Enhancing Model Parallelism in Neural Architecture Search for Multidevice System

Cheng Fu and Huili Chen
University of California, San Diego

Zhenheng Yang
Facebook AI Research

Farinaz Koushanfar
University of California, San Diego

Yuandong Tian
Facebook AI Research

Jishen Zhao
University of California, San Diego

***Abstract*—Neural architecture search (NAS) finds favorable network topologies for better task performance. Existing hardware-aware NAS techniques only target to reduce inference latency on single CPU/GPU systems and the searched model can hardly be parallelized. To address this issue, we propose *ColocNAS*, the first *synchronization-aware*, *end-to-end* NAS framework that automates the design of parallelizable neural networks for multidevice systems while maintaining a high task accuracy. ColocNAS defines a *new search space* with elaborated connectivity to reduce device communication and synchronization. ColocNAS consists of three phases: 1) offline latency profiling that constructs a lookup table of inference latency of various networks for online runtime approximation; 2) differentiable latency-aware NAS that simultaneously minimizes inference latency and task error; and 3) reinforcement-learning-based device placement fine-tuning to further reduce the latency of the deployed model. Extensive evaluation corroborates ColocNAS's effectiveness to reduce inference latency while preserving task accuracy.**

Digital Object Identifier 10.1109/MM.2020.3004538
Date of publication 26 June 2020; date of current version 1 September 2020.

0272-1732 © 2020 IEEE
Published by the IEEE Computer Society

Deep neural networks (DNNs) are increasingly adopted in various fields due to their unprecedented performance. Enormous architecture-level advancements have been proposed to improve the performance and efficiency of DNN execution.[1] In the meantime, neural architecture search (NAS)[2,3] enables a new line of research that aims to design efficient DNN topology for target hardware platforms. Existing hardware friendly NAS techniques[4–6] target to identify efficient models on single CPU/GPU systems by introducing the notion of $FLOP_s$, model size, or predicted latency into the searching process. However, in modern real-time and cloud computing scenarios, multiple computation components co-exist and are shared by different tasks or users.

There are two types of parallelism when executing an neural network (NN) on a multiple-device platform: 1) model parallelism partitions operations into different parts and assign them onto different devices; and 2) data parallelism distributes the inference data onto multiple devices and duplicate the entire model on each device. However, data parallelism has several limitations. 1) Data parallelism can increase the throughput of the application but not the end-to-end inference latency, thus it is not suitable for real-time applications. 2) Data parallelism may incur severe communication overhead during the setup stage since the inputs need to be distributed to each available device. 3) Data parallelism is infeasible when the model size is too large to fit on a single device with constrained memory. With the increasing model size of SOTA NNs, the fact in 3) is becoming more stringent, especially for edge devices with very limited memory. In contrast, emerging multidevice computing systems and device-to-device communication techniques (e.g., 5G communication, NVLINKs, and PCIe Gen 4) can benefit model parallelism. Because our goal is to reduce application latency, model-level parallelism is a promising candidate.

The designed search spaces of conventional NAS can be classified into two main categories (see Figure 2). 1) *Layer-wised search.* Existing latency-oriented model searching methods[4,5] explore model architectures that are analogous to *chain-like* wired mobilenetV2 with the building block shown in Figure 2(a). This topology is hard to be paralleled across multiple computation components due to data dependence. 2) *Cell-structure search.* Another line of methods[2,7] propose to search a building block, called *cell*, on small tasks (e.g., CIFAR-10) and transfer the searched cell for training on a large task (e.g., ImageNet). The model in this space have certain levels of parallelism due to the concurrent execution of different blocks inside each cell. However, the concatenation (*concat*) operations between cells prevents further parallelism since they synchronize the execution and increase communication overhead. To overcome these constraints, ColocNAS is motivated to automate the design of suitable NN architectures for multi-device platforms to achieve low inference latency and high task accuracy.

