Co-Exploring Neural Architecture and Network-on-Chip Design for Real-Time Artificial Intelligence

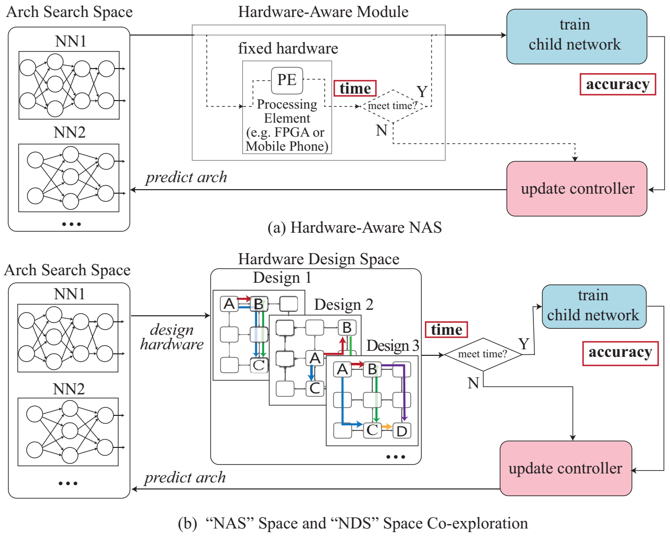
Lei Yang*, Weiwen Jiang†, Weichen Liu‡, Edwin H. M. Sha§, Yiyu Shi†, Jingtong Hu*
*Department of Electrical and Computer Engineering, University of Pittsburgh, US
†Department of Computer Science and Engineering, University of Notre Dame, US
‡School of Computer Science and Engineering, Nanyang Technological University Singapore, Singapore
§School of Computer Science and Software Engineering, East China Normal University, China

Abstract— Hardware-aware Neural Architecture Search (NAS), which automatically finds an architecture that works best on a given hardware design, has prevailed in response to the ever-growing demand for real-time Artificial Intelligence (AI). However, in many situations, the underlying hardware is not pre-determined. We argue that simply assuming an arbitrary yet fixed hardware design will lead to inferior solutions, and it is best to co-explore neural architecture space and hardware design space for the best pair of neural architecture and hardware design. To demonstrate this, we employ Network-on-Chip (NoC) as the infrastructure and propose a novel framework, namely NANDS, to co-explore NAS space and NoC Design Search (NDS) space with the objective to maximize accuracy and throughput. Since two metrics are tightly coupled, we develop a multi-phase manager to guide NANDS to gradually converge to solutions with the best accuracy-throughput tradeoff. On top of it, we propose techniques to detect and alleviate timing performance bottleneck, which allows better and more efficient exploration of NDS space. Experimental results on common datasets, CIFAR-10, CIFAR-100 and STL-10, show that compared with state-of-the-art hardware-aware NAS, NANDS can achieve 42.99% higher throughput along with 1.58% accuracy improvement. There are cases where hardware-aware NAS cannot find any feasible solutions while NANDS can.

I. Introduction

Neural Architecture Search (NAS) has been proposed to automatically generate neural networks for a machine learning task [1–6]. By achieving competitive or even better accuracy against human-invented solutions, it successfully breaks down the expertise barrier and widens the use of neural networks. Most recently, targeting a fixed hardware platform, hardware-aware NAS has been proposed to respond to the rapid increase of real-time AI, where the timing constraint is considered in addition to NAS exploration, as shown in Figure 1(a).

On the other hand, an observation is that the architectures automatically found by NAS are more irregular and complicated than human-invented ones. For instance, there usually exist a lot of skip connections between layers, which incurs a large amount of data movement. Coupled with high throughput and low latency requirement in real-time AI [7–12], it renders a system with a single processing element hard to satisfy the timing specifications. Network-on-Chip (NoC), which provides highly parallel computation and high-bandwidth on-chip communication, has been explored to accelerate neural architectures [13, 14]. Recently, Xilinx has released Versal Platform [15], which employs NoC to connect processing system (PS), AI Engines accelerators, and programmable logic (PL).



(a) Hardware-Aware NAS

(b) "NAS" Space and "NDS" Space Co-exploration

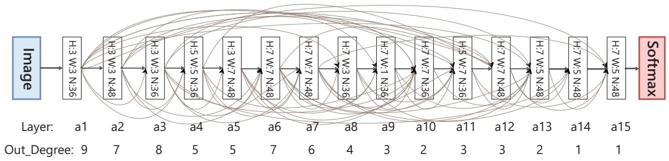
Fig. 1. Comparison of (a) hardware-aware NAS with an arbitrary yet fixed hardware design; (b) proposed NANDS in this work.

