AnaCoNGA: Analytical HW-CNN Co-Design Using Nested Genetic Algorithms

AnaCoNGA: Analytical HW-CNN Co-design using Nested Genetic Algorithms

Nael Fasfous[*], Manoj Rohit Vemparala[†], Alexander Frickenstein[†],
Emanuele Valpreda[‡], Driton Salihu[*], Julian Höfer[§], Anmol Singh[†], Naveen-Shankar Nagaraja[†],
Hans-Joerg Voegel[†], Nguyen Anh Vu Doan[*], Maurizio Martina[‡], Juergen Becker[§], Walter Stechele[*]
[*]*Department of Electrical and Computer Engineering, Technical University of Munich, Germany*
[†]*Autonomous Driving, BMW Group, Germany*
[§]*Department of Electrical Engineering and Information Technology, Karlsruhe Institute of Technology, Germany*
[‡]*Department of Electronics and Telecommunications, Politecnico di Torino, Italy*

***Abstract*—We present AnaCoNGA, an analytical co-design methodology, which enables two genetic algorithms to evaluate the fitness of design decisions on the layer-wise quantization of a neural network and the allocation of hardware (HW) resources. We embed a hardware architecture search (HAS) algorithm *into* a quantization strategy search (QSS) algorithm to evaluate the hardware design Pareto-front of each considered quantization strategy. We harness the speed and flexibility of analytical HW-modeling to enable parallel HW-CNN co-design. With this approach, the QSS is focused on seeking high-accuracy quantization strategies which are guaranteed to have efficient hardware designs at the end of the search. Through AnaCoNGA, we improve the accuracy by 2.88 p.p. with respect to a uniform 2-bit ResNet20 on CIFAR-10, and achieve a 35% and 37% improvement in latency and DRAM accesses, while reducing LUT and BRAM resources by 9% and 59% respectively, when compared to a standard edge variant of the accelerator.**

I. Introduction

Accelerating modern convolutional neural networks (CNNs) for embedded applications presents challenges across multiple domains. The constrained nature of embedded hardware places tight budgets on memory and compute resources, while safety-critical applications place minimum bounds on task-related prediction accuracy. These hardware (HW) and software (SW) constraints can lead to contrasting design targets.

An example of such a HW-SW co-design scenario is the numerical quantization of CNNs and the HW design of a bit-serial accelerator. Quantization is a standard approach to reduce the bitwidth of operands and the complexity of the arithmetic operations of a CNN [1]–[3]. The advantages of quantization extend to simpler arithmetic HW, higher data movement efficiency, lower energy requirements, and lower latency. Moreover, CNNs have been shown to benefit from layer-wise and datatype variable numerical precision [4]. To extract the mentioned benefits of variable numerical precision, a HW accelerator can employ bit-serial computation units [5]. Such an accelerator can have an array of spatially distributed computation units and a distributed on-chip buffer to efficiently provide the computation array with data.

For the CNN designer and the HW engineer to effectively arrive at a solution that meets an application's constraints, a large and complex solution space must be explored. This motivates the design of a lightweight, easily reconfigurable HW-model, which can be used to evaluate design choices in this complex space [6]–[8]. Harnessing the speed and flexibility of such models can enable development of the HW ***parallel*** to the CNN design phases without prohibitive synthesis or cycle-accurate simulation bottlenecks.

In this paper, we embed HW architecture search (HAS) ***into*** quantization strategy search (QSS), in a **nested** genetic algorithm (GA) formulation. The main contributions of this work can be summarized as follows:

- Formulating an analytical HW-model for the execution of CNN workloads on a state-of-the-art bit-serial accelerator [5], allowing fast exploration and evaluation of HW performance and resource utilization, without the need for costly synthesis or simulation.
- Compressing CNNs through layer-wise, datatype-wise quantization strategy search (QSS) and automating the design of a bit-serial accelerator through HW architecture search (HAS), by formulating two multi-objective GAs, circumventing the need for handcrafted reward functions.
- We insert the HAS loop into the QSS loop. For each potential quantization strategy proposed by QSS, the HAS loop efficiently evaluates a 4-D HW design Pareto-front. After synthesis, the HW-CNN co-designed pair achieve a 35% and 37% reduction in latency and DRAM accesses, while achieving 2.88 p.p. higher accuracy compared to a 2-bit ResNet20-CIFAR-10 executing on a standard edge variant of the accelerator.

II. Related Work

A. HW-aware CNN Design

The authors of HAQ [4] propose a reinforcement learning (RL) exploration scheme to determine HW-aware, layer-wise quantization levels for weights and activations of a CNN model. The reward function is evaluated after executing the inference of the quantized CNN on a fixed, pre-synthesized FPGA design. AutoQ [9] is an incremental improvement of [4], with the RL-agent exploring quantization levels in a layer-wise manner for activations and channel-wise for weights. Inference performance is estimated using a model of a bit-serial accelerator. In APQ [10], a joint model architecture-pruning-quantization search is proposed. Pre-trained and pruned sub-networks are extracted from a once-for-all network, mixed-precision quantization is applied thereafter. An energy/latency look-up table is used to provide the HW feedback during the search. The works in this section are analogous to our standalone quantization strategy search (QSS) loop, presented in Sec. III-C.

