This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Data-Flow Graph Mapping Optimization for CGRA with Deep Reinforcement Learning

Dajiang Liu, Shouyi Yin, *Member, IEEE,* Guojie Luo, *Member, IEEE,* Jiaxing Shang, Leibo Liu, *Member, IEEE,* Shaojun Wei, Yong Feng and Shangbo Zhou

***Abstract*—Coarse-grained reconfigurable architectures (CGRAs) have drawn increasing attention due to their flexibility and energy efficiency. Data flow graphs (DFGs) are often mapped onto CGRAs for acceleration. The problem of DFG mapping is challenging due to the diverse structures from DFGs and constrained hardware from CGRAs. Consequently, it is difficult to find a valid and high quality solution simultaneously. Inspired from the great progress in deep reinforcement learning for AI problems, we consider building methods that learn to map DFGs onto spatially-programmed CGRAs directly from experiences. We propose RLMap, a solution that formulates DFG mapping on CGRA as an agent in reinforcement learning (RL), which unifies placement, routing and PE insertion by interchange actions of the agent. Experimental results show that RLMap performs comparably to state-of-the-art heuristics in mapping quality, adapts to different architecture and converges quickly.**

***Index Terms*—CGRA, DFG, Mapping, Reinforcement Learning**

I. Introduction

Runtime-reconfigurable architectures that enable near ASIC performance without or with a little sacrificing programmability are urgently required for computation-intensive algorithms. Software Defined Hardware (SDH)[1] is a promising technique to achieve this goal, resulting in the ability to run computation-intensive algorithms at very low cost, and consequently, enables pervasive use of high energy-efficient solutions for a wide range of applications. Many works[2][3] achieve high performance on data throughput on FPGA. With higher energy-efficiency than FPGA, Coarse-Grained Reconfigurable Architectures (CGRA)[4][5][6][7] becomes a typical representative of SDH. The Process Element Array (PEA) in CGRA can be dynamically reconfigured to adapt to the change of applications.

According to the execution manner, the mapping algorithms mainly fall into two board categories, spatial mapping and temporal mapping. In spatial mapping[8][9][10][11], the functionality of PEA would not be changed once it is configured. So, spatial mapping needs less configuration context, therefore, it has advantages of more power efficiency. In temporal mapping[12][13], the functionality of PEA changes with time and modulo scheduling is commonly used to reduce initiation interval. However, the performance of temporal mapping would drop off precipitously when the configuration cost could not be hidden in that target CGRA. In this paper, we mainly address the problem of spatial mapping on CGRA.

Data-Flow Graph (DFG), as the kernel of computation-intensive applications, is the key for compiling of CGRA. In the compiling flow, computation-intensive parts are firstly partitioned from applications[14][15]. Then, these computation-intensive parts are transformed into DFGs. Finally, these DFGs are optimized and mapped onto the PEA of CGRA. As DFGs are the kernels of various of application, DFG mapping is a research spot in CGRA[16][13][17]. The DFG mapping is usually performed in three steps[13]: scheduling, placement and routing. Scheduling assigns the operation of DFG to control steps. Placement assigns operations of DFG to PEs and routing assigns edges of DFG to hardware routes among PEs. As PEA often has sparse hardware connections among PEs, placement and routing are often closely coupled for effective mapping. In addition to the direct connections among PEs, routing PE (RPE) insertion is often used to increase the routability of mapping.

In order to improve the mapping efficiency, several mapping methods have been previously proposed: SPKM approach[8], pattern-based approach[9][10] and DFGNet approach[11]. In SPKM approach, column-wise scattering, RPE insertion and row-wise scattering are iteratively adopted to minimize the number of utilized rows under valid mapping. In both column-wise scattering and row-wise scattering steps, integer linear programming (ILP) is used to find the optimal solution. In this approach, node placement and RPE insertion are separated in three iterative steps, and it tends to find suboptimal solution as the solution is searched in a separated optimization space. In pattern-based approach, a cluster-by-cluster mapping is accelerated by placing patterns, and then anytime algorithms are used to find the optimal mapping solution. In the patterns for node clusters, placement and RPE insertion are previously elaborated. Although it achieves much better speedup, the scalability is limited as it highly depends on a pattern database. Also, the placement and RPE insertion in patterns are not flexible. In DFGNet approach, a DFG is firstly preprocessed (including routing node insertion) and then mapped onto PEA in node-by-node manner guided by a convolution neural network trained by supervised learning. This method is rather fast and produce high-quality solutions. However, as samples need to be previously labeled in supervised learning, it becomes

D. Liu, J. Shang, S. Zhou and Y. Feng are with the College of Computer Science, Chongqing University, Chongqing 400044, China. S. Yin, L. Liu, and S. Wei are with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China. G. Luo is with the Center for Energy-Efficient Computing and Applications, Peking University, Beijing 10871, China. (*Corresponding author: Dajiang Liu, e-mail: liudj@cqu.edu.cn*)

This work was supported in part by National Natural Science Foundation of China (No. 61804017), Fundamental Research Funds for the Central Universities (No. 2018CDXYJSJ0026), National Natural Science Foundation of China (No. 61702059), National Key Research and Development Program of China (No. 2017YFB1402400) and Frontier and Application Foundation Research Program of CQ CSTC (No. cstc2017jcyjAX0340)

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

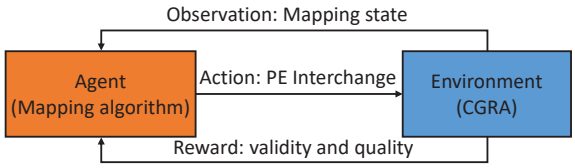

Fig. 1. An overview of RL based DFG mapping on CGRA

difficult for mapping on larger CGRAs.

We take a step back to understand some reasons for why DFG mapping problems on CGRA are challenging:

- The compiling process is complex and often impossible to be modeled accurately. In mapping process, we not only need to consider the validation of placement and routing, but also need to consider the quality improvement. In particular, the problem of valid mapping is an NP-complete problem, and it is hardly to judge whether an intermediate state is good or bad until the last step.
- As the diversity of application and the reconfigurability of hardware, the optimization space of mapping tends to be too large to be enumerated. To address this problem, previous approaches divide mapping into small and separate problems, which in turn loose the opportunity to find a solution closer to the optimum.

In this paper, we attempt to provide a viable alternative to human-generated heuristics for DFG mapping on CGRA using machine learning. Recent success of applying machine learning to other challenging decision-making domains suggests that this idea may not be too far-fetched. Especially, reinforcement learning (RL) has become an active area in machine learning research. RL trains an agent that learns to make better decisions directly from experience interacting with the environment. The agent can learn from scratch by a reward that it receives representing how well it is doing on a task. RL has a long history, but recently it is combined with deep learning, called deep reinforcement learning (DRL). DRL can create artificial agents to achieve human-level performance across many challenging domains such as playing video games[18][19], Computer Go[20][21], etc.

Revisiting the above challenges, we believe RL approaches are very suitable for DFG mappings on CGRA. First, similar decisions are often made in similar graphs or subgraphs, which generates an abundance of training data for RL algorithms. Second, the PEA in CGRA usually keeps regular (e.g., 2D arrays) and small (e.g., 4×4 to 8×8) for the reason of area and power. Therefore, the state space and action space can be well represented using RL. Third, RL can model complex systems and decision-making policies as deep neural networks (DNN) analogous to the models used for game-playing agents. As the DNN has generalization ability, unseen raw can also be well handled toward good results. Finally, it is possible to train for objectives that are hard-to-optimize directly because they lack precise models if there exist reward signals that correlate with the objective.

In this paper we propose DRL based approach to spatially map a DFG onto CGRA more efficiently. As shown in Fig. 1, the approach takes into account information of environment (mapping state) by performing series of experiments (PE interchange action mentioned below) to understand which operations of DFG should be placed on which PE of PEA, which edges of DFG should be routed by which connections of PEA and how to arrange the placement and routing so that the a valid and high-quality mapping can be obtained. Our contributions are summarized as follows:

- We formulate DFG mapping problem as an agent of algorithm in RL, which can learn DFG mapping from scratch.
- We define agent actions as neighbor PE interchanges, combing placement, routing and routing PE insertion in a closely-coupled way. It results in a large space that is prone to find a solution closer to the optimal one.
- We propose a reward signal that properly reflects how well an action is doing for the mapping task. This well designed reward can not only accelerate the training to convergence, but also help the agent to get valid and high-quality mapping.

The remainder of the paper is organized as follows: Section II gives background of CGRA mapping and reinforcement learning. Then, section III states the proposed method and Section IV shows the experimental results. Finally, we conclude in Section V.

