CASCO: Cascaded Co-Optimization for Holistic Neural Network Acceleration

Bahador Rashidi
Huawei Technologies Canada Co., Ltd
Edmonton, Canada
bahador.rashidi@huawei.com

Shan Lu
Huawei Technologies Canada Co., Ltd
Edmonton, Canada
shan.828346@huawei.com

Kiarash Aghakasiri
Huawei Technologies Canada Co., Ltd
Edmonton, Canada
kiarash.aghakasiri1@huawei.com

Chao Gao
Huawei Technologies Canada Co., Ltd
Edmonton, Canada
chao.gao4@huawei.com

Fred Xuefei Han
Huawei Technologies Canada Co., Ltd
Edmonton, Canada
fred.xuefei.han1@huawei.com

Zhisheng Wang
Huawei Technologies Co., Ltd.
Shanghai, China
wangzhisheng1@hisilicon.com

Laiyuan Gong
Huawei Technologies Co., Ltd.
Shanghai, China
gonglaiyuan@hisilicon.com

Fengyu Sun
Huawei Technologies Co., Ltd.
Shanghai, China
sunfengyu@hisilicon.com

***Abstract*—Automatic design space exploration of neural accelerators has become essential to maintain high productivity in deep learning acceleration. Previous accelerator co-design approaches mainly focus on exploring hardware and software design spaces assuming individual mapping of neural network layers is deployed on hardware independently while neglecting the vast yet crucial joint effect of layer fusion. To address this shortcoming, we propose CASCO, a cascaded and holistic co-optimization aware of all co-design aspects, including the accelerator's hardware architecture, on-chip operator mapping, and off-chip layer fusion optimization. We then propose an efficient joint-optimization algorithm framework for exploring the joint-design space efficiently by adaptive search resource allocation. Empirical results show that CASCO outperforms the existing co-design framework HASCO by a large margin. Especially, when training on the same networks, CASCO produces generalizable HW design on unseen neural network applications with EDP reduction from $1.4\times$ to $3.2\times$.**

I. Introduction

Many contemporary applications of artificial intelligence (AI) are built on deep neural networks (DNNs), including image recognition [1], image synthesis [2], natural language processing [3]. The ability of DNNs to create powerful representations from raw sensory information accounts for their superiority. However, a significant quantity of data must be provided as input for DNNs to create useful representations, which raises the cost of computational complexity. Although conventional graphics processing units (GPUs) dominate the market today for DNN computing gear, there are an increasing number of specialized DNN accelerators available, e.g. Google TPU [4], Diannao [5], Eyeriss [6], Ascend [7], etc.

To keep up the rapid evolution of neural network architectures, developing a holistic solution combining HW acceleration and SW mapping onto the specialized HW has been identified as essential for tensor applications [8]–[10]. This HW-SW co-design features fast execution by taking advantage of parallel computation while preserving high energy efficiency by producing optimal hardware and software designs jointly. Some of existing HW-SW co-design are HASCO [8], FAST [9], and DIGAMMA [10]. One innovation of HASCO is the design of an intermediate representation (IR) to reduce the size of the joint HW-SW search space, but the resulting design space is still in the order of billions. The second innovation of HASCO is the use of the Multi-objective Bayesian optimization (MoBo) algorithm to regulate the exploration of the hardware design space and the use of FlexTensor [11] as a subroutine to facilitate software mapping exploration. FAST optimizes both hardware design parameters and software mapping options concurrently. FAST employs the black-box optimizer Vizier [12] to explore the HW configurations design space and employs random search (i.e. the default SW mapping search in TimeLoop [13]) for mapping optimization. As a post-processing procedure, FAST uses an operator fusion integer linear programming (ILP) solver for operator fusion. DIGAMMA [10] is an HW-SW co-design solution that searches for accelerator HW and SW configurations simultaneously. This solution is predicated on HW-SW intermediate encoding of MAESTRO [14], and uses evolutionary search for co-exploration.

