Hardware/Software Co-Exploration for Graph Neural Architectures on FPGAs

Qing Lu*, Weiwen Jiang†, Meng Jiang‡, Jingtong Hu§, and Yiyu Shi¶
*‡¶Dept. of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA
†Dept. of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA
§Dept. of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA, USA

***Abstract*—The success of graph neural networks (GNNs) in the past years has aroused growing interest and effort in designing the best models to handle graph-structured data. Meanwhile, the neural architecture search (NAS) technique has been witnessed to rival against human experts in discovering efficient network topology. Most recently, it has been applied to the field of GNN engineering. However, despite the growing interest in GNN accelerator designs, existing works on graph neural architecture search all concentrate on software (SW) and do not consider hardware (HW) constraints at all, which often leads to sub-optimal system performance when the resulting networks are deployed on hardware accelerators. To address this problem, in this paper we propose a SW-HW co-design framework, namely FGNAS, for automating the search and deployment of GNNs on FPGAs. Experimental results on common benchmark datasets Cora, CiteCeer, and PubMed show that compared with a two-step method built from the state-of-the-art graph NAS framework, FGNAS achieves up to 4.8% improvement accuracy and 3x speedup under the same hardware constraint.**

I. Introduction

Graph neural networks (GNNs) are the state of the art in solving machine learning problems represented in graph forms, including social networking [1], [2], molecular interaction [3], [4], and problems in Electronic Design Automation (EDA) [5], etc. As a result, GNN has attracted a great deal of research interest in deep learning community for both software (SW) [6], [7] and hardware (HW) [8], [9].

Similar to many other neural networks, the performance of GNN significantly depends on its neural architecture, and hence considerable effort has been put into tuning its computational components [10]. Among the existing algorithms, message-passing has set the ground of spatial-based convolutional graph neural networks, from which most recent breakthroughs are derived [11]. As the algorithmic variation increases, to identify better sub-structures of GNN tends to be substantially challenging due to the design space exponentially growing. On the other hand, however, the improvement of feature-extracting ability is still highly demanded.

Soon after being proposed by [12], neural architecture search has become a mainstream research topic of machine learning. It has been demonstrated NAS is promising to surpass the human experts and meanwhile liberate their laborious effort [13]. Although the original NAS using reinforcement learning method suffers from timing inefficiency problem that following works strive to solve [14], [15], it is well established thus adapted to be used for searching novel GNNs.

Quite lately, [16] has designed the first graph NAS framework. Based on the state-of-art GNN methodology, Graph NAS has formulated the layered design space that is preferred to the controller. Besides, parameter sharing strategy is also adopted. Coincidentally, [17] has also used reinforcement learning to automate graph neural network design on similar search space but with split controllers. The search process is well guided in an incremental manner such that the sampling efficiency is boosted. Both of these works have improved the accuracy of GNN against existing hand-crafted networks, indicating NAS is the future solution for graph-based learning.

However, the aforementioned works only focus on the neural architectures but not the associated hardware accelerators. To facilitate the deployment of GNNs, hardware accelerators such as Field Programmable Gate Arrays (FPGAs) are often required to boost operational and computational efficiency [18], [19]. A few e-commerce companies have also developed dedicated platforms in their data centers, maintaining a gigantic graph of billions of nodes [20]. The importance and benefits of hardware-awareness in NAS have already been widely demonstrated for convolutional neural networks (CNNs) [21]–[25]. Yet the exploration of hardware-aware graph NAS remains a missing piece in the literature.

Extending hardware-awareness to graph NAS, however, is not a straight-forward task. The structure of GNN is substantially different from other DNN models with highly irregular data dependency. Accordingly, the design for optimal hardware accelerator is challenging with a large space to explore. On the other hand, hardware-aware NAS typically requires a regular hardware design space that can be easily parameterized. To address this challenge, in this paper we use FPGA as the hardware platform and first propose a linear design framework for GNNs such that hardware parameters can be sampled in a layered manner. We then propose a hardware/software co-design framework to explore the best GNN architecture and the associated FPGA design. Experimental results on common GNN datasets such as Cora, CiteCeer and PubMed show that compared with a two-step exploration method built from the state-of-the-art graph NAS framework [17] our co-desing framework can yield up to 4.8% higher accuracy and 3x speedup under the same hardware constraints. To the best of our knowledge, this is the first hardware-software co-exploration framework for GNNs.