The high design flexibility in NoC, however, brings about a challenge for HW-aware NAS, which needs a fixed NoC design to serve as the target hardware. This in turn requires everything in Noc Design Search (NDS) space to be pre-determined, including partition architectures, mapping partitions, and routing on NoC. However, all these cannot be optimally decided until the neural architecture is known after performing HW-aware NAS. To cope with it, a straightforward way is to assume an arbitrary yet fixed NoC design as the target platform to explore NAS space; however, it will apparently result in inferior solutions due to the huge number of candidates in NDS space.

In this work, we propose to co-explore NAS space and NDS space through a novel framework, namely NANDS, as illustrated in Figure 1(b). It aims to identify the best neural architecture and the NoC design, such that the accuracy can be maximized while the system throughput can satisfy the real-time constraint. Specifically, we use a reinforcement learning-based NAS controller [1, 6] as the backbone to explore NAS space. We then integrate it with a multi-phase manager, which guides the exploration by changing the weights between accuracy and throughput in the objective of each phase for enhanced efficiency. We explore NDS space by applying efficient NoC design methodologies for task partition, mapping and

*Lei Yang and Weiwen Jiang make equal contribution.

978-1-7281-4123-7/20/31.00©2020*IEEE*

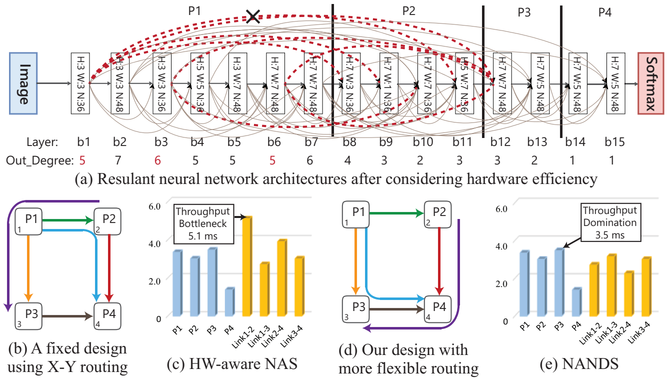


(a) The neural network architecture found by NAS

Platforms / Operation Time	Single Processing Element	4 Processing Elements	
		Bus Interconnection	2-D Mesh NoC
Computation (ms)	12.4	3.4	3.4
Data transmission (ms)	—	14.7	6.2

(b) The timing performance of network implementations on different platforms

Fig. 2. Timing performance of a neural network by NAS [1].



(a) Resulant neural network architectures after considering hardware efficiency

(b) A fixed design using X-Y routing

(c) HW-aware NAS

(d) Our design with more flexible routing

(e) NANDS

Fig. 3. Comparison results of HW-aware NAS and the proposed NANDS.

routing. In addition, we formulate a timing performance model to capture the throughput of NoC design, based on which we develop techniques to detect and alleviate the performance bottleneck.

The main contributions of this work are:

1) We propose NANDS, a framework that opens up freedom to co-explore NDS and NAS. To the best of the authors' knowledge, this is the first work that explores NoC and neural architecture co-design.
2) We devise a multi-phase manager to guide exploration to gradually converge to solutions with the best accuracy-throughput tradeoff.
3) We propose bottleneck detection and alleviation techniques to better explore NDS space for higher timing performance.

Evaluation results on common datasets, including CIFAR-10, CIFAR-100 and STL-10, show that the proposed co-exploration framework can significantly outperform the state-of-the-art HW-aware NAS. Specifically, NANDS can achieve an average of 42.99% higher throughput together with 1.58% accuracy improvement on STL-10, compared with HW-aware NAS. In addition, there exist cases where HW-aware NAS cannot find any feasible solutions, while NANDS can due to the adequate exploration of the large NDS space.

II. Motivation

This section will show the motivations of (1) employing NoC as the infrastructure for the neural architectures identified by NAS, and (2) co-exploring NAS space and NDS space.

A. NoC infrastructure for NAS implementation

The architectures automatically explored by NAS have more irregular and complicated structures than the human-invented ones. For instance, as shown in Figure 2(a), for the neural network with 15 layers obtained by NAS in [1], we observe that the generated architecture (1) contains up to 8 different types of kernels leading the use of a uniform design to be inefficient; (2) involves a lot of skip connections between layers resulting in a large amount of data movement. These coupled with the high throughput and low latency requirement in real-time AI [8, 9] render a system with a single processing element (e.g., a single FPGA) hard to satisfy timing specifications. NoC, which provides highly parallel computation and efficient on-chip communication, is a promising platform to address these challenges. This can be observed from the results in Figure 2(b), where the timing performance of the neural network on 2-D Mesh NoC ($6.2ms$) outperforms that on single-PE ($12.4ms$) and bus-based ($14.7ms$) platforms by 2× and 2.37×.

