partitioning (SFP) scheme that considers scheduling objectives during partitioning, guided by the expected net cut probabilities in high-quality solutions (Sections IV-A–IV-C). To estimate the net cut probability p(e) for each net e, we employ a customized GCN model that generates powerful net embeddings (Section V-B). The model is trained using a tailored loss function (Section V-A) and a large dataset of solutions collected from the emulation flow.

To enhance scalability, we have adopted a hybrid partitioning strategy that applies the SFP scheme only at the last target hardware hierarchy, as depicted in Fig. 3. This strategy enables the GCN model to effectively handle extremely large-scale circuits, including those with over 100 million nodes, either in parallel or sequentially. Otherwise, it would be impractical to accommodate the entire large-scale circuit on a single GPU. Furthermore, the learning difficulty at this last

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

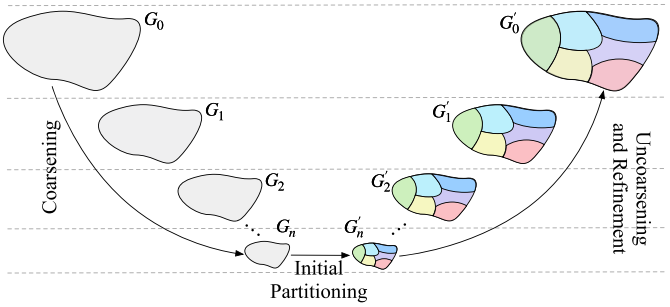


Fig. 5. Multilevel partitioning framework, where we denote the input graph after k rounds of coarsening as G_k and the corresponding partitioned graph at the same coarsening level as G'_k.

level is relatively lower compared to previous levels, as it is closest to the scheduling stage. Consequently, the GCN model can incorporate features that are more closely correlated to the subsequent scheduling process, thereby enhancing the accuracy of estimations.

IV. Scheduling-Friendly Partitioning Scheme

To enhance scheduling performance and emulation efficiency, we propose a novel emulation partitioning scheme guided by net cut probability. Our algorithm aims to properly assign cells to each processor cluster considering both the intercluster bandwidth usage and subsequent scheduling objectives.

Our partitioner is built upon the well-known multilevel partitioning approach [23] shown in Fig. 5, which comprises three main components: 1) coarsening; 2) initial partitioning; and 3) refinement. The approach begins with the coarsening of the input graph (netlist), where sets of vertices (cells) are contracted and replaced with single vertices in the new hypergraph. Once the hypergraph size is sufficiently reduced, initial partitioning of the coarsest hypergraph is carried out. Errors from incorrectly assigned cells can be rectified during the uncoarsening and refinement phase, where the contractions are reversed, and optimization techniques, such as local search, are utilized.

Before diving into the details, we give some related notations and definitions. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a set of n vertices $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, and a set of m edges $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$, where each edge connects two vertices. Furthermore, a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a generalization of graphs where a hyperedge or net $e \in \mathcal{E}$ may connect more than two vertices. A netlist can be naturally represented as a hypergraph by taking each cell as a vertex. A vertex v_i is incident to a net e if $v_i \in e$. $I(v_i)$ denotes the set of all incident nets of v_i, and $I(v_i, v_j) = I(v_i) \cap I(v_j)$ denotes the set of common incident nets of v_i and v_j. $\phi(e, i)$ denotes the number of incident vertices of a net e belonging to a partitioning block P_i. The set $\Gamma(v_i) = \{v_j | \exists e \in \mathcal{E} : \{v_i, v_j\} \subseteq e\}$ denotes the neighbors of v_i. Sometimes, the hypergraph is weighted such that each vertex v is assigned with a weight $c(v) \in R^+$ and each net e is assigned with a weight $w(e) \in R^+$. Furthermore, p(e) represents the estimated cut probability of a net e.

Algorithm 1 Guided Hypergraph Coarsening

Input: $G_0 = (V, E, W, P)$: input hypergraph; t: the target size of the coarsest graph;
Output: multilevel coarsened graphs $G_1, G_2 \cdots G_n$;

```
s ← |V|, i ← 0, V_0 ← V;
while s > t do
    Shuffle(V_i);
    for u ∈ V_i do
        if not_coarsened(u) then
            v* ← argmax_{v∈Γ(u),not_coarsened(v)}(r(u, v));
            contract(u, v*) to update V_i;
        end if
        s ← s − 1;
    end for
    update (E,W,P) to get G_i = (V_i, E_i, W_i, P_i);
    i ← i + 1;
end while
```

A. Guided Netlist Coarsening

The goal of netlist coarsening is to reduce the netlist size by merging vertices (cells) together. In each level of the coarsening hierarchy, the contracted vertices on the current level are forced to be assigned to the same block in the following levels. This has a significant impact on emulation scheduling performance, as it largely determines which cells are allocated to the same processor cluster. To bridge the gap between emulation partitioning and scheduling, it is of great importance to carefully design the coarsening algorithm.

Prevailing multilevel coarsening algorithms employ either matching [10], [23] or clustering [24], [25] on each level of the coarsening hierarchy. Matching-based coarsening methods consider pair-wise vertex contraction while clustering-based approaches group vertices and conduct group-wise contraction. In our framework, we choose the matching-based coarsening strategy due to its high efficiency and good scalability, which is crucial for dealing with large-scale circuits.

The fundamental distinction between different coarsening algorithms is the rating function used to determine which vertices should be matched or clustered together. An ideal rating function should encourage vertex contractions that are consistent with high-quality partitioning solutions. In terms of our scenario, the quality of a solution is measured by the scheduling performance instead of the straightforward partitioning metrics like cut size of connectivity. This hinders the direct application of previous coarsening algorithms and raises the demand for designing a customized rating function. Nevertheless, the customization is nontrivial as the scheduling performance is hard to perceive at the partitioning stage.

