SmartMap: Architecture-Agnostic CGRA Mapping using Graph Traversal and Reinforcement Learning

Fábio T. Ramos[1], Pedro E. F. Realino[1], Wagner A. Junior[1], Alex B. Vieira[2],
Ricardo S. Ferreira[1], José Augusto M. Nacif[1]
[1]Science and Technology Institute, Federal University of Viçosa, Brazil
[2]Computer Science Department, Federal University of Juiz de Fora, Brazil
{fabio.ramos, pedro.realino, wagnerdjunior, ricardo, jnacif}@ufv.br, alex.borges@ufjf.br

***Abstract*—Coarse-Grained Reconfigurable Architectures (CGRAs) have been the subject of extensive research due to their balance between performance, energy efficiency, and flexibility. CGRAs must be capable of executing a dataflow graph (DFG), which depends on a compiler producing quality valid mappings with feasible running time performance and portable mapping DFGs on different CGRA architectures. Machine learning-based compilers have shown promising results by presenting high quality and performance but offer limited portability. Moreover, some approaches do not explore efficient placement methods or do not demonstrate whether scaling to more challenging, less connected architectures. This paper presents SmartMap, an architecture-agnostic framework that uses an actor-critic reinforcement learning method applied to a Monte-Carlo Tree Search (MCTS) to learn how to map a DFG onto a CGRA. This framework offers full portability using a state-action representation layer in the policy network instead of a probability distribution over actions. SmartMap uses a graph traversal placement method to provide scalability and improve the efficiency of MCTS by enabling more efficient exploration during the search. Our results show that SmartMap has 2.81x more mapping capacity, a 16.82x speed-up in compilation time, and consumes fewer resources compared to the state-of-the-art.**

***Index Terms*—CGRA Mapping, Deep Reinforcement Learning, DRL, Graph Neural Network, GNN, Monte-Carlo Tree Search, MCTS, Zigzag Traversal, YOTO**

I. Introduction

There are three critical points in designing an integrated circuit for general-purpose processing: 1) performance, to execute instructions quickly; 2) flexibility, to process different applications; and 3) energy efficiency, to reduce power consumption. Different architectures excel in each of these three aspects, but Coarse-Grained Reconfigurable Architectures (CGRAs) stand out by achieving a good balance in all three [1]. However, their use depends on efficiently mapping the application instructions on the CGRA architecture.

A CGRA is an array where each position contains processing elements (PEs) connected by an on-chip network. The PEs execute operations, and the network shares the results between connected PEs. CGRAs must be able to execute a dataflow graph (DFG) extracted from a loop, where the vertices represent operations (load, store, arithmetic, etc.) and the edges represent the flow of these operations. The mapping process is an NP-Complete problem and involves: 1) placement: placing all the DFG nodes into the PEs of the CGRA; 2) routing: ensuring routing of the DFG edges using the CGRA network; and 3) scheduling: guaranteeing a valid schedule in such a way that the operations arrive simultaneously at the PEs. Thus, solving this problem requires a compiler (mapping method) that exhibits three main characteristics: 1) quality, the ability to produce valid mappings; 2) performance, the ability to produce mappings in a short or feasible time; and 3) portability, the ability to map DFGs onto different CGRA architectures.

Compilers based on metaheuristics [2], [3] and linear optimization [4], [5] model an objective function to be optimized. With a general-purpose function, these approaches can exhibit high portability. Still, due to the large search space and the stochastic nature of metaheuristics, it takes a long time to find valid solutions, especially for medium and large DFGs suffering from low scalability. In this context, graph traversal methods [6]–[11] offer greater scalability by using information from the DFG to identify and eliminate poor decisions. However, stochasticity also limits the quality of large DFGs. Moreover, some approaches address the mapping problem using graph theory [12]–[15] or clustering techniques [16]–[18]. PathSeeker [19] utilizes the mapping information in the event of failure to rearrange it and attempt to find a valid mapping instead of restarting the mapping process. Lastly, machine learning compilers are composed of techniques such as GNNs (Graph Neural Networks) [20]–[24], Monte Carlo Tree Search (MCTS) [20], reinforcement learning (RL) [20], [22]–[25], and transformer [25]. For instance, LISA [21] uses a GNN to learn to generate labels from the DFGs and guide the SA during the search. We discuss the remaining machine learning compilers in Section III.

This paper presents SmartMap, a framework based on RL, MCTS, GNN and graph traversal for CGRA mapping. We use a state representation layer to encode the state in a latent space through a neural network and GNN, which the actor-critic network uses to output the expected return and probabilities of taking each legal action in this state. Unlike standard policy (or actor) networks, we design a state-action representation layer to produce the probability of taking an action in a state rather than the distribution of probabilities over the actions,

978-3-9826741-0-0/DATE25/© 2025 EDAA

decoupling the neural network from the CGRA size. Then, to provide a more scalable and efficient approach, the state and environment were formulated according to the YOTO (You Only Traverse Once) placement method [8], a graph traversal approach. With this modeling, the agent can better use the MCTS by running simulations in promising states.

