NAF: Deeper Network/Accelerator Co-Exploration for Customizing CNNs on FPGA

Wenqi Lou, Jiaming Qian, Lei Gong*, Xuan Wang, Chao Wang*, Xuehai Zhou
University of Science and Technology of China, Hefei, China
{*louwenqi, qj0387, wgg*}*@mail.ustc.edu.cn* {*leigong0203, cswang, xhzhou*}*@ustc.edu.cn*

***Abstract*—Recently, algorithm and hardware co-design for neural networks (NNs) has become the key to obtaining high-quality solutions. However, prior works lack consideration of the underlying hardware and thus suffer from a severely unbalanced neural architecture and hardware architecture search (NA-HAS) space on FPGAs, failing to unleash the performance potential. Nevertheless, a deeper joint search leads to a larger (multiplicative) search space, highly challenging the search. To this end, we propose an efficient differentiable search framework NAF, which jointly searches the networks (e.g., operations and bitwidths) and accelerators (e.g., heterogeneous multicores and mappings) under a balanced NA-HAS space. Concretely, we design a coarse-grained hardware-friendly quantization algorithm and integrate it at a block granularity into the co-search process. Meanwhile, we design a highly optimized block processing unit (BPU) with key dataflow configurable. Afterward, a dynamic hardware generation algorithm based on modeling and heuristic rules is designed to perform the critical HAS and fast generate hardware feedback. Experimental results show that compared with the previous state-of-the-art (SOTA) co-design works, NAF improves the throughput by 1.99×∼6.84× on Xilinx ZCU102 and energy efficiency by 17%∼88% under similar accuracy on the ImageNet dataset.**

I. Introduction

Deep neural networks (DNNs) have drastically evolved in recent years to make breakthroughs in numerous computer vision tasks. However, the high computational complexity and memory requirements of DNNs often cause performance and energy efficiency issues, especially in resource-constrained devices. Hence, extensive research efforts have been devoted to tackling this problem. On the algorithm level, techniques such as model compression [1] and neural architecture search (NAS) [2] emerge to optimize the model size. On the hardware level, various dedicated accelerators [3]–[5] have been designed with dedicated micro-architectures and mapping strategies to improve deployment efficiency. Nevertheless, algorithms and accelerators are not independent, and optimizing one side alone could often harm another, yielding suboptimal solutions [6].

To address the sub-optimality, algorithm designers and hardware developers start exploring opportunities for joint optimization to achieve better specialization and acceleration. Due to the customizability and reconfigurability, FPGAs have become the ideal platform to quickly provide optimized support for different NNs in the co-design process. Nonetheless, previous network/accelerator co-search efforts have not fully considered the customizability of FPGAs [7]–[9], thereby failing to exploit the potential. It is reflected in two aspects: (i) Quantization as a *de facto* step in FPGA deployment, where the design of the quantizer and the choice of bitwidth affect both accuracy and hardware efficiency, has not been consistently considered [10]. (ii) Since the computational diversity between layers still exists in current NAS methods [11], only searching for the architectural sizing of accelerators while overlooking a vital HAS (e.g., multicores, dataflow, and mappings) inevitably leads to dynamic hardware underutilization issues [12], [13]. However, a comprehensive search that includes the above factors will face the following challenges. First, the exponentially increasing search space makes the optimized solution more sparse, and how to balance NA-HAS space involves a compromise between search cost and quality. Second, the quantization strategy and quantizer will significantly influence the NA-HAS space and the underlying hardware design difficulty. Third, a vast and complex HAS space makes the setup cost of the latency *lookup table* or *predictor* methods prohibitive and also makes the hardware feedback delay of prior analytical hardware generation algorithms unacceptable. To tackle these challenges, we propose a deeper **N**etwork/**A**ccelerator co-exploration framework for **F**PGAs, named **NAF**, which jointly searches the network and accelerator architecture to maximize accuracy and hardware efficiency. The main contributions of this work are:

- We design a hardware-friendly quantization algorithm and a dedicated, configurable hardware basic unit (BPU) to provide accuracy and performance guarantees. Then, a block-grained bitwidth search and BPU-grained hardware search are adopted to facilitate the NA-HAS process.
- We perform accurate performance and resource modeling to the accelerator and efficiently tailor the mapping space based on heuristic rules. Compared to [12], our genetic algorithm-based hardware generation algorithm yields similar performance but is 9.6× faster.
- Experimental results show that NAF efficiently conducts an extensive yet balanced NA-HAS. The searched CNN-accelerator pair improves the throughput by 1.99×∼6.84× and energy efficiency by 17%∼88%, with similar accuracy, compared to the SOTA co-design works.