B. Co-Exploration outperforms SOTA HW-aware NAS

NAS techniques with mono-objective of accuracy can easily become useless in scenario with tight timing constraints. To address this problem, state-of-the-art (SOTA) NAS framework considers timing performance of the explored architectures on fixed hardware in NAS process (HW-aware NAS). It can find a simpler architecture for better timing performance. Targeting on a 2×2 NoC with fixed hardware configuration by X-Y routing in Figure 3(b), HW-aware NAS can identify a simpler architecture in Figure 3(a) with only 0.32% accuracy loss. As a result, the latency is reduced from $6.2ms$ to $5.1ms$ (by 17.07%) as shown in Figure 3(c). The proposed co-exploration approach NANDS can further improve timing performance by opening freedom to jointly explore NAS and NDS spaces. As shown in Figure 3(d)(e), NANDS can identify a better NoC design (with customized routing) to further reduce the latency to $3.5ms$.

III. NANDS Framework and Implementation

In this section, we will present **NANDS**, a multi-phase framework to identify the best pair of neural architecture and NoC design, such that the accuracy of machine learning tasks can be maximized while achieving real-time performance.

A. Problem Definition and Design Principles

We aim to co-explore NAS space and NDS space to find neural architecture and NoC design pairs with the maximized accuracy and efficiency. Formally, *given a specific dataset and a NoC platform, the problem is how to automatically and efficiently generate a pair of neural architecture and NoC design, such that both the accuracy for machine learning task on the given dataset and the inference throughput on NoC can be maximized.* The neural architecture is represented by a set of hyperparameters, including the number of channels, kernel size, strides, etc.; while the NoC design needs to determine the partitioning, mapping and routing. The above problem is a bi-criteria optimization problem. In this paper, we focus on maximizing accuracy with a throughput constraint. However, the proposed techniques can be applied for the counterpart problem to maximize throughput with an accuracy constraint.

To achieve the above goal, we highlight three principles:

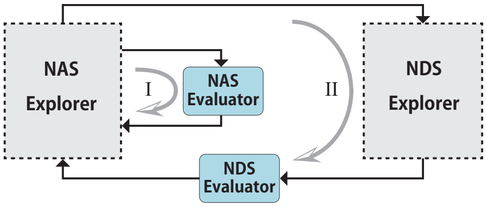


Fig. 4. An overview of two exploration loops in NANDS.

$\mathcal{P}1$. *Accelerate the search process by improving the efficiency of NDS exploration.* In NAS, the long training time will limit its implementation. When it comes to co-exploration, if NDS exploration can be conducted efficiently, it enables us to immediately terminate the training process once it detects that the architecture cannot satisfy the timing constraint.

$\mathcal{P}2$. *Balance the computation workload and network traffic to avoid performance bottleneck.* Unlike the existing approaches targeting on a uniprocessor system, when it comes to NoC, system throughput cannot be optimized by independently maximizing computation efficiency on processors; moreover, the computation workload and network traffic should be well balanced.

$\mathcal{P}3$. *Minimize contention of inter-processor data transmission on NoCs.* Architectures explored by NAS have complicated dependencies and irregular structures. Directly applying them on NoC with a deterministic routing strategy (e.g., X-Y) may cause a large number of NoC contentions, which will further become timing performance bottleneck. To achieve the maximum throughput, we aim to minimize network congestion.

We will first present NANDS, and then introduce the detail implementations which follows the above design principles.

B. NANDS Framework

We propose a multi-phase framework, namely NANDS, to jointly explore NAS space and NDS space. As illustrated in Figure 4, NANDS contains two explorations loops, which further involve four components: NAS Explorer, NDS Explorer, NAS Evaluator, and NDS Evaluator. Two evaluators are designed to assist the explorers: (1) NAS Evaluator takes a neural architecture from NAS Explorer as input, and generates architecture accuracy; (2) NDS Evaluator takes a hardware implementation from NDS Explorer as input, and generates system throughput. Results from two evaluators will be feedback to NAS Explorer to generate neural architectures, called child networks. Then, NDS Explorer takes child network as input to generate its tailored hardware implementation. Overall, the above processes follow two exploration loops, Loop I and II.

Loop I: neural architecture search. NAS Evaluator will train the child networks generated by NAS Explorer, and verify their accuracy. The accuracy will be the reward "R_A" and sent back to NAS Explorer to update hyperparameters.

Loop II: automatic hardware design. NDS Evaluator tests the throughput of the resultant implementation from NDS Explorer. The throughput will be another reward "R_T" and sent back to NAS Explorer to update hyperparameters.

Next, a matched management is required to control the action flow in NANDS, that is, determining which loops to take effect in the exploration. In this paper, we first figure out three possible exploration patterns, based on which we propose a multi-phase management strategy to switch patterns.