II. GNN Design on FPGA

To facilitate the search process, in this section we propose a linear design framework for FPGAs such that hardware parameters can be sampled in a layered manner. First, the computations are carried by generic units with reusability for all the layers. Second, the hardware parameters are used to scale the parallelism degree of the processing units, facilitating the area-timing tradeoff.

A. Modeling

We adopt a generic FPGA design model that is widely used for CNN accelerators proposed by [26] and adapted it to the GNN accelerator design. For each layer four stages are pipelined consisting of the linear transform, attention coefficient computation, aggregation, and nonlinear operation. The messages in-between consecutive stages are registered. Two buffers are employed to resolve the read/write conflict by alternately accessing the main memory and serving the computational units. As mentioned above, this model is fully scalable in the dimension of the embedded features based on the parameters defined.

B. Mixed Precision

We also consider the mixed-precision scenario in our design where data are quantized using different bit width. Like the other parameters, quantization parameters are also arranged by layer so data in the same layer share the same format. As the methods for quantizing are plentiful and have significant impact on the model accuracy, we adopt the post-training quantization (PTQ) and linear quantization as follows.

Given the quantization interval Δ and range bounded by B_{min} and B_{max}, the quantization of real number x is

$$\hat{x} = clip(\lfloor x/\Delta \rceil \times \Delta, B_{min}, B_{max}), \tag{1}$$

where $\lfloor \rceil$ is rounding to integers. For the fixed-point format with sign, Δ, B_{min} and B_{max} are determined by the number of bits allocated to the integral (bi) and fractional (bf) part as

$$\Delta = 2^{-bf}, B_{min} = -2^{bi}, B_{max} = 2^{bi} - \Delta. \tag{2}$$

Consequently, in the mixed-precision design, four parameters are added to the search space, namely wi, wf for the weights, and ai and af for the activation. With the mixed precision, the hardware space exponentially increases, and the components in our FPGA model requires to be configured by bitwidth. We rely on the HLS tool of Xilinx to synthesize all configurations to profile the sizes and latency information. It is noted the impact of quantization on hardware significantly vary among operators.

III. FGNAS Framework

In this section, we delve into the details of the software/hardware co-exploration framework for GNNs based on the GNN design in Section II. There are three main components comprising FGNAS, namely the controller, the FPGA model builder, and the GNN model trainer. For each layer of the child network, our controller generates the parameters of three types defining the network topology, hardware realization, and the precision. With each sample of the controller, a hardware model will be firstly constructed and evaluated against the predefined constraints. Since most samples may not be implementable, their training are circumvented and rewards assigned to be 0; otherwise the network will be built, trained and validated. Finally, when a mini-batch of samples are evaluated, the parameters of the controller will be updated once. The process terminates after a pre-defined certain number of episodes.

A. Problem Formulation

The problem of jointly searching graph neural network architectures and hardware design can be formulated as the following. Given an architecture space $\mathcal{A}$, each sample $a \in \mathcal{A}$ characterizes a hardware space $\mathcal{H}(a)$. The objective is then to find the optimal architecture and hardware design pair $\langle a^*, h^* \rangle$ such that $a^* \in \mathcal{A}$ and $h \in \mathcal{H}(a^*)$. With the target dataset D_t for training and D_v for validation, the accuracy of a design can be measured as $acc_t(a, h)$ and $acc_v(a, h)$, respectively, while the hardware performance $hp(a, h)$ is independent of the data. As the neural architecture sample is parameterized by the weights w, we define the optimality point of the design as

$$\begin{aligned} a^* &= \arg\max_{a \in \mathcal{A}} acc_v(a(w^*), h^*) \\ s.t. : w^* &= \arg\max_{w} acc_t(a(w), h^*) \end{aligned} \tag{3}$$

and at the same time

$$\begin{aligned} h^* &= \arg\max_{h \in \mathcal{H}(a^*)} hp(a^*, h) \\ s.t. &: hp(a^*, h^*) \geq spec \end{aligned} \tag{4}$$

where $spec$ is the hardware specification required to be satisfied by the design.