Designing an efficient model to achieve model parallelism across devices is a nontrivial task. It is challenging because of the following. C1) The searched architecture needs to keep a high accuracy, while reducing its runtime overhead. C2) While NNs with more complicated wiring patterns have higher chance for model parallelism, predicting their execution time is challenging for latency-aware NAS. C3) The actual latency of a specific network architecture in a multidevice

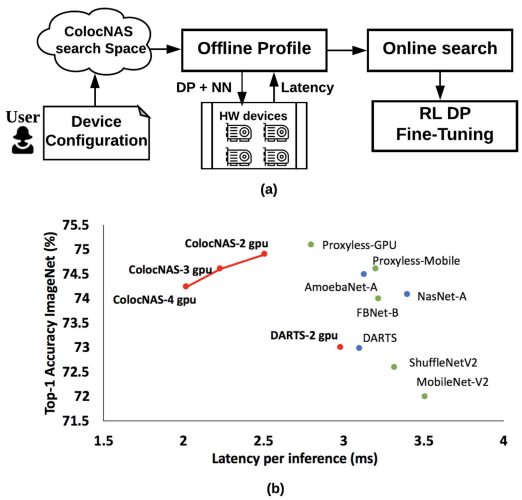

Figure 1. (a) ColocNAS design overview and (b) roadmap between accuracy and latency of NAS methods. DP+NN refers to device placement (DP) policy and neural network (NN) architecture for deploying. ColocNAS significantly outperforms other NAS methods, when running on multiple devices.

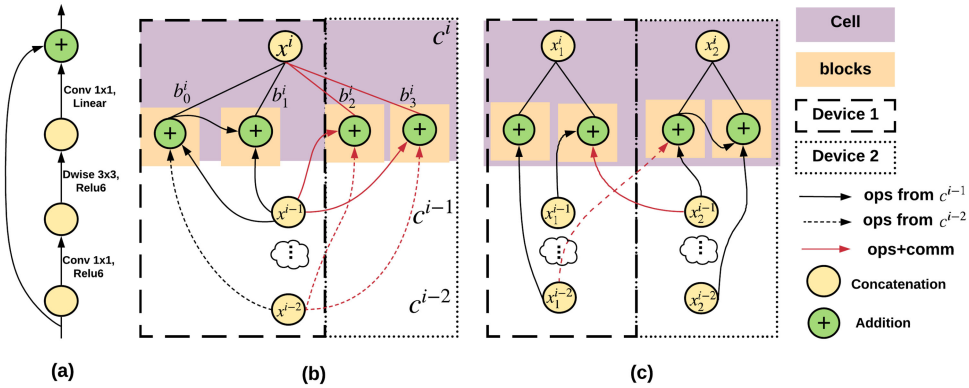

Figure 2. Example of basic building blocks in three types of search space. (a) Layer-wise searched model. (b) Cell-based searched model. (c) New search space proposed in ColocNAS. The yellow circle refers to the concatenation node. b_n^i denotes the nth block in ith cell (i.e., layer).

setting depends on the device placement policy, which is unknown ahead of time.

To tackle these challenges, we propose *ColocNAS*, a differentiable NAS framework that effectively searches network architectures for the given hardware environment. ColocNAS resolves all the aforementioned challenges (C1–C3) using the following solutions [as shown in Figure 2 (a)]. S1) ColocNAS designs a new search space with less synchronization and communication overhead by exploring elaborate connectivity patterns in the NN. S2) Based on the observation that similar computation graphs yield closer latency, ColocNAS leverages offline latency profiling and k-nearest neighbors (kNN) with graph similarity to predict latency for searched model in the online phase. S3) ColocNAS employs uniform workload distribution as the *expert placement policy* to facilitate offline latency profiling and to guide online latency-aware NAS. This policy leverages the intrinsic structure features of the DNN in the new search space. After online NAS, the placement of the searched model is further fine-tuned using a reinforcement learning (RL)-based algorithm to reduce its runtime on the target platform.

Our framework offers a holistic solution to designing efficient and accurate NNs on multidevice systems. Extensive experiments show that ColocNAS reduces inference latency by a large margin, while achieving a competitive accuracy and latency as shown in Figure 1(b). Our work sheds light on a new dimension (device-level parallelism) of hardware-aware NAS algorithms.

ColocNAS DESIGN

ColocNAS uses *differentiable neural architecture search* by combining two gradient-based methods[4,7] to solve the problem of topology design. We formulate the neural architecture search problem as a nonconvex optimization problem as shown in

$$\min_{\alpha \in A} \min_{w_\alpha} L(\alpha, w_\alpha). \tag{1}$$

A promising architecture yields small latency and high task accuracy. Here, A is a new search space proposed in ColocNAS, $\alpha \in A$ is a set of continuous variables that specify a possible architecture, w_α is the weight parameter of the network. L is the loss function that penalizes both accuracy degradation as well as the increase of inference latency. The design flow of ColocNAS is detailed in Algorithm 1.