B. *CNN-aware HW Design*

AutoDNNchip [11] proposes a framework for automated ASIC/FPGA design of a HW accelerator for a given performance target and a specific CNN model. The design space is explored with a performance model to select candidate HW architectures, followed by a run-time simulation to optimize their pipelines. MAGNet [12] is an accelerator generator for CNNs based on a reconfigurable, tile-based spatial array. A baseline accelerator is iteratively mapped and evaluated for a target CNN and then tuned using Bayesian search. Both [11] and [12] support mixed-precision computation, but do not explore the layer-wise quantization search space. The works which fall under this category resemble our HW architecture search (HAS) optimization loop presented in Sec. III-D.

C. *Joint HW-CNN Co-design*

NHAS [13] aims to find an optimal quantized CNN architecture using an evolutionary algorithm. An efficient HW dimensioning for the compute array and on-chip memory is searched to accelerate a pool of CNN workloads used as a benchmark. After the HW is configured, the CNN search space is explored. The HW evaluation follows a look-up table approach due to the smaller quantization search space considered. This **sequential** approach of co-design can be enhanced by including the HW design search *within* the CNN search loop. Other works which target joint HW-CNN co-design are [14] and [15], both of which include the HW's performance in the reward function of an RL-agent and **iteratively** tune both the CNN and HW architectures. Fine HW-level details, such as scheduling schemes and quantized execution, are not explored in [14], as the optimization loop targets optimally partitioning the CNN workload over a pool of FPGAs. In [15], layer-wise quantization is not supported.

In this work, we propose a **nested co-design** approach to perform HW design **parallel** to the quantization search, leading to a tight coupling between the HW and CNN, without iteratively or sequentially switching between the two domains. The classification of the mentioned works is shown in Tab. I.

TABLE I
CLASSIFICATION OF HW-CNN OPTIMIZATION METHODS.

Classification	**HW Metrics**	**CNN Compression**	**HW Design**	**Parallel Co-design**
QSS [4], [9], [10]	✓	✓	✗	✗
HAS [11], [12]	✓	✗	✓	✗
QSS+HAS [13]–[15]	✓	✓	✓	✗
AnaCoNGA	✓ (Nested)	✓	✓ (Nested)	✓

III. METHODOLOGY

In this section, we present three main components of this work, (1) the analytical accelerator model, (2) the quantization strategy search (QSS) algorithm, and (3) the hardware architecture search (HAS) algorithm, followed by AnaCoNGA.

A. *Bit-Serial Accelerator Modeling for BISMO*

Convolutional and fully-connected layers can be lowered into a general matrix multiplication (GEMM) by representing the weight tensor W^l and activation tensor A^{l-1} of layer l as 2-D matrices $\mathtt{Mat}_W$ and $\mathtt{Mat}_A$ (Eq. (1)). The dimensions m and n represent the rows and columns of each matrix.

$$\begin{gathered}\mathtt{Mat}_W \in \mathbb{R}^{m_W \times n_W},\ \mathtt{Mat}_A \in \mathbb{R}^{m_A \times n_A} \\ A^l = \mathtt{Conv}(W^l, A^{l-1}) = \mathtt{Mat}_W \times \mathtt{Mat}_A\end{gathered} \qquad (1)$$

Note that transposing both matrices and switching their order would also produce the convolution result. Therefore, we will refer to the matrix positions instead of the datatype for the remainder of the text. LHS is the left-hand side matrix, while RHS is the right-hand side. For readability, m rows and n columns will appear as subscripts of the corresponding matrix when referring to its dimensions, e.g., $\mathtt{LHS}_n$ is the number of columns of the left-hand side matrix.

The BISMO accelerator [5], abstracted in Fig. 1, is composed of a $\mathtt{D}_m \times \mathtt{D}_n$ array of processing elements (PEs). Each PE is responsible for the dot-product of one row of the LHS against one column of the RHS. Due to the bit-serial decomposition of the GEMM operation, the same row and column must be computed as many times as the bitwidths of its operands necessitate. This decomposition is elaborated in [5]. Typically, the $\mathtt{D}_m \times \mathtt{D}_n$ is much smaller than the layer's $\mathtt{LHS}_m \times \mathtt{RHS}_n$. The computation must be broken down into smaller $\mathtt{D}_m \times \mathtt{D}_n$ sized tiles. Furthermore, each PE can perform $\mathtt{D}_k$ binary dot-products in parallel, whereby the row-column dot-product is computed in tiles of $\mathtt{D}_k$, if the inner product of LHS and RHS is greater than $\mathtt{D}_k$. To maintain structured parallelism across the computation array, the LHS and RHS matrices are padded to obtain matrices that are divisible by the dimensions of the array. With the padded matrices, the number of tiles necessary to complete the computation can be expressed in Eq. (2).

