MapZero: Mapping for Coarse-grained Reconfigurable Architectures with Reinforcement Learning and Monte-Carlo Tree Search

Xiangyu Kong*
Yi Huang*
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China

Jianfeng Zhu
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China

Xingchen Man
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China

Yang Liu
Innovation Institute of High Performance Server, GBA
Guangzhou, Guangdong, China

Chunyang Feng
Innovation Institute of High Performance Server, GBA
Guangzhou, Guangdong, China

Pengfei Gou
HEXIN Technologies Co., Ltd.
Guangzhou, Guangdong, China

Minggui Tang
HEXIN Technologies Co., Ltd.
Guangzhou, Guangdong, China

Shaojun Wei
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China

Leibo Liu†
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China

ABSTRACT

Coarse-grained reconfigurable architecture (CGRA) has become a promising candidate for data-intensive computing due to its flexibility and high energy efficiency. CGRA compilers map data flow graphs (DFGs) extracted from applications onto CGRAs, playing a fundamental role in fully exploiting hardware resources for acceleration. Yet the existing compilers are time-demanding and cannot guarantee optimal results due to the traversal search of enormous search spaces brought about by the spatio-temporal flexibility of CGRA structures and the complexity of DFGs. Inspired by the amazing progress in reinforcement learning (RL) and Monte-Carlo tree search (MCTS) for real-world problems, we consider constructing a compiler that can learn from past experiences and comprehensively understand the target DFG and CGRA.

In this paper, we propose an architecture-aware compiler for CGRAs based on RL and MCTS, called MapZero – a framework to automatically extract the characteristics of DFG and CGRA hardware and map operations onto varied CGRA fabrics. We apply Graph Attention Network to generate an adaptive embedding for DFGs and also model the functionality and interconnection status of the CGRA, aiming at training an RL agent to perform placement and routing intelligently.

Experimental results show that MapZero can generate superior-quality mappings and reduce compilation time hundreds of times compared to state-of-the-art methods. MapZero can find high-quality mappings very quickly when the feasible solution space is rather small and all other compilers fail. We also demonstrate the scalability and broad applicability of our framework.

CCS CONCEPTS

• **Computer systems organization** → **Reconfigurable computing**; • **Software and its engineering** → **Retargetable compilers**.

KEYWORDS

Coarse-Grained Reconfigurable Architecture; Compiler; Graph Neural Network; Reinforcement Learning

ACM Reference Format:
Xiangyu Kong, Yi Huang, Jianfeng Zhu, Xingchen Man, Yang Liu, Chunyang Feng, Pengfei Gou, Minggui Tang, Shaojun Wei, and Leibo Liu. 2023. MapZero: Mapping for Coarse-grained Reconfigurable Architectures with Reinforcement Learning and Monte-Carlo Tree Search. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3579371.3589081

1 INTRODUCTION

Nowadays, due to the end of Moore's Law and Dennard scaling, compute-intensive fields such as artificial intelligence, digital signal processing, biomedicine, and aerospace are placing higher demands on the performance and power consumption of spatial accelerators. Among the prevailing accelerators, coarse-grained reconfigurable architecture (CGRA) [3, 11, 19, 26, 28, 30, 34, 40, 46, 51, 54, 64, 71, 86] has a solid competitive advantage due to its low non-recurring engineering (NRE) costs, high energy efficiency, high throughput, and flexibility after fabrication [41, 44].

*Both authors contributed equally to this research.
†Corresponding author: Leibo Liu (liulb@tsinghua.edu.cn)

ISCA '23, June 17–21, 2023, Orlando, FL, USA.

ACM ISBN 979-8-4007-0095-8/23/06…$15.00
https://doi.org/10.1145/3579371.3589081

CGRA compilers map data flow graphs (DFGs) extracted from applications onto CGRAs, playing a fundamental role in fully exploiting hardware resources for acceleration. To be specific, the goal of a CGRA compiler is to place the operations of the input application onto the appropriate processing elements (PEs) and to ensure that all the operands and outputs are correctly routed. The compilation process is also known as mapping, which implies three steps: scheduling, placement, and routing. In this paper, scheduling is contained in placement. However, CGRAs have diverse computational arrays and network topologies, which places higher demands on the compiler. Thus, efficient and generic compilation algorithms are urgently required to accommodate a variety of emerging CGRA architectures. Due to the complex topological and timing constraints of CGRA, the search space of a valid mapping increases exponentially with the number of PEs and DFG nodes, resulting in tens of minutes or even hours of compilation time [13]. This poses a major challenge for compilers.

The existing compilers for CGRAs can be divided into architecture-specific and architecture-agnostic. Architecture-specific compilers generally apply heuristic approaches that take advantage of specific architectural features to find a quality solution [1, 2, 4, 10, 16, 21, 35, 36, 73, 74, 80, 83]. Architecture-agnostic compilers can be further classified into meta-heuristic [13, 18, 32, 37, 45, 46] and combinatorial optimization-based [9, 13, 17, 22, 43, 68, 72]. Representative meta-heuristic methods include Simulated Annealing (SA), Genetic Algorithms (GA), Machine learning (ML), etc. These methods are comparatively efficient, yet existing implementations suffer from high development costs and unsatisfactory solution quality. Moreover, for cases with tight constraints, they do not promise to find a feasible solution within an acceptable time limit. Yet combinatorial optimization-based techniques such as Integer Linear Programming (ILP) and Boolean satisfiability (SAT) are theoretically guaranteed to find the optimal solution, but the large time overhead is a shortcoming.

The fast evolution of modern CGRAs puts architecture-specific heuristic compilers at risk of premature obsolescence. Compilers that can be ported to different architectures are the trend. Meta-heuristic and combinatorial optimization-based compilers are gaining more attention these years. However, there remains a large gap from developing a portable and highly-efficient compiler. From the perspective of search space exploration, the meta-heuristic compiler randomly changes over existing unsuccessful mapping results to explore feasible solutions, while the combinatorial optimization-based compiler uses a systematic backtracking algorithm to solve for the optimal solution. The existing mainstream compilers [13] inevitably require a long compilation time. For instance, the ILP-based compiler CGRA-ME [13] requires hours of compilation time for complex DFGs. Whilst some subsequent approaches [7, 68] speed up the search by reducing the search space through alternative high-level hardware resource abstraction and skillful traversal, the time overhead is still suboptimal and these methods suffer from the problem of not being scalable to large DFGs.

Essentially, all the aforementioned methods are traversal searches of the search space. Traditional compilers using heuristics or combinatorial optimization can hardly learn from past experiences [13, 16, 17, 35–37, 45]. In their case, each compilation process is independent, which means recompiling after one failed attempt does not enhance the likelihood of finding a solution, and unfavorable search space will be explored repeatedly. Traditional compilers are also clumsy in exploring the search space due to the lack of understanding of DFG and CGRA architectures. For instance, the placement of an ancestor DFG node potentially affects whether subsequent nodes can be mapped successfully, so it requires the compiler to have the foresight to consider in advance the congestion that may be caused by multiple fan-outs of that node. On the other hand, the connectivity and functions of the selected PE of a CGRA as well as its neighboring PEs also have a profound impact on the subsequent mapping. If the compiler has such insights, it will be able to choose the best mapping scheme for the target node with respect to the current resource utilization status on the CGRA and relative node dependencies in the DFG. While lacking this knowledge, the compiler would make a significant search effort to backtrack or remap until it gets a workable solution. Therefore, in order to pursue both efficiency and mapping quality, we need compilers that can **learn from past experience and intelligently explore** the search space.

For this purpose, we resort to ML solutions. However, existing ML-based methods [33, 37, 39, 79] exploit only DFG features while ignoring hardware properties, and also lack the potential to acquire historical knowledge. We observe that the compilation process can be viewed as a sequential task, and reinforcement learning (RL) is a de facto candidate for such tasks [61]. In this context, compilers using reinforcement learning can effectively explore the design space by allowing an agent to interact with the predefined mapping environment and collect trajectories and rewards of different mapping strategies, with which it can learn to achieve better mapping results. Enlighted by [47], we believe that the mapping process of CGRA can be analogous to the placement and routing of chip blocks. Encoding the hardware architecture and DFGs with Graph Neural Networks (GNNs) and exploring the search space with RL can be a viable compilation method. From another perspective, node placement and Go are similar in many ways. Therefore, we consider introducing some ideas from AlphaZero [59] and MuZero [57], specifically Monte-Carlo Tree Search (MCTS) and reinforcement learning (RL), to design an efficient and portable compilation algorithm for CGRAs.

For the purpose of efficient and quality compilation, the compiler (the RL agent) needs to adequately understand the characteristics of the DFG and also the hardware properties of the target CGRA, including the number of PEs, their functionalities, and interconnection relationships. We build a hardware abstraction for CGRA suitable for a GNN front-end, and we carefully design the node features. We also devise a featured representation for DFGs. We choose Graph Attention Network (GAT), an attention-based GNN architecture, to compute the embeddings of each node in the graph-structured data by attending over its neighbors with corresponding attention coefficients, with which the compiler can thus gather information about the routing capacity and congestion level, the reachability between parent and child DFG nodes, the heterogeneous functionality of PEs, etc. With this information and knowledge gathered from the DFG together, the agent can anticipate potential congestion and consider the impact of the choice of PE for the current DFG node on the whole mapping process in a comprehensive manner. We utilize MCTS to allow the agent to

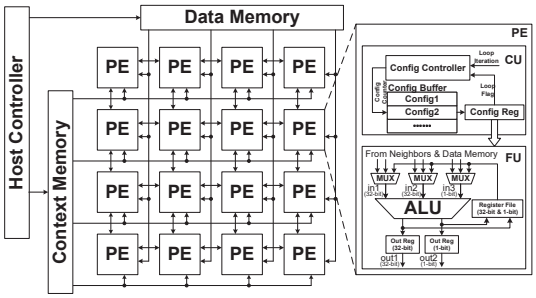

Figure 1: A 4×4 2-D mesh-like CGRA.