Algorithm 1. ColocNAS Design Flow.

INPUT: Prototyping Hardware Devices (τ)**; Device Number** $(\mathcal{N}_\tau)$**; ArchTable Size** (N_{arch})**; Cell Number** N_c**; Possible Operations** (Ops)**.**
OUTPUT: Fine-tuned Device Placement Policy P_{RL}**; Model** M_{out}**.**
1: **Offline Profile:**
$M_i \leftarrow Random_Generate(\mathcal{N}_\tau, Ops, N_\tau, N_c)$
$P_i \leftarrow Expert_Placement_Policy(M_i)$
$Arch_Table \leftarrow Hardware_Profile(M_i, P_i, \tau)$
for $i = 1, ..., N_{\text{arch}}$
2: **Online Searching and Training:**
$M_{\text{arch}} \leftarrow Searching(Arch_Table, Ops, N_\tau, N_c)$
$M_{out} \leftarrow Training_Model(M_{\text{arch}})$
3: **RL Fine-Tuning:**
$P_{RL} \leftarrow RL_FineTuning(M_{out}, \tau)$

Search Space

Previous works search models with layer-wise or cell-based structures as shown in Figure 2 (a) and (b). The layer-wise structure is suitable for a *single* CPU device as the identified model topology is very regularized and its execution is fast due to data locality. However, the resulting chain-like model is not suitable when multiple devices are available for parallelism, as each layer can only start execution when the outputs of all its previous layers are computed. For the cell structure search space, its evaluation can be paralleled. Yet, the *concatenation* at the end of each cell node works as a *synchronization* unit that collects all the results inside the cell blocks before starting the computation in the next cell. This *synchronization* incurs severe communication overhead (the red arrow in Figure 2) and hinders computation parallelism. As such, we proposed a new search space that reduces the participants of the synchronization unit in order to minimize the delay of computation as shown in Figure 2(c). ColocNAS uses the same definition of search blocks and cells while introducing a new block connectivity pattern by "duplicating" the concatenation nodes.

The traditional cell consists of an ordered sequence of N blocks where each block has two input edges (i, j) as shown in Figure 2(b). An edge can be selected from two sources. i) previous blocks in the current cell. ii) The output of the previous two cells. Each edge is associated with an operation that is determined after the search $[o(i, j)]$ is performed. Assuming the inputs of the block are x^1 and x^2, the output of the block is computed as follows:

$$b_j = \sum_{i=1}^{2} o^{(i,j)}(x^i). \tag{2}$$

The intermediate results from the N blocks are concatenated together to form a single cell output x^i. Since we identify that this concatenation node is the bottleneck for achieving device-level parallelism, the new search space is defined as follows.

Instead of reducing towards a single concatenation node in each cell, the blocks of the ith cell will be evenly connected to C different concatenation nodes, resulting in outputs $x^i = [x_1^i, x_2^i, ..., x_c^i]^T$. For the nth block in the ith cell, we define *block connectivity* parameters β_n^{i-1} as the probability over each possible connection between the current block to any concatenation node in $(i-1)$th layer. It will be used to compute the weighted sum input $\overline{x_n^{i-1}}$ from previous layer $i-1$ as

$$\begin{aligned}\overline{x_n^{i-1}} &= \mathrm{softmax}(\beta_n^{i-1}) \cdot x^{i-1} \\ &= \sum_{c=1}^{C} \frac{\exp(\beta_{n,c}^{i-1})}{\sum_{c'=1}^{C} \exp(\beta_{n,c'}^{i-1})} x_c^{i-1}.\end{aligned} \tag{3}$$

Here, β_n^{i-1} is a vector of size 1-by-C. $\overline{x_n^{i-2}}$ is computed from β_n^{i-2} using the same method as (3).

Note that ColocNAS explores optimal network architectures by solving the bilevel optimization problem defined in (1). The optimization variable $\alpha = \{\beta, \mathrm{ops}\}$ where β is the block connectivity parameter and the softmax of ops is the probability vectors over the predefined set of DL operations for every possible connection (the candidate operations are the same as given by Cai *et al.*[5]). At the end of the search, we choose the connectivity of the nth block in ith cell from the previous two cells as $x_c^{i-1} = \arg\max_c \beta_n^{i-1}$ and $x_c^{i-2} = \arg\max_c \beta_n^{i-2}$.