In our framework, the knowledge about scheduling performance is embodied in the net cut probability distribution, which is more concrete and easier to utilize. Specifically, we can prioritize vertices connected by likely-not-to-cut nets while avoiding the contraction of likely-to-cut nets. Based on the above rule, we design a guided rating function $r(v_i, v_j)$ for a pair of vertex (v_i, v_j) as (2), where β is a hyperparameter. The second term in (2) represents weighted heavy edge rating that

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

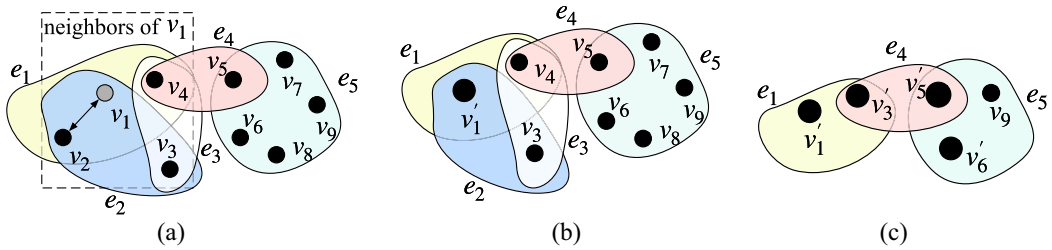


Fig. 6. Example of our netlist coarsening algorithm. Suppose the vertices are visited in the order $v_1, v_2, v_3, \ldots, v_9$, and v_2 has the highest rating with v_1 among the neighbors of v_1. (a) Find contraction partner for v_1. (b) Contract v_1 with its partner. (c) Final coarsened hypergraph.

favors vertex pair (v_i, v_j) connected by a large number of heavy nets with small size and low cut probability. Meanwhile, the first term discourages the contraction of vertex pairs connected by nets with high cut probability

$$r(v_i, v_j) = \left(\beta \exp\left(1 - 2\max\left\{p(e) | e \in I(v_i, v_j)\right\}\right) - \beta\right) + \sum_{e \in I(v_i, v_j)} \frac{w(e)(1 - p(e))}{|e| - 1}. \quad (2)$$

Following KaHyPar [12], our algorithm works in the first choice (FC) manner and runs for multiple rounds, as illustrated in Algorithm 1. Within each round, the vertices are traversed in random order, and we identify the contraction partner for each vertex v_i as its neighboring vertex $v_j \in N(v_i)$ with the highest rating $r(v_i, v_j)$, as shown in Fig. 6(a). The selected vertex pair (v_i, v_j) is then immediately merged into a new vertex with a weight of $w(v_i) + w(v_j)$. Once this occurs, both v_i and v_j are locked and excluded from subsequent contractions in this round. A round stops once all vertices have been visited, and the contracted vertices form the vertex set for the next round. The whole coarsening process is terminated when either the number of vertices drops below a certain threshold, or no eligible vertex remains for consideration.

At the end of each round, we need to update the nets for the next round since the coarsening process may lead to changes in the net set. For instance, some nets can be entirely removed if they shrink to size one, e.g., e_3 in Fig. 6. And these single-vertex nets need to be removed from the net set of the coarsened graph. Meanwhile, some nets may share the same set of incident vertices after coarsening, e.g., e_1 and e_2 in Fig. 6, which we refer to as parallel nets. Each group of parallel nets can be merged into a single net with recalculated weight and cut probability. The weight of a merged net is given by the sum of the original ones while its cut probability is updated using the averaged value.

B. Guided Initial Partitioning

During initial partitioning, the vertices of the coarsest graph are evenly distributed to k processor clusters, i.e., the total vertex weights of each cluster are roughly the same. This is critical for efficient hardware emulation as it helps in balancing the workload across different clusters. It is also important to note that a balanced initial partitioning solution at the coarsest level also infers a balanced solution on any subsequent levels. In fact, the initial partitioning process largely impacts the efficacy of the subsequent refinement process and the final solution quality. Therefore, an ideal initial partitioning algorithm should provide a high-quality solution while maintaining the balance constraint.

Algorithm 2 Guided Hypergraph Bipartitioning

Input: $G = (V, E, W, P)$: coarsest hypergraph;
Output: bipartition of G;
1: $v_b \leftarrow argmax_{v \in V}(s(v))$;
2: $v_s \leftarrow BFS(v_b)$;
3: $P_1 \leftarrow \{v_s\}$, $P_2 \leftarrow V \setminus \{v_s\}$;
4: **while** `not_balance`(P_1, P_2) **do**
5: $\quad v^* \leftarrow argmax_{v \in P_2}(g(v))$;
6: $\quad$ move v^* from P_2 to P_1;
7: $\quad$ **for** $u \in \Gamma(v^*) \cap P_2$ **do**
8: $\quad\quad$ update g(u);
9: $\quad$ **end for**
10: **end while**

As mentioned before, the quality of a partitioning solution in our framework is measured by the final scheduling performance, which is evaluated and reflected by the net cut probability distribution. In this part, we present a customized initial partitioning algorithm, aiming to keep in accordance with high-quality solutions while deviating from bad solutions. Our algorithm follows the recursive bisection paradigm, which is known to be superior to direct k-way methods for solving the hypergraph partitioning problem [12]. The approach involves computing a bipartition of the coarsest hypergraph and then recursively applying bisection to each resulting block. After $log_2(k)$ rounds, the vertices are evenly distributed to k processor clusters.

The bipartition algorithm we employ, as depicted in Algorithm 2, is based on the greedy hypergraph growing (GHG) technique [25], which has been widely used in hypergraph partitioning algorithms. GHG works by iteratively expanding a subgraph around a seed vertex until a valid bipartition is obtained. The choice of the seed vertex is crucial for the success of the algorithm [25], as it determines the initial configuration of the growing process.

To align the processor cluster assignment with the scheduling objective, we suggest choosing a vertex far from the expected cut boundary of high-quality solutions (with superior scheduling performance), as shown in Fig. 7. This helps prevent the growing process from unintentionally crossing the boundary and deviating from the desired high-quality solutions. To determine the seed vertex, a scoring mechanism

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

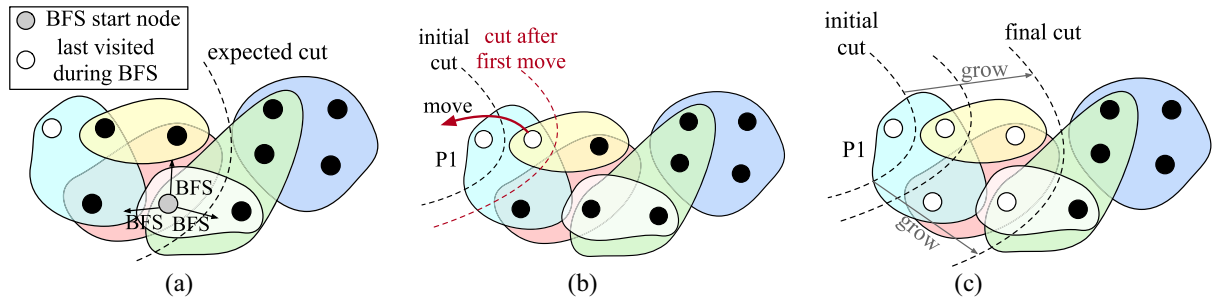


Fig. 7. Example of our initial bipartition algorithm. We start by selecting a seed vertex far from the expected cut boundary, as shown in (a). Then, we gradually move vertices to the growing cluster according to (4) until balance, as shown in (b). (c) Final bipartition.