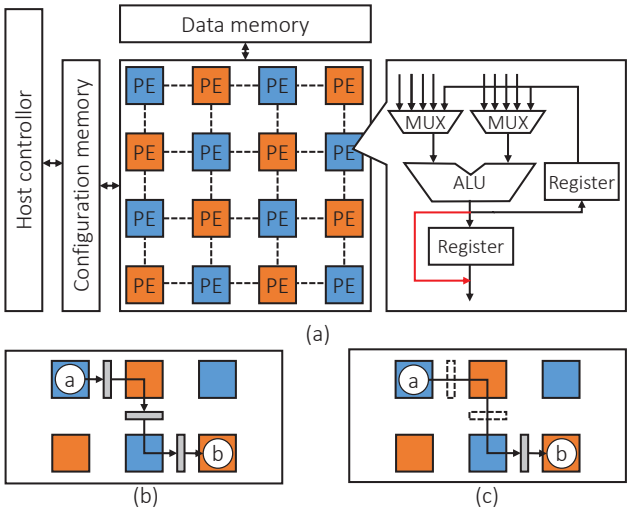

Fig. 2. (a) A general CGRA architecture, (b) A registered routing, (c) An unregistered routing

II. Background and Related Works

In this section, we provide a brief introduction to the required background in CGRA, DFG mapping and DRL. Then, we present related works in recent years.

A. Target Architecture

As shown in Fig. 2(a), a CGRA is typically constituted of a host controller and a 2-D processing element array (PEA), where the host controller is responsible for running the operation system and irregular programs, and the PEA

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

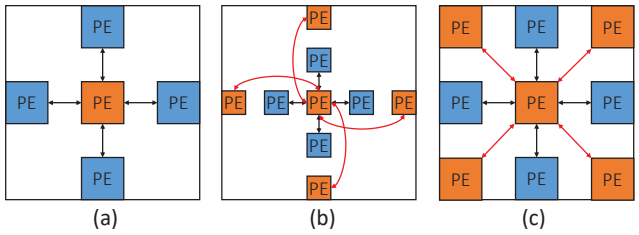

Fig. 3. PE interconnection styles: (a) 4-way routing, (b) 4-way-1-hop routing, (c) 8-way routing

executes computation-intensive workloads. The processing element (PE), usually comprising one or more arithmetic logic units (ALUs), can perform arithmetic operations and communicate with each other using a spatial interconnection. The output register in a PE could be bypassed for routing value not crossing clock edge, as the red line show in Fig. 2(a). Fig. 2(b) and (c) illustrated two cases of using routing PE. If data communicated between PEs must be stored in registers at each hop, operation b must be executed three cycles after operation a. On the other hand, if registers are bypassed during data routing, b can be executed immediately after a. After placement and routing, the output registers of PEs could be trivially selected for bypass according to dependence latency.

The connections between PEs are usually sparse[22] for power and area consideration, and therefore only neighbor or near neighbor PEs are connected. Fig. 3 depicts three commonly used interconnection styles, including 4-way interconnection (i.e., mesh routing), 4-way-1-hop interconnection (i.e., mesh plus routing) and 8-way interconnection.

B. DFG Mapping on CGRA

Given a DFG and a PEA, the DFG mapping on CGRA problem is to find a valid and optimal mapping from the DFG $D = (V, E)$ to the PEA graph $C = (P, L)$, where V, E, P and L represent the set of operations in DFG, the dependencies among operations, the set of PEs and the interconnections among PEs, respectively. The application mapping is a function $\phi : D \rightarrow C$, which in turn implies two functions, $\phi_V : V \rightarrow P$ and $\phi_E : E \rightarrow S$. The natation S is the path set, i.e., the combination of interconnections (L) among PEs and the size of S is $2^{|L|}$.

Definition 2.1: (Routing PE) Given p_i, q_i be the PEs in C, $l_i = (p_i, q_i)$ be the connection between p_i and q_i, and $(l_0, l_1, \cdots, l_n) \in S$ $(n \leq |L|)$ be the connection path of $e = (u, v)$. The routing PE set of e is defined as $R_e = \{q | \forall l = (p, q), q \neq \phi_V(v) \wedge q \neq \phi_V(u)\}$. Then, the routing PE set of the whole mapping is defined as $R = \bigcup_{e \in E} R_e$.

Definition 2.2: (Valid Mapping) Given a DFG $D = (V, E)$ and PEA $C = (P, L)$, a valid mapping is defined as flows:

1) $\forall u \in V$, $\exists p \in P$ such that $\phi_V(u) = p$.
2) $\forall u, v \in V$, if $u \neq v$, then $\exists p, q \in P \wedge p \neq q$ such that $\phi_V(u) = q$, $\phi_V(v) = q$.
3) $\forall u \in V, \phi(v) \notin R$.
4) $\forall e = (u, v)$, if $u \neq v$ then $\exists \phi_E(e) = (l_0, l_1, \cdots, l_n) \in S$ such that $\phi_V(u) = p_0$, $\phi_V(v) = q_n$ and $q_i = p_{i+1}$.
5) $\forall e_1, e_2 \in E$, if $e_1 \neq e_2$ then $R_{e_1} \bigcap R_{e_2} = \emptyset$.

In definition 2.2, the conditions 1), 2) and 3) state that each operation in D has a target PE and each PE can hold only one operation node or routing node, i.e., no operation conflict is permitted in the valid mapping. The condition 4) and 5) state that each edge in D has a connection path in C and the connection paths for different edges should not share routing PEs.

C. Reinforcement Learning

In reinforcement learning, an agent improves its behavior by interacting with an environment, as shown in Fig. 1. At each time step t, the agent observes one state s_t, and is required to choose an action a_t on the state. Following the action, the state of the environment transfers to a new state s_{t+1} and the agent receives a reward r_t. The state transition process is assumed to be a Markov Decision Process (MDP), i.e. the state transition probabilities and rewards depend only on the state of the environment s_t and the action taken by the agent a_t.

The goal of the agent is to select actions in a way that maximizing future rewards by interacting with the environment. The future reward is assumed to be discounted by a factor of γ per time-step, and the future discounted return at time t is defined as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where T is the time-step at which the task terminates. The optimal action-value function $Q^*(s, a)$ is also defined as the maximum expected return achievable by following any strategy, after seeing some sequence s and then taking some action a, $Q^*(s, a) = \max_\pi E[R_t | s_t = s, a_t = a, \pi]$, where π is a policy mapping states to actions. The policy can be discrete values over actions or probability distributions over actions.

The solve of policy π falls into two categories: value-based method and policy-based method. The value-based method first calculates the action-value function, then deduces the policy using simple strategies like ϵ-greedy. This method needs less training data and is suitable for task of small action space. The policy-based method directly parameterizes the policy and it is fit for task of continuous or large action space. In this work, we formulate action as neighbor PE interchange (see sections below) and the action space is rather small. Therefore, value-based method is adopted in this work.

The optimal action-value function obeys an important identity known as the Bellman equation basing on the following intuition: if the optimal value $Q^*(s', a')$ of the state s' at the next time-step was known for all possible actions a', then the optimal strategy is to select the action a' maximising the expected value of $r + \gamma Q^*(s', a')$.

$$Q^*(s, a) = E_{s' \sim \varepsilon} [r + \gamma \max Q^*(s', a') | s, a] \qquad (1)$$

where ε indicates the environment in reinforcement learning. Specifically, as shown in Fig. 1, ε means the target processing element array (PEA) in a CGRA. The basic idea behind value-based reinforcement learning algorithm is to estimate action-value function by Bellman equation update, $Q_{i+1}(s, a) = E[r + \max_{a'} Q_i(s', a') | s, a]$. As iteration i increases to infinity, the value of iteration algorithm converge

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

to the optimal action-value function. In task of large state-action space, the basic approach becomes impractical. Instead, it is common to use a function approximator to estimate the action-value function, $Q(s,a;\theta) \approx Q^*(s,a)$. Recently, Q-network[18], a deep neural network function approximator with weights θ, shows great success in video games and computer Go. The deep Q-network (DQN) can be trained by minimizing a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i,

$$L_i(\theta_i) = E_{s,a\sim\rho}\left[(y_i - Q(s,a;\theta_i))\right] \tag{2}$$

where $y_i = E_{s'\sim\varepsilon}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}|s,a\right]$ is the target for iteration i and $\rho(s,a)$ is a probability distribution over sequences s and actions a. The parameters from the previous iteration θ_{i-1} are held fixed when optimizing the loss function $L_i(\theta_i)$. Instead of preparing fixed targets in supervised learning, the targets here depend on the network weights. Therefore, there are two neural network are involved in DQN, target network and evaluation network. The target network is responsible for providing targets and the other one is responsible for improving the weights at every iteration. Periodically, the weights in evaluation network are cloned to the target network. Base on the loss function, the weights can be updated by performing gradient descent using gradient as follows:

$$\nabla_{\theta_i} L_i(\theta_i) = E_{s,a\sim\rho;s'\sim\varepsilon}\left[(y_i - Q(s,a;\theta_i))\,\nabla_{\theta_i} Q(s,a;\theta_i)\right] \tag{3}$$

It is difficult to compute the full expectations in the above gradient. Expediently, it is often to optimize the loss function by stochastic gradient descent. If the weights are updated after every time-step, the expectations are replaced by single samples from the behaviour distribution ρ and the environment ε respectively.