However, the above formulation is challenging to implement. In the case where the hardware specification relates to multiple objectives, e.g. area and latency, the hardware performance is not a scalar and hence the optimization is ambiguous. In practice, the design is acceptable as long as the hardware constraints are met. In order to optimize the hardware design, one can set more and more strict constraints to the aspect of interest. Therefore, we relax the optimization of hardware performance to the hardware eligibility, and reformulate the problem as

$$\begin{aligned} a^* &= \arg\max_{a \in \mathcal{A}} acc_v(a(w^*), h) \\ s.t. : w^* &= \arg\max_{w} acc_t(a(w), h) \end{aligned} \tag{5}$$

and

$$\begin{aligned} &\exists h \in \mathcal{H}(a^*) \\ s.t. &: hp(a^*, h) \geq spec. \end{aligned} \tag{6}$$

It is worth mentioning when the hardware constraint has multiple dimensions, the $\geq$ symbol applies to every dimension simultaneously.

In this work, we rely one the recurrent neural network (RNN) to jointly optimize both the GNN architecture and its hardware design. As such, the reinforcement learning NAS framework is restructured to co-explore the software and hardware spaces. Based on the above formulation, our framework aims to discover the best neural architectures which are guaranteed to be implementable under specific constraints.

B. Search Space

We divide the search space into two sub-spaces: architecture space and hardware space. For each layer of a GNN, the search spaces are the same, so the same types of parameters are sampled. For illustration convenience, we divide the parameters of a single layer and describe them as follows.

1) architecture space: The architecture space contains the parameters that define the operational mechanism of a graph network. At the time of writing, the topologies of GNNs share message-passing computational flow characterized by graph-wise convolution, and only vary in the way embedded features are generated and combined. In consequence, we define the architecture space regarding the tuning of sub-structures.

Basically, three separate stages are cascaded in each layer: (1) the embedding from last layer are linearly converted; (2) messages between each connected pair of nodes are weighted; and (3) new features of neighbouring nodes are aggregated to produce new embedding. Following the three operations, five parameters are included in the architecture space: embedding dimension, attention type, aggregation type, number of heads and activation function.

2) Hardware Space: Based on the design in Section II-A, the computation of GNN for inference are all parallelizable in terms of the features of the same embedding. As a large dimension would require exponentially complex computation, it is necessary to divide the vector-wise operation into sub-tasks. Therefore, we choose the size for grouping the features as a key parameter to scale the hardware.

Almost all the main tasks can be divided, and we summarize them into four cases:

1) For the embedding to transform from T_i to T_o features, two parameters t_i and t_o are used for grouping them separately.
2) The attention coefficients possibly also require linear operation but the output is a scalar, so we only divide the input by size of t_{attn}.
3) The aggregation is similar to the above case. And we assign parameter t_{aggr} for it.
4) Lastly, the nonlinearity requires one-to-one operation on the feature vector. As this is probably the most challenging operation for hardware, we also group the features into size of t_{act}.

In addition to the architectural and hardware space, we also consider the mixed-precision design which play important roles in both software and hardware performance. In this case, the quantization space also needs to be explored and details is discussed in Section II-B.

TABLE I: Basic information on the statistics of the datasets and our configuration in usage.

Dataset	Cora	CiteSeer	PubMed
# Training Nodes	140	120	60
# Validation Nodes	500	500	500
# Testing Nodes	1000	1000	1000
# Input Features	1433	3703	500
# Classes	7	6	3
learning rate	0.01	0.01	0.01
weight decay	0.0005	0.0005	0.001
Latency (ms)	0.8/0.9/1.0	0.8/0.9/1.0	7/8/9
#LUT/#FF	10k/100k	10k/100k	10k/100k
DSP	10/100	10/100	100/1000

TABLE II: Design space explored by our framework and the actual values used in the experiment.