Through comparison experiments, we find that by using Gumbel softmax[4] over β_c^i in (3) can improve both the accuracy and latency of the searched model compared to the softmax function.

Given $\overline{x_n^{i-1}}, \overline{x_n^{i-2}}$ as the computed input from the previous two cells for the ith cell, the output of the each blocks b_i^n would be the weighted sum over ops[7] during architecture search.

Offline Hardware Bounded Latency Profiling

ColocNAS integrates a latency-aware, gradient-based NAS for the target devices into the searching process to make the low latency model preferable. Recall that the optimization variable α in our search space is a set of probabilistic variables, thus the inference latency given a specific choice of α is also a random variable lat. We use the *expectation* value of the latency variable to assess the quality of the architecture choice α. However, computing the latency expectation over a architecture probability $\mathbb{E}_{\mathrm{lat}}(p_\alpha)$ on the given devices is challenging because of the following. 1) Measuring the real latency value of each architecture during the searching process

is infeasible due to the prohibitive latency cost. 2) The latency of a complex graph is hard to predict while considering the delay incurred by data movement. 3) The latency value of an architecture choice is highly dependent on the placement policy, which makes it hard to compare the optimal latency values of different network topologies. To resolve the above problems, the first stage of ColocNAS is an *offline latency profiling* step that measures the inference time of diverse network architectures on the target devices.

Besides, to disentangle the complexity of model placement on the given hardware, we use a predefined *expert placement policy* for all searched architectures. In particular, our expert placement policy uniformly distributes each concatenation node and the associated operations onto all existing devices. By doing so, we reduce the overhead of synchronization and the cross-device communication compared to the traditional cell-based search space as shown in Figure 2(c).

It is also intractable to profile the latency values of all the architectures in the entire search space ($\sim 10^{20}$ models). Leveraging the observation that similar NNs yield closer runtime latency, we propose a method to *approximate* the latency of a complicate wired NN using *kNN*. We randomly generate architectures and profile their latency values on the target hardware using *expert placement policy* to obtain a lookup table $ArchTable$. The distance between two NNs can be measured by using *graph edit distance*[8]. To approximate the latency expectation over an architecture parameter, we first convert α into the corresponding probability variable p_α which includes the following two parts:

$$p_{ops} = \text{Softmax}(ops) \tag{4}$$

$$p_\beta = \text{GumbelSoftmax}(\beta). \tag{5}$$

Given a specific choice of the probability parameter $p_\alpha = \{p_\beta, p_{\text{ops}}\}$, we sample N neural network architectures $\text{arch}_1, \text{arch}_2, ..., \text{arch}_N$. The expectation value of the latency variable is then approximated as

$$E_{\text{lat}}(p_\alpha) \approx \frac{1}{N}\sum_{n=1}^{N} k\text{NN}(\text{arch}_n, \text{ArchTable}) \tag{6}$$

where the $kNN(\text{cot})$ function returns the average latency of the top-k closest architecture latencies for input arch_n.

Online Latency-Aware Architecture Searching

ColocNAS aims to find a network architecture with low latency and high task accuracy. As such, we define the loss function of ColocNAS's online searching phase as follows:

$$L(\alpha, w_a) = CE(\alpha, w_a) + \lambda_1 L_{\text{lat}} \tag{7}$$

$$L_{\text{lat}} = \exp(-||p_\alpha - \widehat{p_\alpha}||_2) \cdot E_{\text{lat}}(p_\alpha). \tag{8}$$

Here, $CE(\alpha, w_a)$ is the cross-entropy loss given the architecture parameter α and the weights w_a. λ_1 is the scaling factor within the range [0,1] that controls the tradeoff between accuracy and latency. $E_{\text{lat}}(p_\alpha)$ is the expected latency of the specific architecture parameter obtained from (6). For the latency term, we add a regularization term $\exp(-||p_\alpha - \widehat{p_\alpha}||_2)$ where $\widehat{p_\alpha} = \frac{1}{N}\sum_{n=1}^{N} \text{arch}_n$ to make the latency term differentiable to the architecture parameters α. If the sample mean value of the architecture probability $\widehat{p_\alpha}$ is far from the true one p_α, the corresponding latency term will be weighted less. Note that after the model is searched, we train the model from scratch to obtain the final accuracy.