is used. The incident net cut probability score is computed for each vertex v as (3), which represents the likelihood of a vertex being adjacent to the expected cut boundary. Vertices with higher scores are more likely to be near the cut boundary. By starting a breadth-first-search (BFS) from the vertex with the highest score and selecting the last visited vertex as the seed vertex, the algorithm ensures that the seed vertex is located at a considerable distance from the expected cut boundary

$$s(v) = \sum_{e \in I(v)} p(e). \tag{3}$$

Once the seed vertex is identified, the graph-growing process begins with a subgraph P_1 containing only the seed vertex, and the remaining vertices form another subgraph P_2. The algorithm maintains a priority queue of remaining vertices, organized based on a gain function written as (4). The gain function evaluates the score of each remaining vertex $v \in P_2$, considering its connectivity and cut probability with the growing subgraph P_1. This prioritization ensures that vertices with strong connections to the growing subgraph are considered first. At each step, the vertex with the maximum gain is removed from P_2 and added to the growing subgraph P_1. Subsequently, the gains of its neighboring vertices which are currently in the priority queue are updated, and those not in the priority queue are inserted. This growing procedure continues until a predetermined bipartition balance criterion is met, e.g., the total vertex weights of P_1 equals that of the remaining part

$$g(v) = \sum_{e \in I(v): \phi(e,P_1) \neq 0} w(e)(1 - p(e)). \tag{4}$$

C. Guided Refinement

We now turn to our guided refinement algorithm, which aims to eliminate errors caused in the previous two stages by moving the wrongly assigned cells to their expected processor cluster.

Our implementation follows the k-way Fiduccia–Mattheyses (FM) local search paradigm [26] that progressively improves a solution by applying localized changes, e.g., relocating a vertex from one processor cluster to another. At each step, the change that yields the maximum gain in the objective function, i.e., reduced intercluster connectivity, while not breaking the workload balance constraint, is selected, thereby allowing the search to eventually converge to an enhanced solution. Usually, the refinement runs for several passes to achieve substantial improvement.

To align refinement with scheduling objectives, we incorporate the estimated net cut probability into the gain function. Let $m_{i,j}(v)$ denote the operation of moving a vertex v from P_i to P_j, then the gain $g_{i,j}(v)$ of $m_{i,j}(v)$ is defined as (5), where γ is a hyperparameter. The primary principle here is to ensure a reduction, or at least no worsening, of intercluster connectivity. Otherwise, the local search paradigm becomes meaningless. Therefore, we opt to embed the estimated net cut probabilities as extra penalty terms so that a solution with positive total gain will always lead to reduced intercluster connectivity. Meanwhile, the added penalty can help avoid vertex relocations that are inconsistent with the subsequent scheduling stage. Specifically, we reduce the gain of a move $g_{i,j}(v)$ if it leads to a decrease in the connectivity $\lambda(e)$ of likely-to-cut nets ($\phi(e,i) = 1$ and $p(e) > 0.5$) or if it leads to increasing connectivity $\lambda(e)$ of likely-not-to-cut nets ($\phi(e,j) = 0$ and $p(e) < 0.5$)

$$\begin{aligned} g_{i,j}(v) = &\sum_{\substack{e \in I(v) \ : \\ \phi(e,i)=1}} (w(e) - \gamma \max\{0, p(e) - 0.5\}) \\ &- \sum_{\substack{e \in I(v) \ : \\ \phi(e,j)=0}} (w(e) + \gamma \max\{0, 0.5 - p(e)\}). \end{aligned} \tag{5}$$

Once we decide to perform a move $m_{i,j}(v)$, i.e., moving vertex v from P_i to P_j, we first update the gains for all moves of unlocked neighboring vertices of v following the delta-gain updates [25]. After the gain updates, the decided move $m_{i,j}(v)$ is performed, and the moved vertex v is then locked and excluded from any further moves in the current pass. The local search continues until all vertices are locked, after which we reverse all moves until we reach the state that yields the maximum total gain throughout the entire pass. Subsequently, all vertices are unlocked again to initiate a new pass, and the algorithm is repeated until no further improvement is attained.

V. Learning Expected Net Cut Probability

The estimated net cut probability serves as the cornerstone of our scheduling-friendly emulation partitioning scheme. In this work, we develop a machine learning-based net cut probability estimator trained using a large dataset collected from an industrial emulation flow. Our estimator employs a unique loss function and a novel GCN architecture.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

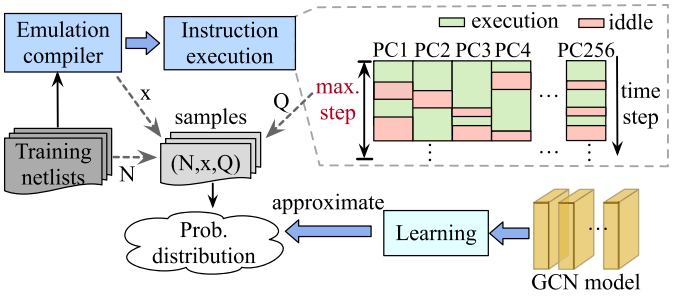


Fig. 8. Illustration of our learning flow.

A. Customized Loss Function

Taking inspiration from Neural Diving [27], we propose to learn the net cut probability from both good and bad partitioning solutions rather than solely from good ones. Our learning process begins with sampling numerous feasible solutions, as depicted in Fig. 8. Specifically, we run the emulation compilation flow multiple times for each training circuit design, collecting the partitioning solution and corresponding scheduling performance measured by the maximum instruction execution step. Note that the sampled solutions need not be near-optimal, which significantly reduces the difficulty and effort needed. Although potentially time-consuming, this data collection step is performed only once and outside the training loop. Let $x = \{x_1, \ldots, x_n\}$ represent a partitioning solution for a circuit consisting of n nets, where x_k is a binary variable indicating whether the kth net e_k is cut ($x_k = 1$) or not ($x_k = 0$). We then define the following probability distribution over the sampled partitioning solutions to a given circuit netlist $\mathcal{N}$:

$$p(x|\mathcal{N}) = \prod_{k=1}^{N_e} p(x_k) = \frac{\exp(-Q(x))}{\sum_{x' \in X} \exp(-Q(x'))}. \tag{6}$$

Here, N_e denotes the number of nets in $\mathcal{N}$, and $Q(x)$ depicts the quality of solution x measured by the maximum instruction execution step (smaller the better). Obviously, solutions with superior scheduling performance are assigned higher probabilities. Furthermore, we make a simplified assumption to reduce computational complexity: each net follows the same Bernoulli distribution and all nets are conditionally independent of each other. It is worth mentioning that there are various alternative options for modeling the probability distribution [28], [29], e.g., Gaussian distribution, which may be explored in future work.