Instead of executing the neural network layer by layer, which could incur substantial communication costs between on-chip and off-chip memory data transmission, with layer-fusion, the computation can be reorganized so that multiple layers are processed on-chip concurrently. For large neural networks with high-dimensional features, the exploitation of on-chip data locality can significantly alleviate the strain of off-chip memory communication. In prior work, individuals

979-8-3503-4859-0/DATE24/© 2024 EDAA

have investigated opportunities for reusing on-chip data [6], [15] while the hardware setting is fixed and proposed different network scheduling techniques [15], [16] to reduce off-chip memory traffic during neural network inference. Determining the optimal layer fusion schedule for a given DNN workload and accelerator architecture is difficult and necessitates the exploration of a vast design space containing numerous trade-offs. Often, layer fusion scheduling relies on a description of the device's memory layout in order to determine the bandwidth and capacity of each section. For example, DeFiNES [17] presents a tool that enables fast exploration of the depth-first scheduling space for DNN accelerators through analytical modeling. DeFiNES shows good modeling accuracy at the end-to-end neural network level, which enables it to achieve up to 10× speedup than other solutions. In [18], a framework called Optimus is developed that supports both offline and online layer fusion scheduling; it uses an analytical memory cost function to direct the search procedure toward a memory-optimal fusion solution. Experiment results demonstrate that Optimus reduces off-chip memory accesses and obtains energy efficiency over the baselines on processors of different architectures and dataflows. The follow-up work Olympus [19] further introduces a comprehensive analysis of memory optimization techniques for DNN processors and presents a theoretical lower bound of memory overhead for DNNs. Other approaches [15], [20]–[22] follow similar schemes but vary in detail.

In this paper, we develop an off-chip memory-aware HW-SW co-optimization framework to automatically achieve low computational latency, energy usage, and production cost for the DNN accelerator by imposing layer fusion to the HW-SW joint-design optimization. Specifically, our contributions are summarized as follows.

- We propose a new HW-SW co-optimization framework, which takes in a DNN model and generates an accelerator design configuration that aims to run the model as a whole efficiently. We include off-chip memory cost to the co-optimization procedure by leveraging layer fusion.
- To explore the large co-design spaces, we propose an efficient multi-level cascaded co-optimization algorithm. For hardware search, we introduce a novel Multi-objective Bayesian optimization (MoBo) that uses batch sampling with an adaptive surrogate model learning procedure for generating high-quality design points. For software search, we propose a genetic algorithm to help guide achieving optimal layer-fusion, hence minimizing overall energy-delay production efficiency. This together forms a three-level co-optimization process where we adopt *successive halving* for adaptive search budget spending to timely explore this ultra-large joint search space.

II. CASCO

Fig. 1 shows the architecture of the CASCO. In stage I, a synchronized search for the fusion and operator mapping is conducted. The outer layer, Stage II, forms a search for the hardware configuration that is coupled with the selected choice in Stage I.

A. Accelerator HW Architecture Search

a) Design Space: We use hardware primitives to define hardware design space for exploration. The primitives are categorized into two parts: the *hardware architecture primitives* and *dataflow primitives*. The architecture primitives include memory level, memory capacity, PE number, PE shape, and network-on-chip (NoC) topology. The memory level describes the number of different levels of on-chip and off-chip buffers e.g. L1, L2, and DRAM. The memory capacity indicates the capacity at each level. The PE number and shape specify the PE array shape and scale. NoC topology is an abstraction of how each PE can communicate with the other. The dataflow primitives are defined by partitioning size, loop unrolls, and replication loop, where all these three primitives define a subset of loop organizations.

b) MoBo with Early Stopping: A MoBo is used to guide HW search space exploration. To enable effective learning of the surrogate model in MoBo, we propose a high-fidelity update strategy for adaptive data selection. This high-fidelity update strategy ensures that only high-fidelity HW configurations are used to update MoBo surrogate model, bringing the following advantages into HW-SW co-exploration by training the surrogate model more appropriately:

- It increases the probability of finding a better HW configuration that leads to a better PPA.
- It improves the search efficiency for finding the global optimum design point with the unique HW configuration, layer fusion group, and SW mapping.