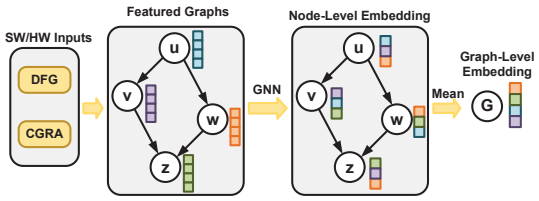

Figure 2: Graph embedding generation via GNN.

interact with the environment so as to explore the design space and generate training data for a deep neural network (DNN), which in turn guides the search and expansion in MCTS. In this way, the agent self-learns mapping strategies obeying design constraints and pursuing quality solutions offline. Since GAT is directly applicable to inductive learning, which means graphs that are completely unseen during training can also be properly processed, the trained network can quickly generate mapping for new DFGs online afterward. Also, different hardware architectures can reuse the same network without significant design overhead, ensuring portability.

In summary, this paper makes the following key contributions:

- We propose MapZero, a compiler framework for CGRA which improves understanding of the compilation process by employing GAT to embed both the DFGs and the graph representation of CGRA. We construct hardware abstraction feasible for GNN training and design necessary node features. This embedding algorithm can be applied to various shapes of DFGs and CGRAs and yield meaningful intermediate state representations for the subsequent tasks.
- To the best of our knowledge, MapZero firstly proposes to use Reinforcement Learning and Monte-Carlo Tree Search in the CGRA compiler. MCTS searches for the next state with a certain strategy learned by RL, so its tree shape will theoretically move in a more favorable direction toward successful mapping. Assisted by GAT, together they can construct an efficient search space exploration scheme.
- We integrate the above ideas and implement a portable compiler to perform CGRA placement and routing automatically and intelligently search the search space. We evaluate the compiler on prevailing and classic CGRA topologies and compare the quality and efficiency of mapping with state-of-the-art compilers. Experimental results demonstrate hundreds of times compilation time reduction while always achieving minimal initiation intervals (i.e., cycles between the start of consecutive loop iterations). MapZero can find high-quality mappings very quickly when the feasible solution space is rather small and all other compilers fail.

2 BACKGROUND AND MOTIVATION

In this section, we first introduce the concept of CGRA and its compiler. Then we review background concepts related to GNN, MCTS, and RL and discuss the rationale of their application to the context of CGRA compilation. Finally, we present a motivational example to show the insights that endorse our work.

2.1 Coarse-Grained Reconfigurable Architecture

Coarse-Grained Reconfigurable Architecture (CGRA) is a typical type of reconfigurable spatial accelerator that supports spatial and temporal computation [41]. Fig. 1 shows a typical type of mesh-like CGRA. It comprises a host controller, on-chip context and memories, processing element arrays (PEAs), etc. Each PEA consists of spatially parallel processing elements (PEs), which operate independently of each other. PEs can be homogeneous, heterogeneous, or general, and are interconnected through point-to-point connections, or mesh-like or crossbar-based topologies. The diversity of PE functionality and interconnections makes a generic compiler extremely challenging.

To adapt the compiler to different CGRA architectures, researchers have devised a handful of algorithms based on combinatorial optimization or meta-heuristic methods to solve the mapping of data flow graphs to hardware abstractions [44]. Mapping can be spatial or temporal. Temporal mapping enables CGRA to execute larger programs through reconfiguration. In the case of temporal mapping, the architecture can be abstracted as a time-extended CGRA (TEC) [16] or a modulo routing resource graph (MRRG) [45], both imply spatio-temporal information. The ultimate goal of mapping is to identify the spatial and temporal coordinates of every node in the DFG within a series of constraints of data dependency, exclusiveness, timing, etc. It is a sequential search task with an immense search space.

2.2 Graph Neural Networks

To effectively generate embeddings for graph-structure inputs, namely DFG and CGRA, a method that is both computationally economical in large graphs and can deal with unseen nodes is a prerequisite. Graph Neural Network (GNN) [47] has the potential to achieve these goals by extracting structural and feature information from input graphs. It aims to learn the embedding vectors $\mathbf{h}_u$ of each node u via gathering information of its neighbors. Fig. 2 demonstrates the course of embedding generation. Graph Attention Network (GAT) [67] is a type of GNN that has the advantage of being inductive (generalizable to unseen nodes) and not limiting the number of neighboring nodes. GAT leverages the multi-head attention mechanism [66] to assign different relevant weights to the neighboring nodes. Varied attention factors are promising for learning heterogeneous hardware structures. The details of the principle of GAT will be presented in the subsequent section.

2.3 Reinforcement Learning

The reinforcement learning (RL) problem can be modeled as a Markov decision process (MDP), which addresses the problem of

how an agent maximizes the cumulative reward it can obtain in a complex, uncertain environment [61]. RL consists of two parts: the agent and the environment. In our case, the agent is the compiler and the environment contains information about CGRA structure and compilation constraints. The agent tries to learn the optimal policy π^* under each state to maximize the discounted cumulative return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

where $R_{t+1}, R_{t+2}, \cdots$ are the sequence of rewards received after time step t, G_t is the expected discounted return, γ is the discount rate, $0 \leq \gamma \leq 1$.

The state-value function for policy π and state s, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \tag{2}$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable under policy π, S_t is the state representation at time t.

With the optimal policy π^*, the agent is meant to get a valid CGRA mapping with the lowest routing penalty. The optimal value function v^*, or the Bellman optimality equation, expresses the fact that the value of a state under π^* equals the expected return for the best action from that state:

$$\begin{aligned} v^* &= \max_a \mathbb{E}_{\pi^*}[G_t|S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1})|S_t = s, A_t = a] \end{aligned} \tag{3}$$

where A_t is the action taken at time t.

Deep reinforcement learning (DRL), is reinforcement learning with function approximation by deep neural networks (DNN). Policy π and value v can be approximated using DNNs. Numerous DRL algorithms have shown extraordinary performance over traditional RL, such as DQN [49], A3C [48], PPO [58], etc. Recently, DRL has a number of promising applications in the field of computer architecture already [27, 38, 39, 75, 76, 84].

2.4 Monte-Carlo Tree Search

Monte Carlo Tree Search (MCTS) [15], is a method for making optimal decisions by sampling and then building a search tree, combining the generality of stochastic simulation with the accuracy of tree search. It is broadly used in artificial intelligence problems, especially in combinatorial games. The growing attention MCTS has received is mainly due to the success of the computerized Go game and its potential application to a wide range of difficult problems.

In AlphaZero [59] and MuZero [57], MCTS is incorporated with DRL to improve performance. The policy and value estimation in MCTS are obtained from a neural network based on RL.

2.5 Challenges

2.5.1 Search Space Complexity. The intricacy of the search space in CGRA compilation is a significant challenge that necessitates effective exploration. In the case of II (Initiation Interval) = 1, mapping a DFG with 14 nodes onto a 4×4 CGRA has $\frac{16!}{2} \approx 10^{13}$ total possibilities, while only a few of them are valid. The search space grows exponentially with the number of PEs in CGRA and the number of nodes in DFG: mapping a 60-node DFG onto an 8×8 CGRA has up to $\frac{64!}{24} \approx 10^{87}$ possibilities. If II > 1, the problem complexity is even higher. Leveraging continually improved domain knowledge, RL and MCTS offer an effective framework for search space exploration. Since the agent has the competence to learn domain knowledge, our framework has the potential to efficiently explore greater search space.

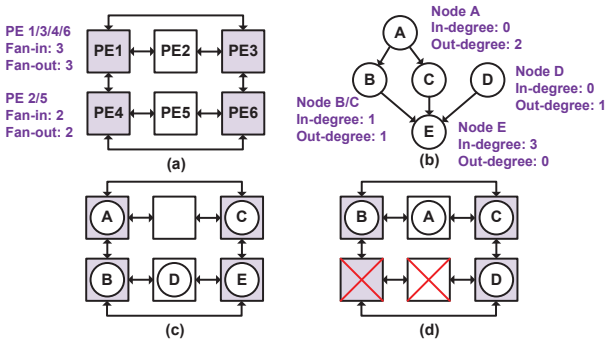

Figure 3: A motivational case study for mapping a 5-node DFG onto a 2× 3 CGRA. The fan-in and fan-out of CGRA PEs, in-degree and out-degree of DFG nodes, are noted to demonstrate part of the properties of the two graph structures (more comprehensive features are desired in reality). (a) Target CGRA. (b) Target DFG. (c) A successful mapping with knowledge of both DFG and CGRA structure and features. (d) A failure mapping without any knowledge.

2.5.2 Motivational case study. The DFG and hardware characteristics are vital to the mapping process. In this case study, we demonstrate the mapping process on a 2×3 CGRA. Fig. 3(a) shows its PEA structure. PEs in different positions are not equivalent, shaded PEs have stronger routing capabilities. For simplicity, we consider the case II= 1, which can also be described as spatial mapping. Fig. 3(b) is the target DFG. The PE selected for node A will potentially impact the successful mapping of node E. This PE has to be connected to two other PEs (for node B and C), both of which can reach a third PE, which is reserved for node E. It is important that this assigned PE for node E should ensure a sufficient number of fan-in connections.

Fig. 3(c)(d) show a successful and a failed mapping, respectively. The improper choice of parent nodes will cause the mapping to fail. On the contrary, by mastering knowledge about both the DFG and the CGRA structure, the compiler can know that: 1) node B and C are both children of node A and parents of node E. 2) node E has 3 fan-ins. 3) node E should be placed on a PE with superior routing capability. 4) PE 0, 2, 3, and 5 have more routing resources, etc. The above information will promote more visionary placements for each node. This example only involves a 2×3 PEA and a small DFG. The compiler would face greater challenges given a large PEA with complex interconnections and DFGs with much more operations. To this end, we propose MapZero, which can take advantage of both the hardware and software features in concert to efficiently explore the search space of CGRA mapping.