D. Related Works

In recent years, the problem of DFG mapping has been well discussed in the literature. Most existing approaches falls into one or several categories, including integer linear programming (ILP) approach, simulated annealing approach, partitioning approach, analytic approach and machine learning approach.

ILP approach usually formulates the mapping problem into clear representation in mathematics and it is possible to find the optimal solution. Due to high time complexity, ILP approach[23][24] is suitable for problem of moderate size. Therefore, it is frequently used to solve a sub-problem in mapping algorithms.

Simulated annealing (SA) approach is widely used to solve complex problems due to its generality and accessibility. It is not only applied to CGRAs[25][26], but also widely applied to FPGAs[27][28][29]. SA-based approach can achieve good mapping solutions but are often inefficient for large problems.

Partitioning approach has strong scalability and is efficient to even large problems, such as SPKM[8], NTUPlace[30], KernelPartion[31], etc. The basic idea of partitioning is divide and conquer. These algorithms perform very fast. However, the mapping result may be suboptimal due to the separated searching space.

Analytical approach can achieve good quality results[32][33] for its efficiency and scalability. Analytical approaches firstly generate an optimized mapping allowing placement overlapping. Then, overlapping elements are separated iteratively. Consequently, it can be very fast even for large problems. However, the solution quality can suffer due to the local nature of final adjustments.

Machine learning approaches are fast and efficient as they exploit previous experiences, including pattern-based approach and supervised learning based approach. Pattern-based method [10] uses patterns to index into a dictionary that provides suggestions for good spatial arrangements of nodes onto relatively homogeneous hardware. The scalability is poor and has no generality as the pattern dictionary is static. DFGNet[11] method trains a neural network to provide suggestions for good mapping by supervised learning. It is only suitable for very small CGRAs as data labeling for supervised learning becomes impossible for large CGRAs.

As SPKM combines the advantages of several approaches above and focuses on spatial mapping on CGRA, we use SPKM as the basis for comparison.

III. Method

In this section, we present our method for DFG mapping on CGRA using RL. We formulate the problem in subsection III-A and describe how to represent it as an RL task in subsection III-B. Then, we introduce how to train the learning algorithm and how to use the trained agent in real DFG mapping in subsection III-C.

A. Objective and Problem

The objective of spatial mapping should balance power, area, and performance. The exact value is difficult and time consuming to compute. Similarly, we define the same cost function as that in [10] for three reasons: 1) we address the same problem of DFG spatial mapping, 2) routing PEs are considered in both work, and 3) specific factors are obtained from power simulations run on a ASIC process using the third-part simulation tools. The specifical form of objective is as follows:

$$O(s) = \sum_{i=1}^{n_n}(\sum_{j=1}^{n_p} I_{i,j}) + (2000\times N_{op}) + (800\times N_{pg}) + (400\times N_{nop}) \tag{4}$$

where n_n, n_p, N_{op}, N_{pg} and N_{nop} indicate the number of nodes, number of parents of a node, the number of ALUs performing an operation, the number of PEs used as routing PE and the number of empty PEs. Parameter N_{nop} is determined by counting the number of empty PEs in the smallest rectangular area that contains the PEs that have already been assigned operations. A larger area solution may have more empty ALUs and more ALUs used as passgates. If all else is equal, this solution will thus incur a greater cost than a more compact

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

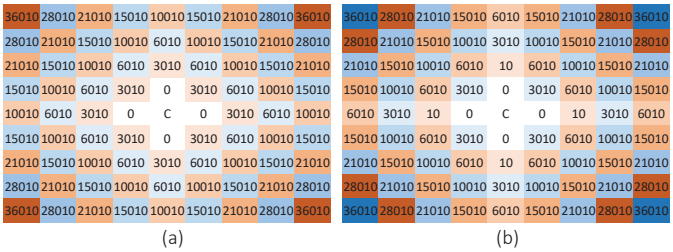

Fig. 4. Connection cost for (a) mesh routing PEA and (b) mesh-plus routing PEA

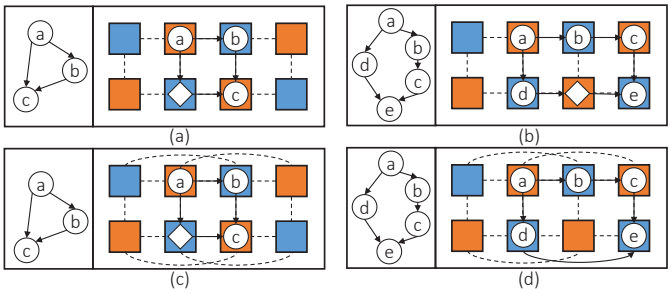

Fig. 5. Necessary routing PEs needed: (a) a 3-node circle needs 1 RPE on mesh PEA, (b) a 5-node circle needs 1 RPE on mesh PEA, (c) a 3-node circle needs 1 RPE on mesh-plus PEA and (d) a 5-node circle needs 0 RPE on mesh-plus PEA.

solution. The cost factors, 2000, 800 and 400, used in Equation (4) were obtained based on power simulations run on a 90-nm ASIC process from Synopsys using the Synopsys PrimeTime-PX tool in work [10]. These cost factors are relevant in a relative sense. If better cost factors or cost factors in different ASIC processes are available to replace Equation (4), it can be substituted trivially.

The interconnection cost of edges in DFGs falls into two categories: edges $I_{i,j}^{val}$ supported by the architecture and edges $I_{i,j}^{inv}$ not supported by the architecture.

For edges supported by the architecture, the interconnection cost can be obtained by Synopsys PrimeTime-PX tools in the target process. For example, as shown in Fig. 4, the interconnection costs between neighbor PEs for mesh and mesh-plus routing architecture are almost 0s ("0"s in Fig. 4(a) and (b)) for both architectures and the interconnection cost between near neighbor PEs is 10 ("10"s in Fig. 4(b)) for mesh-plus routing architecture in 90 nm ASIC process.

For edges that are not supported by the architecture, interconnect cost is a heuristic cost and there are two basic rules to determine their interconnection cost. 1) The cost value of the unsupported interconnection should be larger than any other cost factors, such as the cost factors for operation PE, routing PE and empty PE, in Equation (4). The reason is that legal routing is a precondition for DFG mapping on CGRA and it should have the higher weight as compared to the mapping quality. 2) The cost for long distance edge should have higher cost than that for short distance edge, where the distance indicates the Manhattan distance between the PEs of the source operation and target operations. As the work in [10], the heuristic interconnection cost not supported by the architecture is as follows:

$$I_{i,j}^{inv} = 500 \times {d_{i,j}}^2 + 500 \times d_{i,j} + 10 \tag{5}$$

where $d_{i,j}$ is the manhattan distance between the mapped PE of operation i and operation j. As shown in Fig. 4(a) and (b), the diagonal interconnection has manhattan distance of 2, and the interconnection cost is 3010, which is greater than the cost factor (2000) of operation PE. The longest interconnection, shown in Fig. 4, has manhattan distance of 8, and the interconnection cost is 36010, which is greater than the cost factor of other shorter interconnections.

In order to determine the terminal condition for RL, we also attempt to find the lower bound of objective function (O_{lb}) in Equation (4). To get the lower bound of objective function, we assume that only necessary routing PEs are inserted, the minimal rectangular mapping area is available and all edges are validly routed. The number of necessary routing PEs (N_{pg}^*) is highly related to both the number of circles with odd number of operations in the DFG and the routing style of the target architecture. For mesh routing PEA, at least one routing PE is needed for a circle with odd number of operations. As shown in Fig. 5(a) and (b), the 3-operation circle and the 5-operation circle are both inserted with a routing PE to get a valid mapping. For mesh-plus routing PEA, at least one routing PE is needed for 3-operation circle and it is not necessary to insert a PE for circles with the number operation greater than 3. As shown in Fig. 5(c), a routing PE is inserted for the 3-operation circle to get a valid mapping. In Fig. 5(d), no routing PE is inserted for the 5-operation circle as there is a 1-hop interconnection to deliver the long edge from operation d to e.