Specifically, it works as follows. CASCO first samples a batch of N hardware candidates in each MoBo iteration, to minimize the acquisition function (e.g. Hyper-volume improvement). The software-stack search is conducted concurrently for each hardware within the batch. We allocate more software-level exploration budgets to *promising* hardware candidates while less on unfavorable ones using a novel early-stopping rule, which is explained next. Then, at the end of each MoBo iteration, the surrogate model is refined with high-fidelity data points collected from that iteration by using the high-fidelity update strategy. A fidelity scalar, v_{ParEGO} is defined as follows:

$$v_{ParEGO} = \max_{j \in 1,2,3,4} (w_j y_j) + \rho Y^T W \tag{1}$$

Here, the L2-norm distance $d = ||v_{ParEGO - v_{ParEGO}^{Best}}||_2$ is measured for each HW configuration, and v_{ParEGO}^{Best} is the up-to-date smallest fidelity scalar value by the current iteration. We introduce an upper update limit (UUL) parameter as the selection threshold for high-fidelity HW configurations such that HW configurations with $d \leq UUL$ are selected to update the surrogate model. The d of these high-fidelity HW configurations is added to a dataset D. We update UUL at each iteration as α-percentile of D, with the confidence of 95% by default. In this way, D is always updated with new high-fidelity HW configurations. As UUL tends to decrease over time, pushing the selection criterion becomes more strict as iteration advances on. This significantly increases the likelihood of selecting true high-quality data points.

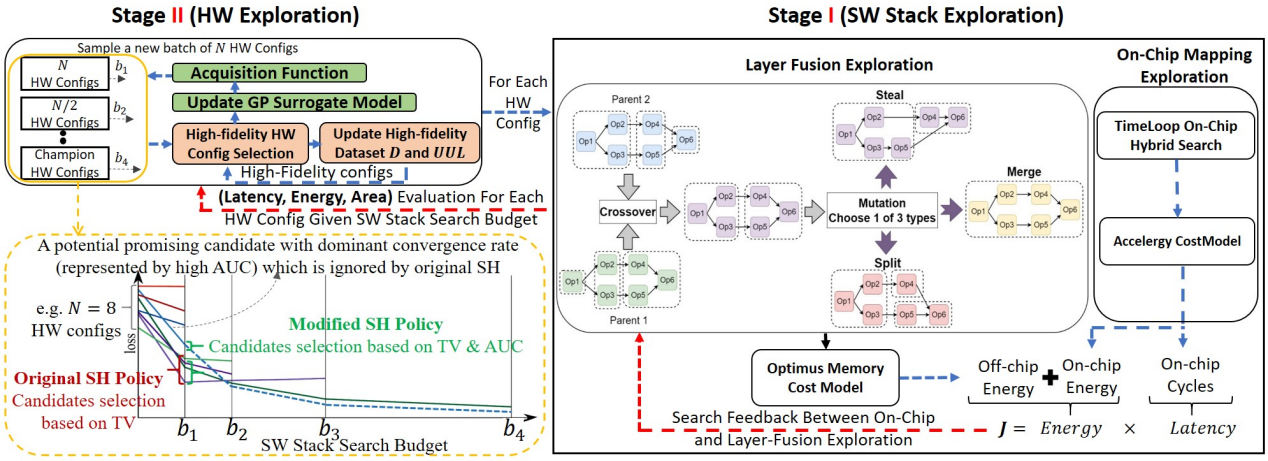

Fig. 1. CASCO full-stack accelerator exploration workflow

Another key component of the CASCO's HW search is to stop early less-promising hardware configuration(s) by monitoring their intermediate performance during software-level exploration. With that aim, CASCO integrates a successive halving (SH) [23] within the MOBO HW search process. This successive halving enables concurrent software mapping search with parallel hardware evaluation while discarding the unfavorable design points from the huge search space on the fly. More specifically, a batch of hardware candidates is sampled in each MOBO iteration. After the batch is sampled, a software-stack search is conducted concurrently for each hardware. In CASCO, we introduce a modified version of SH (MSH) such that for candidate selection, it considers both terminal value (TV) and area under the curve (AUC) for dropping low-quality design points. By contrast, the original SH selects the succeeding candidates only based on terminal value (TV) at the end of the current round of budget b_i such that only the best half of candidates are selected for further exploration. Here, our new rule is designed with the observation that the HW configurations with relatively steep convergence rates are also likely to be promising. In other words, if those steep convergence candidates were given a second chance to be evaluated with a higher budget in the next round (i.e. selected as succeeding candidates), they might outperform those with the best TVs.

B. SW Stack Schedule Search

Cost Model. As shown in Fig. 1, in CASCO, we develop a genetic algorithm (GA) to find the best fusion groups with the lowest co-design objective $J = (\text{Off-Chip Energy} + \text{On-Chip Energy}) * cycles$. Off-chip energy is calculated solely from the fusion groups in a specific candidate. An operator mapping search is required to calculate the on-chip energy and latency for each unique fusion group. Hence, we perform operator search as a subroutine for the layer fusion search. The off-chip energy consumption is measured by the memory cost model as in Optimus [18], which captures the minimum achievable off-chip memory overhead for the fused layer groups.