Pattern 1: Pure NAS. We only conduct Loop I. NAS Explorer will only receive the reward from NAS Evaluator for updating hyperparameters. This pattern is time-consuming since NAS Evaluator needs to train child networks from scratch.

Pattern 2: Pure NDS. We only perform Loop II. NAS Explorer updates hyperparameters only based on the reward from NDS Evaluator. Benefiting from the avoidance of training child networks and the efficiency of evaluating system throughput, this pattern will be much faster than Pattern 1.

Pattern 3: Co-Exploration. We co-explore two search spaces and update NAS Explorer by the rewards from both NAS and NDS Evaluators. Since two exploration loops are conducted simultaneously, we can prune the invalid child networks (in terms of throughput) at the early stage, and terminate their training process. Hence, this pattern is faster than Pattern 1.

Based on three patterns, we present a management strategy to switch patterns in NANDS. Pattern switch is analog to determine which kind of reward (from NAS Evaluator R_A and/or NDS Evaluator R_T) to be used in calculating reward R for NAS Explorer. Specifically, reward can be calculated as follows:

$$R = \alpha \times R_A + (1 - \alpha) \times R_T \tag{1}$$

where α is a regulate variable. It is clear that (1) $\alpha = 1$ stands for *Pattern 1*; (2) while $\alpha = 0$ stands for *Pattern 2*; otherwise, (3) $0 < \alpha < 1$ stands for *Pattern 3*.

Based on the regulator, we manage the exploration in multi-phase control flow. We adjust α by a predefined function like adjusting learning rate. Let N be the total number of iterations that NANDS will explore. First, when iteration $I \leq \beta \cdot N$ (β is a regulated variable, as well as γ in the following text), we aim to improve accuracy, and initialize α as a small number (e.g., 0.9). Then, when $I > \beta \cdot N$ and $I \leq \gamma \cdot N$, we decrease α to around 0.5. Third, we set $\alpha = 0$ to explore the best throughput. Note that we do not use *Pattern 1* ($\alpha = 1$) since it is the slowest one and cannot guarantee hardware efficiency.

C. NANDS Implementation

Figure 5 reveals the details of Explorers in Figure 4, which contains three main components. In NAS Explorer, ❶ "NAS Controller" is responsible for taking rewards from evaluators to predict hyperparameters to generate child networks. In NDS Explorer, ❷ "NoC Design" is to generate hardware design (including partition, mapping and routing) for the input network on the given NoC. ❸ "Bottleneck Detection and Alleviation " in NDS Explorer is designed to maximize throughput of NoC.

❶ **NAS Controller.** As shown on the left part of Figure 5, NAS controller will explore NAS space to identify new child networks. In this paper, we implement NAS controller based on reinforcement learning approach. RNN is employed to predict the hyperparameters, and the weights of RNN will be updated based on the reinforcement learning using the rewards from evaluators. Unlike the implementation in [1] with mono-criteria (accuracy), NAS controller in NANDS will take both accuracy and throughput to update RNN. Specifically, we employ Formula 1 to generate reward. Not limited to reinforcement learning-based controllers, our framework can also support other approaches, such as evolutionary algorithms.

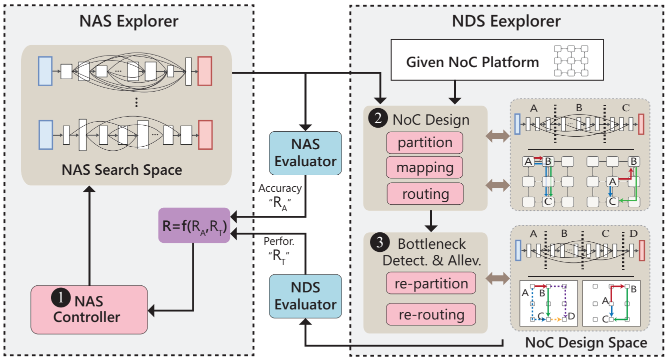


Fig. 5. An overview of the implementation of NANDS framework.

❷ **NoC Design.** Followed by $\mathcal{P}1$-$\mathcal{P}3$, child network will be deployed to NoC in three steps: (1) partition a child network to balance computation, (2) map each partition to a processing element in NoC, (3) determine routing path for data transmission according to data dependency among layers.

The fundamental of computation balanced partition is a multi-way graph partition problem [16]. We apply the algorithm proposed in [16] to get the partitions in time complexity of $O(|E|)$. Then, the partitions will be assigned to processors on NoCs by Dominant Sequence Clustering (DSC) approach [17] with time complexity of $O(|E|log|E|)$. On top of that, we employ X-Y routing algorithm to construct the routing path for a pair of partitions having data dependency. We simply employ above techniques in the implementation, while our presented framework is applicable for any partition, mapping, and routing algorithms.

