A CGRA based Neural Network Inference Engine for Deep Reinforcement Learning

Conference Paper · October 2018

DOI: 10.1109/APCCAS.2018.8605639

3 authors, including:

Minglan Liang
Chinese Academy of Sciences

Zheng Wang
Shenzhen Institute of Advanced Technology (SIAT), Shenzhen, China

A CGRA based Neural Network Inference Engine for Deep Reinforcement Learning

Minglan Liang*†, Mingsong Chen*, Zheng Wang†, Jingwei Sun‡,
*School of Information and Communication, Guilin University of Electronic Technology, Guilin, China
†Shenzhen Institutes of Advanced Technology, Chinese Academy of Science, Shenzhen, China
‡PetroChina Jidong Oilfield Company, Tangshan, China

***Abstract*—Recent ultra-fast development of artificial intelligence algorithms has demanded dedicated neural network accelerators, whose high computing performance and low power consumption enable the deployment of deep learning algorithms on the edge computing nodes. State-of-the-art deep learning engines mostly support supervised learning such as CNN, RNN, whereas very few AI engines support on-chip reinforcement learning, which is the foremost algorithm kernel for decision-**自主的 **making subsystem of an autonomous system. In this work, a Coarse-grained Reconfigurable Array (CGRA) like AI computing engine has been designed for the deployments of both supervised and reinforcement learning. Logic synthesis at the design frequency of 200MHz based on 65nm CMOS technology reveals the physical statistics of the proposed engine of 0.32mm^2 in silicon area, 15.45 mW in power consumption. The proposed on-chip AI engine facilitates the implementation of end-to-end perceptual and decision-making networks, which can find its wide employment in autonomous driving, robotics and UAVs.**

***Index Terms*—computer architecture, reinforcement learning, CGRA**

I. Introduction

The recent advancements in neural networks have demonstrated their success in a wide range of application domains, such as computer vision, natural language processing and gaming engines. Although questions still exist on the difficulty in training and detection accuracy of neural network-based algorithms, there is inarguably the increasing demand on the computing power to support increasingly evolving network structures. Traditionally, the deployment of networks is mostly on CPU and GPU-based platforms, resulting in either less computing performance or huge power consumption, which 禁止 抑制 inhibits the wider application scope of neural networks in the terminal nodes of Internet-of-Things.

Recently, both academia and industry put huge efforts into designing a domain specific accelerators, especially for the convolutional neural networks (CNN) [1] [2]. The Cambricon team designed the first neural network processor and instruction set supporting CNN [3]. The Eyeriss architecture builds dedicated convolution modules by minimizing data movement energy with successful tape-out [4]. By adopting quantization techniques [5], CNN accelerators can achieve significant reduction in on-chip memory usage without significant loss on classification accuracy.

This work was supported by NSFC with Grant No. 61702493, 61561014, Shenzhen S&T Funding with Grant No. KQJSCX20170731163915914, SIAT Innovation Program for Excellent Young Researchers (2017001). Minglan Liang (ml.liang@siat.ac.cn) and Mingsong Chen (cms@guet.edu.cn) contribute equally to this work. Zheng Wang is corresponding author (zheng.wang@siat.ac.cn)

On the other hand, the computing community has also witnessed the applications of neural networks in non-perceptual domains such as decision-making, which is the key attempt to realizing towards human-level intelligence. In the domain of control systems, decision making through reinforcement learning has shown progressive achievements. The Deep Q-Networks (DQN), which was originally proposed in [6], uses multi-layer neural networks to implement the Q-function in order to greatly save the lookup-table for storing Q-values. The Deepmind team demonstrates the ground-breaking contribution in [7] by combining CNN and DQN to realize beyond human-level performance in Atari gaming platform. AlphaGo [8] and AlphaGo Zero [9], which are designed with reinforcement learning techniques, beat the top human chess players. Besides, in the domain of autonomous vehicle, increasingly works have been reported by using reinforcement learning for increasing driving behaviors of AI system [10]. Consequently, there is a high probability that future neural networks are capable of performing end-to-end tasks in perception, decision-making and actuation.