When N_{pg}^* necessary routing PEs are inserted, the total number of nodes occupying PEs can be obtained, $N_{node} = N_{op} + N_{pg}^*$. Then, we can get the area of the minimal rectangular $S_{min}(N_{node})$ that can cover the N_{node} nodes on the target PEA without consideration the specific placements of these nodes. Next, the number of empty PE can be easily calculated by $N_{nop} = S_{min}(N_{node}) - N_{node}$. Finally, the lower bound of the objective function can be presented as follows:

$$O_{lb} = 2000 \times N_{op} + 800 \times N_{pg}^* + 400 \times N_{nop} \tag{6}$$

In Equation (6), the item $2000 \times N_{op}$ is a constant for different solutions as the number of operations in a DFG is invariant. Compared to Equation (4), the item of interconnection cost is neglected here as we assume that all edges are validly mapped. We also note that, in Equation (6), the items $800 \times N_{pg}^*$ and $400 \times N_{nop}$ are theoretical value. If we directly used it as a condition of terminal mapping state, it may never be satisfied. Therefore, we add a relaxation factor (β) to Equation (6) and it becomes:

$$O_{lb}^* = 2000 \times N_{op} + \beta(800 \times N_{pg}^* + 400 \times N_{nop}) \tag{7}$$

where β is greater than 1 and is determined empirically. In the experiments, we find the training process can converge quickly when β is set to 1.1. With the value of relaxed lower

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

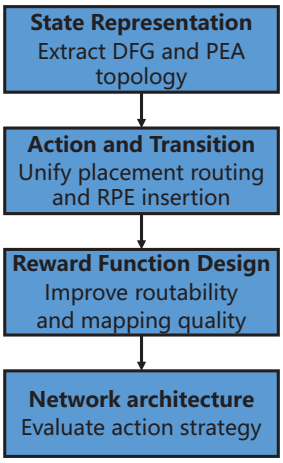

Fig. 6. Formulation steps of our approach

.

bound of objective function, we define the valid mapping states as terminal states once their objective function values get less than O^*_{lb}.

Based on the objective function in Equation (4) and the valid mapping definition in section II, the problem of our approach can be represented as follows:

Problem 1. Given a DFG $D = (V, E)$ and a PEA $P = (P, L)$, find a mapping $\phi : D \to P$ to *minimize* cost O *subject to* ϕ is a valid mapping.

In the problem defined above, it is assumed that the number of operation in D should be less than or equal to the number of PE in P. If the larger DFGs appear, they can be prepartitioned using algorithms like the work in [31].

B. RL Formulation

As show in Fig. 6, the RL formulation of DFG mapping on CGRA involves four important aspects: 1) state representation extracting topologies of DFG and CGRA, 2) action representation unifying placement, routing and RPE insertion in a closely-coupled manner, 3) reward function design to present how well an action is doing on mapping validity and quality, and 4) network architecture to evaluate action strategy. In this subsection, the first 3 steps are elaborated. The network architecture will be studied in the experimental parts.

State representation. We represent the mapping state (i.e., the placement and routing result) as distinct images(see Fig. 7 for illustration). The image in Fig. 7(c) shows a mapping state from DFG D (Fig. 7(a)) to a 4×4 PEA (Fig. 7(b)). In the mapping, shown in Fig 7(b), ont only are the 8 operations in the DFG placed, but also a routing node, a', is inserted and placed on PE 5. We use a $W^2 \times H^2$ image to represent the mapping state, where W and H indicate the width and height of the PEA. For simplify, the unfilled cells in the image shown in 7(c) are all zeros. The image can not only present the operation's placement (ϕ_V), but also present the edge's routing path (ϕ_E), including the routing PE inserted. The image is divided into $W \times H$ blocks and each block is further divided into $W \times H$ cells. Each block indicates the mapping situation

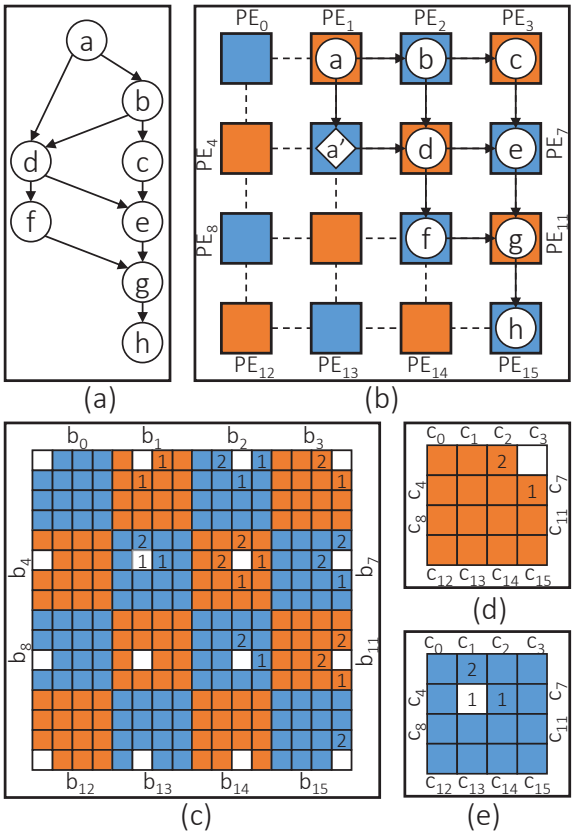

Fig. 7. An example of state representation. (a) A DFG D, (b) a mapping state of D, (c) the state representation of the mapping, (d) the cell representation of block b_3, and (e) the cell representation of block b_5

.

of the corresponding PE as follows: 1) the cell j of block i ($i = j$) indicates the types of PE, 0 for operation PE and 1 for routing PE. 2) the cell j of block i ($i \neq j$) indicate the dependence of operation on PE i, 0 for no edge between operations on PE i and PE j, 1 for an edge from the operation on PE i to the operation on PE j and 2 for an edge from the operation on PE j to the operation on PE i. If all the cells of block i are zeros, the PE i is not mapped with any operation. For example, block b_3 in Fig. 7(d), $c_2 = 2$ indicates that there is a dependence from operation f on PE 2 to operation c on PE 3, and $c_7 = 1$ indicates that there is a dependence from operation c on PE 3 to operation e on PE 7. Fig. 7(e) gives a another example on block b_5, where $c_5 = 1$ indicates PE 5 is a routing PE transferring data from operation a on PE 1 to operation d PE 6.

With the state representation above, the mapping information for a DFG on a specific PEA can be extracted completely. In such state representation, as each PE can only hold one operation, operation conflicts can be naturally avoided satisfying condition 2) in definition 2.2. Also, PE interchange action (see below) further guarantees one PE can hold less or equal to 1 operation (including routing node) if the initial mapping state has no operation conflict. Similar to the images for mapping states, the PEA graph $C = (P, L)$ can also be represented by a such image. Consequently, mapping validation can be easily checked by perform AND operation on state image and

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

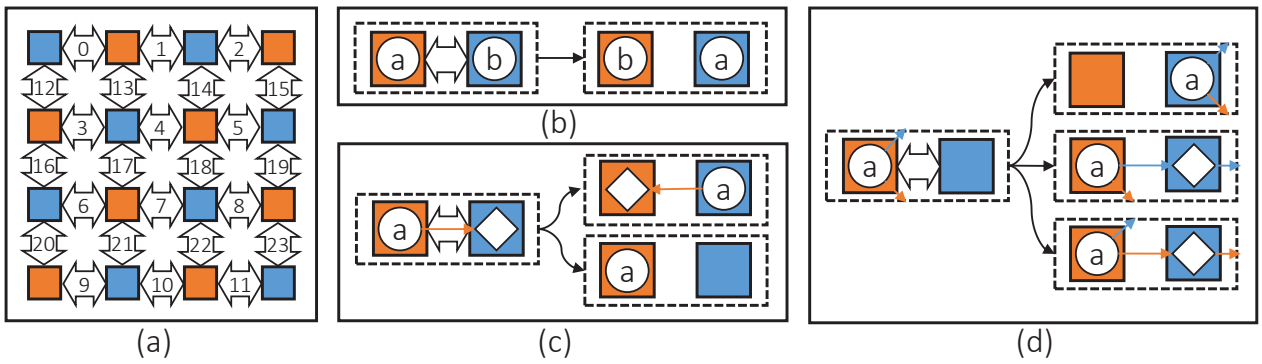

Fig. 8. Action space of state update. (a) Interchanges on neighbor PEs, (b) interchange on 2 operations, (c) interchange on an operation and its routing node, and (d) interchange on an operation and an empty PE

PEA image. Moreover, this state representation can be easily extended to heterogeneous PEA by improve the encoding of type cell (c_j in $b_i, i = j$).

Action representation. At each time step, the mapping algorithm will change the placements of operations or routing nodes. It would require a large action space of size $N \times W \times H$ at each time step, where N is the number of operation in the input DFG. Considering the maximum of N is $W \times H$, the action space is up to $(W \times H)^2$. In addition, operation conflict may occur if more than one operations are mapped on a PE. In order to keep the action space small and avoid operation conflict, we define action as interchange of operations on neighbor PEs (see Fig. 8 for illustration) as follow:

Definition 3.1: Given $u, v \in V$ be the operations in $D = (V, E)$, $p, q \in P$ be the PEs in PEA $C = (P, L)$, $p = \phi_V(u) \wedge q = \phi_V(v)$ and $D(p, q) = 1$, action $\psi(p, q)$ is defined as a new placement pair $(p = \phi_V(v), q = \phi_V(u))$ that interchanges the position of operation u and v.