Given the probability distribution $p(e)$ on training designs, our target is to approximate it using a generative model $p_\theta(e)$ parameterized by θ. The model seeks to capture the net cut frequency in high-quality solutions, or in other words, to maximize the probability of high-quality solutions. Mathematically, our learning objective can be formulated as (7), where N_c is the number of training circuits, N_s is the number of sampled solutions for each design, $\mathcal{N}^i$ is the ith netlist, $x^{i,j}$ denotes the jth sampled solution to the ith netlist, and α is a hyperparameter

$$\max \sum_{i=1}^{N_c} \sum_{j=1}^{N_s} \frac{\exp\left(-\alpha Q(x^{i,j})\right)}{\sum_{k=1}^{N_s} \exp\left(-\alpha Q(x^{i,k})\right)} p_\theta\left(x^{i,j}|\mathcal{N}^i\right). \tag{7}$$

TABLE I
GCN INPUT FEATURES

Name	Size	Description
(cell) type	5	one-hot encoding that depicts cell type: LUT, memory, I/O, register.
(cell) in degree	1	number of incoming edges to a cell
(cell) out degree	1	number of outgoing edges from a cell
(cell) is_boundary	1	whether the cell is adjacent to the cut boundary resulted from partitioning in previous hardware hierarchies
(cell) slack	1	slack of a cell
(cell) weight	1	contracted cells w. larger weights
(net) weight	1	contracted parallel nets w. larger weights

The loss function can then be defined as

$$L = -\sum_{i=1}^{N_c} \sum_{j=1}^{N_s} \frac{\exp\left(-\alpha Q(x^{i,j})\right)}{\sum_{k=1}^{N_s} \exp\left(-\alpha Q(x^{i,k})\right)} \log\left(\prod_{k=1}^{N_e^i} \left(p_\theta\left(x_k^{i,j}\right) + \epsilon\right)\right) \tag{8}$$

where N_e^i denotes the number of nets in the ith netlist, and $x_k^{i,j}$ is the binary cut variable for the kth net in the solution $x^{i,j}$. Here, we apply the log probability to avoid vanishing loss value and add a small term $\epsilon = 1 \times 10^{-10}$ to deal with the zero probability issue. Guided by (8), our model learns to maximize the cut probability of nets that frequently appear in high-quality solutions.

B. Customized GCN Architecture

Due to the graph nature of netlists, the emerging GCN [30], [31], [32] becomes our FC to generate the cut probability for each net. In this section, we propose a customized GCN architecture better suited to our task than the vanilla GCN.

To effectively represent a netlist, we convert it into a heterogeneous graph structure comprising two types of vertices and five types of edges, as depicted in Fig. 9. Considering the significance of direction in scheduling, we distinguish messages from various directions for each net vertex. Our designed graph structure facilitates information aggregation related to both partitioning and scheduling, ultimately resulting in powerful net embeddings. For input features, we select ones that are relevant to either partitioning or scheduling, as listed in Table I.

Given the graph representation of a circuit design, we employ a customized GCN model to generate the net cut probabilities. Specifically, we design a 2-level directed message-passing scheme to propagate information on the constructed graph, as shown in Fig. 9. Let $e \in V_n$ represent a net vertex, $\Gamma_c^d(e)$ and $\Gamma_c^s(e)$ denote the set of drive/sink cells of e, respectively, and h_v^0 denote the input feature of vertex v. Our message-passing scheme begins by aggregating incident cell messages to the corresponding net vertices, which can be written as (9). Here, σ is an activation function, W_1 is a learnable weight matrix, M denotes the mean operation followed by an MLP, and || represents a concatenating operation

$$h_e = \sigma\left(W_1\left[h_e^0 \,||\, M_1^d\left(\left\{h_v^0 : v \in \Gamma_c^d(e)\right\}\right) \,||\, M_1^s\left(\left\{h_v^0 : v \in \Gamma_c^s(e)\right\}\right)\right]\right). \tag{9}$$

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

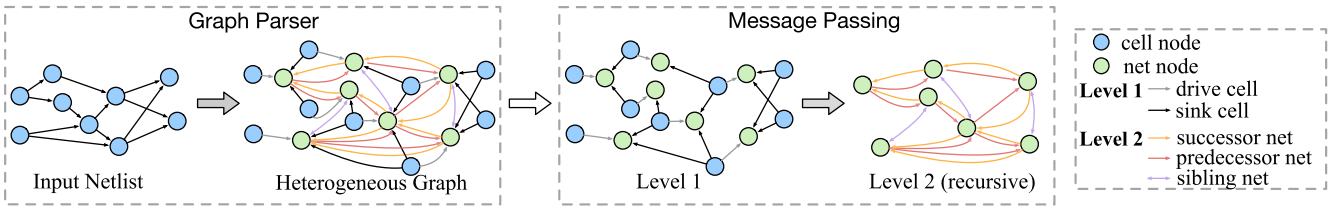


Fig. 9. Transformation from netlist to our designed graph structure. Our GCN follows a 2-level message-passing scheme that first aggregates cell information to nets and then propagates messages between nets.

The generated net embeddings then serve as input features to $Level_2$, where we conduct an iterative message propagation between the net vertices. Similarly, let $\Gamma_n^p(e)$, $\Gamma_n^{sc}(e)$, and $\Gamma_n^{sl}(e)$ denote the set of predecessor/successor/sibling net vertices of a net vertex e, respectively. Then, the message passing in the ith iteration on $Level_2$ can be described as (10), as shown at the bottom of the page.

The final net embeddings after k layers are then fed into an MLP with an output dimension of one and Sigmoid activation to get the estimated net cut probability $p(x_e)$ for each net e as (11), where $x_e = 1$ represents e is cut and $p(x_e = 0) = 1 - p(x_e = 1)$, θ represents the learnable weights of all the W_1, W_2, and MLP in (9)–(11)

$$p_\theta(x_e = 1) = \frac{1}{1 + \exp\left(-\text{MLP}\left(h_e^k\right)\right)}. \tag{11}$$

VI. Experiments

A. Settings

Our scheduling-friendly emulation partitioning scheme is implemented using C++, with corresponding hyperparameters set as $\beta = 2$ [see (2)], and $\gamma = 2$ [see (5)] for all cases. Moreover, our GCN-based estimator is built using the popular deep learning libraries, DGL [33] and Pytorch [34]. Regarding the learning hyperparameters, α in (8) is set to 10, the dimension of net embeddings is set to 64, and all the MLPs are instantiated with three layers and a hidden dimension of 128. Our framework is trained/evaluated on a Linux system with 18 Intel Xeon Gold 6154 cores (3.00 GHz) and a single GPU.