Admitting the various designs in CNN accelerators, very few accelerators have been designed to support reinforcement learning on-chip. Even though the work in [11] and [12] proposes the attempt of on-chip realization of reinforcement learning and DQN network, the major challenges are the seamless integration of supervised and reinforcement learning, which should be fundamentally addressed from the instruction-set and architecture perspectives. In this work, we proposed a novel computing engine supporting both on-chip supervised and reinforcement learning. The highly reconfigurable engine is inspired from the Coarse-grained Reconfigurable Array (CGRA) architecture. By decoding the layer-wise instruction, the engine is reconfigurable for deploying various network layers with special architecture support for DQN-based reinforcement learning. Explicitly, the proposed on-chip DQN engine implements fast iteration of Q-values, which leads to on-line action-taking while recording the complete history space of states and actions for the sake of off-line training through accumulated experiences. The simulation and physical results demonstrated that the proposed engine resulted in a small area of 0.32mm^2 with 15.45 mW power consumption.

The work is organized as follows. Section II introduces prior knowledge on Q-learning based reinforcement learning. Section III presents CGRA-like neural network accelerator supporting reinforcement learning. Section IV shows the simulation and physical results on the proposed design. Section

V concludes the work with discussion on future work.

II. Deep reinforcement learning

Reinforcement learning focuses on the task in which the agent interacts with the environment by selecting a series of actions. To realize this, we adopt a neoteric algorithm, a deep Q-network (DQN), which is a combination of reinforcement learning and deep neural network. In DQN, the agent interacts with the environment through observing the states (s), taking actions (a) and accumulate reward (r). The goal of the agent is to maximize accumulative reward by taking better actions.

The agent selects an action from currently observed s_t at each time-step t according to ϵ-greedy strategy, which chooses a random action with probability ϵ. Otherwise the action with maximal Q-value is chosen as $a_t = maxQ(s_t, a_t; \theta_i)$. Whenever an action is taken, the agent gets an instant reward r, then comes to a new state s_{t+1} at the next time-step. The agent's experience (s_t, a_t, r_t, s_{t+1}) is recorded during the exploration. The agent finishes exploration when any target state (destination, failing) is reached.

During the learning phase, applying Q-leaning updates by mini-batches of agent's experience (s_t, a_t, r, s_{t+1}) drawn randomly from the stored sample experiences, computing each target Q of each current state for iteration i through the following equation:

$$Target\_Q = r + \gamma maxQ(s_{t+1}, a_{t+1}; \theta_i) \tag{1}$$

in which, r is the reward, and $maxQ(s_t, a_t; \theta_i)$ is discounted by γ factor at each time-step t and θ_i are the weights of Q-neural network at iteration i. We get the following loss functions $\mathrm{L}(\theta_i)$ between current state-action value $Q(s_t, a_t; \theta_i)$ approximated by deep Q-network and the target Q of current state at this time-step. The Q-network will be updated by minimizing the loss functions through stochastic gradient descent at iteration i.

$$\mathrm{L}(\theta_i) = \mathrm{E}[(Target\_Q - Q(s_t, a_t; \theta_i))^2] \tag{2}$$

III. CGRA for reinforcement learning

The CGRA architecture [13] attracted extensive research in the past decades. The major properties of CGRA are its high reconfigurability as well as computing performance. CGRA associates several coarse-grained computing units in its processing elements (PEs), hence achieving better processing power than Look-up-Table (LUT) based FPGA architecture. Both modes of computing and connectivity are configurable, which make CGRA an ideal candidate for data-intensive algorithms such as signal processing [14] [15]. We observe that neural networks exhibit high similarity with data flow oriented signal processing algorithm and design the proposed CGRA processor for the domain of neural computing.

A. Baseline architecture

a) Top-level architecture: As shown in Figure 1, the top-level CGRA architecture follows the stream-in and stream-out design principle, where data streams are directly fetched from either on-chip SRAM or off-chip DRAM to the shift registers and writes to those storages after processing. Compared to most accelerators designed with systolic array multipliers [16], the proposed top-level architecture adopts 1-D PE array and saves significant power by removing high fanout crossbar for data fetches. Inter-layer data and coefficients of small-scale neural networks are prone to be stored in SRAM due to high energy efficiency while those from state-of-the-art networks can be only kept off-chip and streamed in dynamically. To facilitate this, a central controller which fetches and decodes network instructions is designed. In total 128 PEs are placed on-chip and sharing the same input data stream, whose length can be arbitrary.