In definition 3.1, D is the manhattan distance (distance between two points in a grid based on a strictly horizontal and/or vertical path). The condition $D(p, q) = 1$ greatly reduce the action space without loss of completeness. The completeness of action ψ can be represented as follows:

Theorem 3.1: Let $C = (P, L)$ be the PEA, $D = (V, E)$ be the DFG and ψ be the neighbor PE interchange action. $\forall u, v \in D$, if $p = \phi_V(u) \wedge q = \phi_V(E)$, $\exists$ a series of action $(\psi_1, \psi_2, \cdots, \psi_n)$ such that $p = \phi_V(v) \wedge q = \phi_V(u)$.

Proof 3.1: As $\forall p, q \in P$, there is a series of PEs $(p_1, p_2, \cdots, p_n)$ such that $D(p, p_1) = 1, D(p_1, p_2) = 1, \cdots, D(p_n, q) = 1$. We first construct an action sequence $(\psi(p, p1), \psi(p_1, p_2), \cdots, \psi(p_n, q))$ that making $q = \phi_V(u), p_n = \phi_V(v)$. Then we construct an action sequence $(\psi_(p_n, p_{n-1}), \cdots, \psi(p_1, p))$ that making $p = \phi_V(v)$. Consequently, the operations on p and q are interchanged.

According to the operations mapped on neighbor PEs, interchange action is further classified into three cases:

- interchange neighbor PEs with an operation and its routing node includes 2 sub-actions: 1) swapping the target PEs of the operation and its routing node, 2) merging the route node to the operation and the target PE of the routing node becomes a empty PE. As shown in Fig. 8(c), operation a and its routing node a' swapped in the first case in sub-action 1, and routing node a' is merged into operation a in sub-action 2.
- interchange neighbor PEs with only 1 operation on one of them includes 3 sub-actions: 1) move the operation to the empty PE, 2) add a routing node on the empty PE for one sink of the operation, 3) add a routing node on the empty PE for the other sink of the operation. In case 2) and 3), if there is only one sink for the operation, then just add a routing node on the empty PE for the only one operation. As shown in Fig. 8(d), operation a is moved to blue empty PE in sub-action 1, a routing node is added to the blue sink on the blue empty PE in sub-action 2, and a routing node is added to red sink on the blue empty PE in sub-action 3.
- other cases: the positions of the two operations are swapped. Accordingly, the edges of operations follow the new positions. As show in Fig. 8(b), the target PE of operation a and b is swapped.

From the discuss above, we note that there are at most three sub-actions in an interchange, and in most cases, there is only one sub-action in an interchange. Therefore, we don't increase the action space for sub-actions and the size of the whole action space is $2WH$-W-H. If more than one sub-actions occur in an interchange, these sub-actions are selected with equal probabilities. It can cover all the basic operations of DFG mapping, placement, routing and RPE insertion, and these basic operations could be closely coupled, which offers more opportunities to find a solution closer to the optimal one.

Rewards. We need a proper reward signal to guide the agent towards good solutions for our objective, minimizing the cost O while keeping mapping valid. Based on the cost function in Equation (4), the step reward is represented as follows:

$$R(s, a) = O(s) - O(s') \tag{8}$$

where s' indicates the new state by taking action a on state s. With the reward deduced from mapping objective, the agent could receive intensive and correct signals indicating how well the agent is doing at every time-step. Therefore, the agent can learn how to map DFGs efficiently and optically.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

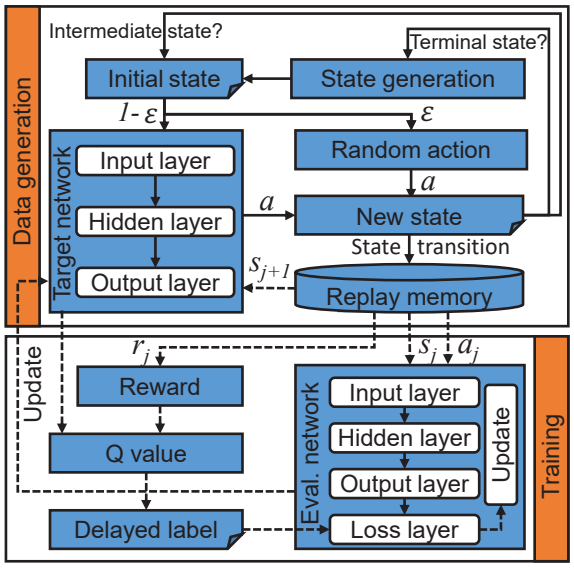

Fig. 9. Data generation and training flow of our approach

.

Algorithm 1 Training algorithm with memory replay

Input:

Initialize replay memory D to capacity N
Initialize the neural network with random weights
for episode = 1, M **do**
  Initialize state s_0 with random generation
  for t=1, T **do**
    With probability ϵ select a random action a_t otherwise select $a_t = \arg\max_a Q^*(s_t, a; \theta)$
    Execute action a_t in PEA emulator and observe reward r_t and image s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    Set $y_j = r_j$ for if s_{j+1} is terminal state, otherwise set $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta)$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ as Equation (3)
    Update $\hat{Q} = Q$ every K steps
  end for
end for

C. Data Generation and Training

We present the action-value of RL as a neural network (called Q-network) which takes a mapping state image described above as input, and outputs the approximate action-value of all possible actions. As shown in Fig. 9, the running flow of our approach involves data generation and training. In order to make learning more efficient, target network freezing [18] is used in the flow. Therefore, there are two Q-network, named target network and evaluation network, in the data generation phase and training phase, respectively. These two networks have the same structure and work as follows:

In the data generation phase, 1) we first generate initial state s_t by random DFG mapping on PEA. 2) Then, ϵ-greedy policy is used to select an action a_t for the state, i.e., with the probability of ϵ selecting a random action and with the probability of 1-ϵ selecting a predicted action by the target network. ϵ decreases with time-step from 1.0 and becomes fixed when reaching to 0.05. 3) Next, the reward r_t and new state s_{t+1} are generated by executing the selected action a_t on the initial state s_t. 4) Finally, we store the transition (s_t, a_t, r_t, s_{t+1}) into a replay memory. In step 3), if the new state is a terminal state, we go back to step 1. Otherwise, we send the new state to the target network in step 2). Repeat the above steps over and over again, we generate millions of transition samples and update the samples in the replay memory in a first-in-first-out way. The random generated DFG in step 1) should satisfy 4 conditions: a) the number of operations in the state involved DFG is from 4 to $W \times H$. b) there is no operation conflict in the mapping state, c) the number of input edges of each operation (i.e., in-degree) in the DFG is less than or equal to 2, and d) the sum of in-degree and out-degree (i.e., the number of output edges of an operation) is less than or equal to the number of connections of the target PE.

In the training phase, 1) we first sample a mini-batch (s_j, a_j, r_j, s_{j+1}) if the replay memory is large enough. 2) Then, the state s_{j+1} is sent to the target network to find the maximal action value $Q' = \max_a Q(s_{j+1}, a; \theta)$. 3) Next, the delayed label is generated using equation $y_j = r_j + Q'$. 4) Meanwhile, state s_j and a_j are sent to the evaluation network to obtain the reference value $Q*$ of a_t on s_t. 5) Then, by calculating the gradient in Equation (3) from y_j and $Q*$, the weights of the evaluation network in update at each iteration. As target network freezing [18] technique is used for more efficient learning, the weights of evaluation are periodically cloned to the target network at every K steps. The details of training are presented in Algorithm 1.

After training, the DFG mapping becomes intuitive and simple. By using the trained network on ϵ-greedy policy, with ϵ fixed to 0.05, the mapping state is updated step by step and gradually becomes valid and optimized.

IV. Experiment Results

To demonstrate the effectiveness of the proposed mapping approach, RLMap, we conduct a series of experiments, including experiments of get more valid mappings, experiments of getting high-quality mapping, experiments of learning convergence, a case study, experiments of scalability to different architectures and experiments of compilation time.

A. Setup

Benchmarks In testing phase, we conducted experiments on 16 DFG kernels from benchmark MiBench, Spec2006 and PolyBench with 4 to 58 operations most widely used by academic researchers for CGRA mapping. The number of operations and edges for each of the DFG kernel are summarized in Table I in the order of operation number.