A dataset composed of 30 real-world industrial circuit designs is prepared for training and testing, as detailed in Table II. The dataset encompasses a broad spectrum of applications, including but not limited to communication, control, cryptography, digital signal processing (DSP), and processor designs. Each design is unique, with no two designs originating from closely similar scenarios such as neural networks with varying kernel sizes. Moreover, the dataset ensures that there are no overlapping designs, meaning one case is not a subset of another. Due to commercial confidentiality, we hide the design names and use case1, case2, ..., and case30 instead. We pick five designs with around five million cells to form the training dataset and use the left 25 designs for testing. The construction of the training dataset involved running an industrial emulator and randomly generating 10 000 emulation compilation solutions for each training design, a process that took about a week. Following this, a GCN model was trained on the generated dataset for 1000 epochs, a process that took roughly 6 h. It is important to note that the training phase is a one-time process for a given hardware emulation platform. Once the model has been trained, it can be used repeatedly for any new unseen designs, eliminating the need for further fine-tuning.

Once the model is well trained, it is incorporated into the industrial emulator using our SFP scheme and evaluated on unseen testing designs. For commercial security reasons, we refer to the emulator as "industrial tool." The baseline industrial tool consists of a massive array of interconnected Boolean processors that are organized hierarchically as depicted by Fig. 3. The communication cost between different hierarchical levels is determined based on the characteristics of the hardware system.

The number of processors assigned to a specific design is automatically determined in our experiments using a usage ratio $R = (N_m/N_a)$. Here, N_m represents the minimal number of processors needed to accommodate all the LUTs in the netlist, and N_a represents the number of actually assigned processors. Here, N_m represents the minimal number of processors needed to accommodate all the LUTs in the netlist, and N_a represents the number of actually assigned processors. Generally, a margin is necessary to ensure validity, hence R is set to a value less than 1. In our experiment, the usage ratio is set to 0.8, indicating that we maintain a margin of 20%.

B. Results on Industrial Designs

We first evaluate our proposed method on the real-world industrial designs. The performance comparison between the original industrial tool and the updated version guided by our approach is presented in Table II. Note that the original industrial tool has already employed a set of heuristic methods during compilation to address the gap between partitioning and scheduling, which are orthogonal to our proposed techniques. To ensure fairness, we conducted 50 runs of both emulation compilation flows for each circuit design using random seeds, and reported the best, worst, and median solution quality (measured by the maximum execution step) found during

$$h_e^i = \sigma\left(W_2^i\left[h_e^{i-1} || M_2^{i,p}\left(\left\{h_u^{i-1} : u \in \Gamma_n^p(e)\right\}\right) || M_2^{i,sc}\left(\left\{h_u^{i-1} : u \in \Gamma_n^{sc}(e)\right\}\right) || M_2^{i,sl}\left(\left\{h_u^{i-1} : u \in \Gamma_n^{sl}(e)\right\}\right)\right]\right) \tag{10}$$

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

TABLE II
PERFORMANCE ON INDUSTRIAL DESIGNS, IN COMPARISON TO THE INDUSTRIAL EMULATION COMPILATION FLOW WITHOUT INTEGRATING OUR METHOD. HERE, IND. REPRESENTS THE INDUSTRIAL EMULATION TOOL, PARSGCN REPRESENTS THE EMULATION TOOL THAT INCORPORATES OUR METHOD, *#steps* DENOTES THE TOTAL STEPS (CLOCK CYCLES) NEEDED TO COMPLETE THE EMULATION OF A DESIGN. INFER. IS FOR INFERENCE, WHICH GIVES THE RUNTIME OF CONSTRUCTING THE HETEROGENEOUS GRAPH AND PREDICTING THE CUT PROBABILITY AND COMP. IS FOR COMPILATION. FOR COMMERCIAL SECURITY REASONS, WE NORMALIZE ALL THE *#steps* AND RUNTIME VALUES BY THE CORRESPONDING PERFORMANCE OF THE INDUSTRIAL TOOL

Design		Size ($\times 10^6$)		#instruction execution steps						Runtime			
		#nodes	#nets	min		max		median		Ind.	ParSGCN		
				Ind.	ParSGCN	Ind.	ParSGCN	Ind.	ParSGCN		Infer.	Comp.	total
Train	case1	3.48	3.41	1.000	0.820	1.000	0.467	1.000	0.665	1.000	0.004	1.087	1.102
	case2	3.66	3.52	1.000	0.880	1.000	0.566	1.000	0.554	1.000	0.008	1.105	1.122
	case3	3.70	3.60	1.000	0.818	1.000	0.405	1.000	0.494	1.000	0.007	1.208	1.123
	case4	4.45	4.39	1.000	0.664	1.000	0.983	1.000	0.994	1.000	0.027	1.131	1.163
	case5	6.74	6.71	1.000	0.905	1.000	0.628	1.000	0.657	1.000	0.033	1.105	1.146
	AVG. Train	4.41	4.33	1.000	0.817	1.000	0.610	1.000	0.673	1.000	0.016	1.127	1.131
Test	case6	0.52	0.53	1.000	0.950	1.000	0.788	1.000	0.886	1.000	0.019	1.106	1.143
	case7	1.07	1.06	1.000	0.957	1.000	0.867	1.000	0.891	1.000	0.042	1.063	1.121
	case8	2.95	2.94	1.000	0.952	1.000	0.774	1.000	0.948	1.000	0.004	1.140	1.148
	case9	3.09	3.03	1.000	0.613	1.000	0.445	1.000	0.594	1.000	0.025	0.969	0.998
	case10	3.20	3.18	1.000	1.000	1.000	0.924	1.000	0.991	1.000	0.077	0.732	0.814
	case11	3.22	3.13	1.000	0.547	1.000	0.416	1.000	0.503	1.000	0.048	1.075	1.130
	case12	3.25	3.16	1.000	0.853	1.000	0.591	1.000	0.529	1.000	0.052	1.103	1.163
	case13	3.63	3.17	1.000	0.997	1.000	0.859	1.000	0.968	1.000	0.050	1.194	1.253
	case14	4.62	4.53	1.000	0.993	1.000	0.839	1.000	0.920	1.000	0.079	1.077	1.077
	case15	9.56	9.61	1.000	0.898	1.000	0.618	1.000	0.858	1.000	0.024	0.943	0.975
	case16	10.35	10.62	1.000	1.000	1.000	0.691	1.000	0.931	1.000	0.026	1.144	1.180
	case17	10.40	10.41	1.000	0.979	1.000	0.983	1.000	0.980	1.000	0.028	1.040	1.078
	case18	11.27	11.28	1.000	0.991	1.000	0.660	1.000	0.926	1.000	0.019	1.018	1.045
	case19	11.79	11.81	1.000	0.984	1.000	0.627	1.000	0.836	1.000	0.021	0.933	0.963
	case20	17.18	17.30	1.000	0.921	1.000	0.975	1.000	0.994	1.000	0.015	0.927	0.951
	case21	17.57	17.94	1.000	0.600	1.000	0.803	1.000	0.539	1.000	0.014	1.098	1.121
	case22	17.69	18.32	1.000	0.626	1.000	0.433	1.000	0.499	1.000	0.013	0.914	0.936
	case23	18.67	19.21	1.000	0.948	1.000	0.589	1.000	0.909	1.000	0.014	1.101	1.124
	case24	20.79	21.30	1.000	0.671	1.000	0.969	1.000	0.584	1.000	0.011	1.195	1.215
	case25	21.08	21.63	1.000	0.995	1.000	0.898	1.000	0.972	1.000	0.015	1.082	1.107
	case26	26.62	26.67	1.000	0.896	1.000	0.789	1.000	0.755	1.000	0.010	1.098	1.117
	case27	29.27	29.37	1.000	0.496	1.000	0.546	1.000	0.504	1.000	0.009	0.997	1.014
	case28	39.90	40.43	1.000	0.582	1.000	0.723	1.000	0.804	1.000	0.007	1.002	1.018
	case29	138.63	140.02	1.000	0.942	1.000	0.737	1.000	0.832	1.000	0.002	1.188	1.197
	case30	181.96	184.15	1.000	0.577	1.000	0.948	1.000	0.995	1.000	0.004	1.102	1.119
	AVG. Test	17.76	17.95	1.000	0.836	1.000	0.740	1.000	0.806	1.000	0.025	1.050	1.080