RL-Based Device Placement Fine Tuning

Once the model is searched and trained, the model can be directly deployed on to the target hardware systems using the expert placement policy that uniformly distributes the workload to each available device. However, this heuristic-based device placement policy may not be optimal considering the heterogeneous property of the underlying computing platforms. To further optimize the latency of the searched architecture Γ, the last step of the ColocNAS leverages policy gradient for device placement on the given hardware. We apply the RL-based method proposed by Mirhoseini *et al.*[9] where the policy network is an attentional autoencoder that takes the NN graph as input. Each input in the encoding sequence is the concatenated embedding of operation type, connectivity and output shape of each block in the graph. The decoder sequentially generates the placement policy P.

The goal is to learn the policy $\pi(P|\Gamma;\theta)$ to minimize the objective $J(\theta) = \mathbb{E}_{P\sim\pi(P|\Gamma;\theta)}[R(P)|\Gamma]$. Here, $R(P)$ is the expectation of the reward (execution time).

The policy gradients are computed via the REINFORCE equation[10] as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{P\sim\pi(P|\Gamma;\theta)}[R(P)\cdot\nabla_\theta \log_p(P|\Gamma;\theta)]. \quad (9)$$

By sampling K placements following the placement policy probability $P\sim\pi(\cdot|\Gamma;\theta)$, the expectation of reward $R(P)$ can be estimated and the policy gradient is computed as

$$\nabla_\theta J(\theta) \approx \frac{1}{K}\sum_{i=1}^{K}(R(P_i)-B)\cdot\nabla_\theta \log_p(P|\Gamma;\theta). \quad (10)$$

Here, B is the mean value of the rewards computed from the K sampled placements. ColocNAS leverages the intrinsic structure of the architecture in the new search space and performs device placement only for blocks and concatenation nodes [see Figure 2 (c)]. As a result, the training of the policy network $\pi(P|\Gamma;\theta)$ converges much faster compared to the original one by Mirhoseini *et al.*[9]

EXPERIMENTS

In this section, we demonstrate ColocNAS's effectiveness of model-level parallelism and accurate classification performance on CIFAR-10 and ImageNet tasks compared to the state-of-the-art NAS methods.

Experimental Setup

We run ColocNAS on different device configurations and evaluate the searched architectures in terms of its test accuracy and inference latency.

Searching phase setup. ColocNAS searches two types of cells, namely, a *normal cell* and a *reduction cell*. We put the reduction cell at $1/3$ and $2/3$ location of the neural architectures. After each reduction cell, we double the number of output channels in the network. The operations searched in the reduction cell has stride$=2$.

The model that uses on the searching phase has eight cells. Larger networks can be built by stacking multiple normal cells in between the reduction cells. We alternatively update the two variables (α and w_α) to solve the bilevel optimization problem in (1). In particular, we use second-order approximations to update the architecture parameter α. All possible operations in the search space and relevant searching details are the same as DARTS.[7]

Hardware platforms and corresponding search space. To prove the generality of ColocNAS on different platforms, we test our method on two types GPUs and six different device configurations. 1) 2/3/4 Tesla K80 GPUs with Gen3 PCIe device connection. 2) 2/3/4 Tesla V100 GPUs with NVlink connections. The device placement algorithm and latency profiling for cell structure are implemented in Tensorflow v1.14. For each device setting, we set the number of concatenation nodes to be the same as the number of devices to facilitate the *expert placement policy*. Since the device configuration determines the search space as shown in Figure 2, we name the search space with 2/3/4 GPUs as *ColocNAS-SP-2*/*ColocNAS-SP-3*/*ColocNAS-SP-4*. We denote the searched architectures as *ColocNAS-b4c2*/*ColocNAS-b6c3*/*ColocNAS-b4c4* which have 4/6/4 number of blocks and 2/3/4 concatenation nodes in each cell for hardware settings with 2/3/4 GPUs, respectively.