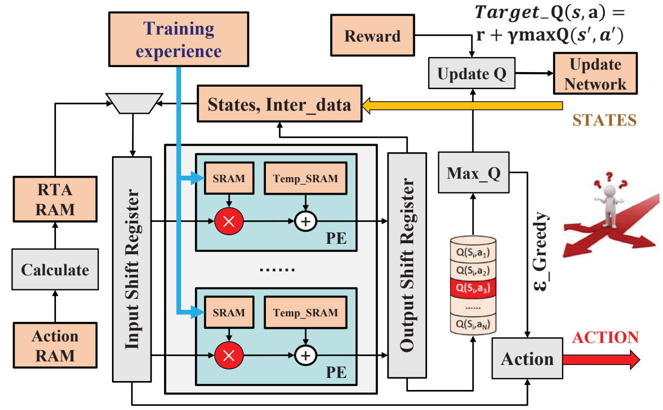

Fig. 1. Baseline CGRA architecture and supports for DQN

b) Processing element: The PE, also named as the artificial neuron, is composed of controlling modules, storage modules and algorithmic units, which are present in Figure 2. The control modules comprise configuration registers, a processing state machine and an address generator unit. The basic calculation module includes a multiplier, an adder and an activation function supporting ReLU operation. Storage modules include parameter SRAM, temporary SRAM and accumulative register. Parameter SRAM keeps the network coefficients which are updated when new network layer is decoded, while the temporary SRAM is used for keeping intermediate data before writing to output registers. Currently the PE supports operating modes of convolution, pooling, fully connected and state-action network for DQN. Various processing mode shares the same computing resource in the PE, such as multiply-accumulator units and parameter SRAM, with the main difference in the operating mode of the address generator.

c) State transition: Figure 3 partially shows the state transition diagram of the proposed CGRA engine. The states can be categorized into sub-groups. Group I implements initialization states such as loading initial data and coefficients. Group II realizes instruction load, decode and execution of 128-stage pipeline of vector processing. Group III is dedicated designed for DQN-based reinforcement learning. Other states implement data output and error management.

B. Architecture supports for reinforcement learning

To support DQN procedure on-chip, the baseline CGRA architecture is extended with dedicated support for DQN,

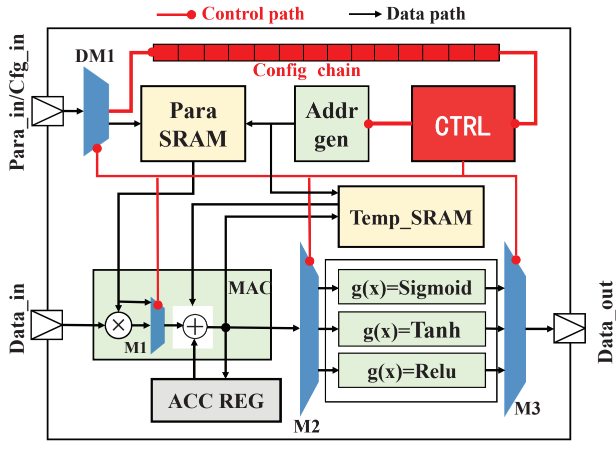

Fig. 2. Architecture of processing element

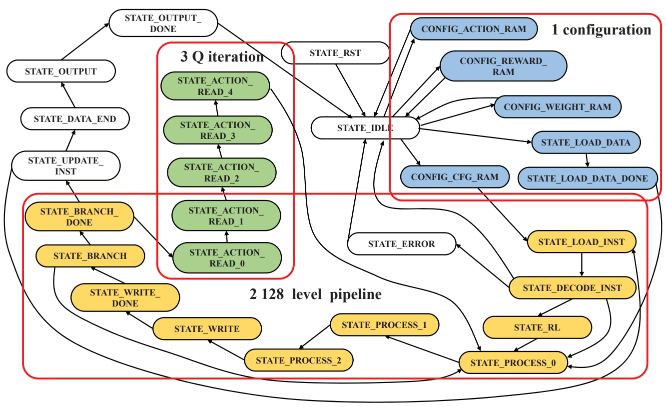

Fig. 3. Central controller state machine

including neural temporal SRAM, action definition RAM and real-time action RAM, controller states for action iteration and action selection logic.