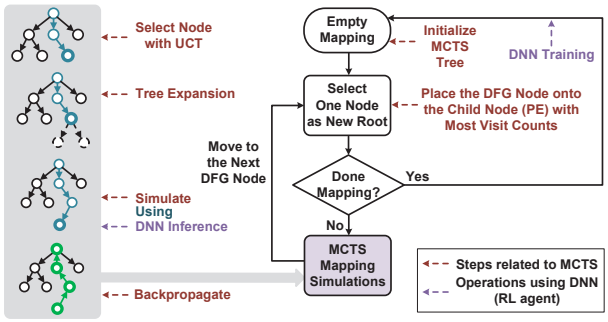

Figure 4: Workflow of the proposed compilation framework.

3 METHODOLOGY

3.1 Overview

Fig. 4 presents the workflow of MapZero, showing how RL and MCTS work together. In the process, an MCTS tree is first initialized with a root node, which means an empty mapping. Tree nodes will denote the updated mapping states with a new DFG node assigned to a certain PE. The framework checks if the mapping is completed. If not, a round of MCTS mapping simulations will commence, as shown in the gray box. Each iteration in the round consists of four steps:

Selection: Starting from the root node R, recursively select child nodes using the Upper Confidence Bound (UCB) for trees (UCT) [31], which balances the exploitation of known benefits with exploration that encourages access to nodes that are relatively unvisited, until a leaf node is reached.

$$UCT(v) = \overline{X_j} + 2C_p\sqrt{\frac{2\log n}{n_j}} \tag{4}$$

where $\overline{X_j}$ is the mean reward from node j, n and n_j are the numbers of times the parent node and the child node j have been visited, respectively, C_p is a constant parameter.

Expansion: If the aforementioned leaf node is not the terminal node, add child nodes to it according to valid actions (available PEs).

Simulation: The simulation starts from the expanded node and uses DNN to infer a value referring to potential final mapping quality.

Backpropagation: Update previously visited nodes in the MCTS tree with the statistics gleaned from the simulation.

After the specified number of iterations, the most visited (i.e., most promising) child node of the previous root is selected as the new root. When the mapping is done, trajectories including rewards, observations of states, actions, and value estimates collected from the MCTS tree are exploited to train the agent. The whole process will run through a number of iterations until the agent has acquired sufficient domain knowledge and is able to efficiently generate a quality mapping. Detailed algorithms will be discussed later.

3.2 Graph Embedding Generation

The key role of the GNN is to extract essential information about the topology and node features of the input graphs. The information is then encoded into low-dimensional vector representations for downstream RL tasks. The features of DFGs and CGRA hardware are significant for the agent's learning to understand the environment.

3.2.1 DFG Feature Vector. In a DFG, each node is encoded into a 10-dimension node feature vector: (1) id (2) scheduling order obtained by topological sorting (3) scheduled time slice (4) scheduled modulo time slice (5) in-degree (6) out-degree (7) opcode (8) has self-cycle or not (9) number of DFG nodes to be mapped in the same modulo time slice (10) the id of assigned PE.

3.2.2 CGRA Hardware Feature Vector. In a CGRA, each PE is encoded into a 7-dimension node feature vector: (1) id (2) in-degree (3) out-degree (4)-(6) the boolean values of PE functionalities (whether this PE can perform logical, arithmetic, and memory access operations) (7) the id of the mapped DFG node.

Note that feature vectors change with the state. The CGRA hardware of each modulo time slice has a separate graph representation. For CGRAs with circuit-switched mesh interconnects epitomized by HyCube [28], it is assumed that each crossbar switch is driven by clockless repeaters that can be configured to either let signals bypass asynchronously to the next hop, or to stop and receive the incoming data, so data can be sent across multiple PEs within a single cycle. This cycle is between two time slices.

3.2.3 Graph Embedding. We employ the graph attention network (GAT) [67] to generate graph embeddings. The node embedding can be calculated by aggregating feature vectors from neighboring nodes and concatenating the features of K-independent attention mechanisms. Formally, the layer-wise propagation of each graph attention layer can be formulated as follows:

For a vertex u, calculate the attention coefficients with its neighbors $v \in N(u)$, as in Eq. 5-6, where the transformed features of vertices u, v are concatenated; then a maps the concatenated high-dimensional features to a scalar.

$$e_{uv} = a\left(\mathbf{W}\mathbf{h}_u \parallel \mathbf{W}\mathbf{h}_v\right) \tag{5}$$

$$\alpha_{uv} = \frac{\exp\left(LeakyReLU(e_{uv})\right)}{\sum_{m\in N(u)}\exp\left(LeakyReLU(e_{um})\right)} \tag{6}$$

LeakyReLU is a nonlinear function defined by Eq. 7.

$$LeakyReLU(x) = \begin{cases} x & x \geq 0 \\ cx & x < 0 \end{cases} \tag{7}$$

Eq. 8 describes the attention-based graph convolution, where α_{uv}^k is the normalized attention coefficient of the node v to u calculated by the k^{th} attention mechanism, $\mathbf{W}^k$ is the layer-specific weight matrix of the corresponding input linear transformation, σ is a nonlinear function. $\mathbf{h}'_u \in \mathbf{R}^{n\times d\times k}$ is the matrix of activations from the u^{th} layer, which stacks the hidden layers of each node, assuming that there are n nodes with feature dimension d and attention mechanism k.

$$\mathbf{h}'_u = \Big\|_{k=1}^{K}\ \sigma\left(\sum_{v\in N(u)}\alpha_{uv}^k\mathbf{W}^k\mathbf{h}_u\right) \tag{8}$$

After multiple layers, the learned node embeddings are summarized by mean pooling to create a final representation vector.

3.2.4 Integration with RL. The graph embedding networks act as the front end of the RL network. To integrate with RL, we concatenate the two embedding vectors with the metadata of the current node, including the node id and relevant features.

3.3 RL Problem Formulation

We target the problem of CGRA mapping, whose objective is to determine the spatio-temporal coordinates of DFG nodes on the architecture with no constraint violation and routing penalty minimized. Fig. 6 demonstrates the specific usage of RL in this framework. The inputs to the framework are the target DFG and CGRA. Their node features are extracted and fed into a graph embedding network. The generated embedding is then processed by the RL network to obtain the policy and value estimation of the current state. The RL network updates weights according to the routing penalty feedback from the environment, with which the RL agent gradually learns to find optimized mapping solutions. For the mapping of neighbor-to-neighbor interconnect style CGRAs, placement and routing are coupled, whereas they are decoupled for architectures with circuit-switched mesh interconnects. In the latter case, after the agent has performed the placement action, Dijkstra's algorithm finds the shortest path for routing, if any. The key elements of the Markov Decision Process (MDP) are defined as follows.

State s_t: the state representation consists of three parts,

(1) CGRA hardware abstraction H: a directed graph that portrays the information of CGRA PEs and routing resources in the current modulo time slice, implying their computing and routing capabilities.
(2) DFG D: a directed graph that contains detailed features of each DFG node, entailing information about other nodes that compete with this node for resources and that constrain its resource selection due to data dependencies.
(3) Metadata M: metadata of the current DFG node to be mapped, including its id and relevant features.

Action a_t: the action space contains available coordinates in the CGRA at time t. Placing a node onto an already occupied PE on the time slice to which this node is scheduled is illegal, thus we mask the invalid actions. Once a spare PE is selected by the current node, we modify the corresponding features in H and D to denote this selection.

Reward r_t: the reward given at each action is the negative routing penalty introduced by the mapping of the current node, calculated based on the position of its parent nodes. The routing penalty is determined by whether a valid routing can be found after the placement action. A good action refers to selecting a PE so that the node can connect to its parent nodes via interconnections, thus receiving a trivial penalty. A bad action involves a selection that will incur routing failures, thus resulting in a large penalty. Unlike typical RL algorithms, our reward is given at each time step to enable more efficient learning. The agent gets a final return based on whether the mapping was successful or not. Minimal routing penalty under the condition of successful mapping is thus achieved through minimizing the value of cumulative returns.

The environment contains the necessary resource, routing, and timing constraints to guarantee that the solutions found are valid.

3.4 Neural Network Structure

The neural network model learns to maximize the expected reward (minimize the routing penalty) and improve mapping quality over episodes. Fig. 5 demonstrates the network structure used in the framework. The model is composed of two components: representation and prediction. The representation network is a multi-view embedding network to synthetically explore the observation information and produce an intermediate state vector. The prediction network consists of two output heads to compute the policy and value, respectively. The details of the model will be specified in the following parts.

3.4.1 Network components. The representation network generates graph embeddings for the observed DFG and CGRA hardware, whose features evolve with the state. Both of the graph embeddings are critical guidance for the subsequent prediction. We deploy graph attention network (GAT) to generate low-dimensional embeddings for them to distill implications about the physical characteristics of CGRA and interactions between DFG nodes. The network should also know about the current node to be mapped, so we employ a Fully Connected Layer (FC) to generate embedding for its metadata. After we have obtained all relevant embeddings, we concatenate them and pass the result through a Multilayer Perceptron (MLP) to get the joint intermediate state vector.

The intermediate state vector is then fed into the prediction network, where the policy network gives a probability distribution over actions and the value network estimates the value of the current state. The output dimensions of MLPs and FCs used in the prediction network are labeled in Fig. 5.