Latency Prediction Results

Recall that ColocNAS applies the kNN method to estimate end-to-end latency. We sample 10 000 architectures that are mapped using the expert placement policy to available GPUs on both CIFAR-10 ($32\times32\times3$) and ImageNet ($224\times224\times3$) datasets. We set the total number of cells to be 14 and 20 for ImageNet and CIFAR-10, respectively. For CIFAR-10 and ImageNet for evaluation, we added one/two initial stem cells to downscale the raw images, respectively.

Figure 3 shows the effectiveness of our kNN-based latency predictor. The latency is profiled in search space *ColocNAS-SP-4* with ImageNet input (batch size is 32) on 4 Tesla K80 GPUs. We use 5% of the profiled data for testing. The profiling takes 2.5 days on 8 GPUs. The RMSE error is 0.22 ms on the test data, which is adequate for the latency prediction of complicated graphs. Note that this offline profiling is a one-time profiling process, that can be used to search different NN architectures when tuning the λ_1 parameters in (7).

Results of Latency-Aware NAS

Cifar-10 Benchmark After the architecture is searched, we train the weight parameters of the resulting network for 1500 epochs using a batch size of 96 and Adam optimizer. As shown in Table 1, ColocNAS maintains a competitive test error rate (2.36% on average) in different hardware settings. In the meantime, ColocNAS reduces the inference latency by a large margin (1.24×/1.34×/1.50× faster on 2/3/4 Tesla K80 GPUs compared to DARTS 2-GPU) on multiple-devices thanks to the parallelizable search space and differentiable latency guidance. ColocNAS uses a search space with elaborated connectivity and an online kNN-based latency lookup, thus the search cost is higher than DARTS (see Table 1). Empirical results show that ColocNAS is still among the fastest NAS techniques due to our gradient-based method and CIFAR-10 proxy.

ImageNet Results Using the normal and reduction cell found by the latency-aware searching step, we stack 14 cells to build an NN for ImageNet evaluation. The network is trained for 360 epochs with power cosine learning rate and SGD optimizer with nesterov-momentum.

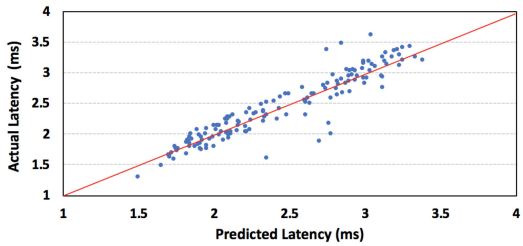

Figure 3. Effectiveness of kNN-based end-to-end latency predictor on the ImageNet data with 4 Tesla K80 GPUs. We determine $K = 5$ using cross-validation, which yields RMSE 0.22 ms on the test set.

The comparison results are shown in Table 2. Compared to the state-of-the-art NAS methods, our model shows a competitive error rate on the test dataset (25.47% on average). Furthermore, our model outperforms all other state-of-the-art

Table 1. Performance comparison between ColocNAS and the state-of-the-art NAS methods classification. The test error and inference latency on CIFAR-10 benchmarks are shown here. For cell-based method, the latency is tested by reconstructing the model using open-source code from DARTS. [7] "-" indicates the model for CIFAR-10 is not publicly available. We use a batch size of 32 for all latency evaluation. The latency of ColocNAS is tested on 2/3/4 Tesla K80 GPUs after RL fine-tuning and "DARTS 2-GPU" are tested on 2 GPUs using RL placement. Other baselines are tested on a single Tesla K80 GPU.

Model	Search Space	Search Method	Search Cost (GPU hours)	# Params	Test Error (%)	Latency (per image)	Normalized Reduction (%)
DenseNet-BC	-	manual	-	25.6M	3.46	14.8 ms	0.0
NAONet	cell	gradient	4.8K	10.6M	3.18	8.4 ms	-43.2
PNAS	cell	SMBO	6K	3.2M	3.41	3.14 ms	-78.8
Amoeba-A [11]	cell	evolution	76K	3.2M	3.12	3.10 ms	-79.1
Amoeba-B [11]	cell	evolution	76K	2.8M	2.55	3.04 ms	-79.4
DARTS (2nd) [7]	cell	gradient	24	3.2M	2.76	3.12 ms	-78.9
NASNet-A [2]	cell	RL	48K	3.3M	2.65	3.24 ms	-78.1
DARTS 2-gpu [7]	cell	gradient	24	3.2M	2.76	3.01 ms	-79.7
ColocNAS-b4c2	new space	gradient	41	3.4M	**2.31**	**2.43 ms**	**-83.6**
ColocNAS-b6c3	new space	gradient	49	3.5M	**2.35**	**2.24 ms**	**-84.9**
ColocNAS-b4c4	new space	gradient	51	3.4M	**2.42**	**2.01 ms**	**-86.4**