The contributions of this work are threefold. First, we propose SmartMap, a novel RL-MCTS-based compiler with a general model using a state-action representation layer in a policy network, allowing a single agent to share mapping knowledge and map a DFG across different CGRA sizes. Second, SmartMap can efficiently map from straightforward to more challenging CGRAs by integrating YOTO. Our approach reached 2.81x more mapping capacity and 16.82x speed-up in compilation time than the state-of-the-art. Third, we demonstrate that it is possible to improve machine learning (ML) compilers by using an efficient placement method rather than relying solely on machine learning techniques. SmartMap is open source and freely available [26].

The remainder of this work is as follows: Section II presents background concepts, Section III compares SmartMap with related work, Section IV details the methodology, Section V presents the results, and Section VI presents the conclusion.

II. Background

A. CGRA Classification

If all PEs have the same functionality, we classify the CGRA as **homogeneous** and otherwise as **heterogeneous**. A CGRA can also be **synchronous** if it synchronizes the PE executions according to a global clock, **asynchronous** if it achieves synchronization through control signals, or **elastic** if it uses buffers between PEs to ensure that all operations routed to them arrive simultaneously. Considering the network, a CGRA can be **neighbor-to-neighbor (N2N)** if it sends data to directly connected PEs in one cycle, as seen in systems like ADRES [27], HReA [28], and MorphoSys [29]; or **circuit-switched crossbar** if it can send data to PEs that are k-hops away in one cycle, as demonstrated by HyCUBE [30].

Fig. 1 depicts the interconnection topologies. The N2N topology includes various interconnection styles, such as a **mesh**, which connects a PE with neighbors one hop away in the north, south, west, and east directions; a **one-hop**, which connects PEs with neighbors up to two hops away, also in the four directions; a **diagonal**, which connects a PE to other PEs one hop away along its diagonal; and a **toroidal**, which connects PEs on the border with PEs on the opposite border in the same row and column.

B. Graph Traversal - You Only Traverse Once

YOTO [8] is a simple greedy traversal approach proposed to minimize buffer usage in elastic CGRAs. The algorithm captures correlations between multiple outputs in the DFG by performing a zigzag traversal, a bi-directional depth-first traversal method [8]. This algorithm returns a list of edges in the form $L = \{(a_1, b_1), ..., (a_k, b_k)\}$, representing the order in which the edges were traversed. YOTO uses this list for

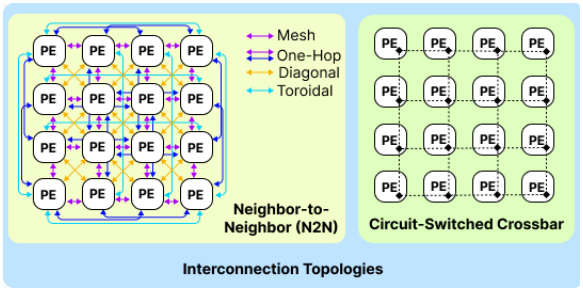

Fig. 1. Interconnection topologies.

node placement, prioritizing the placement of node a_1 first. Then, YOTO places all edges sequentially, where at the time of placing edge i, node a_i is an already placed node, and node b_i is the node under placement in a PE connected to node a_i's PE placement. Note that this greedy approach assumes that nearby nodes in the DFG should be placed in adjacent PEs, reducing the search space by eliminating non-optimal placements.

C. Reinforcement Learning Summarization

Reinforcement learning methods train an agent to find an optimal policy, a plan to maximize the accumulated discounted rewards (or return). The policy can either map a state to an action or produce a probability distribution over possible actions. Additionally, the agent can use a value function to predict the expected return in each state. The policy is implicit when using a value function, meaning the selected action has the highest return. Methods using a value function are known as value-based methods, while methods that rely solely on the policy are called policy-based methods. Some methods use both a policy to make decisions (actor) and a value function to evaluate how good or bad those decisions were (critic), known as actor-critic methods. Lastly, deep reinforcement learning employs neural networks to estimate policy and value functions. See [31] for in-depth knowledge.