Target Architectures We consider CGRAs with different PEA sizes and different routing styles to fully evaluate our

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE I
STATISTICS OF THE BENCHMARK DFGS

DFGs	#OPs	#EDGEs	DFGs	#OPs	#EDGEs
bcyl	4	5	calvir	16	8
adi	6	8	jquant2	18	17
filter	6	4	mshift	20	22
wrf	5	5	vsolve	21	20
seidel-2d	9	9	fdtd-apml	21	22
pppm	12	9	clincs	29	32
places	13	12	ftt	40	32
wayinit	17	16	jfdctflt	58	44

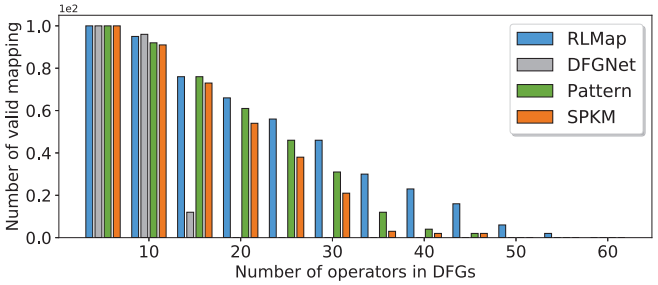

Fig. 10. Map more DFGs on 8×8 mesh-routed CGRA

proposed approach. The PEA size differs from 4×4 to 8×8, which covers most CGRAs in the literature. The routing style varies from mesh (routing with 4 neighbor PE connected) to mesh plus (routing with 4 neighbor PEs and 4 one-hop PEs connected), which cover the routing manners of most CGRAs in the literature.

Neural network architecture We built the RLMap prototype described in using a neural network with an input layer, a hidden layer and an output layer. The size of input layer and output layer is $W^2 \times H^2$ and $2 \times W \times H$-W-H, as described in subsection III-B. Empirically, in our mapping problem, we use a full connected layer as the hidden layer as it captures more details from mapping state than others. The number of neurons in the hidden layer also varies for CGRAs with different PEA size (see discussions below).

Baselines

In testing phase, all the experiments were performed on the same Linux workstation with an Intel Xeon 2.4 GHz CPU and 64GB memory. Since SPKM[8], pattern-based mapping[10] and DFNet[11] are typical and excellent approaches for CGRA spatial mapping, we compared the quality of our placement results with the three approaches. For SPKM and DFGNet, the cost function of the approach is modified to the objective in Equation (4). For pattern-based mapping approach (we use "Pattern" below for simplify), we use the patterns and arrangements listed in Fig. 10 in [10] as the dictionary.

B. Map more applications

Mapping validation is a basic condition of the mapping problem. It becomes more difficult when the available routing resources get sparser, such as mesh-routed PEA. In order to demonstrate the ability of getting valid mapping, we try to map 100 randomly generated DFGs onto an 8×8 mesh-routed PEA using different approaches. As some of the DFGs can not be validly mapped originally, we terminate the mapping process after 2 hours. Fig 10 plots how many DFGs can be validly mapped by each approach. The horizontal axis in Fig. 10 represents the number of nodes that each DFG contains, and vertical axis represents the number of valid mappings out of 100 DFGs. From Fig. 10, we note that it is more difficult to find a valid mapping when the number of nodes in each DFG increases. When the number of node is less than 10, all the four mapping method perform well and they almost find the same number of valid mappings. When the number of node is 15, DFGNet performs worse as it only supports DFGs with nodes up to 16. When the number of node is more than 50 (approximate the size of PEA), pattern-based method and SPKM method can hardly find a valid mapping, while our proposed method can still find several valid mappings. On average, our proposed approach can obtain 21.7× and 34.3× more valid mappings than pattern-based method and SPKM method. As the trained Q-network in our method can predict good mapping manner, our proposed approach can find more valid mappings within the same period of time.

C. Get High-Quality Mappings

The reward designed in Equation (8) can not only make the agent get valid mapping, but also can make agent get low cost mapping. Table II presents the detailed costs of different approaches on the benchmark described in Table I targeting to an 8×8 mesh routed PEA, where DFGNet is not included as it only supports small DFGs. In table II, N_{op}, N_{pg}, N_{nop} and N_{hop} indicate the number of operators, the number of routing PEs, the number of empty PEs in the smallest rectangular and the number of 1-hop connections, respectively. Our approach, RLMap, can get 16 valid mappings while other two approaches can only get 14 valid mappings out of 16. For kernels $filter$, $pppm$ and $calvir$, the three approaches almost obtain the same cost, as the structures of these kernels are relative simple. Consequently, all approaches can find the optimal solutions. It is noteworthy that our approach can still find valid mappings for complicated kernels $clincs$ and $jfdctlt$ while other approaches all fails. The kernel $clincs$ has 29 operations, 32 edges and 6 cycles and some of the edges are heavily entangled. Therefore, It is quite difficult to find the optimal, even a valid solution, for this kernel. However, RLMap can still find a solution with 3 routing PEs and 7 1-hop connections with the help of Q-network prediction. As for kernel $jfdctlt$, the number of operations is 58, which is quite close to the size of 8×8 PEA. Consequently, it is also difficult to find a valid mapping for it. With the efficient action-value prediction from Q-network, our approach still finds a solution with 2 routing PE and 2 1-hop connections. On average, our approach can get 5.66% and 2.91% less costs as compared to SPKM and pattern-based approaches.

D. Training and Convergence

The training time is highly related to the convergence behavior of the Q-network. In reinforcement learning of our work, we randomly generate initial mapping states for various

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE II
COMPARISONS OF MAPPING COST WITH SPKM AND PATTERN BASED APPROACHES

DFGs	N_{op}	N_{pg}			N_{nop}			N_{hop}			$Cost$		
	All	SPKM	Pattern	RLMap	SPKM	Pattern	RLMap	SPKM	Pattern	RLMap	SPKM	Pattern	RLMap
bycl	4	3	3	3	2	1	1	1	1	1	1.12e4	1.08e4	1.08e4
adi	6	3	4	4	3	2	2	1	0	0	1.56e4	1.60e4	1.60e4
filter	6	0	0	0	0	0	0	0	0	0	1.20e4	1.20e4	1.20e4
wrf	5	0	0	0	1	0	0	1	1	1	1.04e4	1.00e4	1.00e4
seidel-2d	9	1	1	0	2	2	3	1	1	1	1.96e4	1.96e4	1.92e4
pppm	12	0	0	0	0	0	0	0	0	0	2.40e4	2.40e4	2.40e4
places	13	0	0	0	1	2	2	1	0	1	2.64e4	2.68e4	2.68e4
wayinit	17	0	0	0	3	1	1	1	3	1	3.52e4	3.44e4	3.44e4
calvir	16	0	0	0	0	0	0	0	0	0	3.20e4	3.20e4	3.20e4
jquant2	18	0	0	0	2	3	9	2	3	2	3.96e4	3.72e4	3.68e4
mshift	20	1	1	1	15	12	2	4	3	2	4.68e4	4.56e4	4.16e4
vsolve	21	0	0	0	7	7	3	0	0	0	4.48e4	4.48e4	4.32e4
ftt	40	0	0	0	24	8	0	16	8	1	8.98e4	8.33e4	8.00e4
fdtd-apml	21	5	3	1	12	10	2	6	4	3	5.09e4	4.84e4	4.36e4
clincs	29	-	-	3	-	-	6	-	-	7	-	-	6.29e4
jfdctt	58	-	-	2	-	-	4	-	-	2	-	-	1.19e5
Normalized	1.00	1.44	1.33	1.00	4.39	2.67	1.00	2.62	1.85	1.00	1.06	1.03	1

DFGs. Then, we select actions by ϵ-greedy policy, generate state transitions and store these transitions in a replay memory with the maximal size equaling to $5.0e5$. When the number of transitions in replay memory is beyond $1.0e5$, a batch of transitions of size 32 are sampled and training is started.

In reinforcement learning, however, accurately evaluating the progress of an agent during training can be challenging. Since the evaluation metric of reinforcement learning, as suggested by [18], is the total reward the agent collects in an episode or game averaged over a number of games, we usually periodically compute it during training. During training, we track the learning performance by recording the learning rate, average loss per epoch, average episode reward per epoch and average number of episodes (n_{ae}) per epoch. The epoch size (n_{ep}) is set to 50000. Fig. 11 shows the learning curves for 6×6 PEA during training with relaxation factor $\beta = 1.1$, including learning rate curve, loss curve, average reward curve over epochs and average number of episodes over epochs. As shown in Fig. 11(a), the learning rate is decayed exponentially from $2.5e$-3 to $2.5e$-3. The loss, as depicted in Fig. 11(b), decreases with the time-step and gets close to $5.0e$-5 at time-step $8.0e6$. The average episode reward increase with time-step and become convergence at time-step $3.8e6$, which indicates our approach can always get gains when getting into convergence. As shown in Fig. 11, the number of episodes per epoch can demonstrate the effectiveness of our approach directly. As the maximal time-steps per episode (n_{mt}) is set to 10000 and the epoch size (n_{ep}) is 50000, the lower bound of n_{ae} is $\frac{n_{ep}}{n_{mt}} = 5$. When the time-step is less than $2.0e6$, n_{ae} is kept to 5 because the Q-network is still stupid and can hardly get a valid mapping. When time-step is larger than $2.0e6$ and less than $5.0e6$, n_{ae} becomes larger on average as the Q-network becomes smart to handle the mapping. When the time-step is larger than $5.0e6$, n_{ae} becomes even larger on average as the training becomes convergent. We note that the variance of n_{ae} is very large even though the Q-network gets convergent. This is because that the initial mapping state is randomly generated and some of them may cannot be validly mapped originally. When it happens, n_{ae} is also relative small.