II. Related Work

A. Neural Architecture Search

Neural architecture search (NAS) has recently flourished to automate the design of top-performing DNNs. Nevertheless, early NAS works [2] employ reinforcement learning (RL) as the search engine, suffering from prohibitive search costs (e.g., 83 GPU-days). To solve this problem, [14] formulates the NAS problem in a continuously differentiable manner, significantly reducing the search time (several GPU-days). It directly searches the supercell's generic structure and then repeatedly stacks the searched supercell across all the layers to construct the final network. Subsequently, a supernet structure

*Corresponding authors: Lei Gong and Chao Wang

978-3-9819263-7-8/DATE23/© 2023 EDAA

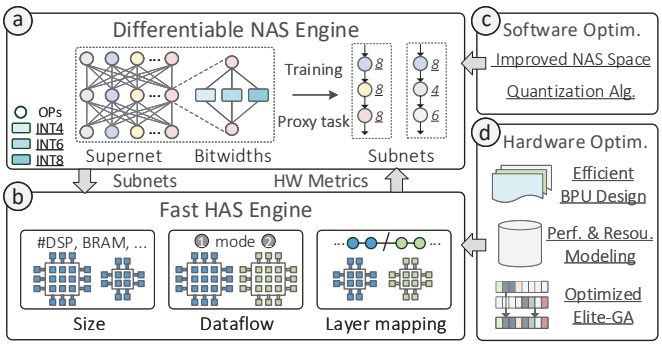

Fig. 1: The overview of our NAF co-search framework.

is proposed in [11] to strike a better trade-off between accuracy and efficiency by enabling layer-wise diversity. Moreover, it points out that adopting the hardware-agnostic metric (e.g., FLOPs) in the NAS process makes the model difficult to meet the requirements of real-world applications. Accordingly, hardware-aware NAS works emerge, taking more effective hardware metrics (e.g., latency and power) from pre-collected *lookup tables* or pre-built *predictors* to regularize the NAS process. However, these efforts only target off-the-shelf hardware without exploring the hardware design space.

B. Network and Accelerator Co-Search

Given that various ASIC/FPGA-based accelerators [3], [4], [12], [15] for DNNs and the rise of NAS, network/accelerator co-search becomes essential to enable specialization and acceleration. Early, an RL-based approach is adopted to search the network and the FPGA accelerator's parallel size [16], but the search-cost problem limits its scalability. Then, a differentiable co-search involving bitwidth is presented in [7]. It models execution time with a hardware-agnostic metric (FLOPs per resource) to find the parallel factors and only searches bitwidths of weights without considering quantizer optimization. In [9], a hardware performance *predictor* is built offline by sampling to fast feedback hardware metrics for the sizing problem of its single-core accelerator. However, the difficulty and cost of training the *predictor* will be prohibitive when HAS space increases due to the discrete and intricate relationship between accelerator design and hardware metrics. [6], [8], [13] explore a finer-grained generic HAS space (e.g., tiling and parallel dimension, size, and mappings) but only for the ASIC design. Although FPGA design is addressed in [8], it treats FPGAs as the counterparts of ASICs, neglecting their flexibility. Thus, in this work, we propose the NAF framework to efficiently explore an optimized NA-HAS for FPGA platforms.

III. Network/Accelerator Co-Exploration

Given the target datasets, FPGA platform settings, and FPS requirements, NAF automatically generates matched CNN-accelerator pairs to achieve a sufficient trade-off between performance and accuracy. The framework (see Fig. 1) includes two components: (a) the differentiable NAS engine and (b) the fast HAS engine to enable joint searching for network structures (e.g., operation types (`OPs`) and bitwidths (`INT4/6/8`)) and accelerator architectures (e.g., heterogeneous multicores and dataflow). Subsequently, we deeply optimize the co-search process from software and hardware perspectives (c-d). First,

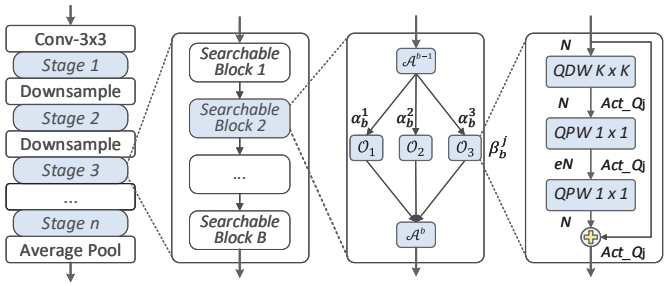

Fig. 2: The supernet structure, where the *Block* has the same meaning as the *layer* in [11]. `K` and `e` represent kernel size and the expansion ratio. `Act_Q`$_i$ is the activation quantization node. `QDW` means quantized depth-wise convolution.

we propose an improved NAS space with a hardware-oriented quantization algorithm (Sec. IV). Meanwhile, we specify the basic unit of HAS by an efficient BPU design (Sec. V). Finally, a fast hardware generation algorithm is designed based on accelerator modeling and heuristic rules (Sec. VI). Overall, the co-search problem is formulated as a bi-level optimization:

$$\min_{\alpha,\beta} \; L_{val}(\omega^*, \alpha, \beta) + \lambda L_{hw}(\alpha, \beta, \gamma^*) \tag{1}$$

$$s.t. \quad \omega^* = \arg\min_{\omega} L_{train}(\omega, \alpha, \beta) \tag{2}$$

$$s.t. \quad \gamma^* = \arg\min_{\gamma} L_{hw}(\alpha, \beta, \gamma) \tag{3}$$

$$s.t. \quad hw_{cost}(\alpha, \beta, \gamma^*) \leq hw_{limit} \tag{4}$$

where ω is the supernet weights; α, β, and γ are the architecture distribution parameters, data bitwidth parameters, and accelerator design parameters, respectively. For instance, α_b^i (see Fig. 2) represents the architecture parameter assigned to the i-th candidate operator in the b-th searchable block of the supernet. L_{val} and L_{train} are the validation and training loss of the supernet, respectively. L_{hw} is the hardware-cost loss of the sampled subnet, influenced by the network structure, bitwidth, and the accelerator. λ is a hyperparameter that controls the trade-off between terms and is dynamically adjusted during training based on the relationship between the target and current FPS. In particular, NAF dynamically generates an optimized FPGA accelerator (γ^*) for the subnet to fast support hardware evaluation in a complex and large HAS space.