Space	Symbol	Value
Embedding Dimension	d	4, 8, 12, 16, 32, 64
Attention Type	$attn$	"constant", "gat", "gcn"
Aggregation Type	$aggr$	"add", "max", "mean"
Number of Heads	k	1, 2, 4, 8, 16
Activation Function	act	"relu", "tanh", "sigmoid", "elu"
Linear Group Size	t_{in}/t_{out}	1, 2, 3, 4, 5
Attention Group Size	t_{attn}	1, 2, 4, 8
Aggregation Group Size	t_{aggr}	1, 2, 4, 8
Activation Group Size	t_{act}	1, 2, 3, 4, 5
Integer Bit Width	ai/wi	1, 2, 3
Fraction Bit Width	af/wf	0, 1, 2, 3, 4, 5, 6

C. Algorithm

Reinforcement learning is applied in our design as the searching backbone. As we have parameterized the design of both architecture and hardware and formatted these parameters by layer, one RNN can be employed to sample the parameters sequentially as actions from the respective list of options. For the sampled design, the hardware performance is analyzed using our FPGA model, under the constraints of computatonal resources and latency. Only if the sample hardware design meets the hardware specifications, will the software design be trained and tested on the dataset. The reward for the sample $\langle a, h\rangle$ is:

$$R(a,h) = \begin{cases} 0, & hp(a,h) < spec \\ acc_v(a,h), & \text{otherwise} \end{cases} \tag{7}$$

This way, the training can be circumvented as possible and the search can be faster than pure NAS.

Once the reward is obtained, the parameter θ of the controller is updated following the policy gradient rule [27]:

$$\nabla J(\theta) = \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\gamma^{T-t}\nabla_\theta \log \pi_\theta(a_t|a_{(t-1):1})(R_k - b) \tag{8}$$

where $J(\theta)$ is the expected reward at the initial step.

The controller is configured as the following. The number of steps T equals the total number of parameters to be sampled; the batch size for updating θ is $m = 5$ episodes; the reward is not discounted so $\gamma = 1$; and baseline b is the exponential moving average of the reward with a decaying factor of 0.9.

IV. Experimental Results

A. Datasets and Experimental Setups

In the experiments, three datasets are used for benchmarking the performance on transductive learning, namely Cora, CiteSeer, and PubMed. The statistics and training configuration is listed in Table I. The setting for training on these datasets follows that of [17]. Since the volume and complexity of the datasets vary largely, the hardware of the search is constrained differently and accordingly.

The experiments are carried out using single Nvidia 1080Ti graphic processing unit (GPU), and Intel 8700K CPU. We use Xinlix FPGA devices with 100 MHz clock rate for profiling the latency. It is noted that since we constrains the hardware, comparing the accuracy to the state-of-art networks are not quite sensible and instead we evaluate the searching efficiency against the two baseline methods.

B. Baseline Methods

To evaluate the ability and efficiency of FGNAS, we involve two methods for the comparison.

Random Search. We perform a random search approach as a baseline for search efficiency. The random search results can reflect the distribution of candidate solutions in specific design space. As will be shown later, for certain data and hardware constraints, the random search can render decent result already.

Disjoint Search. To show the advantages of HW/SW co-design, we perform a graph NAS without hardware-awareness using a state-of-the art framework [17], and then optimizes the FPGA design for the identified GNN architecture.

TABLE III: The best accuacry result on different datasets.

	Dataset	Cora	CiteCeer	PubMed
Constraints	Latency	1 ms	1 ms	7 ms
	#LUT/#FF	100k	100k	100k
	#DSP	100	1000	1000
Accuracy	Random Search	69.9%	72.0%	80.0%
	Disjoint Search	69.9%	68.5%	65.6%
	FGNAS (Ours)	71.5%	72.4%	82.4%

We test the searching efficiency of our method using different hardware parameters in latency, number of LUTs/FFs and number DSPs. The result on Cora is shown in Table IV.

C. Searching Details

Table II demonstrates the search space used in the experiments. During the search, the controller is updated with SGD algorithm and the learning rate is set to 0.1. When a two-layer child network is sampled and the hardware efficiency is satisfied, the network will be trained using Adam optimizer for 200 epochs to obtain accuracy. The accuracy will be then feedback to update controller. The depth of the child networks as two layers. The validation is performed after every epoch, from which the highest will be taken as the reward to the controller. By rule of thumb, we set the depth of the child networks as two layers.