3.4.2 Intrinsic principle of training. The policy and value outputs of the network reflect the decision-making capability of the RL agent. The agent learns to maximize the objective function (minimizing the routing penalty) gradually via training. At each step t, the quadruple (s_t, a_t, r_t, s_{t+1}) is collected. Once a batch of data has been collected, the weights of the network are updated through stochastic gradient descent. The policy $\pi(a_t|s_t)$ represents a distribution of probability over the actions. The optimization of π is based on the policy gradient theorem that relates the gradient of the objective function and π: $\Delta_\theta \log \pi(a|s)A(s,a)$. $A(s,a)$ is the advantage that calculates the difference between the return of action a and the estimated value of state s. On the other hand, the value $V(s)$ quantifies the score of a state s, and the optimization of $V(s)$ is based on the mean square error of Bellman's function mentioned in Section 2.3.

3.5 Search Space Exploration

MCTS is known as an effective method for space exploration as it can leverage historical states while learning. Algorithm 1 presents the RL-MCTS-based search space exploration algorithm. In the context of CGRA mapping, MCTS starts from a root node and recursively applies a learned policy to place DFG nodes onto the CGRA fabric and traverse down the tree. Once a new node is reached, it uses the DNN to produce a value estimation of the node. After a simulation is completed, the result is backpropagated through all the passed nodes in the trajectory. MCTS stores previously observed DFG and hardware status as nodes in the MCTS tree. Each node is

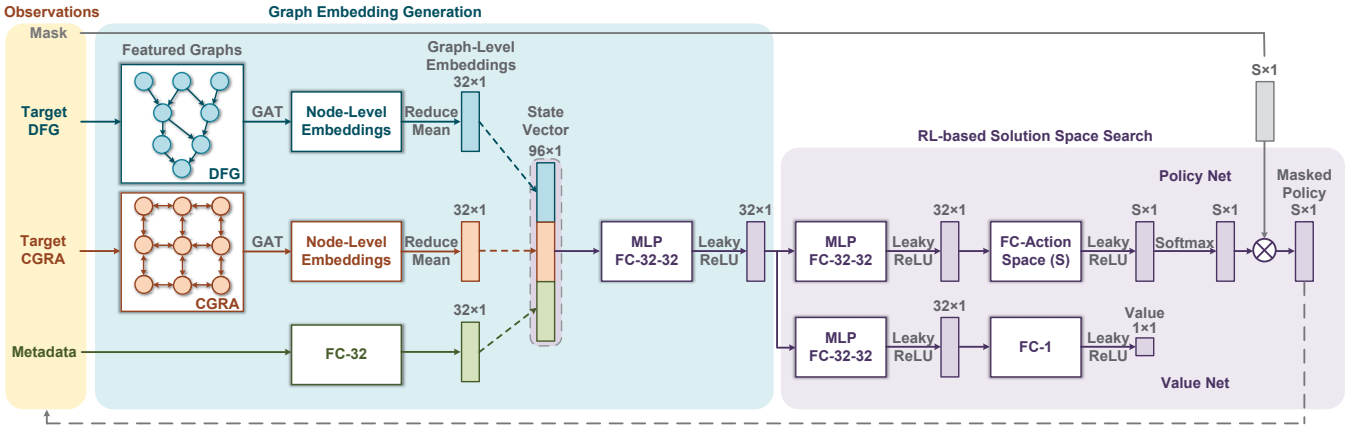

Figure 5: Neural network architecture. A graph embedding generator uses GAT to encode information about the DFG and CGRA hardware status, and FC to encode the current node to be placed. Concatenating the graph embeddings, the prediction network then outputs a probability distribution over available placement locations $\pi(a_t|s_t)$, and an estimate of the expected reward for the current placement, respectively. The output dimensions of MLPs and FCs are labeled in the figure.

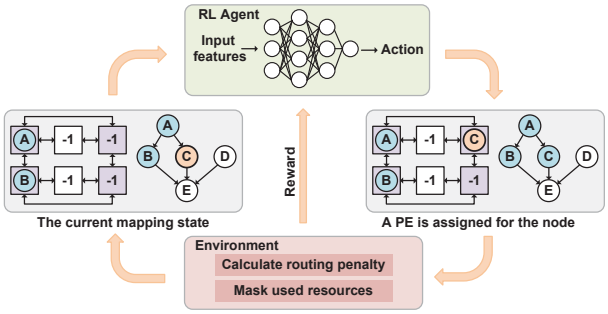

Figure 6: RL-based DFG mapping on CGRA. Given a DFG and CGRA, the RL agent sequentially decides where to assign the current unmapped node. The environment will give back a reward immediately after each action is taken.

labeled with an estimated value representing the future returns for exploration starting from it.

MCTS can provide additional insights during state exploration and help narrow the scope of exploration to a few promising branches [57] to learn optimal DFG mapping efficiently. The execution time of space search is mainly influenced by the number of expanded nodes at each expansion stage; a larger number of expanded nodes means it is more likely to find a solution in difficult cases, at the cost of a longer running time. In actual use, once a valid solution is found in the simulation phase under the MII constraint, the whole mapping procedure ends.

3.6 Training Strategy

3.6.1 Data Augmentation. We apply data augmentation to the training process in order to alleviate the lack of training data and make the model more robust. By analyzing the symmetry of the target CGRA, we flip, shift, and rotate the searched mapping results to get more (s, π, r) groups.

3.6.2 Pre-training. While it would be possible to use this framework directly to obtain mapping results for unseen DFGs through multiple iterations, using a pre-trained agent for inference can produce quality mappings very quickly. In the mapping process, most of the time overhead lies in the network inference, so we use multi-threading during execution. For faster convergence, we generate a

Algorithm 1 RL-MCTS-based search space exploration

Input: the graph representation of DFG and CGRA; metadata; mapping rules; history data
Output: policy (action probabilities) p, value v

initialize neural network weights θ_0 randomly;
for *each interation* i **do**
  initialize s_0;
  // termination conditions:
  // 1. all nodes have been mapped
  // 2. no available PE exists
  for *each step* t, *until termination at step* T **do**
    // execute an MCTS search with the previous neural network
    // each edge in the search tree stores a prior probability // $P(s, a)$, a visit count $N(s, a)$, and an action value $Q(s, a)$
    while *computation resource remains* **do**
      select the edge with maximum UCT;
      expand the leaf node and evaluate its state s by the neural network, store policy logits P in the outgoing edges;
      increment the visit counts $N(s, a)$ and record the mean action values $Q(s, a)$ of the traversed edges;
      store the state, policy, reward (s, π, r) groups of the traversed trajectory in the self-play buffer;
    end while
  end for
  give a final reward;
  // update network weights θ_i
  sample a batch of (s, π, r) among the self-play buffer;
  get estimated policy and value $(p, v) = f_{\theta_i}(s)$;
  update the network by minimizing the loss function via stochastic gradient descent:
  $\theta_{i+1} = \arg min \frac{1}{|Batch_i|T} \Sigma_{b \in Batch_i} \Sigma_{t=0}^{T} ((r - v)^2 - \pi \log p)$
  where the gradient is clipped to avoid gradient explosion;
end for

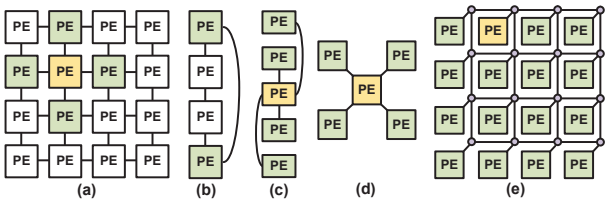

Figure 7: PE interconnection styles. (a) 4× 4 mesh-like CGRA. (b) Toroidal connection. (c) 1-hop connection. (d) Diagonal connection. (e) Circuit-switched crossbar.

Table 1: Target CGRAs used in the evaluation.

	Mesh	1-hop	Diagonal	Torroidal	Crossbar
HReA	✓	✓	✓	✓	
MorphoSys	✓	✓		✓	
ADRES	✓	✓		✓	
8×8 baseline	✓	✓	✓		
16×16 baseline	✓	✓	✓	✓	
HyCube					✓

random set of DFGs and feed them to the network in the order of ease to hard, which draws on the idea of curriculum learning [50], i.e., gradually introducing more difficult tasks to speed up training. When higher quality solutions are expected, the pre-trained agent can be further fine-tuned on the particular DFG.

When mapping a new DFG with the pre-trained agent, we allow backtracking when traversing down the search tree. Once the PE assignment for a node is found to yield an undesirable reward, we unmap it and allow the agent to perform a different action. Backtracking allows the agent to have minor errors and timely remediate them, which can enhance the success rate of mapping with little time overhead.

4 EVALUATION

In this section, we evaluate the compilation time overhead, mapping quality, and generality of our design compared with state-of-the-art CGRA compilers: CGRA-ME (ILP and SA) [13] and LISA [37].

4.1 Experimental Setup

4.1.1 Target Architectures. To demonstrate the generality of our design, we evaluate the framework over notable architectures: HReA [40], MorphoSys [60], ADRES [46], and HyCube [28]. The PE interconnections used are listed in Fig. 7. The topologies of target architectures are varied combinations of them, covering the routing manners of most CGRAs in the literature. The PEA size differs from 4×4 to 16×16. Each PE is assumed to have five constant units, two load units, one ALU, one store unit, and one output register (except ADRES). In ADRES, PEs in the same row share the same bus connection to the memory. Table 1 lists the target CGRAs.

4.1.2 Benchmarks. We extract loop kernels from Microbench [14], ExPRESS benchmark [29], and Embench-IoT [6] with operations widely used by academic researchers for CGRA mapping. DFGs are generated by LLVM and optimized for the target architectures (node balancing and memory access alignment operation elimination). The number of operations and edges for each DFG kernel are summarized in Table 2 (listed in alphabetical order). We choose relatively challenging benchmarks (with high PE occupancy and diverse workloads) to evaluate the compilers. The unrolled cases are for scalability evaluation.

Table 2: Statistics of the benchmark DFGs (u means unrolled).