$$\begin{gathered}\mathtt{T}_m = \mathtt{Padded\_LHS}_m/\mathtt{D}_m, \quad \mathtt{T}_n = \mathtt{Padded\_RHS}_n/\mathtt{D}_n, \\ \mathtt{T}_k = \mathtt{Padded\_LHS}_n/\mathtt{D}_k\end{gathered} \qquad (2)$$

Knowing the size of the padded matrices and the number of tiles necessary to complete the GEMM operation, the size of each tile can be computed in bytes according to Eq. (3), where $\mathtt{LHS}_{\text{bits}}$ is the numerical precision of the LHS elements.

$$\mathtt{LHS\_T}_{\text{bytes}} = \frac{\mathtt{Padded\_LHS}_m \cdot \mathtt{Padded\_LHS}_n \cdot \mathtt{LHS}_{\text{bits}}}{\mathtt{T}_m \cdot 8} \qquad (3)$$

DRAM requests relating to the LHS depend on both $\mathtt{T}_m$ and $\mathtt{T}_n$, whereas RHS elements are only requested $\mathtt{T}_n$ times (see Eq. (4)). This is a function of BISMO's standard scheduler

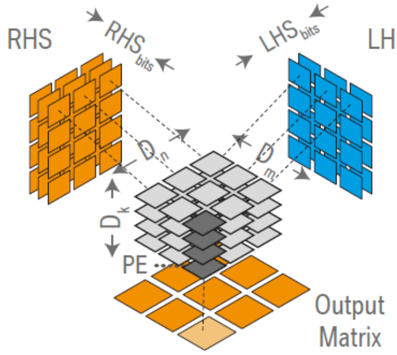


Fig. 1. High-level abstraction of a bit-serial accelerator [5]: The dimensions $\mathtt{D}_m, \mathtt{D}_n, \mathtt{D}_k$ determine the tiling degree of matricies RHS and LHS.

maintaining reuse of the RHS matrix. Since each $\mathtt{T}_m$ of the LHS must be computed against all tiles $\mathtt{T}_n$ of the RHS, the scheduler keeps the RHS tiles on chip until they have been used exhaustively. When a new tile of RHS is loaded, all $\mathtt{T}_m$ tiles of the LHS must be called again to be computed with it.

$$\begin{aligned} \mathtt{DRAM}_{\mathtt{LHS}} = \mathtt{T}_m \cdot \mathtt{T}_n \cdot \mathtt{LHS\_T}_{\text{bytes}}, \quad \mathtt{DRAM}_{\mathtt{RHS}} = \mathtt{T}_n \cdot \mathtt{RHS\_T}_{\text{bytes}} \\ \mathtt{DRAM}_{\mathtt{Result}} = \mathtt{LHS}_m \cdot \mathtt{RHS}_n \cdot 4, \text{ 32-bit write-back} \\ \mathtt{DRAM}_{\mathtt{Total}} = \mathtt{DRAM}_{\mathtt{LHS}} + \mathtt{DRAM}_{\mathtt{RHS}} + \mathtt{DRAM}_{\mathtt{Result}} \end{aligned} \tag{4}$$

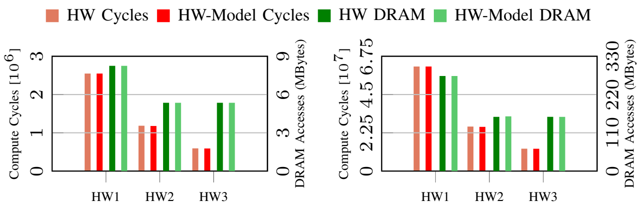
Finally, looking at the cycles spent for computation, each bit of each tile of each matrix must be computed against the bits from the other matrix. Additional cycles are spent as part of the pipeline for each bit combination on each $\mathtt{T}_m$ computed against $\mathtt{T}_n$. The BISMO accelerator overlaps data transfers with computation, resulting in Eq. (5) being sufficiently accurate for design space exploration.

$$\begin{aligned} \mathtt{Compute\_Cycles} = (\mathtt{T}_m \cdot \mathtt{T}_n \cdot \mathtt{T}_k \cdot \mathtt{LHS}_{\text{bits}} \cdot \mathtt{RHS}_{\text{bits}}) + \\ \mathtt{T}_m \cdot \mathtt{T}_n \cdot (8 \cdot (\mathtt{LHS}_{\text{bits}} \cdot \mathtt{RHS}_{\text{bits}} + 1) + 3) + 2 \cdot \mathtt{T}_n \end{aligned} \tag{5}$$

With this analysis, we are able to evaluate workload execution metrics with respect to HW parameters such as compute array dimensions $\mathtt{D}_m, \mathtt{D}_n, \mathtt{D}_k$, as well as on-chip buffer sizes for LHS and RHS without having to synthesize the HW each time. We use the analytical model introduced in this section to perform fast exploration and design of the HW as an example. It is important to note that AnaCoNGA is not limited to this HW analytical model; the optimization loops introduced in the next sections can potentially be used to harness the speed and flexibility of more advanced fast analytical HW models in literature, such as CoSA [6], GAMMA [8], or Timeloop [7].