IV. Neural Architecture Space Design

A. Supernet Structure

In NAF, we follow the recent widely adopted layer-wise architecture space design [7], [9], [11], in which each searchable block can choose one operator from the operator space $\mathcal{O}$ (see Fig. 2). BatchNorm (BN) and ReLU layers are omitted in `OP`'s structure (see the rightmost of Fig. 2) for brevity.

In this paper, we argue that the design of NAS should consider underlying hardware characteristics, not just model size and accuracy. Therefore, we adapt the space of NAS here. First, inspired by [17], we uniformly set the convolution kernel size of DW to 7×7 (except the 3×3 in the first two stages). Because compared with the small kernel size, DW7×7 does not bring noticeable computation and access overhead for the overall model but has better data locality. Hence, increasing the search space of K cannot bring high gains. Second, we expand the space of e to $\{2, 4, 6\}$ when $K = 3$, and $\{4, 6, 8\}$

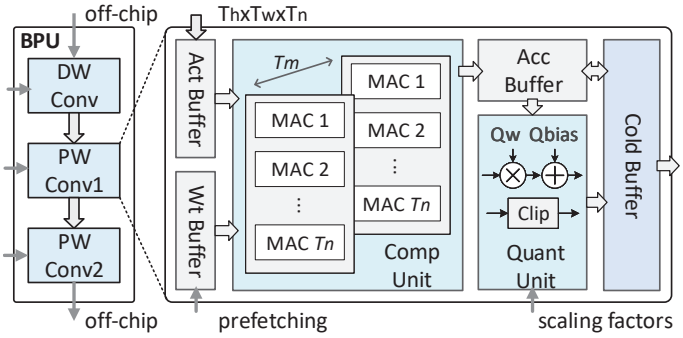

Fig. 3: Simplified block diagram of the block processing unit (BPU). Dataflow can be configured according to parameters; Branch-add operation is integrated into the PW Conv2 core.

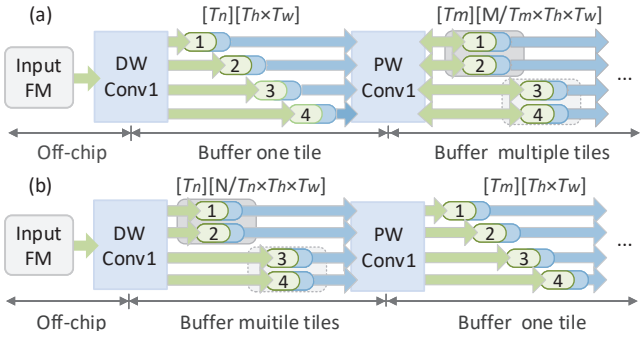

Fig. 4: Two candidate fine-grained dataflow of the BPU. (a) mode ❶: PW1-in, PW2-out and (b) mode ❷: PW1-out, PW2-in. -in/out means input/output reuse; $[P][D]$ means buffer size. P and D denote parallelism and depth, respectively.

when $K=7$. This setting reduces the search space and is in line with design experience, i.e., increasing the image channel depth when reducing the image resolution [2]. The choice of bitwidth affects accuracy, memory access, and hardware performance, which should be consistently considered but often overlooked in prior works. Considering the accuracy degradation of ultra-low bitwidth (<4) and the actual FPGA deployment, we set the candidate quantization bitwidth to $\{4, 6, 8\}$ for each searchable block. Given that $|\mathcal{O}|=3$ and the supernet consists of 21 layers, the NAS space size of NAF is $9^{21} \approx 9.85\text{E+}20$.

B. Hardware-friendly Quantization

Since the design of most quantization algorithms neglects the underlying hardware [18], it is not easy to reap practical benefits by directly integrating the quantization algorithm into the co-design process. As such, in terms of bitwidth search, we make the following optimization. First, considering the search cost and extra hardware overhead, we use a block-grained search strategy (i.e., the layers inside the block have the same bitwidth) with a per-layer quantization granularity (i.e., each layer rather than channel inside the block shares a scaling factor). Afterward, we quantize the entire computational graph (see the rightmost of Fig. 2) instead of only convolutional layers to ensure the throughput of the block when deployed. Moreover, we design a hardware-friendly quantizer.