D. CGRA Mapping Problem Complexity

Considering $|PEs|$ as the number of PEs in an architecture and $|DFG|$ as the number of nodes in a DFG, the number of possibilities for assigning all DFG nodes in separate PEs is given by: $(|PEs|)!/(|PEs| - |DFG|)!$. In other words, as the number of nodes in a DFG and the number of PEs increase, the solution space grows factorially. For example, the number of possibilities for placing a DFG with the number of nodes equal to the number of PEs in architectures of size 4×4, 8×8, and 16×16 is approximately 20.92×10^{12}, 126.89×10^{87}, and 857.82×10^{504}, respectively. Additionally, Fig. 2 shows that as the number of connections in an architecture decreases, the rate of valid solutions reduces drastically. We have used A* for routing, starting from nodes ordered topologically. *V_x_E_y* represents a DFG with *x* nodes and *y* edges, and the number of valid solutions for the bars with unseen values is minimal but not zero.

Thus, CGRA mapping can be more straightforward at a higher cost with more connections in the architecture or

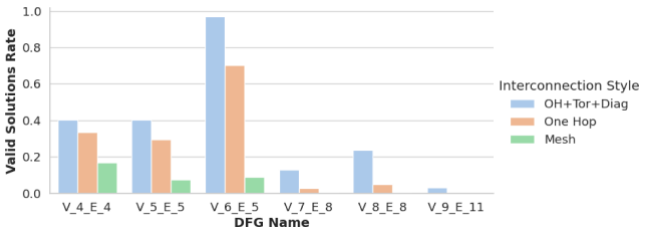

Fig. 2. Valid solution rates considering all placements for different DFGs and 3x3 CGRA interconnection styles.

more difficult at a lower cost with fewer connections. A compiler must be capable of scaling its results from more-connected to less-connected architectures to minimize costs without compromising quality.

III. Related Work

In this Section, we compare SmartMap to the state-of-the CGRA mapping frameworks.

A. MapZero

MapZero [20] is a framework that uses neural networks, GNN, RL, and MCTS to learn how to map a DFG onto a CGRA from scratch. Although MapZero has shown promising results, it is unclear whether the results scale to more challenging CGRAs. The following sections outline the main similarities and differences between our work and MapZero [20].

1) MCTS: MapZero [20] can lose efficiency as the number of connections in the architecture decreases because it may spend simulations during MCTS on multiple actions that lead to poor decisions. For example, generally, if two nodes in a DFG are neighbors, good placements arrange these nodes on PEs with direct connections in the CGRA if these PEs are available. Thus, in this case, simulating placements where these nodes are placed far apart and using other PEs for routing results in poor simulations. This issue is even worse for less connected architectures, as the number of directly connected PEs decreases. Additionally, MapZero allows MCTS simulations to continue on a node even when facing an invalid routing, which can result in a simulation following a path that will not produce a valid mapping. MCTS is crucial for improving mapping results and can be better leveraged by pruning poor decisions during the search. In contrast, our compiler does not deal with these issues, since the actions modeled by YOTO contain only good PEs for decision-making. Moreover, when SmartMap reaches an invalid routing in a state, it marks this state as a terminal, and no further actions will be allowed.

2) Neural Network: Regarding the neural network, since the graph embedding generation and value network layers proposed in MapZero [20] are well designed, we incorporate these layers into our neural network. However, in the policy network, we use an state-action representation layer to operate directly on pairs of state-actions instead of a fully connected layer with a predefined output equal to the CGRA size.

3) Techniques: In summary, our work employs Proximal Policy Optimization (PPO) during pre-training and a different equation for the MCTS selection step.

B. Other ML-based CGRA Mappings

Recent works have integrated GNNs to learn and generate graph and node embeddings in a latent space [20], [22]–[24], which an agent uses during RL. In [22], the authors use GCN [32] and RL to learn how to choose a PE for a DFG node to achieve a valid mapping. E2EMap [24] improves these results by adopting reverse mapping, i.e., the agent learns to choose a DFG node for a PE instead of a PE for a DFG node. To alleviate the limitations of local representations in GNNs, [23] integrates an attention module to obtain global representations of the graphs. TransMap [25] includes an attention module, using transformers [33] with Laplacian embeddings as position encoding along with a type embedding layer, similar to that used in BERT [34]. However, all these works exhibit intermediate portability, meaning they do not provide a general, single model capable of acquiring and utilizing prior knowledge across different DFGs and CGRAs.

In contrast, we design our neural network to address these limitations. Although it is possible to use all these methods directly for mapping, a pre-trained model allows for faster mapping. Lastly, MapZero [20] and TransMap [25] only use masks to invalidate the selection of already chosen PEs based on a topological order for placement. In other words, these methods do not use an efficient search exploration, which could make the approach more scalable and efficient.

IV. Methodology

A. CGRA Mapping Problem Formulation

In this work, the mapping provided by a compiler must achieve an exact solution: 1) Placement of all nodes of the DFG; 2) Routing of all edges of the DFG using the network of the CGRA, ensuring that this routing does not pass through PEs that are already performing an operation or serving as routing; 3) Schedule the execution of all nodes so that all information that needs to reach a PE arrives simultaneously.