Benchmarks	Vertices	Edges	Benchmarks	Vertices	Edges
accumulate	21	25	mac	12	14
arf	54	86	mac2	40	46
cap	42	47	matmul	26	28
conv2	18	20	mults1	34	38
conv3	28	31	mults2	42	48
filter_u	180	201	mulul	97	108
huf_u	592	720	sort_u	328	400
h2v2	68	71	stencil_u	141	159
jpegdct_u	255	295	sum	8	9

4.1.3 State-of-the-art compiler. We choose CGRA-ME [13], a prevailing open-source CGRA compilation framework based on Integer Linear Programming (ILP). CGRA-ME has the merit of being compatible with diverse architectures. Its ILP models can be solved by commercial optimization solvers such as Gurobi 9.0.0 [52], which enables multi-core to speed up the process of solution space search. CGRA-ME also supports Simulated Annealing (SA) as another option. SA is currently the most prevalent meta-heuristic compilation algorithm for reconfigurable accelerators [14, 45, 63, 71] due to its high efficiency and generality. A recently proposed GNN-based compilation algorithm, LISA [37], is also involved in our experiment. LISA utilizes labels generated by GNN to improve the traditional SA by directing the mapping process. Since LISA and CGRA-ME (SA) are based on simulated annealing, their mapping processes carry large uncertainty. Thus, for each case, we took the geometric mean of three attempts. All the experiments were performed on the same Linux workstation with 4 Intel Xeon 3.40 GHz CPUs. We implemented the models with PyTorch [53].

4.2 Mapping Quality

Without loss of generality, we chose a number of CGRAs with various interconnection complexities to evaluate our framework. Initiation Interval (II) is the predominant metric for mapping quality evaluation. Lower II means higher performance. To achieve the minimal II (MII), we set MapZero and all the baseline compilers to start with MII and gradually increase the target II if mapping fails under the current II.

We set the time limit to 8 hours. The MCTS tree expands 100 nodes per expansion stage. The number of DFG nodes used in pre-training ranges from 3 to 30. Fig. 8 (a)-(c) shows the mapping quality with ratios of II values obtained by CGRA-ME (ILP), CGRA-ME (SA), LISA, and MapZero relative to MII on HReA, MorphoSys, and ADRES. MapZero can successfully map all the benchmarks and always achieve the MII, while SA and LISA timed out in most cases and ILP failed in the case of large DFGs.

According to our experiments, LISA is only applicable to single-cycle multi-hop interconnect architectures like HyCube [28], where each PE is associated with a fully-connected crossbar which is connected in a grid, and fails on other topologies. We find that LISA may generate improper labels for unfamiliar topologies (e.g. HReA), resulting in always failing to find the correct solution. Thus, we compare the mapping quality of MapZero and LISA on HyCube, as shown in Fig. 8(d), to further demonstrate the generality of our framework. For cases with high PE occupancy, LISA cannot always find a solution. Since our MapZero can obtain the features and

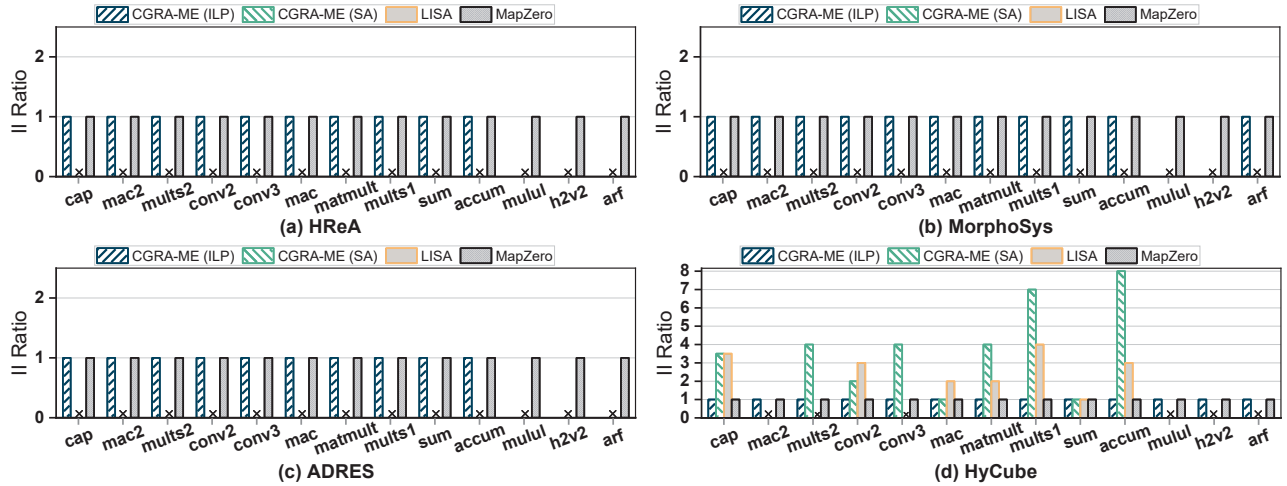

Figure 8: II ratio of multiple compilation methods relative to MII. The lower the II, the better the mapping quality and performance. II of failed mapping is set to 0.

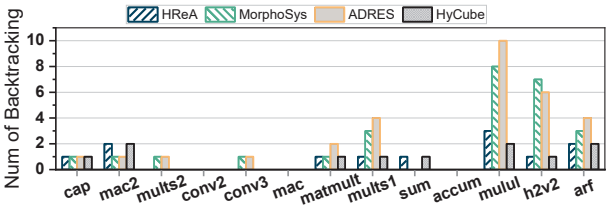

Figure 9: The number of backtracking operations required in the mapping process.

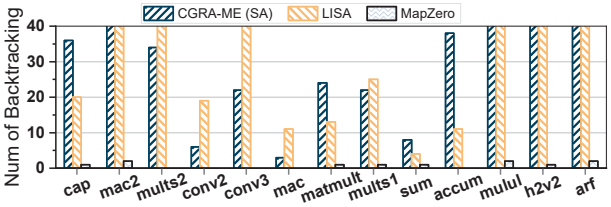

Figure 10: The number of backtracking/annealing operations required by multiple compilation methods.

structures of both DFG and CGRA at the same time, it is easier to explore solutions in the huge search space.

All the above experimental data demonstrates that MapZero can map successfully even if there are few feasible solutions in the search space, owing to the domain knowledge gained by the well-trained agent.

4.3 Time Overhead

Fig. 11 shows the compilation time overhead for ILP, SA, LISA, and MapZero on target CGRAs. We set the target II to MII at the beginning of each mapping process and limit the mapping time to 8 hours. MapZero achieves 50x, 45x, and 274x geo-mean compilation time reduction compared to CGRA-ME (ILP) for HReA, MorphoSys, and ADRES. We compare the time overhead of MapZero and LISA on HyCube and see a 405x reduction in geo-mean compilation time. Also on HyCube, MapZero achieves 214x and 594x compilation time reduction compared to CGRA-ME (ILP) and CGRA-ME (SA), respectively. We exclude the timeout cases when calculating runtime reduction.

Fig. 9 shows the number of backtracking operations required in the mapping process for target architectures. The number of backtracking operations required in most situations is very small and poses no significant time overhead. The agent's decisions are highly accurate. Fig. 10 compares the number of backtracks of MapZero to baseline methods on HyCube. In the case of CGRA-ME (SA) and LISA, times of annealings are counted. The compilation time is not always proportional to the number of annealings as 100 random perturbations are made before each annealing. Since CGRA-ME (ILP) is solved with Gurobi optimizer [52], which uses simplex iterations, cutting planes, and other methods, it is difficult to count backtracks.

In summary, MapZero can save laborious traversal efforts and a well-trained RL agent can generate quality mapping quickly. MapZero outperforms CGRA-ME (ILP), CGRA-ME (SA), and LISA in almost all the cases and can be adaptive to various topologies.

4.4 Training and Convergence

In the training process, we use DNN and UCT value to select and expand the MCTS tree and store the trajectories into a replay buffer of size 10,000. We randomly sample a batch of size 32 once the replay buffer is full and start training. A sampling priority is maintained. Already sampled trajectories will be given a lower priority in the next round of sampling.

To evaluate the agent's progress during training, we track the learning rate, the average loss per epoch, the average reward per epoch, and the routing penalty evaluated at the end of each epoch. Fig. 12 demonstrates the learning curves for HReA. We can see a considerable decline in policy and value loss and a steady ascent in reward. During evaluation, each node placement causing a routing conflict introduces a penalty of -100. In our context, a total routing penalty greater than -100 means successful mapping. Variable routing penalty values are only meant to indicate different PE choices and are irrelevant to the real mapping quality. After enough training, the mapping is valid in all the tests. Since backtracking allows some tolerance for the agent's misjudged actions, the training accuracy is sufficient for a portable, highly efficient CGRA compiler.

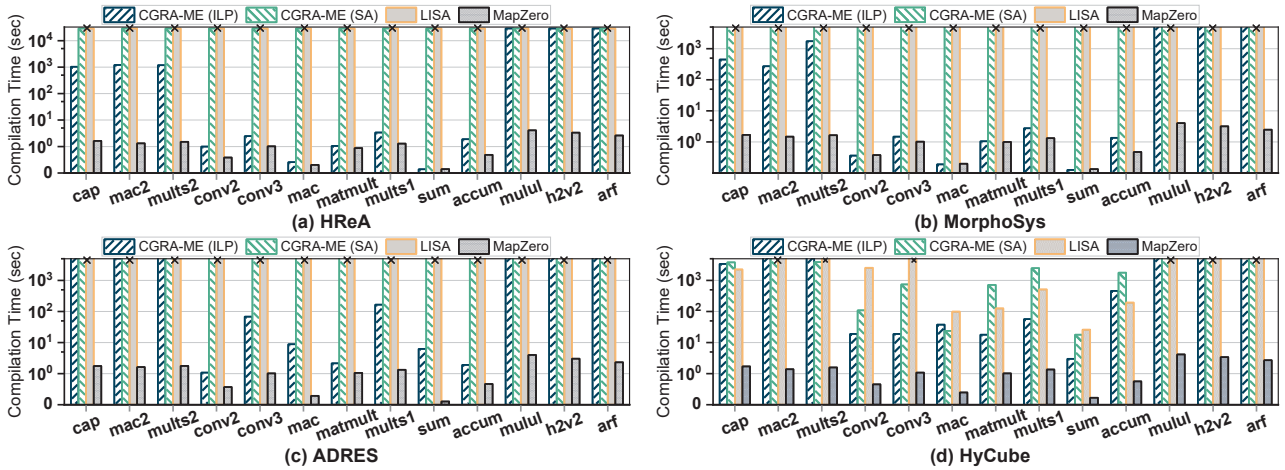

Figure 11: Compilation time comparison of CGRA-ME-ILP, CGRA-ME-SA, LISA, and MapZero on different CGRA topologies. (a) HReA, (b) MorphoSys, (c) ADRES, and (d) HyCube.