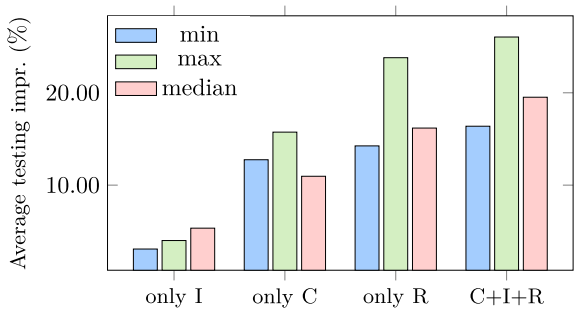


Fig. 10. Performance comparison between single guiding techniques and the full version, where C is for coarsening, I is for initial partitioning, and R is for refinement.

TABLE III
PERFORMANCE COMPARISON BETWEEN OUR CUSTOMIZED GCN AND THE NAIVE GCN, MEASURED BY THE PERFORMANCE IMPROVEMENT OVER THE INDUSTRIAL TOOL

Design	naive GCN			our GCN		
	min	median	max	min	median	max
AVG. train	13.09	24.5	26.65	18.23	39.07	32.73
AVG. test	9.08	14.27	11.54	16.38	26.04	19.52

the 50 runs. As indicated in Table II, our method exhibits significant improvements in solution quality. On average, it enhances the best solution quality by 18.27% and 16.38% for training and testing designs, respectively, and up to over 50%. Furthermore, the worst solution quality is improved by 39.01% and 26.04% for training and testing designs, respectively. These results clearly validate the effectiveness of our proposed framework. Additionally, our methods yield a substantial enhancement in the median solution quality, with an average improvement of 32.73% for training designs and 19.52% for testing designs. This indicates that incorporating our method makes the industrial emulation compilation flow more stable.

We further conduct a runtime analysis to assess the additional overhead incurred by incorporating our SFP scheme, which was measured by the total compilation time. Similarly, we calculate the average runtime based on 50 runs for each design. As shown in Table II, the runtime overhead is relatively small, with an average of 8.04% and a maximum of 25.25% for the 25 tested designs. It is worth noting

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

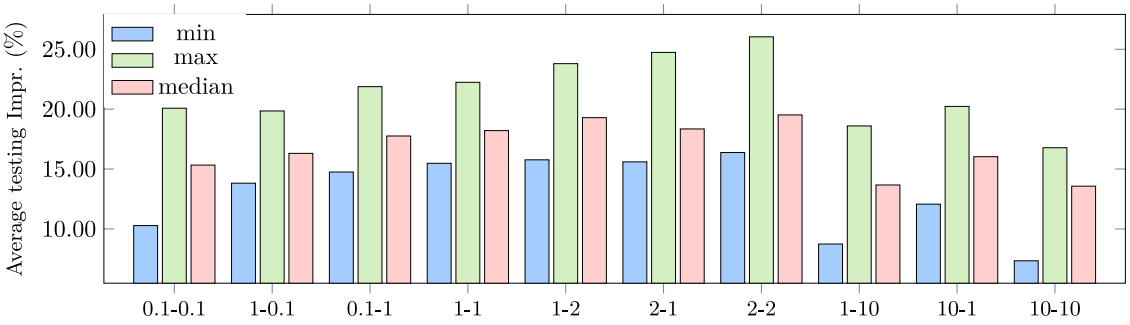


Fig. 11. Impact of the hyperparameters, where the x coordinate a-b represents the setting of $\beta = a, \gamma = b$. Our method performs stably when the hyperparameters are within a reasonable range but suffers from degradation when the hyperparameters are set as extreme values, e.g., 10-10.

TABLE IV
PERFORMANCE ON OPEN-SOURCE DESIGNS, WHERE THE BASELINE METHODS INCLUDE THE INDUSTRIAL EMULATION COMPILATION FLOW WITHOUT INTEGRATING PARSGCN, AND THE ONE THAT USES KAHYPAR TO PARTITION. FOR COMMERCIAL SECURITY REASONS, WE NORMALIZE ALL THE *#steps* VALUES BY THE CORRESPONDING PERFORMANCE OF THE INDUSTRIAL TOOL

Design	Size ($\times 10^6$)		#instruction execution steps								
			min			max			median		
	#nodes	#nets	Ind.	KaHyPar	ParSGCN	Ind.	KaHyPar	ParSGCN	Ind.	KaHyPar	ParSGCN
Or1200	0.46	0.47	1.000	1.035	0.988	1.000	1.046	0.918	1.000	1.054	0.981
Boom	1.09	1.1	1.000	1.059	0.993	1.000	1.137	0.819	1.000	1.081	0.894
HwachaRocket	2.41	2.43	1.000	1.028	0.975	1.000	1.040	0.910	1.000	1.047	0.968
GemminiRocket	10.45	10.74	1.000	1.149	0.865	1.000	1.272	0.908	1.000	1.108	0.799
XiangShan	20.24	20.26	1.000	1.196	0.905	1.000	1.707	0.661	1.000	1.669	0.911
AVG.	6.93	7.00	1.000	1.093	0.945	1.000	1.240	0.843	1.000	1.192	0.911

that for some designs, the overhead may be negative, as the SFP scheme may reduce the time required to find a good partitioning solution, such as expediting the convergence of the FM algorithm. Besides, the overhead of running ParsGCN is one-time as it is incorporated into the emulation compilation stage, which needs to be executed only once for each design. After compilation, we can run the emulation multiple times for each design without recompilation.