The weights (w^l) of the l-th layer are quantized similarly to DoReFa [19] but in a symmetric way (see Eq. 5) to keep the zero point unbiased and facilitate the hardware implementation.

$$W_q^l = quant_k(\tanh w^l / \max(|\tanh w^l|)) \tag{5}$$

where $\texttt{quant}_\texttt{k}(\cdot)$ quantizes the real number input in the range of $[-1, 1]$ to a k-bit signed number output. For activations (x^l), in modern block designs, the presence of branch-add and linear output layers and the reduction of ReLU layers make the output values easy to accumulate. Therefore, we modify the activation quantization function in PACT [20] as Eq. 6 to retain more useful negative values and improve accuracy.

$$y^l = clamp(x^l, -\mathcal{C}^l, \mathcal{C}^l), \quad A_q^l = quant_k(y^l/\mathcal{C}^l) \times \mathcal{C}^l \tag{6}$$

where the activation (x^l) is clipped symmetrically by the $\texttt{clamp}(\cdot)$ function according to the $\mathcal{C}^l$, which represents the clipping level and is learned by training. For back-propagation, the gradient $\partial A_q^l/\partial \mathcal{C}^l$ can be computed as Eq. 7 by estimating $\partial A_q^l/\partial y^l$ as 1 using the Straight-Through Estimator (STE) [19].

$[exp]$ is 1 when the exp is true otherwise 0. $\mathcal{C}^l$ is fixed during inference and thus can be absorbed by the scaling factor.

$$\frac{\partial A_q^l}{\partial \mathcal{C}^l} = \frac{\partial A_q^l}{\partial y^l} \times \frac{\partial y^l}{\partial \mathcal{C}^l} = [x^l \geq \mathcal{C}^l] - [x^l \leq -\mathcal{C}^l] \tag{7}$$

V. Hardware Architecture Space Design

A. BPU Structure

Targeting the block structure, we design a multi-block reusable BPU (see Fig. 3) to provide efficient hardware support while ensuring generality. It contains three separate computing cores optimized for the DW/PW convolution layer to alleviate computational diversity and enables parallel computation of the three cores in a more fine-grained and timely manner. As an example, the PW Conv1 core consists of the activation and weight buffer, accumulation buffer, computing unit, and quantization unit. The cold buffer is used to store the intermediate results and is optional, depending on the dataflow mode. BN and ReLU operations are integrated into the quantization unit. Although layers within the block share the same bitwidth, the three computing cores are separately configurable. Overall, the fundamental idea of the BPU design is to fuse DW and PW operations so that the output tile generated by the former core can be immediately piped to the latter core without writing back to the DRAM. The tile level pipeline lowers latency while reducing off-chip access requirements, making it suitable for bandwidth-constrained platforms such as embedded FPGAs. Specifically, we apply weight reordering and prefetching to ensure off-chip burst transfers. Moreover, two low-precision MACs are packed as $A_q \times (W_q^1 << 16 + W_q^2)$ onto a DSP with 25×18 multiplier support to reduce the DSP consumption.

B. BPU Dataflow

We apply *loop tiling* on the width and height of the image (T_h, T_w) and *loop unroll* on the input and output channels (T_n, T_m) same as the widely adopted method [4], [5]. For the computation order, we only reserve the exploration of data reuse strategies to reduce the HAS space. Typical reuse strategies include input, output, and weight reuse, where due to our fine-grained pipeline structure, we only explore the performance and resource consumption in the first two modes, as shown in Fig. 4. Specifically, we fix the reuse strategy of the DW core, which generates output tiles along the channel

dimension, to significantly reduce the on-chip BRAM overhead caused by keeping intermediate results in the latter PW cores. In mode ❶, the output tile generated by the DW core flows through the PW1 core only once, so all the computations involved must be completed at a time. Therefore, the PW1 core will traverse the output channels in turn and produce M/T_m partial results ($[T_m][T_h \times T_w]$), being saved in its cold buffer. Accordingly, the PW2 core can repeatedly read the input tiles stored in the cold buffer of the PW1 core to generate the output tile and send it off-chip. Mode ❷ has an alternate layer loop order to mode ❶, and thus we will not repeat the process for simplicity.

VI. Network-Accelerator Mapping

A. Accelerator Modeling

1) Resource Modeling: Ignoring ultra-low bitwidth computation (<4), DSP blocks, BRAM, and off-chip bandwidth (BW) are often the limiting factors in the FPGA-based accelerator's design [4]. Thus, we carefully evaluate the usage of DSP and BRAM in the modeling process. The BW is dynamically allocated during the hardware generation process.

DSP usage is closely related to the parallelism and bitwidth of the computing unit. Taking the Xilinx FPGA as an example, overlooking the extra consumption of the quantization operation, the DSP usage of the BPU ($\mathcal{D}_{bpu}$) is shown below:

$$\mathcal{D}_{bpu} = \Psi(q_{max}^{bks}) \times \left(T_n^{dw} + T_n^{pw_1} \times T_m^{pw_1} + T_n^{pw_2} \times T_m^{pw_2}\right) \quad (8)$$

where $\Psi(q)$ is the DSP cost function under different bitwidths. In detail, $\Psi(q) = 1$, when $8 < q \leq 16$; $\Psi(q) = 0.5$, when $5 \leq q \leq 8$, $\Psi(q) = 0$, when $q \leq 4$. Note that `INT8` multiplication still consumes one DSP in DW Conv instead of half due to the limitation that MACs packing requires a common multiplier.