The search stops after sampling 2000 child networks. For the joint and random search, due to the violation of hardware constraints, most samples are invalid. For a fair comparison, we use the total number of valid samples to guide the random search such that the GPU hours would be on the same scale. In the case of disjoint search, the GPU time is determined by the number of episodes and we set 200 for the architecture search and 800 for the hardware search. Each experiment is conducted in 5 runs, and the one with the highest test accuracy is taken for evaluation. Finally, we report the evaluated accuracy.

D. Performance

The results are summarized in Table III and Table **??** It is generally observed that the joint search achieves the best accuracy and/or shorter searching time.

1) Comparing with Random Search: The random search is performant in the sense that the highest accuracy is discoverable at certain hardware constraints. For example, with the setting of 1 ms latency, 100,000 LUTs/FFs and 100 DSPs, the random search achieves the best accuracy among the three methods. However, when the constraints are tighter, the distribution of decent samples are sparser. As a result, the best accuracy covered by searching a fixed number of samples is lower than the other two methods.

The search time of random method is around 1x to 2x of the joint search. The reasons are two-fold: Firstly, the sampled networks are more scattered so their average size is larger. Although the GPU calls are equal, the training time of randomly sampled networks are higher. Another reason is that in order to reach the same number of implementable samples as joint search, much more episodes need to be inspected so the CPU time adds up to a considerable level.

2) Comparing with Disjoint Search: The disjoint search consumes highest time with our setting because 1) more samples are actually trained due to the manual setup; and 2) the architecture found in the first step is larger than average size. It is observed that accuracy is slightly better than random search and in some cases surpass the joint search. However, since the pure architecture search are not aware of the hardware constraints at all, the post-quantization accuracy may degrade severely as decent bit width allocation hardly exists.

V. Conclusions

Neural architecture search is a promising solution for the advancement of graph neural network engineering, but it lacks hardware awareness. In this work we propose to an FPGA-based HW/SW co-design framework, namely FGNAS, that jointly explores the architectural and hardware spaces. Using reinforcement learning, generic hardware model, and mixed precision design, FGNAS performs more efficiently than the random search and a state-of-the-art based disjoint methods. Under different hardware constraints, FGNAS has the best accuracy in majority of the test cases with up to 3x faster runtime.

Acknowledgement

This work is supported in part by National Science Foundation under grant CNS-1822099.

TABLE IV: Performance of the proposed joint search framework under different hardware constraints. Best test accuracy and search time on Cora are used to compare against the baselines.

Constraints			FGNAS (ours)		Random Search		Disjoint Search	
latency (ms)	#LUT/#FF	#DSP	Acc.	Time (h)	Acc.	Time (h)	Acc.	Time (h)
0.8	10,000	10	66.2%	0.56	61.8%	1.12	62.6%	1.87
		100	62.9%	0.87	60.0%	1.22	63.8%	1.65
	100,000	10	67.8%	0.88	62.9%	1.12	64.6%	1.55
		100	68.7%	0.89	64.0%	0.95	68.5%	1.40
0.9	10,000	10	68.1%	0.69	68.0%	1.19	66.0%	1.32
		100	69.2%	1.17	68.9%	1.20	69.0%	1.50
	100,000	10	68.8%	0.99	69.0%	1.44	68.0%	1.69
		100	70.2%	0.88	69.5%	1.44	69.6%	1.70
1.0	10,000	10	68.1%	0.72	67.8%	1.23	66.0%	1.38
		100	70.1%	1.33	69.0%	1.40	69.9%	1.44
	100,000	10	68.8%	1.19	69.2%	1.66	69.0%	1.77
		100	71.5%	1.48	69.9%	1.55	69.9%	1.60

REFERENCES

[1] Q. Tan, N. Liu, and X. Hu, "Deep representation learning for social network analysis," *Frontiers in Big Data*, vol. 2, p. 2, 2019. [Online]. Available: https://www.frontiersin.org/article/10.3389/fdata.2019.00002

[2] M. Nurek and R. Michalski, "Combining Machine Learning and Social Network Analysis to Reveal the Organizational Structures," *arXiv e-prints*, p. arXiv:1906.09576, Jun. 2019.