B. Placement, Routing, and Scheduling

The first two steps are coupled, and the last step occurs at the final stage of mapping. The agent places a node and routes it based on its already placed parent and child nodes. When reaching a terminal state, if placement and routing are valid, scheduling is carried out. Regarding the techniques, placement is performed by the MCTS, and routing is done by the A* algorithm using Manhattan distance as the heuristic function. During scheduling, time 0 is assigned to a PE containing a node at level 0 with the longest path. The scheduling of remaining nodes continues from this point, considering wire usage and node timing.

C. Graph Features Definitions

SmartMap encodes the node and graph information using the following features:

- **DFG node features**: 1) node id; 2) scheduling order obtained by zigzag traversal; 3) number of DFG nodes mapped in the same modulo time slice; 4) presence of a self-loop; 5) scheduled modulo time slice; 6-7) in-degree and out-degree; 8) opcode; and 9) id of the assigned PE.
- **CGRA node features**: 1) node id; 2-4) capacity to perform arithmetic, logical, and memory access operations; 5-6) in-degree and out-degree; and 7) id of the mapped DFG node.

D. Reinforcement Learning Problem Formulation

Reinforcement learning formulates a problem as a Markov Decision Process (MDP). MDPs comprise states, actions, transition and reward functions, initial and terminal states, and a discount factor. The agent uses a state-value function to calculate the expected return by starting from a state s and following the policy π given by Equation 1 where the return is given by Equation 2 and r_{t+k+1} is the reward received after taking action at timestep $t+k$ following the policy π, and γ is the discount factor, ranging between 0 and 1.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s] \tag{1}$$

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

States: include the DFG and CGRA graphs and their features, the features of the node to place in the current state, and the features of the legal PEs, i.e., the PEs that the agent can select in the current state; **Actions:** based on the list returned by the zigzag traversal (see Section II-B), are the unused neighboring PEs of the PE where node a_i is placed, or all PEs if the node to be placed is a_1; **Rewards:** after the placement is the negative number of wires used for routing between the placed node and its already placed parent and child nodes. If the next state is terminal, the reward is summed with 0 if the mapping is valid; otherwise, it is summed with -100 for each unrouted DFG edge, plus an additional -100 (the same large penalty used in MapZero experiments). In the end, the rewards are normalized by 100; **Initial States:** the initial DFG and CGRA, with the first placement node and all PEs considered legal actions. **Terminal States:** are those where all nodes have been placed, no more PEs are available for placement, an invalid routing is encountered, or finding a neighboring PE for node placement is impossible.

E. Monte-Carlo Tree Search

The MCTS [35] is used to simulate actions and identify the one with the highest chance of producing the best future outcome. To achieve this, it contains nodes that store the state, reward, probability, visit count, and expected return. Each node points to its children, indicating that an action taken in the parent's state leads to the child's state.

The MCTS performs n simulations, which consist of four sequential steps: **selecting** a node recursively according to the same equation and constant values used in MuZero for node selection (see Appendix B, Equation 2 in [36]); **expanding** it by adding child nodes based on the legal actions; **simulating** the expected return at the expanded node's state using the neural network; and **backpropagating** (updating) the node values through the traversed path. After n simulations, an action is selected based on the most visited child node of the root.

F. SmartMap Overview

Fig. 3 presents an overview of SmartMap, using a high-level input example of a DFG with 4 vertices and 4 edges and a 2x2 mesh architecture. Since SmartMap employs YOTO to illustrate the states, we assume that the zigzag traversal returned an edge list $L = \{(3,1),(1,0),(0,2)\}$, and the action selected from the initial state was node 0 of the CGRA. We discuss the layers in the following sections.

1) Neural Network: Graph Embedding Generation represents the state information in a latent space using GAT [37] and neural networks, and the policy and value networks produces the probabilities of taking each legal action and the expected value (or return) in the current state, respectively. The policy network receives the valid actions, which are obtained according to the YOTO in the current state, and uses a linear layer (FC) to embed them in the latent space. For each action, a state-action representation is formed to calculate the probability of taking it, which is done through a MultiLayer Percepetron (MLP) with Leaky ReLU, followed by a final linear layer, Leaky ReLU, and softmax function sequentially. This new approach enables the agent to leverage prior knowledge to map DFGs across different architecture sizes.