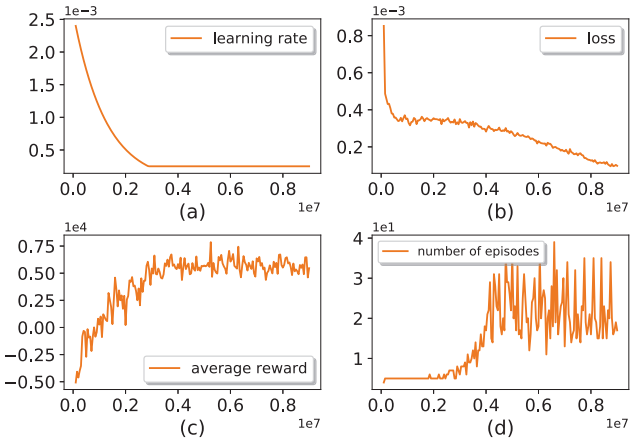

Fig. 11. Learning curves in training

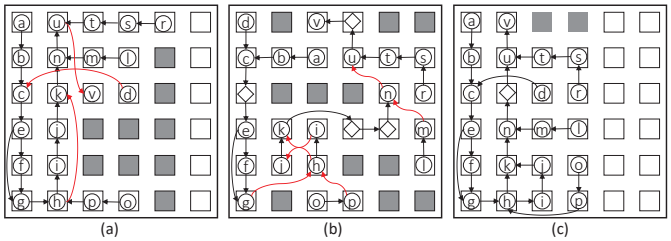

Fig. 12. Case study on kernel $fdtd\text{-}apml$ (a) initial placement, (b) mapping result at time-step $2E6$, (c)mapping result at time-step $8E6$

E. Case Study

To show the gain from our approach, we take the mapping of kernel $fdtd\text{-}apml$ on a 6×6 mesh plus PEA for case study. As shown in Fig. 12, kernel $fdtd\text{-}apml$ has 21 operations in total. Firstly, an initial placement is given, shown in Fig. 12(a), including three invalidly routed edge (see the red arrows). When the time-step of training reaches to $2e6$, the Q-network is still stupid and random actions are often taken in ϵ-greedy policy. Consequently, more invalidly-routed edges and more

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE III
NEURAL ARCHITECTURE FOR DIFFERENT PEA SIZE

PEA size	Input Layer	Hidden Layer	Output Layer
4×4	16×16	256	24
6×6	36×36	512	60
8×8	64×64	1024	112

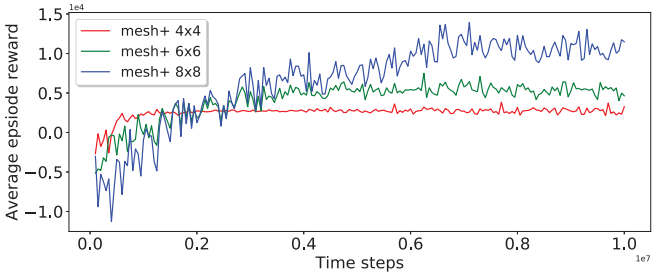

Fig. 13. Training curves of agents for different PEA size

trivial routing PE are generated, as shown in Fig. 12(b). When the time-step of training reaches to $8e6$, the Q-network becomes smart and less exploration is conducted in ϵ-greedy algorithm. Therefore, the better solution is generated by the Q-network from the initial state within 10000 time-steps.

F. Scalability to different architectures

Our approach is applicable to different PEA size by simply modifying the layer size of the neural network. Table III gives the specific layer size set for PEA size of 4×4, 6×6 and 8×8. The sizes of input layer and output layer are strictly determined by the state representation and action representation described in section III. The number of neurons in hidden layer is set 256, 512 and 1024 for 4×4, 6×6 and 8×8 PEAs, empirically. With the Q-network set above, the training curves tracking the agent's average episode reward are demonstrated in Fig. 13. From the training curves, the agent of larger PEA size spends more iterations to get into convergence as the data space and neural architecture size are much larger. The training time-step to get into convergence for 4×4, 6×6 and 8×8 PEAs are about $5e5$, $2e6$ and $7e6$. The time costs of training for three PEAs are also quite different. As shown in Table IV, it spends 5.05 hours, 23.34 hours and 133.23 hours to make the neural networks for 4×4, 6×6 and 8×8 PEAs be convergent, respectively. The training time seems to be exponential to the PEA size and it becomes impractical for very large PEAs. However, the PEA size in real CGRA is usually designed to be small in consideration of power and area. Therefore, our approach is still applicable for most CGRAs.

G. Compilation Time

Although the training time of RLMap is tediously long for larger PEA, the compiling time is acceptable. As shown in Fig. 14, the compilation time of the kernels in Table I of different approaches targeting to 8×8 mesh-routed PEA is presented, where the x-axis indicates the compilation time (seconds). All the approaches terminated after 2 hours. SPKM

TABLE IV
TRAINING TIME FOR DIFFERENT PEA SIZE

PEA size	4×4	6×6	8×8
time(hours)	5.05	23.34	133.23

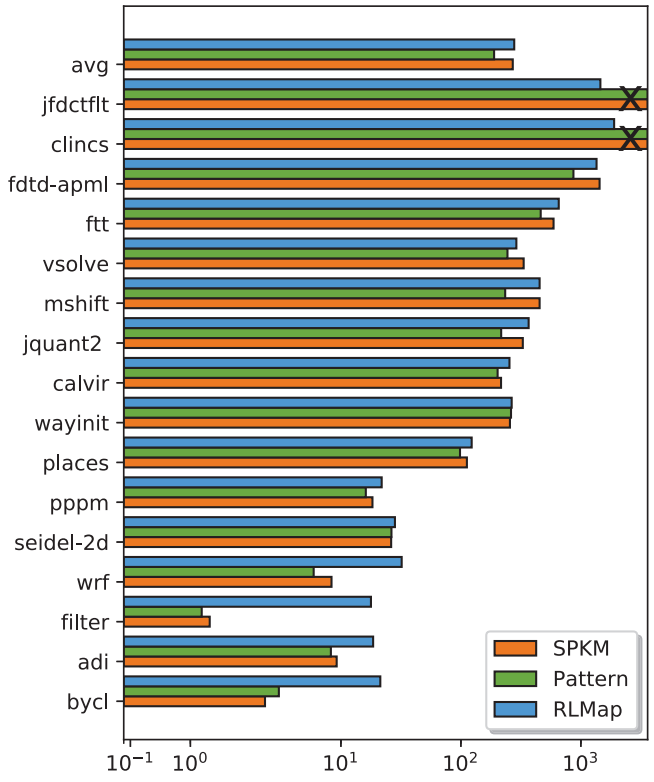

Fig. 14. Compilation time of different approaches

and pattern-based approaches find solutions for 14 kernels and our approach finds solutions for all the 16 kernels. We note that our approach performs worse than other approaches on small kernels. That is because there are millions of operations in the forward prediction process of Q-network at each time-step. As for small kernels, the time-step reduced by our approach is not obvious. Thus, the compilation time, the product of time-step and latency of each step, is still very long for small kernels. As for large kernels, our approach reduces the time-steps greatly. Consequently, the compilation time of our approach also keeps moderate. On average, our approach gets compilation time comparable to that of SPKM.

V. CONCLUSIONS

Traditional CGRA mapping algorithms optimize placement, routing and PE insertion, respectively, and iteratively search global optimized solution among the three steps. This paper has proposed an efficient yet high-quality CGRA mapping approach. It is a novel academic method unifying placement, routing and PE insertion in a deep reinforcement learning framework for CGRA mapping. In the Q-network for state-action evaluation, state, action, reward and network architecture are well designed to guide training to improve routability while reducing mapping cost. Consequently, the proposed

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

approach can perform DFG mappings with high quality. The experimental results have shown that our approach has established the successful use of deep reinforcement learning for high-quality and practical CGRA mapping.

REFERENCES

[1] D. M. T. Office, "Electronics resurgence initiative: Page 3 investments."

[2] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 29.

[3] X. Wei, Y. Liang, T. Wang, S. Lu, and J. Cong, "Throughput optimization for streaming applications on cpu-fpga heterogeneous systems," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 488–493.