BRAM resources primarily provide the required on-chip storage and bandwidth through memory interleaving for the computing unit. For simplicity, we only list the BRAM consumption of the PW1 core in the BPU in mode ❶ as:

$$\mathcal{B}_{pw_1} = 2 \sum \begin{cases} \lceil \frac{q_{max}^{bks} \times T_n^{pw_1}}{bwidth} \rceil \times \lceil \frac{T_h \times T_w}{bdepth} \rceil \\ \lceil \frac{q_{acc} \times T_m^{pw_1}}{bwidth} \rceil \times \lceil \frac{M_{max}^{bks} / T_m \times T_h \times T_w}{bdepth} \rceil \end{cases} \quad (9)$$

where q_{max}^{bks} and M_{max}^{bks} are the maximum bitwidth and output channels of the blocks sharing the same BPU, and q_{acc} is the accumulation buffer bitwidth. $bwidth$ and $bdepth$ are the data width and depth provided by one BRAM, respectively. We ignore the weight buffer because it is usually implemented by LUTs and Flip-Flops (FFs) due to the small block depth.

2) Performance Modeling: We use the number of cycles required for the BPU to complete the workload as the performance feedback. Ignoring the start and exit stage of the pipeline, the latency of the l-th block running on the PW1 core ($\mathcal{L}_{pw_1}^l$) and BPU ($\mathcal{L}_{bpu}^l$) are approximated as follows:

$$\begin{cases} \mathcal{L}_{pw_1}^l = \lceil \frac{M^l}{T_m^{pw_1}} \rceil \times \lceil \frac{N^l}{T_n^{pw_1}} \rceil \times \max(\mathcal{L}_{comp}^{pw_1}, \mathcal{L}_{in}^{pw_1}, \mathcal{L}_{wt}^{pw_1}) \\ \mathcal{L}_{bpu}^l = \lceil \frac{H^l}{T_h} \rceil \times \lceil \frac{W^l}{T_w} \rceil \times \max\left(\mathcal{L}_{dw}^l, \mathcal{L}_{pw_1}^l, \mathcal{L}_{pw_2}^l\right) \end{cases} \quad (10)$$

where we try to make the computation latency ($\mathcal{L}_{comp}^{pw_1}$) cover the latency of loading activations and weights by double buffering and BW allocation. For conciseness, Eq. 10 ignores the pipeline's start and end phases and other details (e.g., the latency of burst off-chip access). We consider these factors and calibrate the modeling using actual latencies obtained by deploying different `OPs` under different hardware configurations. Finally, we restrict the average performance estimation error to 5% to present the actual behavior mapped on the accelerator.

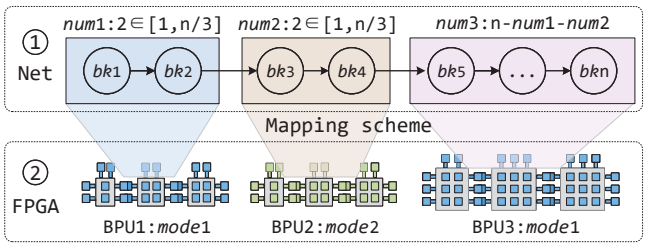

Fig. 5: Example of mapping a subnet on target FPGA with three BPUs using the CTC-based layer allocation strategy.

B. CTC-based Layer Allocation Strategy

Network-accelerator mapping in multi-core scenarios on FPGAs mainly invloves the number of cores and layer allocation strategy, which has been explored in prior works. However, the minute-level search time [10], [12] makes them unacceptable in the NAS framework. Thus, we try to obtain a better compromise between performance and time by pruning the search space.

First, the number of heterogeneous cores ($\mathtt{N_{BPUs}}$) is often related to the specific workload and platform settings, so it is dynamically determined during the search process (see Algorithm 1) to meet the requirements. Second, inspired by [5], we propose a layer assignment strategy based on the computation to communication (CTC) ratio. We find that, similar to the manually designed CNNs, the CTC variance of blocks near the input end in NASNets is larger than the variance of the output end. Thus, letting as few blocks at the network's input end share the same BPU as possible makes sense. Accordingly, we add the variable (num) to describe the number of consecutive blocks assigned to the BPU and limit its search range (see Fig. 5). The parameter variable ($\mathcal{PV}$) describing the BPU is:

$$\mathcal{PV} = [num, mode, T_h, T_n^{dw}, T_n^{pw_1}, T_m^{pw_1}, T_n^{pw_2}, T_m^{pw_2}] \quad (11)$$

When there are multiple BPUs, the number of blocks near the input end that share a BPU does not exceed the average.