B. Model Validation & Real HW Measurements

To validate the proposed analytical HW model, we synthesize three differently dimensioned BISMO accelerators, detailed in Tab. II. HW1 and HW3 represent small and large accelerators, while HW2 has an asymmetric processing element array. We run small and large GEMM operations on all three accelerators by executing all the convolutional and fully-connected layers of ResNet20 for CIFAR-10 (small) and ResNet18 for ImageNet (large), with 4-bits for weights and activations. In Fig. 2, we present the results of our HW-model's accuracy and fidelity compared to real, synthesized HW, for estimating compute cycles and DRAM accesses. The high accuracy and correlation of the measured and estimated values make the model well-suited for design space exploration. We note that this HW-model is used for exploration, however all the results reported in the final Tab. III are on real, synthesized HW, after AnaCoNGA concludes its co-design search.



(b) ResNet20-CIFAR-10 4-bit (c) ResNet18-ImageNet 4-bit

Fig. 2. Validation of the HW-model vs. real HW measurements for compute cycles and DRAM accesses on three BISMO configurations (HW1-3).

TABLE II
HARDWARE CONFIGURATIONS USED FOR MODEL VALIDATION.

Config	$\mathtt{D}_m \times \mathtt{D}_n$	$\mathtt{D}_k$	LHS_Buffer	RHS_Buffer
HW1	4×4	128	128KB	128KB
HW2	4×8	256	128KB	256KB
HW3	8×8	256	256KB	256KB

C. Genetic Quantization Strategy Search (QSS)

The quantization search space for a CNN has a size of Q^{2L}, where Q is the set of possible quantization levels for weights and activations, and L is the number of layers in the neural network. It is important to note that quantization strategy search (QSS) can be applied to any quantization technique (DoReFa [1], PACT [2], or others), as it only tries to find the best bitwidths for each layer and datatype. A multi-objective GA (MOGA) is used to tackle the multi-criteria optimization problem of maximizing accuracy and minimizing HW-related costs. No HW design takes place in this search.

An initial population $\mathcal{P}_0$ is randomly generated, with each genome encoding a quantization strategy, i.e., a quantization tuple $(W^l_{\text{bits}}\ ,\ A^{l-1}_{\text{bits}})$ for each layer of the CNN (explicit, bijective encoding). The genomes of $\mathcal{P}$ are briefly fine-tuned and evaluated based on their task accuracy on a validation set. When using *standalone* QSS, the GA must additionally consider the fitness of the quantized CNNs on HW estimates (DRAM accesses and computation cycles). Based on the three fitness metrics, we identify the *Pareto optimality* of each individual with respect to the population $\mathcal{P}$. The population goes through phases of selection, crossover and mutation for the subsequent generations, producing Pareto-optimal CNN quantization strategies.

D. Genetic Hardware Architecture Search (HAS)

We formulate a second GA that allocates and optimally dimensions the HW. Each individual's genome captures

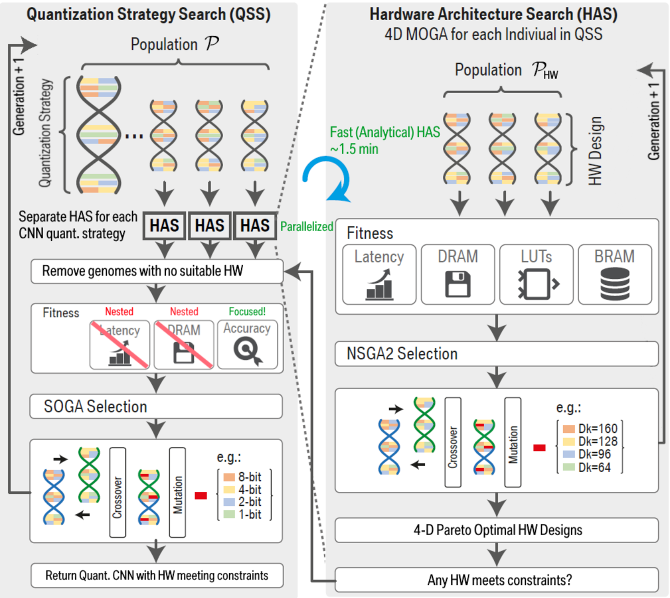


Fig. 3. AnaCoNGA: Each individual from QSS executes its **own** HAS MOGA. Any QSS individual can prove itself efficient on its **own** HW design to get a chance for its accuracy to be evaluated. QSS is relieved from optimizing HW and is transformed to a SOGA (i.e. accuracy focused).