a) Temporal storage for state contribution: For conventional network layers, the local input data can be disregarded after the processing of the current network layer. However, in the state-action layer of DQN, the action part of network input nodes have to be iterated through all possible values until the best Q value is found. Therefore, before the iteration is completed, the contribution from state part of nodes have to be temporarily book-kept and accumulated with contributed values from the action nodes. This is essentially supported by the temporal SRAM in PEs, which keeps the state contribution to output nodes until all action space is iterated.

b) Action definition and real-time action RAMs: The definition of action space includes a number of action nodes and information for individual action, including begin, end values and step size for iteration of each action. Accordingly, action space definition SRAM is designed as shown in Figure 4. Upon processor initialization, the action space is programmed through streamed configuration bits. The CGRA instruction, which defines the state-action layer, triggers the action iteration logic. It fetches action values from the definition RAM and iterates until all predefined ranges of action values are covered. To facilitate such iteration, the real-time action RAM is introduced to temporarily storing the iterating action value and the current action with maximal Q value.

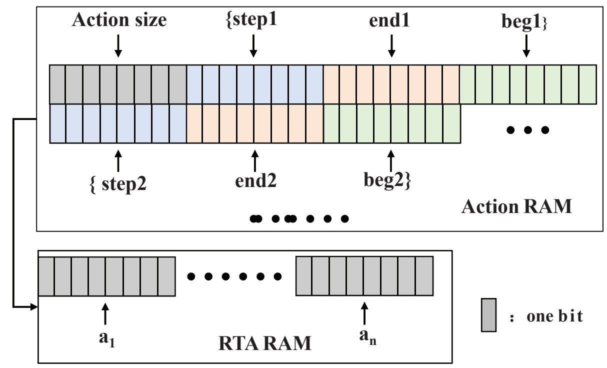

Fig. 4. Action memory design structure

c) Extension of states in controller: As shown in Figure 3, the state machine of baseline CGRA is enhanced with group III states for iterating through predefined action spaces. Compared to a software implementation of action iteration, such hardware- implemented loop achieves significant speed-up while supports the arbitrary size of action space.

d) Action selection logic: In Q-learning the final selected action usually follows the ϵ-greedy principle [6], which means there is a possible chance that random action is selected even though action with maximal Q value is found. The CGRA architecture takes ϵ and a random number seed as inputs in the network instruction, and generates pseudo-random numbers based on shift register and compares the random number with ϵ to determine whether a random action will be taken. Finally, the CGRA writes the current state and selected action into the memory and proceeds to the next state in Q-learning.

C. Interaction between CGRA and host for training

Due to the rich context of reinforcement learning, not all algorithmic kernels are suitable for the on-chip deployment. The proposed CGRA engine accelerates the computation intensive action selection procedure. While the reward calculation, which is more flexible and less parallel in nature, is left for the host CPU for computing. This could be efficiently performed off-line by traversing through the recorded experiences of state and action pairs. The computed reward values are used to adjust the network parameters through standard training procedure of DQN [6], whereas the updated parameters can be loaded into the CGRA chip for further self-exploration. Consequently, the CGRA chip is self-awareness in terms of perception and decision making, while the human is able to update its behavior through controlled off-line training.

IV. Experimental Results

This section explains the experimental methodology, and the physical characteristics of the CGRA IC as well as the time of on-chip Q iteration benchmarked with conventional CPU.

A. Experimental methodology

The CGRA architecture is manually designed using Verilog language and simulated with Modelsim. To generate physical characteristics, Synopsys Design Compiler is utilized for logic synthesis while Cadence EDI for placement and routing.

B. Physical properties

The results of logic synthesis based on UMC 65nm CMOS standard cell technology at 200MHz are shown in Table 1, which includes the power estimation and occupied silicon area for individual hardware module. It is observed that 77.68% of the total power is consumed by the registers, another 19.18% by the clock networks and the rest 3.14% by the combinational logic. The total area of the design is $0.32mm^2$. The relatively small area and power consumption attribute to the reconfigurable design fashion of our CGRA architecture.