Table 2. Transferability classification error on ImageNet benchmarks. For NAS in layer-wise search space, the latency is tested by using open-source evaluation code from the papers. "-" indicates the number is not shown in the original paper or the model is not publicly available. The latency results in the table are tested on Tesla K80 GPUs. The RL setting for the baseline is evaluated on 2-GPU. With more GPUs, the baseline would yield same latency as the models can hardly be parallelized across devices.

Model	Search space	Search method	Search cost (GPU hours)	# Params	# FLOPs	Test accuracy (Top-1 %)	Latency (-RL/+RL)	Normalized reduction (%)
MobileNetV2	-	-	-	3.4M	300M	72.0	3.51/3.51 ms	0.0
ShuffleNetV2 (1.5)	-	-	-	3.5M	299M	72.6	3.32/3.32 ms	-5.4
Amoeba-A [11]	Cell	Evolution	756K	5.1M	555M	74.5	3.13/ 3.02 ms	-14.0
NASNet-A [2]	Cell	RL	48K	5.3M	564M	74.0	3.22/ 3.10 ms	-11.7
DARTS (2nd) [7]	Cell	Gradient	24	4.9M	595M	73.1	3.09/2.98 ms	-15.1
MnasNet-65 [6]	Layer wise	RL	91K	3.6M	270M	73.0	-	-
MnasNet [6]	Layer wise	RL	91K	4.2M	317M	74.0	-	-
FBNet-A [4]	Layer wise	Gradient	216	4.3M	249M	73.0	2.93/2.93 ms	-16.5
FBNet-B [4]	Layer wise	Gradient	216	4.5M	295M	74.1	3.41/3.41 ms	- 2.8
ProxylessNAS-mobile [5]	Layer wise	Gradient	200	6.9M	-	**74.6**	3.95/3.95 ms	+12.5
ProxylessNAS-GPU [5]	Layer wise	Gradient	200	7.1M	-	**75.1**	2.80/2.80 ms	-20.2
ColocNAS-b4c2	New space	Gradient	41	4.7M	546M	**74.9**	**2.67/2.51 ms**	**-28.5**
ColocNAS-b6c3	New space	Gradient	49	4.8M	572M	**74.6**	**2.41/2.23 ms**	**-36.5**
ColocNAS-b4c4	New space	Gradient	51	4.8M	566M	**74.1**	**2.18/2.07 ms**	**-41.0**

NAS methods in terms of the inference latency when deploying in a multiple-device hardware environment. With 2/3/4 GPU settings, ColocNAS achieves 1.57×/1.77×/1.91× execution speedup compared to the ProxylessNAS-mobile on Tesla K80. For Tesla V100 GPUs, ColocNAS yields a similar speedup compared to DARTS and ProxylessNAS. Applying the RL placement, we directly tested models searched for Tesla K80 onto the new hardware systems and the latency per image for ColocNAS-b4c2/b6c3/b4c4 are 0.66/0.61/0.54 ms on 2/3/4 Tesla V100 GPUs, which are 1.12×/1.21×/1.35× faster than DARTS (0.74 ms) and 1.28×/1.39×/1.57× faster than ProxylessNAS-mobile (0.85 ms). This corroborates that ColocNAS preserves a high task accuracy while reducing the latency overhead. We also notice that if without the guidance of the hardware-bounded latency estimation [$\lambda_1 = 0$ in (7)], the latency incurred by ColocNAS will increase by +0.23 ms (ColocNAS-b4c4) on Tesla K80 GPUs. Based on our observation, the latency regularization term encourages computation-efficient operations on the timing critical paths in the model. In addition, it explicitly helps to reduce the number of connectivity across blocks compared to model searching without latency guidance.

Performance Discussion One can observe the trade-off between accuracy and latency from Table 1 and 2. ColocNAS aims to achieve hardware-aware NAS for fast model inference by reducing the communication overhead between multiple computing platforms. However, the reduction in communication would incur accuracy degradation since the information exchange between sub-graphs for each device is reduced.