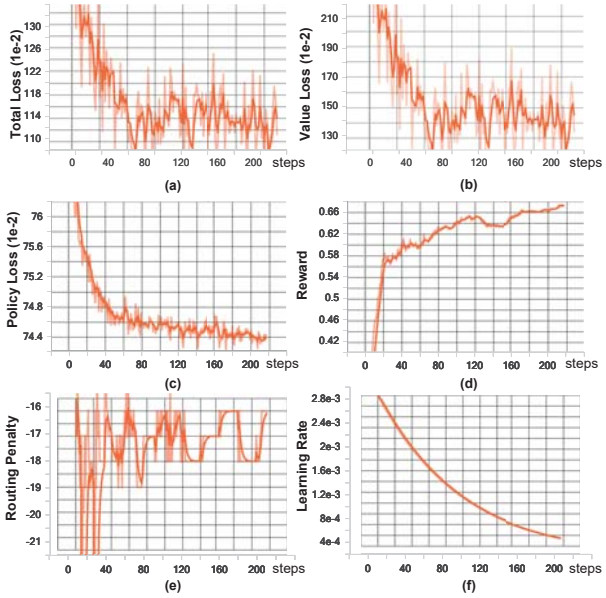

Figure 12: Learning curves in training. The darker-colored curves are smoothed to reveal the trends more clearly. (a) Average total loss per epoch. (b) Average value loss per epoch. (c) Average policy loss per epoch. (d) Average reward. (e) Routing penalty (in evaluation). (f) Learning rate.

4.5 Scalability to Different Sizes

Given that the GNN layer size is only correlated with the dimension of node features, MapZero can be applied to various PEA sizes without requiring changes to the GNN network structure. The sizes of the output layers in the policy network are determined by the action space, i.e. the PEA size. Training time for PEAs of sizes 4×4, 8×8, and 16×16 is 5, 10, and 30 hours, respectively. In the case of 16×16, the MCTS tree expands 200 nodes per expansion stage. We use some unrolled benchmarks to evaluate the scalability of MapZero. Fig. 13 shows the compilation time by MapZero, ILP, SA, and LISA. MapZero successfully finds valid mappings in all cases with the minimal II, while the others fail. It demonstrates that MapZero can scale to varying sizes of PEA and is capable of crunching diverse workloads. While ILP and SA are highly unfavorable in the face of a tremendous search space.

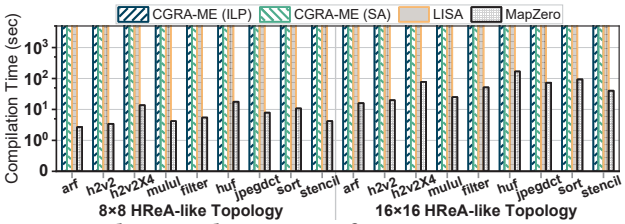

Figure 13: The compilation time of mapping DFGs to 8×8 and 16×16 baseline CGRAs.

4.6 Generality on Heterogeneous Architecture

Our GNN-based graph embedding generator can capture heterogeneous features of architectures. To demonstrate the generality of MapZero, we evaluate it on a heterogeneous CGRA, as shown in Fig. 14. The operations supported by each PE are labeled in the figure. Fig. 15 shows the mapping quality and compilation time ratio to the CGRA-ME (ILP), as well as the number of backtracking operations. MapZero shows good adaptability by achieving the identical II in much less time compared to CGRA-ME (ILP).

4.7 Ablation Study

We perform an ablation study by removing MCTS to verify its contribution to the whole system. We ran 13 cases on each of the 4 CGRAs as in Section 4.2, and only 35 out of 52 were successful in finding MII in time. It shows that removing MCTS results in substantial performance degradation, thereby demonstrating that the integration of MCTS is essential for building an efficient and portable CGRA compiler.

4.8 Discussion

As mentioned before, GNN extracts graph information effectively and an RL agent can actively explore design space and learn from prior experiences. GNN-based RL has become a well-established and widely used paradigm [8, 12, 20, 47, 69, 75, 85]. But for CGRA, there are not many applications yet. Our framework can be further

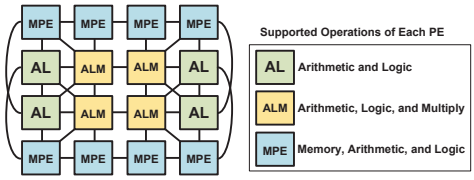

Figure 14: A heterogeneous architecture.

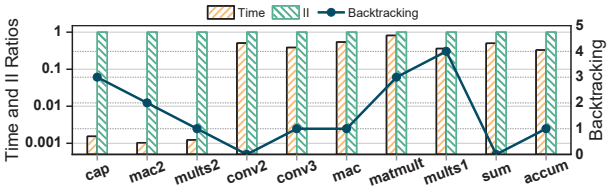

Figure 15: Compilation time and II ratio in comparison to CGRA-ME (ILP), as well as the number of backtracking operations on a heterogeneous architecture.

applied to scheduling [1], design space exploration (DSE) [3, 71], performance evaluation [56, 62], and high-level synthesis [75, 78] with slight modifications to state/action representations. For instance, the state representation of DFG and CGRA hardware is compatible with the DSE task. By analyzing a set of DFGs, the agent can take actions to add or remove PEs, interconnects, or memory ports in order to get the best domain-specific accelerator design under certain metrics. As another example, this framework can also be used for dynamic scheduling of CGRA, where the agent maps DFG nodes onto PEs of different time domain extensions to obtain the minimum makespan. There are many other possibilities worth exploring in future work.

5 RELATED WORKS

5.1 Existing CGRA Compilers

We discuss the latest CGRA compilation techniques from the following two aspects.

5.1.1 Efficient space exploration. Very recently, there have been lines of CGRA compiler research targeting efficient search space traversal [2, 4, 7, 36, 68]. They seek to reduce the search space by introducing search heuristics and employing higher-level abstractions and have achieved great improvements. Yet these works fall within the scope of heuristic or mathematical optimization-based methods, facing challenges of scalability and easy adaptability [37, 44].

5.1.2 Machine learning assisted approach. Some CGRA compilers have leveraged machine learning to perform placement directly or guide mapping via prediction. [8] applies GNN-based RL to do operation placement. DFGNet [79] translates the compilation process into an image-based classification problem using a convolutional neural network (CNN) and utilizes the prediction results to guide mapping. RLMap [39] uses CNN as the front end of the policy network to embed the current mapping state and applies RL to place nodes. [33] uses GNN to estimate the scale of PEs needed for DFGs to aid the genetic algorithm for graph partitioning. LISA [37] uses GNN to analyze DFGs and generate labels via supervised learning as a way to speed up the SA process. Nevertheless, [8] targets a special near-data computing architecture that provides asynchronous messaging through NOC, where the routing constraints important in general CGRA mapping become trivial. RLMap [39], DFGNet [79] and [33] only support DFGs with fewer nodes than the number CGRA PEs (II=1). These methods do not directly support temporal mapping (II>1), which limits their scalability. LISA is limited by special crossbar-based network topologies, and its automatically generated labels do not guarantee good results for new architectures. By contrast, our framework uses GNN to encode CGRA architectures to allow the agent to fully understand hardware characteristics. Furthermore, the integration of MCTS and RL enables more efficient exploration of the space and generation of training sets. Our RL agent can acquire heuristics automatically by interacting with the environment other than through limited knowledge from supervised learning.

5.2 Applications of Machine Learning

5.2.1 Graph Neural Network. GNNs have become a strong candidate for the Electronic Design Automation (EDA) flow for chip design [42] since they can exploit the graph structures of netlists, data flow graphs (DFG), and intermediate RTLs by effectively extracting useful topology information and data dependency as embedding vectors, thus facilitate subsequent steps in logic synthesis [65, 75], floorplanning [77], placement [12, 47], verification [82], analog design [55, 69], etc. In recent works, GNN has also been used to help and guide the compilation of CGRA, such as in LISA [37] and [33].

5.2.2 Reinforcement Learning and Monte-Carlo Tree Search. RL and MCTS have yielded brilliant results in chess and computer games in recent years[57, 59]. They can acquire domain knowledge and historical experience by interacting with the environment, thereby achieving better performance. RL and MCTS have been applied separately or together in the field of computer architecture and EDA to perform NoC design [24, 38, 70], scheduling [20, 25, 85], prefetching [5], placement [12, 47] routing [81], logic synthesis [23, 75], analog design [69], CGRA compilation [39], etc.

6 CONCLUSION

In this work, we target the complex problem of CGRA mapping. We propose MapZero, aiming to effectively generate quality mappings for various CGRAs. MapZero uses GAT to generate embeddings for DFG and CGRA hardware, which are further used in the RL-MCTS-based solution space search. A well-trained RL agent can generalize to new DFGs with a minimum amount of fine-tuning effort. Experimental results show that MapZero outperforms state-of-the-art baseline compilers in time overhead and mapping quality. It can find high-quality mappings very quickly when the feasible solution space is rather small and all other compilers fail. MapZero also shows superior generalizability. We further present insights about the broad applicability of our framework in the field of CGRA.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their thoughtful comments. This work is supported in part by the National Natural Science Foundation of China (Grant No. 62204139), and in part by the Jiangsu Industrial prospect and key core technology - competitive projects under Grant No. BE2020016.