HW design decisions, namely $\texttt{D}_m$, $\texttt{D}_n$, $\texttt{D}_k$, $\texttt{LHS\_Depth}$, and $\texttt{RHS\_Depth}$ at each genetic locus. The fitness criteria of this GA are the HW design's execution performance (compute cycles and DRAM accesses) of a *predetermined* quantized CNN, as well as the amount of FPGA resources (BRAMs and LUTs) it requires for its allocation. To estimate the FPGA resource utilization of a genome, we use the model proposed by Umuroglu et al. [5]. For performance criteria, the model presented in Sec. III-A is used. NSGA-II is applied to this 4-dimensional solution space as it can return a multitude of Pareto-optimal solutions for the designer to choose from. Complex formulation of a reward function is not necessary in the MOGA, relieving the burden of balancing the 4 criteria in a single, handcrafted reward value.

E. AnaCoNGA: Nested HW-CNN Co-design

The QSS and HAS loops present a causality dilemma: what comes first, the optimized HW or the CNN quantization strategy? In Sec. II, we mention existing methods which tackle this challenge sequentially or iteratively. To perform true co-design, both HW and CNN need to be jointly and **concurrently** considered.

One approach is to combine HAS genomes with QSS genomes into one GA. However, this would result in a prohibitively complex, large search space (4.77×10^{37} for ResNet20 on the considered bit-serial accelerator), with many direct and indirect relationships between the HW and quantization parameters. This noisy search space would be challenging for the genetic algorithm, even with well-encoded genomes and genetic operators. The complex joint search space would also necessitate larger populations and generations for the GA, leading to excessive GPU hours. Another approach could be to iterate between the search spaces, similar to [13]–[15]. An iterative approach brings us to the same dilemma, since the HW was initially biased for a different quantization strategy, and a newly found HW-CNN combination is sub-optimal with respect to another combination, which had a different quantization strategy prior.

To tackle this challenge, we **nest** two genetic algorithms, as shown in Fig. 3. On the one hand, the HAS GA requires roughly ∼1.5 minutes to execute for 200 generations and 200 HW genomes and can be *parallelized*. This is due to the fast analytical HW-model in Sec. III-A, and the LUT/BRAM utilization models proposed in [5]. On the other hand, the QSS genetic algorithm requires some epochs of fine-tuning to evaluate the accuracy of a potential quantization genome. This can be a costly fitness evaluation process for larger networks and datasets. When nesting the HAS GA into the QSS GA, we can exploit the **speed** of the HAS loop to evaluate the HW design Pareto-front for **each** considered quantization genome (parallel HAS blocks in Fig. 3). In each HAS experiment, a 4-D Pareto-front of HW designs is generated for the respective quantization genome. The solutions in the 4-D HW Pareto-front are checked to see if any of them meet our target HW constraints. If no solution in the HAS Pareto-front satisfies our HW requirements, then the QSS receives a signal to *remove* the genome's fine-tuning step and assign it a null accuracy, without wasting any GPU training time (feedback line from HAS to QSS in Fig. 3). With this approach, the QSS is relieved from optimizing HW metrics and can now be reformulated into a *single-objective* genetic algorithm (SOGA), which is solely *focused* on improving the accuracy of the quantized CNNs. **The QSS essentially allows each quantization genome to evaluate its own HW design space before accepting them into the population.** Therefore, two radically different QSS genomes could meet the target HW constraints (DRAM, computation cycles, BRAM and LUTs) by finding themselves *specialized* HW designs in their respective HAS explorations. This way, the HW design remains ***flexible*** (undefined) on the scale of the overall experiment, but is guaranteed to exist for any genome which is eventually chosen by the QSS at the end of the search. AnaCoNGA's design loops enable the use of analytical HW-models such as [6]–[8], harnessing their speed and flexibility to achieve parallel HW-CNN co-design.

IV. Experiments

A. Experimental Setup

AnaCoNGA is evaluated on CIFAR-10, CIFAR-100, and ImageNet datasets. The 50K train and 10K test images of CIFAR-10 and CIFAR-100 are used to train and evaluate the quantization strategies. ImageNet consists of ∼1.28M train and 50K validation images. After an ablation study, we set the population size and number of generations to 50 for QSS GAs on ResNet20. Probabilities for mutation and crossover are set to 0.5 and 1.0, respectively. For ResNet56, we reduce the running population size $|\mathcal{P}|$ to 25. For ResNet18-ImageNet experiments, $|\mathcal{P}|$ is set to 25 and the number of generations is reduced to 25. The CNNs trained on CIFAR-10 are fine-tuned for 3 epochs and evaluated on 10K random samples during the search. For ImageNet, we fine-tune for 1.5 epochs before evaluating on the valid-set. The quantization method for ResNet20 experiments is DoReFa [1], while deeper (ResNet56) and higher resolution (ResNet18) experiments use the PACT method [2]. Results denoted with (2,4-bit) indicate 2-bit weights and 4-bit activations. For comparison, binarized variants are trained using the XNOR-Net method [3].