[3] K. Huang, C. Xiao, L. Glass, M. Zitnik, and J. Sun, "Skipgnn: Predicting molecular interactions with skip-graph networks," 2020.

[4] S. Spalević, P. Veličković, J. Kovačević, and M. Nikolić, "Hierarchical protein function prediction with tail-gnns," 2020.

[5] Y. Ma, Z. He, W. Li, L. Zhang, and B. Yu, "Understanding graphs in eda: From shallow to deep learning," in *Proceedings of the 2020 International Symposium on Physical Design*, ser. ISPD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 119–126. [Online]. Available: https://doi.org/10.1145/3372780.3378173

[6] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *arXiv e-prints*, p. arXiv:1901.00596, Jan. 2019.

[7] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated Graph Sequence Neural Networks," *arXiv e-prints*, p. arXiv:1511.05493, Nov. 2015.

[8] Y. Wang, B. Feng, G. Li, S. Li, L. Deng, Y. Xie, and Y. Ding, "Gnnadvisor: An efficient runtime system for gnn acceleration on gpus," 2020.

[9] H. Zeng and V. Prasanna, "Graphact: Accelerating gcn training on cpu-fpga heterogeneous platforms," in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 255–265. [Online]. Available: https://doi.org/10.1145/3373087.3375312

[10] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.

[11] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry," *arXiv e-prints*, p. arXiv:1704.01212, Apr. 2017.

[12] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," *arXiv e-prints*, p. arXiv:1611.01578, Nov. 2016.

[13] L.-C. Chen, M. D. Collins, Y. Zhu, G. Papand reou, B. Zoph, F. Schroff, H. Adam, and J. Shlens, "Searching for Efficient Multi-Scale Architectures for Dense Image Prediction," *arXiv e-prints*, p. arXiv:1809.04184, Sep. 2018.

[14] S. Yan, B. Fang, F. Zhang, Y. Zheng, X. Zeng, H. Xu, and M. Zhang, "Hm-nas: Efficient neural architecture search via hierarchical masking," 2019.

[15] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," 2019.

[16] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graphnas: Graph neural architecture search with reinforcement learning," *ArXiv*, vol. abs/1904.09981, 2019.

[17] K. Zhou, Q. Song, X. Huang, and X. Hu, "Auto-GNN: Neural Architecture Search of Graph Neural Networks," *arXiv e-prints*, p. arXiv:1909.03184, Sep. 2019.

[18] T. Geng, A. Li, W. Tianqi, C. Wu, Y. Li, A. Tumeo, and M. Herbordt, "Uwb-gcn: Hardware acceleration of graph-convolution-network through runtime workload rebalancing," 08 2019.

[19] W. Jiang, E. H.-M. Sha, X. Zhang, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Achieving super-linear speedup across multi-FPGA for real-time DNN inference," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 5s, pp. 1–23, oct 2019.

[20] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "AliGraph: A Comprehensive Graph Neural Network Platform," *arXiv e-prints*, p. arXiv:1902.08730, Feb. 2019.

[21] L. L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu, "Fast hardware-aware neural architecture search," 2020.

[22] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization," *ArXiv*, vol. abs/1811.08886, 2018.

[23] X. Zhang, W. Jiang, Y. Shi, and J. Hu, "When neural architecture search meets hardware implementation: from hardware awareness to co-design," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019, pp. 25–30.

[24] W. Jiang, L. Yang, E. Sha, Q. Zhuge, S. Gu, S. Dasgupta, Y. Shi, and J. Hu, "Hardware/software co-exploration of neural architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, pp. 1–1, 04 2020.

[25] L. Yang, W. Jiang, W. Liu, E. H. M. Sha, Y. Shi, and J. Hu, "Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 85–90.

[26] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 161–170. [Online]. Available: https://doi.org/10.1145/2684746.2689060

[27] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.