2) Pre-Training: The neural network is pre-trained on a set of DFGs, where actions are taken proportionally to the probabilities the policy network returns. This process iteratively repeats until the mappings finish. In the end, the expected return for each state is calculated based on the rewards and values obtained during the mapping process to train the value network. In contrast, the policy network training uses the PPO [38]. Specifically, the weights of the value and policy networks (V_θ and π_θ) are updated via gradient descent according to equations 3 and 4, respectively, along with a Kullback-Leibler divergence penalty and an entropy penalty. Due to space constraints, refer to [38] for more details about objective 4 and the penalties. The advantage ($\hat{A}$) and target return (V_{targ}) were calculated using Generalized Advantage Estimation (GAE) [39].

$$L_v = \max\left((V_\theta - V_{\text{targ}})^2, \left(\text{clip}(V_\theta, V_{\theta_{\text{old}}} - \epsilon, V_{\theta_{\text{old}}} + \epsilon) - V_{\text{targ}}\right)^2\right) \tag{3}$$

$$L_p = \max\left(-\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}, -\text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1-\epsilon, 1+\epsilon\right)\hat{A}\right) \tag{4}$$

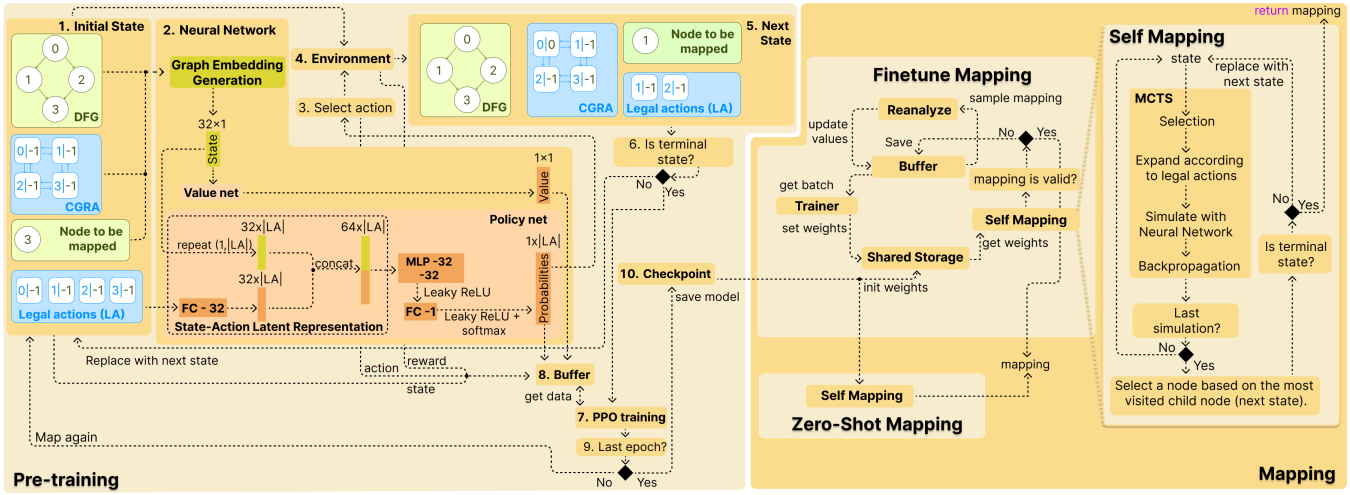

Fig. 3. SmartMap Overview.

3) Mapping: SmartMap can map a DFG in two modes: **zero-shot** and **finetune**. The zero-shot mapping relies solely on MCTS and the neural network. Meanwhile, finetune mapping allows the agent to generate mappings and train its network on these new examples to improve future performance. Regarding the components, there is a replay buffer to store mappings; a shared storage holds the neural network weights and other mapping-related information; a reanalysis component updates the predicted values in each state using the new network weights instead of discarding old examples; and a trainer trains the neural network on a batch of mappings, where the objective used in this stage is the same as in MapZero (Algorithm 1, Line 21 in [20]). It's important to note that each component in the finetune mapping layer runs on different threads, and it is possible to launch multiple threads to generate mappings. If there is a valid mapping, it is returned. Additionally, SmartMap uses backtracking to allow the agent to take another action if an invalid action is chosen. Lastly, the implementation was adapted from a MuZero's reimplementation [40].

G. Experiments

To evaluate our approach, we selected homogeneous architectures capable of executing arithmetic, logical, and memory access operations, ranging from more connected (expensive) to less connected (low-cost), with dimensions varying from 4×4 to 8×8. The architectures include the following interconnect styles: one-hop plus diagonal plus toroidal (OH+Tor+Diag), one-hop, and mesh (see Fig. 1). As for the benchmarks, they were sourced from the E2EMap code repository [24], provided by the authors of [20], and extracted from CGRA-ME [41]. We generated synthetic DFGs to compose the training and evaluation datasets. The evaluation dataset also includes DFGs extracted from real CGRA benchmarks. Moreover, since YOTO assumes that neighboring nodes should be placed close together on the CGRA, all unbalanced DFGs were balanced. Lastly, we compared the results with MapZero [20], which is the main directly related work in this research and was capable of outperforming CGRA-ME (SA and ILP) [41] and LISA [21].