❸ **Bottleneck Detection and Alleviation.** In the implementation of NANDS framework, we take a further step to detect the bottleneck of system throughput, and then alleviate it by re-partition and re-routing, to maximize the throughput.

Bottleneck Detection. After NoC design, network layers are partitioned and assigned onto NoC. Data transmission paths are built by X-Y routing. Based on the network structure, we can obtain the volume of data between a pair of partitions, and then calculate the accumulate latency on each link (using the bandwidth divided by the sum of data on link), denoted as L_{lk} for link lk. Then, in terms of hardware properties (e.g., DSPs, bandwidth for FPGA-based processing element) in NoC, we can obtain computation latency [18], denoted as L_{pe} for pe.

NoC is performed in a pipelined fashion, the overall system throughput is determined by the slowest computation or communication (as shown in Figure 3). Let $LINK$ and PE be a set of links and processing elements in NoC, respectively, and BIT_{in} be the total bits in one input image. Throughput TP is defined as the number of bits of input can be processed in second (bit per second, bps) calculated by:

$$TP = \frac{BIT_{in}}{\max\{\max_{pe \in PE}(L_{pe}), \max_{lk \in LINK}(L_{lk})\}} \tag{2}$$

Based on the above performance model, we identify three kinds of throughput bottlenecks in a NoC system.

Given an NoC, a neural architecture, a NoC design, there are three kinds of throughput (TP) bottlenecks:

B1: *TP is determined by a processing element;*

B2: *TP is determined by a NoC link, and the link is occupied by only one data transmission path;*

B3: *TP is determined by a NoC link, where there are multiple routing paths will go through.*

Based on Formula 2 and Corollary III, we can easily detect B1. Besides, we can classify B2 and B3 by analyzing the number of routing paths that cover the identified NoC link.

Bottleneck Alleviation. After detection, we propose Re-partition and Re-routing techniques to resolve all kinds of performance bottlenecks as presented in Algorithm 1.

Algorithm 1 Bottleneck Alleviation

```
Input: (1) NoC with $PE$ and $LINK$, (2) TP, (3) Bottleneck type, (4) latency
    function $L$, (5) partition $P$, (6) mapping $M$: $p$ to $pe$, (7) routine path $R$.
Output: A NoC design.
 1: if Bottleneck is B1, and TP is determined by $pe_k$:
 2:     Get partition $p_i$, s.t., $M(p_i) = pe_k$;
 3:     if $p_i$ has only 1 layer:
 4:         Cannot remove the bottleneck and terminate;
 5:     else if NoC has available processing element:
 6:         Partition $p_i \rightarrow p_{i1} + p_{i2}$ to minimize their max latency;
 7:     else:
 8:         Find $p_i$'s neighbor $p_k$ with the minimum latency;
 9:         Move layers in $p_i$ to $p_k$ to minimize $max(L_{M(p_i)}, L_{M(P_k)})$;
10: else if Bottleneck is B2, and the only routing path is $pe_i \rightarrow pe_j$:
11:     Merge partitions on $pe_i$ and $pe_j$ to hide communication;
12: else if Bottleneck is B3, and the link is $lk$:
13:     Obtain $pe_i \rightarrow pe_j$ passing $lk$ with the maximum data volume;
14:     Re-routing $pe_i \rightarrow pe_j$ to detour at $lk$;
```

Re-partition is developed to solve B1 and B2. For B1, the performance is dominated by the critical PE (with the maximum latency), which indicates that the corresponding partition is too large. We reduce the computation load by re-partitioning the involved layers (Line 1-9). For B2, there is only one routing path through the critical link, where inter-PE communication dominates the throughput. We merge corresponding partitions to avoid network communication (Line 10-11).

Re-routing will be employed to solve B3. Among multiple routing paths go through the critical link lk, we identify one with the maximum data volume (Line 12-14). We re-route this path by detouring packets when arriving at lk. On 2-D Mesh NoC, there are two squares contains lk, we select the one has less traffic for detouring, where the detoured path will go through other three links in the selected square except lk.

Finally, if performance cannot be improved by Algorithm 1, we will terminate alleviation process and directly employ the previous design. Otherwise, we iteratively conduct Algorithm 1 until it reaches line 4 or a preset iteration upper bound.

IV. Experiment

This section will report the experimental results on evaluating the efficiency of the proposed NANDS framework.

A. Experiment Setup

Datasets: Three image classification datasets are employed (CIFAR-10, CIFAR-100, and STL-10) to verify the efficacy of NANDS. During the search process, we only use the training images, and randomly select 10% of them to build a validated set. Test images are used to test accuracy of the resultant architectures. All images undergo the data augmentation, including upsampling, random cropping and random horizontal flip.