[4] B. Mei, S. Vernalde, H. De Man, and R. Lauwereins, "Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix," 2003.

[5] V. Govindaraju, C. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, "Dyser: Unifying functionality and parallelism specialization for energy-efficient computing," *IEEE Micro*, vol. 32, no. 5, pp. 38–51, 2012.

[6] V. Govindaraju, C. Ho, and K. Sankaralingam, "Dynamically specialized datapaths for energy efficient computing," 2011.

[7] L. Liu, D. Wang, M. Zhu, Y. Wang, S. Yin, P. Cao, J. Yang, and S. Wei, "An energy-efficient coarse-grained reconfigurable processing unit for multiple-standard video decoding," *IEEE Transactions on Multimedia*, vol. 17, no. 10, pp. 1706–1720, 2015.

[8] J. W. Yoon, A. Shrivastava, S. Park, M. Ahn, R. Jeyapaul, and Y. Paek, "Spkm: A novel graph drawing based algorithm for application mapping onto coarse-grained reconfigurable architectures," in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. IEEE Computer Society Press, 2008, pp. 776–782.

[9] G. Mehta, C. Crawford, X. Luo, N. Parde, K. Patel, B. Rodgers, A. K. Sistla, A. Yadav, and M. Reisner, "Untangled: A game environment for discovery of creative mapping strategies," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 6, no. 3, p. 13, 2013.

[10] G. Mehta, K. K. Patel, N. Parde, and N. S. Pollard, "Data-driven mapping using local patterns," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 11, pp. 1668–1681, 2013.

[11] S. Yin, D. Liu, L. Sun, L. Liu, and S. Wei, "Dfgnet: Mapping dataflow graph onto cgra by a deep learning approach," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017, pp. 1–4.

[12] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," in *Computers and Digital Techniques, IEE Proceedings-*, vol. 150, no. 5. IET, 2003, pp. 255–61.

[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*. ACM, 2012, pp. 1284–1291.

[14] D. Liu, S. Yin, L. Liu, and S. Wei, "Polyhedral model based mapping optimization of loop nests for cgras," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, pp. 1–8.

[15] D. Liu, S. Yin, Y. Peng, L. Liu, and S. Wei, "Optimizing spatial mapping of nested loop for coarse-grained reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 11, pp. 2581–2594, 2015.

[16] H. Park, K. Fan, S. A. Mahlke, T. Oh, H. Kim, and H.-s. Kim, "Edge-centric modulo scheduling for coarse-grained reconfigurable architectures," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 166–176.

[17] M. Hamzeh and A. Shrivastava, "Regimap: register-aware application mapping on coarse-grained reconfigurable architectures (cgras)," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 18.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *Computer Science*, 2013.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, 2016.

[21] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[22] F. Bouwens, M. Berekovic, B. De Sutter, and G. Gaydadjiev, "Architecture enhancements for the adres coarse-grained reconfigurable array," in *High Performance Embedded Architectures and Compilers*. Springer, 2008, pp. 66–81.

[23] G. Lee, S. Lee, K. Choi, and N. Dutt, "Routing-aware application mapping considering steiner points for coarse-grained reconfigurable architecture," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2010, pp. 231–243.

[24] J. W. Yoon, A. Shrivastava, S. Park, M. Ahn, and Y. Paek, "A graph drawing based spatial mapping algorithm for coarse-grained reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 11, pp. 1565–1578, 2009.

[25] S. Friedman, A. Carroll, B. Van 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*. ACM, 2009, pp. 191–200.

[26] C. Ebeling, D. C. Cronquist, and P. Franklin, "Rapidłreconfigurable pipelined datapath," in *International Workshop on Field Programmable Logic and Applications*. Springer, 1996, pp. 126–135.

[27] V. Betz and J. Rose, "Vpr: A new packing, placement and routing tool for fpga research," in *International Workshop on Field Programmable Logic and Applications*. Springer, 1997, pp. 213–222.

[28] T. Taghavi, X. Yang *et al.*, "Dragon2005: Large-scale mixed-size placement tool," in *Proceedings of the 2005 international symposium on Physical design*. ACM, 2005, pp. 245–247.

[29] C. Sechen and A. Sangiovanni-Vincentelli, "The timberwolf placement and routing package," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 2, pp. 510–522, 1985.

[30] T.-C. Chen, T.-C. Hsu, Z.-W. Jiang, and Y.-W. Chang, "Ntuplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs," in *Proceedings of the 2005 international symposium on Physical design*. ACM, 2005, pp. 236–238.

[31] G. Ansaloni, K. Tanimura, L. Pozzi, and N. Dutt, "Integrated kernel partitioning and scheduling for coarse-grained reconfigurable arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 12, pp. 1803–1816, 2012.

[32] S.-Y. Chen and Y.-W. Chang, "Routing-architecture-aware analytical placement for heterogeneous fpgas," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[33] T. Luo and D. Z. Pan, "Dplace2. 0: A stable and efficient analytical placement based on diffusion," in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. IEEE Computer Society Press, 2008, pp. 346–351.

Dajiang Liu received the B.S. degree in School of Microelectronics and Solid-state Electronics, University of Electronic Science and Technology, Chengdu, China, in 2009, and Ph.D. degrees in Institute of Microelectronics, Tsinghua University, Beijing, China in 2015. From 2015 to 2017, he has worked in Tsinghua University as a research associate. He currently works as a lecturer at the college of computer science in Chongqing University, Chongqing, China. His research interests include reconfigurable computing and deep reinforcement learning.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2878183, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Shouyi Yin received the B.S., M.S. and Ph.D. degree in Electronic Engineering from Tsinghua University, China, in 2000, 2002 and 2005 respectively. He has worked in Imperial College London as a research associate. Currently, he is with Institute of Microelectronics at Tsinghua University as an associate professor. His research interests include reconfiguable computing, mobile computing and SoC design.

Guojie Luo received the BS degree in computer science from Peking University, Beijing, China, in 2005, and the MS and PhD degrees in computer science from UCLA, in 2008 and 2011, respectively. He received the 2013 ACM SIGDA Outstanding PhD Dissertation Award in Electronic Design Automation and the 10-year Retrospective Most Influential Paper Award at ASPDAC 2017. He is currently an associate professor in the School of EECS, Peking University. His research interests include electronic design automation, heterogeneous computing with FPGAs and emerging devices, and medical imaging analytics.

Jiaxing Shang was born in Guiyang, Guizhou, China in 1987. He received the B.S. and Ph.D. degrees in Control Science and Engineering from Tsinghua University, Beijing, China, in 2010 and 2016 respectively. He currently works as a lecturer at the college of computer science in Chongqing University, Chongqing, China. He is doing the postdoctoral research in Computer Science and Technology at Chongqing University from 2016. His research interests include social networks analysis, data mining, artificial intelligence, and recommender systems. He has published about 20 high quality journal and conference articles, including KBS, WASA, ICONIP, SNA-KDD, Physica A, etc

Leibo Liu received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. degree in Institute of Microelectronics, Tsinghua University, in 2004.He now serves as an associate professor in Institute of Microelectronics, Tsinghua University. His research interests include reconfigurable computing, mobile computing and VLSI DSP.

Shaojun Wei was born in Beijing, China in 1958. He received Ph.D. degree from Faculte Polytechnique de Mons, Belguim, in 1991. He became a professor in Institute of Microelectronics of Tsinghua University in 1995. He is a senior member of Chinese Institute of Electronics (CIE). His main research interests include VLSI SoC design, EDA methodology, and communication ASIC design.

Yong Feng was born in Chongqing, China in 1977. He received the B.S. degree in Computer Applied Technology from Chongqing University in 1999, the M.S. degree in Computer Systems Organization from Chongqing University in 2003, and Ph.D. degree in Computer Software and Theory from Chongqing Univeristy in 2006. He currently serves as a Professor at the College of Computer Science of Chongqing University in China. From July 2007 to July 2010, he did the postdoctoral research in Control Science and Engineering Center at Chongqing University. Dr. Feng has published more than 50 academic papers and 2 monographs.

Shangbo Zhou was born in Guanxi, China. He received the B.S. degree from Gangxi National College in 1985, the M.S. degree from Sichuan University in 1991, both in Mathematics, and Ph.D. degree in Circuit and System from Electronic Science and Technology University. From 1991 to 2000, he was with the Chongqing Aerospace Electronic and Mechanical Technology Design Research Institute. Since 2003, he has been with the College of Computer Science of Chongqing University, where he is now a Professor. His current research interests include artificial neural networks, physical engineering simulation, visual object tracking and nonlinear dynamical system. He has published more than 100 journal and conference papers, including Physical Review E, Neurocomputing, Chaos, Pattern Recognication, Multimedia Tools and Applications, etc.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.