TABLE I
MAIN CHARACTERISTICS OF ACCELERATOR

power(mW)		(%)	frequency(um^2)		(%)
total	15.4609		total	319385	
clock_network	2.9852	19.18	buf/Inv	51593	16.15
combinational	0.4885	3.14	combinational	245570	76.88
registers	12.0919	77.68	others	22230	6.97

C. On-chip Q iteration time

We compare the computation time consumed by different structures of DQN network and the dimensions of the action space. As shown in Figure 5, 1, 2, 4 and 6 nodes of action space are tested on the 2-, 5-, and 10-layer network, comparing the Q iteration time required for each combination. For all tested action space, the on-chip Q iterative time of all three network structures is less than 2 ms. Such iteration time slightly increases with the dimension of action space as well as the size of a network.

We profile the Q iteration time for the same configuration of network and action space on the host machine with Intel i7 processor. Matlab with neural network toolbox with the feature of runtime profiling is used for benchmarking. As shown in Figure 5, the on-chip CGRA achieves at least 100x speed-up compared to the host machine for all configurations, which demonstrates the advantage of the proposed architecture for fast Q iteration of DQN network.

V. CONCLUSION

In this paper, a CGRA architecture is proposed for deploying neural networks of supervised and reinforcement learning. The baseline architecture is highly reconfigurable, with processing elements implementing key functions of the neural network. Extensions on the baseline architecture support fast on-chip iteration of action space in order to compute the best Q-value for reinforcement learning. The architecture achieves 100x speed-up benchmarked with Intel CPU, while only consumes 15.46 mW power on the silicon area of 0.32 mm^2.

The future work includes the designing of self-awareness IC using the proposed architecture and training platform to achieve efficient on-chip perception and decision-making.

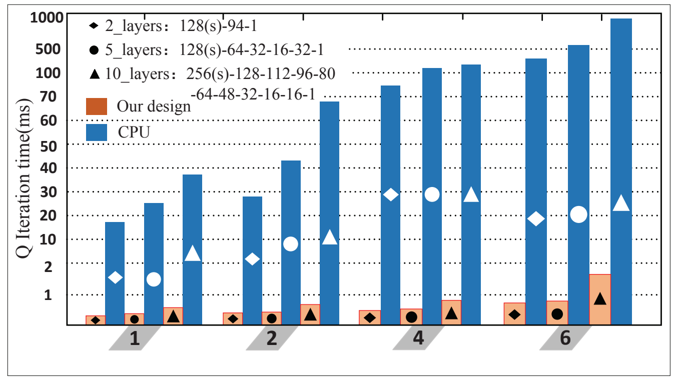

Fig. 5. Comparison of Q iteration time between CGRA and host machine

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[3] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, IEEE Computer Society, 2014.

[4] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[5] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*, pp. 1–8, IEEE, 2016.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[7] 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.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van 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, no. 7587, p. 484, 2016.

[9] 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.

[10] W. Xia, H. Li, and B. Li, "A control strategy of autonomous vehicles based on deep reinforcement learning," in *Computational Intelligence and Design (ISCID), 2016 9th International Symposium on*, vol. 2, pp. 198–201, IEEE, 2016.

[11] J. Park, I. Hong, G. Kim, B.-G. Nam, and H.-J. Yoo, "Intelligent network-on-chip with online reinforcement learning for portable hd object recognition processor," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 2, pp. 476–484, 2014.

[12] P. R. Gankidi, *FPGA Accelerator Architecture for Q-learning and Its Applications in Space Exploration Rovers*. PhD thesis, Arizona State University, 2016.

[13] B. De Sutter, P. Raghavan, and A. Lambrechts, "Coarse-grained reconfigurable array architectures," in *Handbook of signal processing systems*, pp. 449–484, Springer, 2010.

[14] X. Chen, A. Minwegen, Y. Hassan, D. Kammler, S. Li, T. Kempf, A. Chattopadhyay, and G. Ascheid, "Flexdet: flexible, efficient multi-mode mimo detection using reconfigurable asip," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pp. 69–76, IEEE, 2012.

[15] Z. E. Rákossy, D. Stengele, A. Acosta-Aponte, S. Chafekar, P. Bientinesi, and A. Chattopadhyay, "Scalable and efficient linear algebra kernel mapping for low energy consumption on the layers cgra," in *International Symposium on Applied Reconfigurable Computing*, pp. 301–310, Springer, 2015.

[16] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pp. 1–12, IEEE, 2017.