NAS Space: We use ResNet as the backbone to search network architectures. Search space is determined by three decision variables: R, N and C, the number of residual layers,

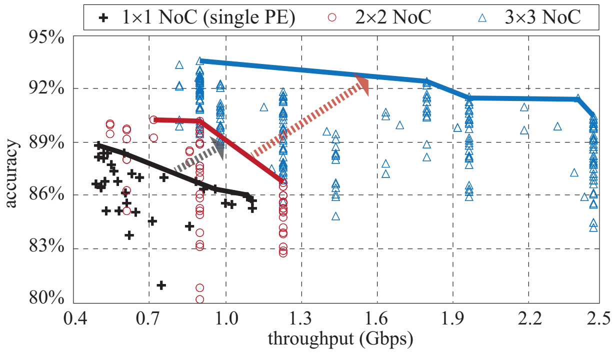


Fig. 6. Pareto frontiers of accuracy-throughput tradeoffs captured by NANDS on CIFAR-10 can be significantly pushed forward with increasing NoC size.

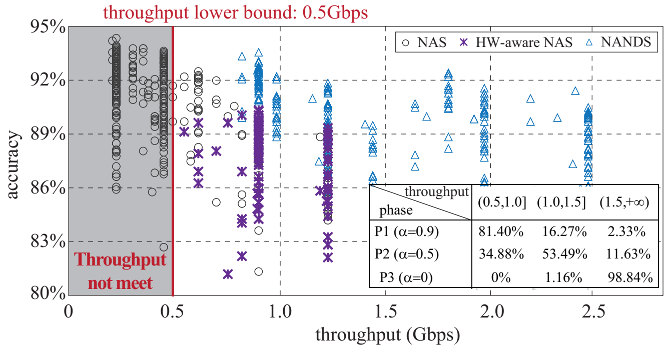
convolutional layers in one residual layer, and channels in each convolutional layer, respectively. We set different search ranges of decision variables in terms of datasets. For CIFAR-10, we explore R in range $[4, 5]$, N in range $[0, 2]$, and C in set $\{32, 64, 128, 256\}$. For CIFAR-100, we explore R in range $[4, 5]$, N in range $[0, 2]$, and C in set $\{64, 128, 192, 256, 512\}$. For STL-10, we explore R in range $[5, 6]$, N in range $[0, 3]$, and C in set $\{96, 128, 192, 256, 512\}$. We explore a larger network for STL-10, because the size of the images in STL-10 is larger ($96 \times 96 \times 3$ pixels) than that of CIFAR-10/100 ($32 \times 32 \times 3$ pixels). To train child networks, we set training parameters (e.g., learning rate, epoch) based on that in ResNet9 [19].

NDS Space: According to the size of neural networks, we employ 2×2 and 3×3 2-D Mesh NoCs. Experiments will be also conducted in the system with one PE for comparison. For target platforms, we adopt Zynq UltraScale XCZU9EG as PEs, which contains 2520 DSPs, and communication among FPGAs are carried out by Aurora IP core provided by Xilinx, which provides a bandwidth of 10 Gb/s between PEs. Accelerator design on each PE follows [18]. Finally, we employ roofline model to obtain computation and communication latency.

B. Comparison results

Scalability on NoC Sizes: Figure 6 compares the Pareto frontiers on accuracy-throughput tradeoff obtained by NANDS with three sizes of NoCs (different types of points) to demonstrate the scalability. For each platform, we obtain its Pareto frontiers by connecting the design points that are not inferior to any others in terms of both throughput and accuracy.

Results in Figure 6 clearly demonstrate that with the increasing NoC size, Pareto frontiers can be significantly pushed forward. Compared with single PE based NAS, the best accuracy can be improved from 88.39% to 90.68% (2×2 NoC) and 93.59% (3×3 NoC). In addition, for solutions with the maximum throughput, the accuracy and throughput for single PE platform are $\langle 88.39\%, 0.50Gbps \rangle$, which are improved to $\langle 90.68\%, 0.72Gbps \rangle$ for 2×2 NoC and $\langle 91.58\%, 2.40Gbps \rangle$ for 3×3 NoC. The main reason is that larger-size NoC provides more computation power, such that it can accommodate larger neural networks with higher accuracy to achieve the same or even higher throughput. This verified the scalability of our proposed approaches. Note that since 3×3 NoC has already provided sufficient hardware for the target NAS space, there is no need to employ larger NoCs. In the following texts, we base the experiments on 3×3 NoC.



throughput phase	(0.5,1.0]	(1.0,1.5]	(1.5,+∞)
P1 (α=0.9)	81.40%	16.27%	2.33%
P2 (α=0.5)	34.88%	53.49%	11.63%
P3 (α=0)	0%	1.16%	98.84%

Fig. 7. Results comparison of the design space exploration.