We use the Xilinx Z7020 SoC on the PYNQ-Z1 board as the target platform for all HW experiments in Tab. III and Fig. 6, with all designs synthesized at 200MHz target clock frequency. For HAS experiments, we set both the population size and generations to 200, since no significant improvement was observed for larger experiments. Mutation and crossover probabilities are set to 0.4 and 1, respectively. In Tab. III, AnaCoNGA's nested HAS GA uses the respective (2,4-bit) configuration's HW performance as its HW constraint.

B. The Quantization Strategy Search (QSS) Loop

To evaluate the *standalone* QSS space, we fix the accelerator dimensions to measure the HW fitness metrics of each potential solution considered by the GA. We choose the HW3 configuration in Tab. II, as it is one of the BISMO configurations proposed in [5] and used as the standard CNN edge accelerator in [4]. Fig. 5 shows 2-D projections of the 3-D search space of the QSS loop compressing ResNet20 for the CIFAR-10 dataset. To visualize the progress of the algorithm, we plot old-generation Pareto-fronts in grayscale (darker points indicate newer generation Pareto-fronts), while red points belong to the final Pareto-front. The projections reveal a loose correlation between DRAM accesses and compute cycles, as well as a convex Pareto-front between prediction accuracy and HW efficiency. From this Pareto-front of quantization strategies, a solution can be chosen to fit the needs of the application. In Tab. III, we detail the results of standalone QSS solutions chosen for ResNet20, and the more difficult search problem of ResNet56 quantization, which has a larger quantization search space (recall QSS space = Q^{2L}), for both CIFAR-10 and CIFAR-100. The results show standalone QSS producing non-dominated strategies with respect to uniform quantization, on the standard BISMO design (HW3).

C. The Hardware Architecture Search (HAS) Loop

To evaluate the standalone HAS loop, we execute the HW search for uniform 8-bit and 4-bit variants of ResNet20. Fig. 4 shows 2-D projections of the final 4-D Pareto-fronts achieved by HAS, optimizing for computation latency, DRAM accesses, and BRAM and LUT utilization. We notice a clear shift in the HW design space when the quantization strategy changes, even for the tested uniform strategies. Another observation is that the shift is not only due to the more efficient execution metrics of 4-bit vs. 8-bit, but also due to new legal scheduling options on differently dimensioned accelerators. This can be seen in the non-overlapping red and blue markers of the BRAM vs. LUTs 2-D projection plot, indicating different HW dimensions being optimal for the 4-bit and 8-bit CNNs, while respecting the resource limitations of the Z7020 FPGA.

We present further HAS results in Tab. III, applied to the strategies found in the QSS experiments of the previous section (denoted as QSS+HAS). From the resulting Pareto-fronts, we synthesize candidates with the lowest execution and DRAM access cycles, without exceeding the resource utilization of the standard BISMO choice from Tab. II. We note that the GA finds non-trivial asymmetric HW configurations ($\mathtt{D}_m \neq \mathtt{D}_n$), which exploit the position of the tensors W^l and A^{l-1} into either LHS or RHS matrices. The asymmetric HW allows the scheduler to swap the position of weights and activations in the middle of the CNN execution, to maintain high computation efficiency and low DRAM accesses, by reusing the datatype placed in the RHS matrix of the computation. This naturally reduces the LUT and BRAM requirements of the design. In Tab. III, the real HW measurements of sequential co-design (QSS+HAS) show a clear advantage to all standalone QSS CNNs, dramatically lowering their DRAM accesses and latency below or equivalent to a 1-bit strategy executing on standard BISMO dimensions from [4], with less LUT and BRAM required for the design.

D. Analysis of AnaCoNGA Co-Designed Solutions

In Tab. III, we show the results of all the considered networks, datasets, and search combinations executed on **synthesized** hardware. We pair the uniform bitwidth CNNs with the handcrafted edge BISMO variant used in [4]. We notice an improvement in task-related accuracy for *all* AnaCoNGA solutions over sequential co-design (QSS+HAS). This can be attributed to the *accuracy-focused* SOGA implemented in the QSS of AnaCoNGA, which leaves the HW architecture search to be handled by the nested HAS MOGA (recall Fig. 3). Furthermore, the nested HAS allows more diverse, high-accuracy quantization individuals to survive through QSS, as each QSS individual can find their *own* HW design to meet the application constraints.