C. Ablation Study

In addition, we conduct an ablation study on our proposed SFP scheme. As shown in Fig. 10, all three guiding techniques contribute to an improvement in performance, with the combination of all three techniques resulting in the most significant improvement. Furthermore, we observed that the gain from a single guided initial partitioning algorithm is much smaller compared to the other two techniques. This could be due to the fact that the gain achieved during the initial partitioning stage is diminished during the subsequent refinement process. However, when our guided initial partitioning and refinement algorithms are combined, superior performance is achieved, demonstrating the effectiveness of our guided refinement approach.

In addition, we carry out an ablation study to assess the effectiveness of our customized GCN architecture. We reimplement a naive GCN as a baseline, which treats our constructed graph as undirected and homogeneous. This baseline GCN is trained on the same dataset for an identical number of epochs and is subsequently used as the cut probability estimator. The results, as shown in Table III, reveal that our customized GCN outperforms the naive version on both the training and testing designs. This suggests that the direction of the message indeed plays a significant role in our task.

Moreover, we conduct experiments to investigate the influence of the hyperparameters associated with our SFP scheme, specifically, β in (2) and γ in (5). As mentioned before, we set $\beta = 2, \gamma = 2$ by default. As depicted in Fig. 11, the performance of our method remains consistent across various values of β and γ. These results further underscore the robustness and generalization capability of our proposed method, given that it is not significantly impacted by the hyperparameters. However, our method might suffer from a large performance drop when some extreme values are used, e.g., $\beta = 10, \gamma = 10$. This suggests that the values of the hyperparameters should be chosen within a reasonable range to avoid performance degradation.

D. Results on Open-Source Designs

To further assess the generalization capability of our proposed method, we conduct experiments on a variety of open-source designs. The designs are obtained from GitHub and Chipyard, including: 1) OpenRISC 1200 (Or1200), a 32-bit scalar RISC with Harvard microarchitecture [35]; 2) Berkeley Out-of-Order (BOOM) Core, an out-of-order superscalar RV64Gcore [36]; 3) Rocket, a 5-stage in-order scalar processor core generator [37], with two different extensions Gemmini DNN accelerator [38] or Hwacha vector unit [39]; and 4) XiangShang, a superscalar out-of-order RISC-V processor with RV64GCBK ISA support [40]. To enhance the transparency and credibility of the results, we include an open-source baseline, KaHyPar [12], for comparison. KaHyPar is one of the state-of-the-art (SOTA) hypergraph partitioning tools. As shown in Table IV, ParSGCN consistently

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

delivers strong performance on the open-source designs, resulting in an average performance gain of 5.5%, 15.7%, and 8.9% in the best, worst, and median solution quality over 50 runs, respectively. Moreover, our method demonstrates promising scalability, as it tends to perform better on larger designs. This is in line with the results on the industrial designs (Table II). Meanwhile, KaHyPar's performance is less impressive as it does not consider any scheduling objectives or information. This outcome further underscores the existing gap between emulation partitioning and scheduling.

VII. Conclusion

Prior studies have noted the importance of efficient functional verification in reducing design cycles. To handle the ever-growing design size, existing processor-based emulation systems perform partitioning during compilation to reduce the problem size of computation scheduling. However, this divide-and-conquer approach often leads to suboptimal emulation efficiency due to the gap between partitioning and scheduling. In this article, we propose a GCN-driven solution to bridge the above gap, which is incorporated into an industrial emulator. Comprehensive experiments on large-scale designs demonstrate the effectiveness of our proposed method, with an average improvement of 16.38%, 26.04%, and 19.52% on best, worst, and median solution quality found in 50 runs. Furthermore, we have demonstrated the good generalization ability of our method, and shown that the runtime overhead is small. Lastly, it is important to acknowledge that our approach does have certain limitations, which present opportunities for future improvements. For example, our method is trained and tested on a specific hardware platform. In the future, we are interested in exploring the possibility of transferring our methods across different hardware systems. Additionally, we are keen on modeling the net cut probability in a more complex manner, which could potentially enhance the performance.

References

[1] L. Scheffer, L. Lavagno, and G. Martin, *EDA for IC System Design, Verification, and Testing*. Boca Raton, FL, USA: CRC Press, 2018.

[2] H. D. Foster, "Trends in functional verification: A 2014 industry study," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2015, pp. 1–6.

[3] P. P. Chu, *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability*. Hoboken, NJ, USA: Wiley, 2006.

[4] M. Graphics. "Mentor Celoce." 2016. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/veloce

[5] "Synopsys ZeBu." Synopsys. 2021. [Online]. Available: https://www.synopsys.com/verification/emulation/zebu-ep1.html

[6] K. Shi, S. Xu, Y. Diao, D. Boland, and Y. Bao, "ENCORE: Efficient architecture verification framework with FPGA acceleration," *ACM Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2023, pp. 209–219.

[7] "Cadence palladium." Cadence. 2015. [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html

[8] A. A. Yazdanshenas, "Hardware design and CAD for processor-based logic emulation systems," M.S. thesis, Dept. Electr. Comput. Eng., Univ. Windsor, Windsor, ON, Canada, 2006.

[9] R. Pu, Y. Sun, P.-H. Ho, F. Yang, L. Shang, and X. Zeng, "Sphinx: A hybrid Boolean processor-FPGA hardware emulation system," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2023, pp. 1–8.

[10] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 1997, pp. 526–529.

[11] Ü. V. Çatalyürek and C. Aykanat, "Patoh (partitioning tool for hypergraphs)," in *Encyclopedia of Parallel Computing*. New York, NY, USA: Springer, 2011, pp. 1479–1487.

[12] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, "High-quality hypergraph partitioning," *J. Exp. Algorithm.*, vol. 27, pp. 1–39, Feb. 2023.