REFERENCES

[1] Mahesh Balasubramanian and Aviral Shrivastava. 2020. CRIMSON: Compute-Intensive Loop Acceleration by Randomized Iterative Modulo Scheduling and Optimized Mapping on CGRAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 3300–3310. https://doi.org/10.1109/TCAD.2020.3022015 Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[2] Mahesh Balasubramanian and Aviral Shrivastava. 2022. PathSeeker: A Fast Mapping Algorithm for CGRAs. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 268–273. https://doi.org/10.23919/DATE54114.2022.9774520 ISSN: 1558-1101.

[3] Thilini Kaushalya Bandara, Dhananjaya Wijerathne, Tulika Mitra, and Li-Shiuan Peh. 2022. REVAMP: a systematic framework for heterogeneous CGRA realization. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, Lausanne Switzerland, 918–932. https://doi.org/10.1145/3503222.3507772

[4] Rami Beidas and Jason H. Anderson. 2022. CGRA Mapping Using Zero-Suppressed Binary Decision Diagrams. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 616–622. https://doi.org/10.1109/ASP-DAC52403.2022.9712571 ISSN: 2153-697X.

[5] Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. 2021. Pythia: A customizable hardware prefetching framework using online reinforcement learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1121–1137.

[6] Andrew Burgess. 2021. Embench-Iot. https://www.embench.org/.

[7] Michael Canesche, Marcelo Menezes, Westerley Carvalho, Frank Sill Torres, Peter Jamieson, José Augusto Nacif, and Ricardo Ferreira. 2021. TRAVERSAL: A Fast and Adaptive Graph-Based Placement and Routing for CGRAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 8 (2021), 1600–1612. https://doi.org/10.1109/TCAD.2020.3025513 Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[8] Andre Xian Ming Chang, Parth Khopkar, Bashar Romanous, Abhishek Chaurasia, Patrick Estep, Skyler Windh, Doug Vanesko, Sheik Dawood Beer Mohideen, and Eugenio Culurciello. 2022. Reinforcement Learning Approach for Mapping Applications to Dataflow-Based Coarse-Grained Reconfigurable Array. https://doi.org/10.48550/arXiv.2205.13675 arXiv:2205.13675 [cs].

[9] Samit Chaudhuri and Asmus Hetzel. 2017. SAT-based compilation to a non-vonNeumann processor. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 675–682. https://doi.org/10.1109/ICCAD.2017.8203842 ISSN: 1558-2434.

[10] Liang Chen and Tulika Mitra. 2014. Graph minor approach for application mapping on cgras. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 7, 3 (2014), 1–25.

[11] Longlong Chen, Jianfeng Zhu, Yangdong Deng, Zhaoshi Li, Jian Chen, Xiaowei Jiang, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2021. An Elastic Task Scheduling Scheme on Coarse-Grained Reconfigurable Architectures. *IEEE Transactions on Parallel and Distributed Systems* 32, 12 (2021), 3066–3080. https://doi.org/10.1109/TPDS.2021.3084804 Conference Name: IEEE Transactions on Parallel and Distributed Systems.

[12] Ruoyu Cheng and Junchi Yan. 2021. On Joint Learning for Solving Placement and Routing in Chip Design. In *Advances in Neural Information Processing Systems*, Vol. 34. Curran Associates, Inc., 16508–16519. https://proceedings.neurips.cc/paper/2021/hash/898aef0932f6aaecda27aba8e9903991-Abstract.html

[13] S. Alexander Chin and Jason H. Anderson. 2018. An Architecture-Agnostic Integer Linear Programming Approach to CGRA Mapping. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465799

[14] S. Alexander Chin, Noriaki Sakamoto, Allan Rui, Jim Zhao, Jin Hee Kim, Yuko Hara-Azumi, and Jason Anderson. 2017. CGRA-ME: A unified framework for CGRA modelling and exploration. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 184–189. https://doi.org/10.1109/ASAP.2017.7995277 ISSN: 2160-052X.

[15] Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*. Springer, 72–83.

[16] Shail Dave, Mahesh Balasubramanian, and Aviral Shrivastava. 2018. RAMP: Resource-Aware Mapping for CGRAs. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465892

[17] Caleb Donovick, Makai Mann, Clark Barrett, and Pat Hanrahan. 2019. Agile SMT-Based Mapping for CGRAs with Restricted Routing Networks. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. 1–8. https://doi.org/10.1109/ReConFig48160.2019.8994781 ISSN: 2640-0472.

[18] Stephen Friedman, Allan Carroll, Brian Van Essen, Benjamin Ylvisaker, Carl Ebeling, and Scott Hauck. 2009. SPR: an architecture-adaptive CGRA mapping tool. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. 191–200.

[19] Graham Gobieski, Ahmet Oguz Atli, Kenneth Mai, Brandon Lucia, and Nathan Beckmann. 2021. Snafu: An Ultra-Low-Power, Energy-Minimal CGRA-Generation Framework and Architecture. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1027–1040. https://doi.org/10.1109/ISCA52012.2021.00084 ISSN: 2575-713X.

[20] Nathan Grinsztajn, Olivier Beaumont, Emmanuel Jeannot, and Philippe Preux. 2021. READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. 70–81. https://doi.org/10.1109/Cluster48925.2021.00031 ISSN: 2168-9253.

[21] Jiangyuan Gu, Shouyi Yin, Shaojun Wei, et al. 2018. Stress-aware loops mapping on CGRAs with dynamic multi-map reconfiguration. *IEEE Transactions on Parallel and Distributed Systems* 29, 9 (2018), 2105–2120.

[22] Yijiang Guo, Jiarui Wang, Jiaxi Zhang, and Guojie Luo. 2021. Formulating Data-arrival Synchronizers in Integer Linear Programming for CGRA Mapping. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 943–948.

[23] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. 2020. DRiLLS: Deep reinforcement learning for logic synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 581–586.

[24] Yong Hu, Daniel Mueller-Gritschneder, and Ulf Schlichtmann. 2018. Wavefront-MCTS: Multi-objective Design Space Exploration of NoC Architectures based on Monte Carlo Tree Search. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1145/3240765.3240863 ISSN: 1558-2434.

[25] Zhiming Hu, James Tu, and Baochun Li. 2019. Spear: Optimized Dependency-Aware Task Scheduling with Deep Reinforcement Learning. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2037–2046. https://doi.org/10.1109/ICDCS.2019.00201 ISSN: 2575-8411.

[26] Yuanjie Huang, Paolo Ienne, Olivier Temam, Yunji Chen, and Chengyong Wu. 2013. Elastic CGRAs. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '13)*. Association for Computing Machinery, New York, NY, USA, 171–180. https://doi.org/10.1145/2435264.2435296

[27] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. 2020. ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning. http://arxiv.org/abs/2009.02010 Number: arXiv:2009.02010 arXiv:2009.02010 [cs, eess].

[28] Manupa Karunaratne, Aditi Kulkarni Mohite, Tulika Mitra, and Li-Shiuan Peh. 2017. HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/3061639.3062262

[29] Ryan Kastner. 2005. ExPRESS - Benchmarks. https://web.ece.ucsb.edu/EXPRESS/.

[30] Changmoo Kim, Mookyoung Chung, Yeongon Cho, Mario Konijnenburg, Soojung Ryu, and Jeongwook Kim. 2012. ULP-SRP: Ultra low power Samsung Reconfigurable Processor for biomedical applications. In *2012 International Conference on Field-Programmable Technology*. 329–334. https://doi.org/10.1109/FPT.2012.6412157

[31] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.

[32] Takuya Kojima, Nguyen Anh Vu Doan, and Hideharu Amano. 2020. GenMap: A genetic algorithmic approach for optimizing spatial mapping of coarse-grained reconfigurable architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 11 (2020), 2383–2396.

[33] Takuya Kojima, Ayaka Ohwada, and Hideharu Amano. 2022. Mapping-Aware Kernel Partitioning Method for CGRAs Assisted by Deep Learning. *IEEE Transactions on Parallel and Distributed Systems* 33, 5 (2022), 1213–1230. https://doi.org/10.1109/TPDS.2021.3107746 Conference Name: IEEE Transactions on Parallel and Distributed Systems.

[34] Xiangyu Kong, Jianfeng Zhu, Xingchen Man, Guihuan Song, Yi Huang, Chenchen Deng, Pengfei Gou, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2023. M2STaR: A Multi-Mode Spatio-Temporal Redundancy Design for Fault-Tolerant Coarse-Grained Reconfigurable Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023).

[35] Mingyang Kou, Jiangyuan Gu, Shaojun Wei, Hailong Yao, and Shouyi Yin. 2020. TAEM: Fast Transfer-Aware Effective Loop Mapping for Heterogeneous Resources on CGRA. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218668 ISSN: 0738-100X.

[36] Jinho Lee and Trevor E. Carlson. 2021. Ultra-Fast CGRA Scheduling to Enable Run Time, Programmable CGRAs. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 1207–1212. https://doi.org/10.1109/DAC18074.2021.9586255 ISSN: 0738-100X.

[37] Zhaoying Li, Dan Wu, Dhananjaya Wijerathne, and Tulika Mitra. 2022. LISA: Graph Neural Network based Portable Mapping on Spatial Accelerators. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, Seoul, Korea, Republic of, 444–459. https://doi.org/10.1109/HPCA53966.2022.00040

[38] Ting-Ru Lin, Drew Penney, Massoud Pedram, and Lizhong Chen. 2020. A Deep Reinforcement Learning Framework for Architectural Exploration: A Routerless NoC Case Study. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 99–110. https://doi.org/10.1109/HPCA47549.2020.00018 ISSN: 2378-203X.