C. Rule-based Heuristic Search

Although we prune the mapping space, the remaining search space still makes the cost of exhaustive or random search unacceptable. Under strict time constraints, we finally employ a genetic algorithm (GA) for its lightness and simplicity to solve our problem. Research shows that GAs can reach competitive results compared to deep reinforcement learning (DRL) [21]. Eventually, we design a fast hardware generation algorithm (see Algorithm 1) based on the elitist genetic algorithm (e-GA) with the following optimizations to speed up the search process: (i) Rule-based coarse-grained search. Considering the possible channels and resolution sizes, we restrict the tiling and parallel sizes in the most likely candidate list (as shown in Tab. I). Hence, the HAS space size after pruning is about `2.49E+20`. (ii) Parallel processing. The evaluation of individuals in GA's

Algorithm 1: Fast hardware generation algorithm

Initialize the target latency (lat_{target}) and hardware constraint
Initialize population size ($\mathcal{P}[\mathcal{M}]$), iteration number ($\mathcal{I}$)
Initialize the latency ($lblock()$) and resource ($rblock()$) model
for c ***in*** $[1, N_{BPUs}]$ **do**
  Randomly initialize each individual in $\mathcal{P}opu[\mathcal{M}]$
  while $iteration < \mathcal{I}$ **do**
    ◁ ***multi-process parallel***
    for ind *in* $[1, \mathcal{M}]$ **do**
      Get the fitness index with $lblock()$ and $rblock()$
      Evaluate: $Fit_{ind} = \boldsymbol{FitScore}(\mathcal{P}opu[ind])$
    Update the top-k global optimal individuals
    for ind *in* $[k, \mathcal{M}]$ **do**
      Select parents (s, t) according to probability
      Crossover: $Temp = \boldsymbol{cross}(\mathcal{P}opu[s], \mathcal{P}opu[t])$
      Mutation: $\mathcal{P}opu[ind] = \boldsymbol{mutat}(Temp)$
    Keep the best $\mathcal{L} = Obj[1]$ and $\mathcal{PV} = \mathcal{P}opu[1]$
  Update the best latency: $\mathcal{L}_{best}$
    ◁ ***early stop the HAS search***
  if $\mathcal{L}_{best} \leq lat_{target}$ **then**
    break
  $\mathcal{I} = \mathcal{I} + \delta_1[c]; \mathcal{M} = \mathcal{M} + \delta_2[c];$
Return best overall latency ($\mathcal{L}_{best}$) with corresponding block-wise latency and hardware parameters ($\mathcal{PV}s$)

population is independent, so the parallel gain for the process is significant. (iii) Early stop the HAS search. It can save search time by quickly evaluating the trivial subnets. δ_1/δ_2 is set to increase the iteration/population number to ensure convergence. Kindly note that our aim is not to find the optimal solution but a suitable optimized solution to quickly reflect the performance level of the subnet on the hardware platform.

TABLE I: Hardware Architecture Search Space

	Candidate values
Cores ($\mathtt{N_{BPUs}}$) **& Bitwidths**	[1, 5]; {4, 6, 8}
Dataflow & BKs (num)	{1, 2}; [1, $\mathtt{block_{nums}/N_{BPUs}}$]
Tiling size (T_h, T_w)	1, 7, 14, 28, 56
Parallel size (T_n, T_m)	2, 4, 6, 8, 12, 16, 24, 32, 48, 64

VII. Evaluation

A. Experimental Setup

At the cost of considerable offline setup time, prior works deliver short hardware (HW) feedback delay within a smaller HAS space, ignoring the exploration of HW update frequency. Thus, to efficiently handle a larger HAS space, we first explore the effect of HW update frequency on the co-search process on the CIFAR10 dataset. The supernets are trained for 60 epochs with batch size 128 and no α or β update for the first 15 epochs. The results (see Fig. 6) show that the hardware metric can effectively regulate the co-search process under different numbers of BPUs. Although the HW update every 40 batches is still valid, its fluctuation is relatively large. As such, we set the HW update frequency to 20 to speed up the search.

NAF performs co-search on a subset of the ImageNet dataset randomly sampled from 100 classes, trained for 120 epochs, where the first 45 epochs do not update α and β. This procedure takes 32.5 hours on two NVIDIA A100 GPUs with a hardware update frequency of every 20 batches. Then, the searched DNN

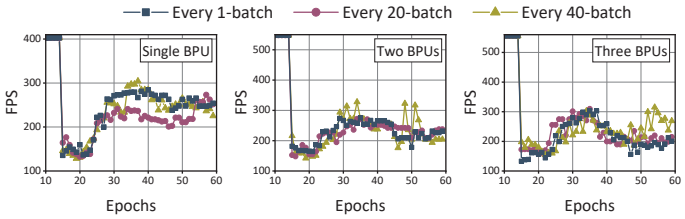

Fig. 6: Hardware update frequency experiments on the CIFAR10 dataset, targeting 200 FPS on Xilinx ZC706.

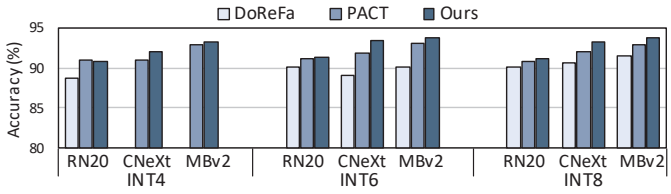

Fig. 7: Quantization algorithm comparison on CIFAR10.

is trained for 300 epochs from scratch on the whole ImageNet training set. The accelerator is synthesized and placed-and-routed with Xilinx Vitis HLS and Design Suite (v20.2), targeting the Xilinx ZCU102 board at 214Mhz. A power meter is plugged in to measure the runtime power performance.