The latency and DRAM accesses of AnaCoNGA and QSS+HAS variants are comparable to or better than a *single-bit* network executing on the handcrafted accelerator. All AnaCoNGA-based HW designs are ***smaller*** (fewer peak binary TOPS) than HW3, but achieve ***better*** performance due to their tightly-coupled dimensioning, which improves their compute efficiency. To better understand the AnaCoNGA HW performance, we split the total execution time and measure the cycles related to compute, as well as the non-overlapping cycles spent on other parts of the pipeline (stall cycles). We present this data in Fig. 6. Although the HAS genetic algorithm is not aware of pipeline stalls, it optimizes for minimal compute cycles and lower DRAM accesses, where, particularly the latter, is correlated with *lower pipeline stalls*. HW designs with these traits naturally bring down stall cycles, leading to higher compute and memory access overlap. For Fig. 6-a, the AnaCoNGA solution indeed has *higher compute cycles* than 1-bit due to its higher bitwidths, which results in a higher accuracy CNN. However, the DRAM accesses are well-optimized resulting in *fewer stall cycles*, ultimately bringing the total latency of the execution *below* 1-bit on

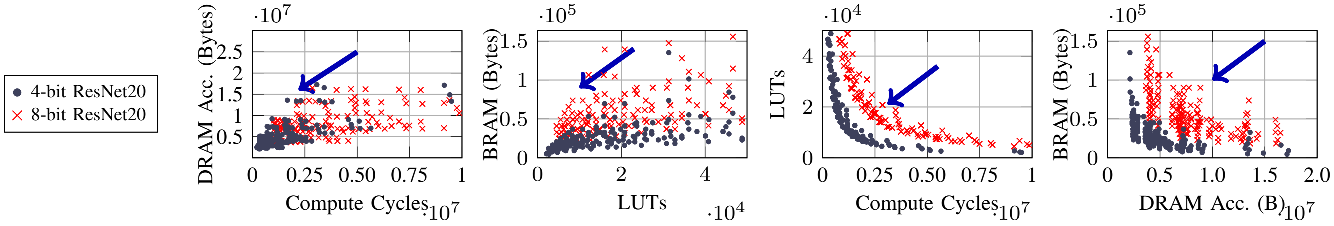


Fig. 4. HAS: 2-D projections of a 4-D Pareto-front in a multi-objective search space. The GA optimizes for HW resources (LUTs, BRAMs) and performance metrics (DRAM accesses, execution cycles) for ResNet20.

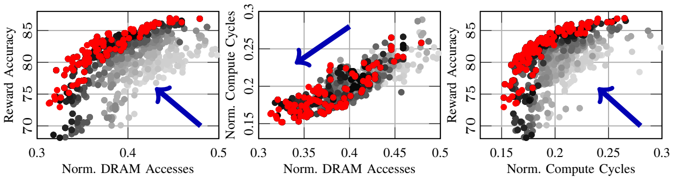


Fig. 5. QSS: 2-D projections of a 3-D Pareto-front for optimal quantization with respect to accuracy, compute cycles, and DRAM accesses on HW3. Compute cycles and DRAM accesses are normalized to an 8-bit execution on HW3. Reward Acc. is before fine-tuning.

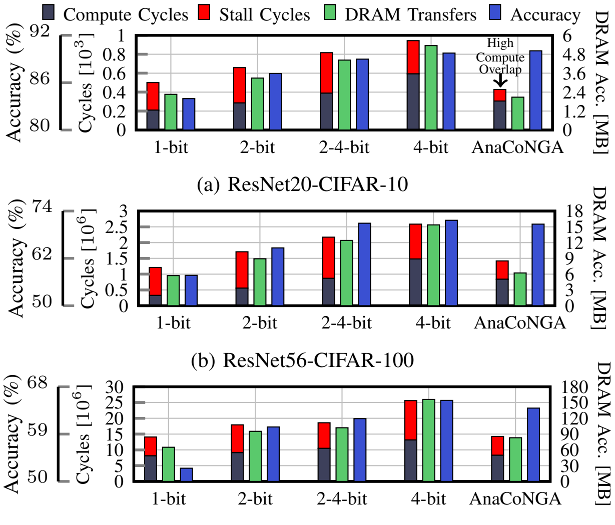
TABLE III
EXPERIMENTS ON CIFAR-10, CIFAR-100 AND IMAGENET DATASETS, WITH RESNET20, RESNET56 AND RESNET18 WORKLOADS. UNIFORM AND STANDALONE QSS ARE EXECUTED ON A STANDARD EDGE VARIANT (HW3) USED IN [4]. **LATENCY AND DRAM ARE MEASURED ON HW.**