Design Space Exploration: In this set of experiment, we compare NANDS framework on CIFAR-10 with the original NAS framework [1], and the state-of-the-art HW-aware NAS [20]. Since NAS does not consider hardware, we collect the explored architectures, and apply the proposed NoC Explorer (the right part in Figure 5) to obtain throughput. For HW-aware NAS, it needs a determined hardware design to predict timing performance. To be fair, we employ the same partition, mapping and routing algorithms as that in Section III-C ❷. We set the throughput lower bound to be $0.5Gbps$ as the real-time constraint, indicating that designs with throughput less than $0.5Gbps$ are invalid (e.g., the left gray zone in Figure 7).

Figure 7 shows the exploration results, where x-axis and y-axis represent throughput and accuracy, respectively. The table in the bottom-right-corner reports the distributions of the explored child networks by NANDS in terms of throughput.

Comparisons among frameworks. First, NAS cannot guarantee timing performance. Although NAS can find solution with the highest accuracy (94.41%), its throughput is merely 0.22 Gbps, which violates timing requirements and is 4.00× lower than that of feasible solution from NANDS (e.g., the throughput of 0.9 Gbps and accuracy of 93.59%). HW-aware NAS and NANDS can meet throughput constraint of 0.5Gbps, but for a tighter constraint (e.g., $> 1.5Gbps$), only NANDS can provide valid solutions. This is because HW-aware NAS did not explore hardware design space. Second, for child networks with the same throughput, NANDS can achieve better accuracy than NAS and HW-aware NAS. For solutions with the throughput of 0.82 Gbps, the best accuracy of NANDS is 93.40%, which is much higher than 88.95% and 89.95% obtained by NAS and HW-aware NAS. This is because NANDS can benefit from exploring hardware design space to accommodate more complicated structures with the same throughput but higher accuracy.

Insights of multi-phase exploration. From table in Figure 7, we can draw another important conclusion: the proposed multi-phase scheme can guide controller to make a better tradeoff between accuracy and throughput. As described in Section III, we applied a multi-phase exploration by adjusting a regulation variable α . At the beginning, α is set to 0.9 for higher accuracy. The throughput of most architectures explored in P1 lies between 0.5 to 1 Gbps. Then, in P2, we adjust α to 0.5, and throughput of the architectures explored gradually shift to interval of 1.0 to 1.5 Gbps. Finally, in P3 with $\alpha = 0$, results mostly reside in high throughput range

TABLE I
COMPARISON OF THE SEARCH TIME BETWEEN PURE NDS, NAS, HW-AWARE NAS, AND NANDS, ON THREE COMMON DATASETS

Dataset	Spec.	Models	Arch. Property			Accuracy		Throughput			Elapsed Time	
	(Gbps)		Depth	Para. ($\times 10^6$)	MACs (GOP)	(%)	degr.	(Gbps)	Sat.	impr.	(minute)	impr.
CIFAR-10	0.50	NAS	9	0.89	0.70	94.41%	0.00%	0.22	×	baseline	1115	baseline
		HW-Aware NAS	8	0.19	0.05	90.95%	-3.46%	0.66	✓	2.94×	164	6.80×
		NANDS (Opt TP)	**8**	**0.20**	**0.06**	**91.58%**	**-2.83%**	**2.40**	✓	**10.66×**	**361**	**3.09×**
		NANDS (Opt Acc.)	**10**	**0.40**	**0.21**	**93.59%**	**-0.82%**	**0.90**	✓	**4.00×**		
CIFAR-100	0.45	NAS	12	1.04	1.02	76.58%	0.00%	0.22	×	baseline	1863	baseline
		HW-Aware NAS	8	0.19	0.07	71.43%	-5.15%	0.28^*	×	1.25×	246	7.57×
		NANDS (Opt TP)	**8**	**0.25**	**0.15**	**72.22%**	**-4.36%**	**0.90**	✓	4.00×	**594**	**3.14×**
		NANDS (Opt Acc.)	**12**	**0.63**	**0.46**	**75.58%**	**-1.00%**	**0.45**	✓	2.00×		
STL-10	0.6	NAS	11	2.95	2.13	76.45%	0.00%	0.45	×	baseline	2928	baseline
		HW-Aware NAS	12	1.70	0.50	74.25%	-2.20%	0.61	✓	1.25×	402	7.28×
		NANDS (Opt TP)	**11**	**2.02**	**1.02**	**75.83%**	**-0.62%**	1.07	✓	2.37×	**1059**	**2.76×**
		NANDS (Opt Acc.)	**13**	**2.65**	**1.45**	**76.45%**	**0.00%**	0.60	✓	1.32×		

"*": relax spec., HW-aware NAS cannot guarantee throughput of 0.45Gbps.

above 1.5 Gbps.