1) Setup: We documented all hyperparameters and configurations in the repository for reproducibility.

V. Results

Table I shows the rates of successful mappings obtained by each method in the experiments, and Table II presents these results for each DFG in the experiments on the 4×4, where V and E represent the number of vertices and edges of the DFG, respectively, and Successful Mapping column indicates whether the mapping was successful or not, which values outside and inside parentheses are related to zero-shot and finetune mapping, respectively; and from left to right, they represent the results for OH+Tor+Diag (All), one-hop (1-Hop), and mesh (Mesh) CGRAs.

TABLE I
Rate of valid mappings obtained in each experiment.

Size	Mode	Connection style	Successful Mapping Rate (%)	
			MapZero	SmartMap
4x4	Zero-shot	Oh+Tor+Diag	50	**90.91**
		One-hop	22.73	**77.27**
		Mesh	4.55	**59.09**
4x4	Finetune	Oh+Tor+Diag	**100**	**100**
		One-hop	68.18	**100**
		Mesh	27.27	**95.45**
8x8	Zero-shot	Oh+Tor+Diag	7.94	**52.38**
		One-hop	4.76	**42.86**
		Mesh	1.59	**30.16**

A. Mappings

The zero-shot results in Table I demonstrate the capability of each method to map a DFG directly, i.e., using only MCTS. SmartMap outperformed MapZero in all experiments, mapping approximately 40.91 $(90.91 - 50)\%$, 54.54 $(77.27 - 22.73)\%$, and 54.54 $(59.09 - 4.55)\%$ more DFGs, respectively, for 4x4

TABLE II
RESULTS FROM THE 4×4 EXPERIMENTS FOR EACH DFG.

DFG	Method	(V, E)	Successful Mapping		
			All	1-Hop	Mesh
$V\_5\_E\_4$	MZero	(5, 4)	✓ (✓)	✓ (✓)	✓ (✓)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$cholesky$	MZero	(6, 6)	✓ (✓)	**X** (✓)	**X** (✓)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$V\_6\_E\_5$	MZero	(6, 6)	✓ (✓)	✓ (✓)	**X** (✓)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$balanced-sum$	MZero	(7, 6)	✓ (✓)	✓ (✓)	**X** (✓)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$V\_7\_E\_7$	MZero	(7, 7)	✓ (✓)	✓ (✓)	**X** (✓)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$V\_8\_E\_9$	MZero	(8, 9)	✓ (✓)	**X** (✓)	**X** (**X**)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$V\_9\_E\_10$	MZero	(9, 10)	✓ (✓)	**X** (✓)	**X** (**X**)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$V\_10\_E\_9$	MZero	(10, 9)	✓ (✓)	✓ (✓)	**X** (✓)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$V\_11\_E\_11$	MZero	(11, 11)	✓ (✓)	**X** (✓)	**X** (**X**)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$balanced-pre$	MZero	(11, 11)	✓ (✓)	**X** (✓)	**X** (**X**)
	SMap		✓ (✓)	**X** (✓)	✓ (✓)
$V\_12\_E\_12$	MZero	(12, 12)	✓ (✓)	**X** (✓)	**X** (**X**)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$cholesky\_unroll$	MZero	(12, 14)	**X** (✓)	**X** (✓)	**X** (**X**)
	SMap		✓ (✓)	**X** (✓)	**X** (**X**)
$atax$	MZero	(13, 14)	**X** (✓)	**X** (✓)	**X** (**X**)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$V\_13\_E\_14$	MZero	(13, 14)	**X** (✓)	**X** (**X**)	**X** (**X**)
	SMap		✓ (✓)	✓ (✓)	**X** (✓)
$V\_14\_E\_14$	MZero	(14, 14)	**X** (✓)	**X** (✓)	**X** (**X**)
	SMap		✓ (✓)	✓ (✓)	**X** (✓)
mvt	MZero	(14, 15)	**X** (✓)	**X** (**X**)	**X** (**X**)
	SMap		**X** (✓)	**X** (✓)	**X** (✓)
$syrk$	MZero	(14, 15)	**X** (✓)	**X** (✓)	**X** (**X**)
	SMap		✓ (✓)	✓ (✓)	**X** (✓)
$V\_15\_E\_15$	MZero	(15, 15)	**X** (✓)	**X** (**X**)	**X** (**X**)
	SMap		**X** (✓)	✓ (✓)	**X** (✓)
$gemm$	MZero	(15, 16)	**X** (✓)	**X** (**X**)	**X** (**X**)
	SMap		✓ (✓)	✓ (✓)	**X** (✓)
$symm$	MZero	(16, 17)	**X** (✓)	**X** (**X**)	**X** (**X**)
	SMap		✓ (✓)	✓ (✓)	✓ (✓)
$doitgen$	MZero	(16, 17)	**X** (✓)	**X** (**X**)	**X** (**X**)
	SMap		✓ (✓)	**X** (✓)	**X** (✓)
$V\_16\_E\_19$	MZero	(16, 19)	**X** (✓)	**X** (**X**)	**X** (**X**)
	SMap		✓ (✓)	**X** (✓)	**X** (✓)