[39] Dajiang Liu, Shouyi Yin, Guojie Luo, Jiaxing Shang, Leibo Liu, Shaojun Wei, Yong Feng, and Shangbo Zhou. 2019. Data-Flow Graph Mapping Optimization for CGRA With Deep Reinforcement Learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 12 (2019), 2271–2283. https://doi.org/10.1109/TCAD.2018.2878183 Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[40] Leibo Liu, Zhaoshi Li, Chen Yang, Chenchen Deng, Shouyi Yin, and Shaojun Wei. 2018. HReA: An Energy-Efficient Embedded Dynamically Reconfigurable Fabric for 13-Dwarfs Processing. *IEEE Transactions on Circuits and Systems II: Express Briefs* 65, 3 (March 2018), 381–385. https://doi.org/10.1109/TCSII.2017.2728814 Conference Name: IEEE Transactions on Circuits and Systems II: Express Briefs.

[41] Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. 2019. A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–39.

[42] Daniela Sanchez Lopera, Lorenzo Servadei, Gamze Naz Kiprit, Souvik Hazra, Robert Wille, and Wolfgang Ecker. 2021. A Survey of Graph Neural Networks for Electronic Design Automation. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*. IEEE, Raleigh, NC, USA, 1–6. https://doi.org/10.1109/MLCAD52597.2021.9531070

[43] Xingchen Man, Jianfeng Zhu, Guihuan Song, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2022. CaSMap: agile mapper for reconfigurable spatial architectures by automatically clustering intermediate representations and scattering mapping process. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ACM, New York New York, 259–273. https://doi.org/10.1145/3470496.3527426

[44] Kevin J. M. Martin. 2022. Twenty Years of Automated Methods for Mapping Applications on CGRA. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 679–686. https://doi.org/10.1109/IPDPSW55747.2022.00118

[45] Bingfeng Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. 2002. DRESC: a retargetable compiler for coarse-grained reconfigurable architectures. In *2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings.* 166–173. https://doi.org/10.1109/FPT.2002.1188678

[46] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. 2003. ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In *International Conference on Field Programmable Logic and Applications*. Springer, 61–70.

[47] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Anand Babu, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. 2020. Chip Placement with Deep Reinforcement Learning. *arXiv:2004.10746 [cs]* (April 2020). http://arxiv.org/abs/2004.10746 arXiv: 2004.10746.

[48] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

[49] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[50] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. 2020. Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint arXiv:2003.04960* (2020).

[51] Tony Nowatzki, Newsha Ardalani, Karthikeyan Sankaralingam, and Jian Weng. 2018. Hybrid optimization/heuristic instruction scheduling for programmable accelerator codesign. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. ACM, Limassol Cyprus, 1–15. https://doi.org/10.1145/3243176.3243212

[52] LLC Gurobi Optimization. 2021. LLC Gurobi Optimization. *Gurobi optimizer reference manual* (2021).

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[54] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. 2017. Plasticine: A reconfigurable architecture for parallel patterns. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 389–402. https://doi.org/10.1145/3079856.3080256

[55] Haoxing Ren, George F Kokai, Walker J Turner, and Ting-Sheng Ku. 2020. ParaGraph: Layout parasitics and device parameter prediction using graph neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[56] Alex Renda, Yishen Chen, Charith Mendis, and Michael Carbin. 2020. Difftune: Optimizing cpu simulator parameters with learned differentiable surrogates. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 442–455.

[57] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609.

[58] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[59] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[60] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J Kurdahi, Nader Bagherzadeh, and Eliseu M Chaves Filho. 2000. MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE transactions on computers* 49, 5 (2000), 465–481.

[61] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[62] Cheng Tan, Chenhao Xie, Ang Li, Kevin J. Barker, and Antonino Tumeo. 2020. OpenCGRA: An Open-Source Unified Framework for Modeling, Testing, and Evaluating CGRAs. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*. 381–388. https://doi.org/10.1109/ICCD50377.2020.00070 ISSN: 2576-6996.

[63] Cheng Tan, Chenhao Xie, Ang Li, Kevin J. Barker, and Antonino Tumeo. 2021. AURORA: Automated Refinement of Coarse-Grained Reconfigurable Accelerators. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1388–1393. https://doi.org/10.23919/DATE51398.2021.9473955 ISSN: 1558-1101.

[64] Christopher Torng, Peitian Pan, Yanghui Ou, Cheng Tan, and Christopher Batten. 2021. Ultra-Elastic CGRAs for Irregular Loop Specialization. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 412–425. https://doi.org/10.1109/HPCA51647.2021.00042 ISSN: 2378-203X.

[65] Ecenur Ustun, Chenhui Deng, Debjit Pal, Zhijing Li, and Zhiru Zhang. 2020. Accurate operation delay prediction for FPGA HLS using graph neural networks. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9.

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[67] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[68] Matthew J. P. Walker and Jason H. Anderson. 2019. Generic Connectivity-Based CGRA Mapping via Integer Linear Programming. http://arxiv.org/abs/1901.11129 arXiv:1901.11129 [cs].

[69] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. 2020. GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[70] Ke Wang, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. 2019. IntelliNoC: A holistic design framework for energy-efficient and reliable on-chip communication for manycores. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–12.

[71] Jian Weng, Sihao Liu, Vidushi Dadu, Zhengrong Wang, Preyas Shah, and Tony Nowatzki. 2020. DSAGEN: Synthesizing Programmable Spatial Accelerators. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 268–281. https://doi.org/10.1109/ISCA45697.2020.00032

[72] Jian Weng, Sihao Liu, Zhengrong Wang, Vidushi Dadu, and Tony Nowatzki. 2020. A Hybrid Systolic-Dataflow Architecture for Inductive Matrix Algorithms. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 703–716. https://doi.org/10.1109/HPCA47549.2020.00063 ISSN: 2378-203X.

[73] Dhananjaya Wijerathne, Zhaoying Li, Thilini Kaushalya Bandara, and Tulika Mitra. 2022. PANORAMA: Divide-and-Conquer Approach for Mapping Complex Loop Kernels on CGRA. (2022), 6.

[74] Dhananjaya Wijerathne, Zhaoying Li, Anuj Pathania, Tulika Mitra, and Lothar Thiele. 2021. HiMap: Fast and Scalable High-Quality Mapping on CGRA via Hierarchical Abstraction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021), 1–1. https://doi.org/10.1109/TCAD.2021.3132551 Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[75] Nan Wu, Yuan Xie, and Cong Hao. 2021. Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning. In *Proceedings of the 2021 on Great Lakes Symposium on VLSI*. 39–44.

[76] Qingcheng Xiao, Size Zheng, Bingzhe Wu, Pengcheng Xu, Xuehai Qian, and Yun Liang. 2021. HASCO: Towards Agile HArdware and Software CO-design for Tensor Computation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1055–1068. https://doi.org/10.1109/ISCA52012.

2021.00086 ISSN: 2575-713X.

[77] Qi Xu, Hao Geng, Song Chen, Bo Yuan, Cheng Zhuo, Yi Kang, and Xiaoqing Wen. 2021. GoodFloorplan: Graph Convolutional Network and Reinforcement Learning Based Floorplanning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021), 1–1. https://doi.org/10.1109/TCAD.2021.3131550 Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[78] Hanchen Ye, Cong Hao, Jianyi Cheng, Hyunmin Jeong, Jack Huang, Stephen Neuendorffer, and Deming Chen. 2022. ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 741–755.

[79] Shouyi Yin, Dajiang Liu, Lifeng Sun, Leibo Liu, and Shaojun Wei. 2017. DFGNet: Mapping dataflow graph onto CGRA by a deep learning approach. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4. https://doi.org/10.1109/ISCAS.2017.8050274 ISSN: 2379-447X.

[80] Shouyi Yin, Xianqing Yao, Dajiang Liu, Leibo Liu, and Shaojun Wei. 2016. Memory-Aware Loop Mapping on Coarse-Grained Reconfigurable Architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 5 (2016), 1895–1908. https://doi.org/10.1109/TVLSI.2015.2474129 Conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems.

[81] Cong Zhang, Huilin Jin, Jienan Chen, Jinkuan Zhu, and Jinting Luo. 2020. A Hierarchy MCTS Algorithm for The Automated PCB Routing. In *2020 IEEE 16th International Conference on Control & Automation (ICCA)*. 1366–1371. https://doi.org/10.1109/ICCA51439.2020.9264558 ISSN: 1948-3457.

[82] Yanqing Zhang, Haoxing Ren, and Brucek Khailany. 2020. GRANNITE: Graph neural network inference for transferable power estimation. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[83] Zhongyuan Zhao, Weiguang Sheng, Qin Wang, Wenzhi Yin, Pengfei Ye, Jinchao Li, and Zhigang Mao. 2020. Towards higher performance and robust compilation for CGRA modulo scheduling. *IEEE Transactions on Parallel and Distributed Systems* 31, 9 (2020), 2201–2219.

[84] Hao Zheng, Ke Wang, and Ahmed Louri. 2021. Adapt-noc: A flexible network-on-chip design for heterogeneous manycore architectures. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 723–735.

[85] Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C. Ma, Qiumin Xu, Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, and James Laudon. 2019. GDP: Generalized Device Placement for Dataflow Graphs. *arXiv:1910.01578 [cs, stat]* (Sept. 2019). http://arxiv.org/abs/1910.01578 arXiv: 1910.01578.

[86] Rong Zhu, Bo Wang, and Dajiang Liu. 2022. RF-CGRA: A Routing-Friendly CGRA with Hierarchical Register Chains. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 262–267. https://doi.org/10.23919/DATE54114.2022.9774601 ISSN: 1558-1101.