In this article, we propose ColocNAS, an end-to-end, differentiable neural architecture searching framework that is aware of the hardware-bounded latency.

Tables 1 and 2 show that the latency improvement of ColocNAS is sublinear with the number of computing devices. This is caused by the nonnegligible communication overhead between devices and system control overhead.

CONCLUSION

In this article, we propose ColocNAS, an end-to-end, differentiable neural architecture searching framework that is aware of the hardware-bounded latency. ColocNAS introduces an innovative parallelizable search space that reduces synchronization/device communication for model parallelism. For the first time, ColocNAS is able to maximize device utilization of multiple computing platforms by decoupling the task of hardware-aware NAS into three steps: offline hardware latency profiling, online latency-aware searching, and RL-based device placement fine-tuning. As a result, ColocNAS automates the design of neural architectures that achieve high task accuracy and low runtime latency for a given hardware setting. We perform extensive experiments and corroborate that ColocNAS reduces the inference latency by a large margin in a multidevice environment while maintaining a competitive task accuracy compared to the state-of-the-art hardware-aware NAS methods.

ACKNOWLEDGMENTS

This work was supported by NSF 1829525 and by the SRC/DARPA Center for Research on Intelligent Storage and Processing-in-Memory.

REFERENCES

1. V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
2. B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 8697–8710.
3. B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=r1Ue8Hcxg
4. B. Wu *et al.*, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 10 734–10 742.
5. H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://arxiv.org/pdf/1812.00332.pdf
6. M. Tan *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 2820–2828.
7. H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX
8. X. Gao, B. Xiao, D. Tao, and X. Li, "A survey of graph edit distance," *Pattern Anal. Appl.*, vol. 13, no. 1, pp. 113–129, 2010.
9. A. Mirhoseini *et al.*, "Device placement optimization with reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.* 2017, vol. 70, pp. 2430–2439, [Online]. Available: JMLR.org.
10. R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3/4, pp. 229–256, 1992.
11. E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, pp. 4780–4789.

Cheng Fu is currently working toward the Ph.D. degree with the University of California, San Diego. His research interests include efficient ML algorithms for hardware and machine learning for binary analysis. His advisor is Professor Jishen Zhao. He interned at Facebook AI research and worked on automating machine learning project. Contact him at cfu@eng.ucsd.edu.

Huili Chen is currently working toward the Ph.D. degree with the University of California, San Diego. Her research interests span diverse aspects of machine learning (ML) systems, including intellectual property (IP) protection, adversarial attack detection, and application of advanced ML algorithms to solve long-standing problems in other domains. Contact her at huc044@ucsd.edu.

Zhenheng Yang is currently a Research Scientist with Facebook AI. His research interests include computer vision and machine learning, specifically weakly supervised learning, 3-D perception, and activity reasoning. Yang received the Ph.D. degree from the University of Southern California. Contact him at zhenhengy@gmail.com.

Farinaz Koushanfar is currently a Professor and Henry Booker Faculty Scholar with the Department of Electrical and Computer Engineering, University of California, San Diego, where she directs the Adaptive Computing and Embedded Systems Lab. She is the Co-Founder and Co-Director of the University of California, San Diego, Center for Machine-Integrated Computing and Security. Koushanfar received the Ph.D. degree from the University of California, Berkeley. She is a Fellow of the Kavli Foundation Frontiers of the National Academy of Engineering. Contact her at farinaz@ucsd.edu.

Yuandong Tian is currently a Research Scientist and Manager with Facebook AI Research. His research interests include theory and practice of deep learning, sequential decision making, and computer vision. Tian received the Ph.D. degree from the Robotics Institute, Carnegie Mellon University. Contact him at yuandong@fb.com.

Jishen Zhao is currently an Assistant Professor with the Computer Science and Engineering Department, University of California, San Diego. Her research spans and stretches the boundary between computer architecture and system software, machine learning, and system codesign. Zhao received the Ph.D. degree in computer science and engineering from the Pennsylvania State University. Contact her at jzhao@eng.ucsd.edu.

IEEE Security & Privacy magazine provides articles with both a practical and research bent by the top thinkers in the field.

- stay current on the latest security tools and theories and gain invaluable practical and research knowledge,
- learn more about the latest techniques and cutting-edge technology, and
- discover case studies, tutorials, columns, and in-depth interviews and podcasts for the information security industry.

computer.org/security