B. Quantization Algorithm Comparison

We quantize ResNet20 (RN20), ConvNeXt (CNeXt), and MobileNet-V2 (MBv2) on CIFAR10 using different quantization algorithms to compare the accuracy. The training parameters remain the same: 128 batch size, 0.01 learning rate, cosine learning rate adjustment, and 300 epochs. Experimental results (see Fig. 7) present that our method achieves the best accuracy on CNeXt and MBv2 under different bitwidths. We can see that DoReFa and PACT lead to different degrees of accuracy degradation on CNeXt and MBv2, respectively. At `INT4`, the accuracy of applying DoReFa is even lower than 80%.

C. Hardware Generation Algorithm Comparison

To evaluate our HAS algorithm's efficiency, we compare it with prior network-accelerator mapping work [12]. For a fair comparison, we also apply the multithreading and coarse-grained optimization to the baseline method. The target NASNets are FBNet-B [11], EDDNet-3 [7], and EfficientNet-

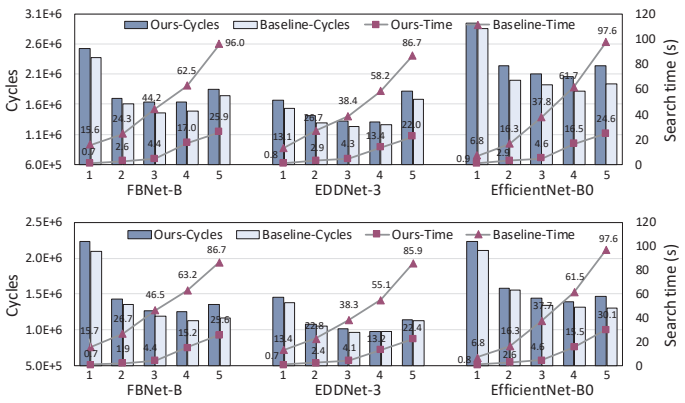

Fig. 8: Comparison of the performance and time cost of hardware generated for different NASNets (`INT8` quantization on the ImageNet) using [12] and our method on ZC706 (top) and ZCU102 (bottom) under different numbers of BPUs.

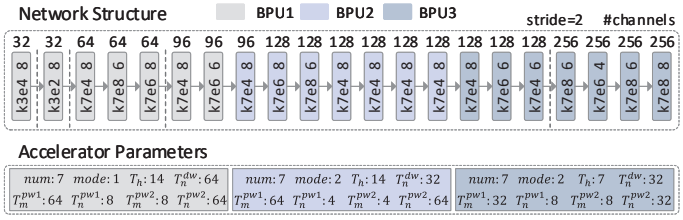

Fig. 9: Searched CNN and the corresponding FPGA accelerator.

B0 [22], and the hardware platforms are ZC706 and ZCU102. The CPU is the Intel Xeon gold 6240. The results (see Fig. 8) exhibit that due to the inter-layer computational diversity of the current NASNets, the BPU-based multi-core structure can yield significant performance gains. However, when $\texttt{N}_{\texttt{BPUs}} > 3$, the search cost evidently outweighs the performance gain, and the coarse-grained search cannot even get an optimized solution in a short time when $\texttt{N}_{\texttt{BPUs}} = 5$. Thus, we set the maximum number of cores to 3. Remarkably, although our best performance is inferior to [12] (within 12%), our search is about 9.6× faster, which is worthwhile in the extensive NA-HAS process.

D. Search Efficiency and Quality Comparison

The searched solution targeting the ImageNet dataset and the Xilinx ZCU102 board is shown in Fig. 9. The accelerator utilizes 2255 DSPs (89%), 781 BRAM36Ks (86%), 226k LUTs (82%) and 263k FFs (48%). To verify the effectiveness of NAF, we compare it with previous SOTA hardware-aware NAS and co-search works regarding search efficiency and quality. For search efficiency, we mainly compare the search space size and search time, as shown in Tab. II. We can see that NAF explores a remarkably larger joint search space in a notably less search time. Compared with the RL-based works [16], [23], they still suffer a much longer search time, despite the small search space they involve. Compared with differentiable co-search works [8], [9], NAF achieves a 2.95×∼4.43× less search time. For search quality, we compare the searched CNN-accelerator pair's accuracy, throughput, and energy efficiency. Tab. III shows that NAF obtains a 1.99×∼6.84× throughput improvement under the same platform. To make a fair comparison across platforms, we set energy efficiency (FPS/W) as the hardware criterion. We find that NAF achieves 17%∼88% energy efficiency improvement with similar accuracy.

TABLE II: Search Efficiency Comparison on ImageNet

Method	NAS Space	HAS Space	Joint Space	Search Time† [GPU-hours]
HS-Co-Opt [16]	2.252+18	1	1.15E+18	266
OFA [24]	2.00E+19	1	2.00E+19	1200
BSW [23]	4.20E+05	8.64E+03	3.63E+09	1000
DIAN [8]	9.85E+20	4.89E+17	4.82E+38	144
Co-Explore [9]	4.30E+7	3.00E+02	1.29E+10	96
NAF (ours)	9.85E+20	2.49E+20	**2.45E+41**	**32.5**

† The reported GPU-hours using from the baseline's original papers.