CGRAs with OH+Tor+Diag, one-hop, and mesh interconnections. For 8x8 CGRAs, this difference was approximately 44.44 $(52.38 - 7.94)\%$, 38.10 $(42.86 - 4.76)\%$, and 28.57 $(30.16 - 1.59)\%$, respectively. These results demonstrate that our approach can more efficiently explore the MCTS by identifying and eliminating poor simulations.

Regarding finetune results, note how the valid mapping rate increases when using the complete framework. Although both methods were able to map all DFGs to the most connected 4×4 CGRA in finetune mode, MapZero tends to perform worse as the number of connections in the architecture decreases. At the same time, SmartMap maintains its mapping capability even for smaller connected architectures, failing on only one DFG in the mesh architecture, reinforcing our approach's scalability. During finetune mapping, the difference in valid mapping rates obtained was approximately 0 $(100 - 100)\%$, 31.82 $(100 - 68.18)\%$, and 68.18 $(95.45 - 27.27)\%$, respectively.

Finally, Table II shows that these experiments are mainly composed of challenging DFGs, i.e., their size is close to that of the architecture. As the DFG size increases and the architecture connections decrease, MapZero tends to fail. SmartMap maintains its mapping capability, only failing for the *chollesky_unroll* DFG on the mesh architecture. Since E2EMap [24] and the methods presented in their work were able to map this DFG with a temporal-spatial approach on a mesh architecture, which is at least twice slower, and our approach successfully mapped larger DFGs, likely, this DFG does not have an exact solution for spatial mapping.

B. Compilation Time and Resource Utilization

We filtered the results to include only the DFGs successfully mapped by both methods on architectures with the same dimensions and interconnection style in both mapping modes. Subsequently, we calculated the average compilation time and number of PEs used, resulting in an average of 80.59 seconds and 15.11 PEs for MapZero and 4.79 seconds and 9.18 PEs for SmartMap, leading to an approximate 16.82x speed-up in compilation time. Our approach is faster and utilizes fewer PEs as intermediate routing, on average.

C. Final Considerations

In all experiments, our approach successfully mapped 194 DFGs, while MapZero mapped only 69, resulting in a mapping capacity that is 2.81x higher. Moreover, experiments on larger architectures and with different methods from the literature are currently in progress, including finetune mapping on 8×8 architectures. Despite this, Table I shows a trend in our approach to achieving better results. Note that one of our contributions is to increase the portability of the compiler to leverage prior knowledge from architectures of different sizes. However, each model was trained separately on each architecture. For MapZero, we applied the same training approach as SmartMap. These separations allow us to isolate and compare the impact of using an efficient placement method on the results. Refer to the repository to see the training and mapping details.

VI. CONCLUSION

This paper presents SmartMap, an RL-MCTS-based framework that introduces a novel approach to increasing the portability of reinforcement learning-based compilers and uses the YOTO placement method to utilize MCTS better, enhancing efficiency and scalability. Our approach has 2.81x more mapping capacity, is 16.82x faster, and uses fewer resources for routing than the state-of-the-art. Moreover, this work demonstrates that combining efficient placement methods with machine learning techniques can create a more powerful compiler than relying solely on machine learning methods.

ACKNOWLEDGMENTS

This study was partially funded by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. Fapemig, CNPq, and Codedepot have also partially funded this work.

REFERENCES

[1] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, "A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–39, 2019.

[2] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," *IEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 5, pp. 255–261, 2003.

[3] T. Kojima, N. A. V. Doan, and H. Amano, "Genmap: A genetic algorithmic approach for optimizing spatial mapping of coarse-grained reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 11, pp. 2383–2396, 2020.

[4] S. A. Chin and J. H. Anderson, "An architecture-agnostic integer linear programming approach to CGRA mapping," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[5] M. J. P. Walker and J. H. Anderson, "Generic connectivity-based cgra mapping via integer linear programming," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 65–73.

[6] R. Ferreira, L. Rocha, A. Santos, J. Nacif, S. Wong, and L. Carro, "A runtime graph-based polynomial placement and routing algorithm for virtual fpgas," in *2013 23rd International Conference on Field programmable Logic and Applications*, 2013, pp. 1–8.