Model	**Work**	**Acc** [%]	**LUT** Util	**BRAM** Blocks	**Latency** K. Cycles	**DRAM** Acc. MB
ResNet20 CIFAR-10	XNOR (1-bit) [3]	83.98	32639	135	501	2.26
	DoReFa (2-bit) [1]	87.16			659	3.29
	DoReFa (2,4-bit) [1]	88.98			817	4.43
	DoReFa (4-bit) [1]	89.75			944	5.35
	QSS (standalone)	89.44			798	4.17
	QSS+HAS	89.44	29687	55	422	1.99
	AnaCoNGA	**90.04**	**29671**	**55**	**428**	**2.08**
ResNet56 CIFAR-10	XNOR (1-bit) [3]	85.61	32639	135	1212	5.73
	PACT (2-bit) [2]	90.28			1710	8.93
	PACT (2,4-bit) [2]	92.97			2172	12.41
	PACT (4-bit) [2]	93.27			2585	15.37
	QSS (standalone)	91.89			2120	11.98
	QSS+HAS	91.89	29643	79	1242	5.44
	AnaCoNGA	**92.31**	**29638**	**79**	**1315**	**5.83**
ResNet56 CIFAR-100	XNOR (1-bit) [3]	57.70	32639	135	1212	5.73
	PACT (2-bit) [2]	64.66			1710	8.93
	PACT (2,4-bit) [2]	70.91			2172	12.41
	PACT (4-bit) [2]	71.65			2585	15.37
	QSS (standalone)	69.52			2054	11.60
	QSS+HAS	69.52	29638	79	1240	5.45
	AnaCoNGA	**70.68**	**29643**	**79**	**1420**	**6.23**
ResNet18 ImageNet	XNOR (1-bit) [3]	52.51	32639	135	14090	64.93
	PACT (2-bit) [2]	60.36			17932	95.23
	PACT (2,4-bit) [2]	61.94			18596	102.07
	PACT (4-bit) [2]	65.40			25609	155.85
	AnaCoNGA	**63.94**	**28035**	**123**	**14250**	**83.07**

a handcrafted BISMO design, while maintaining task-related accuracy *higher* than a uniform 4-bit solution. Similar trends can be observed in Fig. 6 for ResNet56 and ResNet18 as well, achieving lower execution metrics than 2-bit CNNs and maintaining high task-related accuracies.

V. Conclusion

We presented an analytical model for a state-of-the-art bit-serial accelerator [5] to circumvent the alternatives of co-design, which resort to synthesis or cycle-accurate simulations. We formulate the co-design problem as two GAs, QSS and HAS, combined in a novel nested framework to eliminate the need for handcrafted reward functions or iterative switching between the HW and CNN domains. With AnaCoNGA, we improve the accuracy of ResNet20-CIFAR-10 by 2.88 p.p. compared to a uniform 2-bit CNN, and achieve a 35% and 37% improvement in latency and DRAM accesses, while reducing LUT and BRAM resources by 9% and 59% respectively, when compared to an edge variant of the accelerator.



(a) ResNet20-CIFAR-10

(b) ResNet56-CIFAR-100

(c) ResNet18-ImageNet

Fig. 6. Breakdown of execution on real, **synthesized** HW. AnaCoNGA reduces latency and DRAM accesses while maintaining high accuracy.

References

[1] S. Zhou *et al.*, “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” *ArXiv*, vol. abs/1606.06160, 2016.

[2] J. Choi *et al.*, “PACT: parameterized clipping activation for quantized neural networks,” *ArXiv*, vol. abs/1805.06085, 2018.

[3] M. Rastegari *et al.*, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *ECCV*, 2016.

[4] K. Wang *et al.*, “HAQ: Hardware-Aware Automated Quantization With Mixed Precision,” in *CVPR*, 2019.

[5] Y. Umuroglu *et al.*, “Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing,” in *FPL*, 2018.

[6] Q. Huang *et al.*, “Cosa: Scheduling by constrained optimization for spatial accelerators,” *ArXiv*, vol. abs/2105.01898, 2021.

[7] A. Parashar *et al.*, “Timeloop: A Systematic Approach to DNN Accelerator Evaluation,” in *ISPASS*, 2019.

[8] S.-C. Kao *et al.*, “Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm,” in *ICCAD*, 2020.

[9] Q. Lou *et al.*, “Autoq: Automated kernel-wise neural network quantization,” in *ICLR*, 2020.

[10] T. Wang *et al.*, “Apq: Joint search for network architecture, pruning and quantization policy,” in *CVPR*, 2020.

[11] P. Xu *et al.*, “Autodnnchip: An automated dnn chip predictor and builder for both fpgas and asics,” in *FPGA*, 2020.

[12] R. Venkatesan *et al.*, “Magnet: A modular accelerator generator for neural networks,” in *ICCAD*, 2019.

[13] Y. Lin *et al.*, “Neural-hardware architecture search,” in *NeurIPS-W*, 2019.

[14] J. W *et al.*, “Hardware/software co-exploration of neural architectures,” *TCAD*, 2020.

[15] W. Jiang *et al.*, “Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start,” *TCAD*, 2020.