In conclusion, NANDS can guide controller to made better tradeoffs between accuracy and efficiency.

Comparisons on Additional Datasets: Table I reports the experimental results of NANDS and other approaches on three different datasets, including CIFAR-10, CIFAR-100, and STL-10. We report solutions by NANDS in the Pareto frontiers with the maximum throughput ("Opt TP") and the maximum accuracy ("OptAcc."). For NAS [1] and HW-aware NAS [20], we report the finally identified architectures.

Table I shows the consistent results with that in Figure 7: NANDS can make the better accuracy-throughput tradeoffs against state-of-the-art NAS frameworks. Specifically, when NAS generates solutions beyond the real-time constraint, NANDS can find valid solutions with little or no accuracy loss. In addition, NANDS achieves $3.09\times$, $3.14\times$, and $2.76\times$ speedup over NAS, respectively, on three different datasets. Compared with HW-aware NAS, NANDS takes more time in searching since HW-aware NAS does not thoroughly explore the NoC design space, which reduces the quality of results. On STL-10 datasets, NANDS (Opt TP) can achieve 42.99% higher throughput and 1.58% improvement on accuracy. In addition, for CIFAR-100, HW-aware NAS cannot find any feasible solutions to meet 0.45Gbps throughput bound, but our proposed NANDS can.

V. CONCLUSION

In this work, we focus on neural architecture and NoC-based hardware co-design. We propose NANDS framework to co-explore NAS space and NoC design search (NDS) space, which can maximize network accuracy and system throughput. A multi-phase manager is developed to hierarchically explore NAS and NDS spaces, which can guide NANDS to gradually converge to solutions with the best accuracy-throughput tradeoff. On top of it, techniques proposed to detect and alleviate timing performance bottleneck have effectively explored NDS to further improve throughput. Experimental results on CIFAR-10, CIFAR-100 and STL-10 verified the effectiveness of the proposed approach by achieving 42.99% higher throughput along with 1.58% higher accuracy than the state-of-the-art. Besides, there are cases where hardware-aware NAS cannot find any feasible solutions while NANDS can.

ACKNOWLEDGEMENT

This work is partially supported by National Science Foundation under Grant CCF-1820537 and CNS-1822099, and NAP M4082282 and SUG M4082087 from Nanyang Technological University, Singapore, and NSFC 61772094, China, and Edgecortix Inc.

REFERENCES

[1] Barret Zoph et al. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[2] Barret Zoph et al. Learning transferable architectures for scalable image recognition. In *In Proc. of CVPR*, pages 8697–8710, 2018.

[3] Bowen Baker et al. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016.

[4] Esteban Real et al. Large-scale evolution of image classifiers. *CoRR*, abs/1703.01041, 2017.

[5] Lingxi Xie et al. Genetic cnn. In *In Proc. of ICCV*, pages 1388–1397. IEEE, 2017.

[6] Weiwen Jiang et al. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. *arXiv preprint arXiv:1901.11211*, 2019.

[7] Weiwen Jiang et al. Heterogeneous fpga-based cost-optimal design for timing-constrained cnns. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2542–2554, 2018.

[8] Jeremy Fowers et al. A configurable cloud-scale dnn processor for real-time ai. In *In Proc. of ISCA*, pages 1–14, 2018.

[9] Eric Chung et al. Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro*, 38(2):8–20, 2018.

[10] Weiwen Jiang et al. Xfer: A novel design to achieve super-linear performance on multiple fpgas for real-time ai. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 305–305. ACM, 2019.

[11] Weiwen Jiang et al. Achieving super-linear speedup across multi-fpga for real-time dnn inference. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):67, 2019.

[12] Weiwen Jiang et al. Integrating memristors and cmos for better ai. *Nature Electronics*, 2(9):376–377, 2019.

[13] Wonje Choi et al. Hybrid network-on-chip architectures for accelerating deep learning kernels on heterogeneous manycore platforms. In *in Proc. of CASES*, page 13. ACM, 2016.

[14] Wonje Choi et al. On-chip communication network for efficient training of deep convolutional networks on heterogeneous manycore systems. *IEEE Transactions on Computers*, 67(5):672–686, 2018.

[15] Versal: The first adaptive compute acceleration platform (acap). Oct 2018.

[16] George Karypis et al. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998.

[17] N. Koziris et al. An efficient algorithm for the physical mapping of clustered task graphs onto multiprocessor architectures. In *In Proc. of EMPDP*, pages 406–413, 2000.

[18] Chen Zhang et al. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *In Proc. of FPGA*, pages 161–170. ACM, 2015.

[19] Chuan Li. https://lambdalabs.com/blog/resnet9-train-to-94-cifar10-accuracy-in-100-seconds. 2019.

[20] Bichen Wu et al. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *CoRR*, abs/1812.03443, 2018.