VIII. Conclusion

We propose an efficient differentiable network/accelerator co-search framework, named NAF, to generate high-quality solutions for target reconfigurable platforms and datasets. Backed by software and hardware optimizations, it efficiently explores a deeper NA-HAS space. Compared with prior SOTA co-design works, NAF achieves a 1.99×∼6.84× throughput gain on Xilinx ZCU102 and 17%∼88% energy consumption improvement while maintaining similar accuracy.

TABLE III: Search Quality Comparison on ImageNet

Method	Platform	Power [W]	Top-1 Acc[%]	Thpt. FPS	Energy Effi. [FPS/W]
HS-Co-Opt [16]	Xilinx XC7Z015	—	68.0	12.1	—
TCAD'22 [1]	Xilinx ZCU102	—	73.3	36.6	—
EDD [7]	Xilinx ZCU102	—	74.6	125.6	—
Co-Explore [9]	Intel GX1150	43.63	77.6	221.2	5.07
HAO [25]	Xilinx Ultra96	5.5	72.7	44.9	8.16
NAF (ours)	Xilinx ZCU102	26.19	74.9	**250.2**	**9.55**

IX. Acknowledgment

This work is partially supported by the National Key R&D Program of China (under Grant 2017YFA0700900, 2017YFA0700903), National Natural Science Foundation of China (under Grants 62102383, 61976200, and 62172380), Jiangsu Provincial Natural Science Foundation under Grant BK20210123, Youth Innovation Promotion Association CAS under Grant Y2021121, and the USTC Research Funds of the Double First-Class Initiative under Grant YD2150002005.

References

[1] Y. Liang *et al.*, "An efficient hardware design for accelerating sparse cnns with nas-based models," *TCAD*, vol. 41, no. 3, pp. 597–613, 2022.
[2] B. Zoph *et al.*, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018.
[3] Z. Du *et al.*, "Shidiannao: Shifting vision processing closer to the sensor," in *ISCA*, 2015, p. 92–104.
[4] Y. Chen *et al.*, "Cloud-dnn: An open framework for mapping dnn models to cloud fpgas," in *FPGA*, 2019, pp. 73–82.
[5] X. Zhang *et al.*, "Dnnexplorer: a framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator," in *ICCAD*, 2020.
[6] K. Choi *et al.*, "Dance: Differentiable accelerator/network co-exploration," in *DAC*, 2021, pp. 337–342.
[7] Y. Li *et al.*, "Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions," in *DAC*, 2020, pp. 1–6.
[8] Y. Zhang *et al.*, "Dian: Differentiable accelerator-network co-search towards maximal dnn efficiency," in *ISLPED*, 2021, pp. 1–6.
[9] H. Fan *et al.*, "Algorithm and hardware co-design for reconfigurable cnn accelerator," in *ASP-DAC*, 2022, pp. 250–255.
[10] N. Fasfous *et al.*, "Anaconga: Analytical hw-cnn co-design using nested genetic algorithms," in *DATE*, 2022, pp. 238–243.
[11] B. Wu *et al.*, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *CVPR*, 2019.
[12] L. C. Chan *et al.*, "Partitioning fpga-optimized systolic arrays for fun and profit," in *ICFPT*. IEEE, 2019, pp. 144–152.
[13] Y. Lin *et al.*, "Naas:neural accelerator architecture search," in *DAC*, 2021.
[14] H. Liu *et al.*, "Darts: Differentiable architecture search," in *ICLR*, 2019.
[15] C. Wang *et al.*, "A ubiquitous machine learning accelerator with automatic parallelization on fpga," *TPDS*, vol. 31, no. 10, pp. 2346–2359, 2020.
[16] W. Jiang *et al.*, "Hardware/software co-exploration of neural architectures," *TCAD*, vol. 39, no. 12, pp. 4805–4815, 2020.
[17] Z. Liu *et al.*, "A convnet for the 2020s," in *CVPR*, 2022.
[18] W. Lou *et al.*, "Octcnn: A high throughput fpga accelerator for cnns using octave convolution algorithm," *TC*, vol. 71, no. 8, pp. 1847–1859, 2022.
[19] S. Zhou *et al.*, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv:1606.06160*, 2016.
[20] J. Choi *et al.*, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.
[21] S.-C. Kao *et al.*, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in *MICRO*, 2020.
[22] M. Tan *et al.*, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, 2019.
[23] M. S. Abdelfattah *et al.*, "Best of both worlds: Automl codesign of a cnn and its hardware accelerator," in *DAC*, 2020, pp. 1–6.
[24] H. Cai *et al.*, "Once-for-all: Train one network and specialize it for efficient deployment," in *ICLR*, 2020.
[25] Z. Dong *et al.*, "Hao: Hardware-aware neural architecture optimization for efficient inference," in *FCCM*, 2021, pp. 50–59.