[13] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "SpecPart: A supervised spectral framework for hypergraph partitioning solution improvement," in *Proc. 41st IEEE/ACM Int. Conf. Comput.-Aided Design*, 2022, pp. 1–9.

[14] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "K-SpecPart: Supervised embedding algorithms and cut overlay for improved hypergraph partitioning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 4, pp. 1232–1245, Apr. 2024.

[15] S. Karandikar et al., "FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud," *IEEE/ACM Int. Symp. Comput. Archit. (ISCA)*, 2018, pp. 29–42.

[16] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[17] Z. Wang, Z. He, C. Bai, H. Yang, and B. Yu, "Efficient arithmetic block identification with graph learning and network-flow," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, vol. 42, no. 8, pp. 2591–2603, Aug. 2023.

[18] W. Li, Y. Ma, Y. Lin, and B. Yu, "Adaptive layout decomposition with graph embedding neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 5030–5042, Nov. 2022.

[19] Y. Ma et al., "High performance graph convolutional networks with applications in testability analysis," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[20] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–17.

[21] R. Wang et al., "A bi-level framework for learning to solve combinatorial optimization on graphs," *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 34, 2021, pp. 21453–21466.

[22] A. Nazi, W. Hang, A. Goldie, S. Ravi, and A. Mirhoseini, "Gap: Generalizable approximate graph partitioning framework," 2019, *arXiv:1903.00614*.

[23] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 1997, pp. 530–533.

[24] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 1999, pp. 343–348.

[25] U. V. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Trans. Parallel Distrib. Syst. (TPDS)*, vol. 10, no. 7, pp. 673–693, Jul. 1999.

[26] C. Fiduccia and R. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 1982, pp. 175–181.

[27] V. Nair et al., "Solving mixed integer programs using neural networks," 2020, *arXiv:2012.13349*.

[28] C. M. Bishop and N. M. Nasrabadi, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[29] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2014, pp. 1–14.

[30] Z. Wang et al., "FGNN2: A powerful pre-training framework for learning the logic functionality of circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, early access, Jul. 26, 2024, doi: 10.1109/TCAD.2024.3434464.

[31] Z. Wang, S. Liu, Y. Pu, S. Chen, T.-Y. Ho, and B. Yu, "Restructure-tolerant timing prediction via multimodal fusion," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[32] S. Zheng, L. Zou, P. Xu, S. Liu, B. Yu, and M. Wong, "Lay-Net: Grafting Netlist knowledge on layout-based congestion prediction," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2023, pp. 1–9.

[33] M. Wang et al., "Deep graph library: Towards efficient and scalable deep learning on graphs," 2019, *arXiv:1909.01315*.

[34] A. Paszke et al., "Automatic differentiation in PyTorch," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, 2017, pp. 1–4.

[35] S. Titri, N. Izeboudjen, L. Sahli, D. Lazib, and F. Louiz, "Open cores based system on chip platform for telecommunication applications: VoIP," in *Proc. Int. Conf. Design Technol. Integr. Syst. Nanoscale Era*, 2007, pp. 245–248.

[36] A. Amid et al., "Chipyard: Integrated design, simulation, and implementation framework for custom SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, Jul./Aug. 2020.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.

[37] K. Asanovic et al., "The rocket chip generator," Dept. EECS, Univ. California, Berkeley, CA, USA, Rep. UCB/EECS-2016-17, 2016.
[38] H. Genc et al., "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, 2021, pp. 769–774.
[39] Y. Lee, C. Schmidt, A. Ou, A. Waterman, and K. Asanovic, "The Hwacha vector-fetch architecture manual, version 3.8. 1," Dept. EECS, Univ. California, Berkeley, CA, USA, Rep. UCB/EECS-2015-262, 2015.
[40] Y. Xu et al., "Towards developing high performance RISC-V processors using agile methodology," in *Proc. 55th IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2022, pp. 1178–1199.



Ziyi Wang received the B.S. degree from the Department of Computer Science and Technology, Fudan University, Shanghai, China, in 2021. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His research interests include graph learning applications in electronic design automation and logic synthesis.



Wenqian Zhao received the B.Sc. degree in computer science and engineering from The Chinese University of Hong Kong, Hong Kong, in 2019, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering.

His research interests include machine learning for VLSI design automation and hardware-aware deep-learning acceleration.



Yuan Pu received the B.S. degree in computer science from The Chinese University of Hong Kong, Hong Kong, in 2022, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, supervised by Prof. B. Yu.

His research interest includes combinatorial algorithms/AI in EDA.



Lei Chen received the B.S. degree in computer science from the South China University of Technology, Guangdong, China, in 2012, and the Ph.D. degree in computer science from the Hong Kong Baptist University, Hong Kong, in 2016.

He is currently a Senior Researcher with Huawei Noah's Ark Lab, Hong Kong. His research interests include EDA logic synthesis, circuit data modeling, data-driven optimization algorithms, and big data management.



Wilson Wang Kit Thong received the Ph.D. degree from the Department of Electronics Engineering, City University of Hong Kong, Hong Kong, in 2013.

He is a Principal Engineer with Huawei, Hong Kong, working on EDA software toolchain development. He has more than ten years experience in algorithms development for commercial software products, including LTE baseband/protocol stack, IC bonding and packaging control, and compiler for EDA.



Weihua Sheng received the B.Eng. degree from Tsinghua University, Beijing, China, in 2000, and the M.S. and Ph.D. degrees from RWTH Aachen University, Aachen, Germany, in 2003 and 2010, respectively.

He has spearheaded the technology development on multicore programming tools at RWTH Aachen University from 2007 to 2013, which later turned into the foundation of Silexica. His early career roots are in the electronic design automation industry. He has a proven record of successful consultation and collaboration with top-tier technology companies on multicore design tools. He is currently with Huawei Research Center, Hong Kong.



Tsung-Yi Ho (Fellow, IEEE) received the Ph.D. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2005.

He is a Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. His research interests include several areas of computing and emerging technologies, especially in design automation of microfluidic biochips.

Prof. Ho was a recipient of the Best Paper Award at the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems in 2015. He currently serves as the VP Conferences of IEEE CEDA, and the Executive Committee of ASP-DAC and ICCAD. He is a Distinguished Member of ACM.



Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received ten Best Paper Awards from IEEE TSM 2022, DATE 2022, ICCAD 2021 and 2013, ASPDAC 2021 and 2012, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, and SPIE Advanced Lithography Conference 2016, and six ICCAD/ISPD contest awards. He has served as the TPC Chair for ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He is the Editor of IEEE TCCPS NEWSLETTER.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 22,2025 at 03:32:10 UTC from IEEE Xplore. Restrictions apply.