[7] R. Ferreira, L. Rocha, A. G. Santos, J. A. M. Nacif, S. Wong, and L. Carro, "A runtime fpga placement and routing using low-complexity graph traversal," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 2, Mar. 2015. [Online]. Available: https://doi.org/10.1145/2660775

[8] M. Canesche, M. Menezes, W. Carvalho, F. S. Torres, P. Jamieson, J. A. Nacif, and R. Ferreira, "Traversal: A fast and adaptive graph-based placement and routing for cgras," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 8, pp. 1600–1612, 2020.

[9] M. Canesche, W. Carvalho, L. Reis, M. Oliveira, S. Magalhães, P. Jamieson, J. M. Nacif, and R. Ferreira, "You only traverse twice: A yott placement, routing, and timing approach for cgras," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–25, 2021.

[10] R. Ferreira, A. Garcia, T. Teixeira, and J. M. P. Cardoso, "A polynomial placement algorithm for data driven coarse-grained reconfigurable architectures," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI'07)*, 2007, pp. 61–66.

[11] Y.-T. Lai, H.-Y. Lai, and C.-N. Yeh, "Placement for the reconfigurable datapath architecture," in *2005 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2005, pp. 1875–1878.

[12] L. Chen and T. Mitra, "Graph minor approach for application mapping on cgras," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 3, pp. 1–25, 2014.

[13] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "Epimap: Using epimorphism to map applications on cgras," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1284–1291.

[14] M. Alle *et al.*, "Synthesis of application accelerators on runtime reconfigurable hardware," in *2008 International Conference on Application-Specific Systems, Architectures and Processors*, 2008, pp. 13–18.

[15] T. Peyret, G. Corre, M. Thevenin, K. Martin, and P. Coussy, "Efficient application mapping on cgras based on backward simultaneous scheduling/binding and dynamic graph transformations," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, 2014, pp. 169–172.

[16] S. Friedman, A. Carroll, B. V. Essen, B. Ylvisaker, C. Ebeling, and S. Hauck, "SPR: An architecture-adaptive CGRA mapping tool," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2009, pp. 191–200.

[17] X. Man, J. Zhu, G. Song, S. Yin, S. Wei, and L. Liu, "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*, 2022, pp. 259–273.

[18] D. Wijerathne, Z. Li, T. K. Bandara, and T. Mitra, "PANORAMA: Divide-and-conquer approach for mapping complex loop kernels on CGRA," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 127–132.

[19] M. Balasubramanian and A. Shrivastava, "Pathseeker: a fast mapping algorithm for cgras," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 268–273.

[20] X. Kong, Y. Huang, J. Zhu, X. Man, Y. Liu, C. Feng, P. Gou, M. Tang, S. Wei, and L. Liu, "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*, 2023, pp. 1–14.

[21] Z. Li, D. Wu, D. Wijerathne, and T. Mitra, "Lisa: Graph neural network based portable mapping on spatial accelerators," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 444–459.

[22] Y. Zhuang, Z. Zhang, and D. Liu, "Towards high-quality cgra mapping with graph neural networks and reinforcement learning," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[23] A. X. M. Chang *et al.*, "Reinforcement learning approach for mapping applications to dataflow-based coarse-grained reconfigurable array," *arXiv preprint arXiv:2205.13675*, 2022.

[24] D. Liu, Y. Xia, J. Shang, J. Zhong, P. Ouyang, and S. Yin, "E2emap: End-to-end reinforcement learning for cgra compilation via reverse mapping," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 46–60.

[25] J. Li *et al.*, "Transmap: An efficient cgra mapping framework via transformer and deep reinforcement learning," in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2024, pp. 626–633.

[26] F. Ramos, P. Realino, W. Junior, A. Vieira, R. Ferreira, and J. Nacif, "Smartmap code," GitHub Repository, 2024. [Online]. Available: https://github.com/lesc-ufv/SmartMap

[27] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix," in *Field Programmable Logic and Application: 13th International Conference, FPL 2003, Lisbon*, 2003, pp. 61–70.

[28] L. Liu, Z. Li, C. Yang, C. Deng, S. Yin, and S. Wei, "Hrea: An energy-efficient embedded dynamically reconfigurable fabric for 13-dwarfs processing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 3, pp. 381–385, 2017.

[29] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho, "Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE transactions on computers*, vol. 49, no. 5, pp. 465–481, 2000.

[30] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[31] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[32] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[33] A. Vaswani *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[34] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[35] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*. Springer, 2006, pp. 72–83.

[36] J. Schrittwieser *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[37] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *International Conference on Learning Representations*, 2018.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[39] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[40] A. H. Duvaud and Werner, "Muzero general: Open reimplementation of muzero," GitHub repository, 2019.

[41] A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. Kim, Y. Hara-Azumi, and J. Anderson, "Cgra-me: A unified framework for cgra modelling and exploration," in *IEEE 28th intl conference on application-specific systems, architectures and processors (ASAP)*, 2017, pp. 184–189.