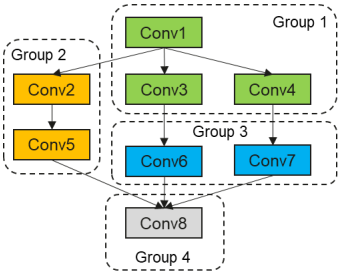

Fig. 2. Example of DAG-based operator fusion.

We adopt the heuristic search method as in Timeloop [13] for mapping search.

Search Space for Layer Fusion. CASCO conducts layer fusion based on *directed acyclic graph (DAG)* representation of a DNN. Many DNNs can be depicted as DAGs $D = (N, E)$, where N represents *nodes* in the neural networks and E is the connections between those nodes. The operator fusion partitions the network model into fused operator groups, e.g., $(Group1, Group2, Group3, Group4)$, as in Fig. 2. From the DAG representation, we can extract a scheduled space for layer fusion.

For the DAG D given in Fig. 2, let $N = \{n_1, \ldots, n_8\}$ represent the operator $\{Conv1, \ldots, Conv8\}$ respectively, and $E = \{e_1, \ldots, e_9\}$. Since the output node, n_l, depends on some previous nodes $\{n_1, \ldots, n_{l-1}\}$, the fusion of nodes inside D cannot be arbitrary. For example, the group $\{n_5, n_8\}$ is invalid, as the calculation of n_8 is dependent on $\{n_5, n_6, n_7\}$. Also, it is worth noting that there exist groupings such as $\{n_1, n_2\}$, $\{n_1, n_3\}$ and $\{n_1, n_4\}$, that all three groups involve node n_1. This incurs the recalculation of n_1. Group $\{n_1, n_2, n_3, n_4\}$ computes n_1 only once and has the best computational efficiency. However, this scheme may require a large on-chip buffer, which is unavailable on edge devices. An observation to leverage in layer fusion is that fusing the nodes into the same group when they share no data dependencies brings no benefits, e.g. group 3 in Fig. 2. We exploit data dependencies either due to sharing the same input with another node in the same fusing group or producing the input data of a node in the

same group.

Evolutionary Search. Prior work [18] relies on dynamic programming (DP) for searching fusion groups. The main idea behind DP is to traverse all valid groups and select the best combination. DP finds the global optimal fusion strategy with a time complexity of $O(2^{max_i|succ(n_i)|}|N|)$, where $succ(n_i)$ is the number of successor nodes. DP search time grows exponentially with the input network size, and we find it less feasible, especially in a co-search setting. Therefore, we propose to use evolutionary search (EA). Our EA algorithm starts with a random set of fusion plans as the initial population. At each step, we select the current best population based on the objective and perform crossover and mutation operations to create the next generation. The new plans are then evaluated and added to the population. This procedure repeats until we reach a pre-defined search budget. The objective for EA is EDP: $J = (\text{Off-Chip Energy} + \text{On-Chip Energy}) * cycles$.

To explore the search space more efficiently, we propose a novel procedure to crossover and mutate existing fusion plans. Specifically, if the network contains N nodes $\{n_1, \ldots, n_N\}$, our crossover procedure randomly selects two parent fusion plans $P_1 = \{Group_1, \ldots\}$ and $P_2 = \{Group_x, \ldots\}$ from the top population. We first visit n_1 and find its containing groups $Group_1$, $Group_x$. We then randomly select one of $Group_1$, $Group_x$ as the new group G for n_1 and mark all nodes in G as visited. Finally, we add G to the new child fusion plan and repeat the same procedure on the next un-visited node n_i until all nodes are visited. After crossover, we randomly apply one of the following mutation procedures:

- **Merge**: randomly merges two neighboring groups into one larger fusion group;
- **Split**: randomly splits a group into two smaller groups;
- **Steal**: randomly selects a boundary node from a group and adds it to a neighboring group.

The crossover operation enables the next generation to inherit and mix favorable groups from top parents. The merge and split mutations allow significant modifications to the child groups, while the steal mutation enables minor changes. Together, these strategies offer balanced exploitation and exploration. Another distinct advantage of EA is that we can freely assign the number of generations to control the search time, whereas in DP we must wait for the entire procedure to finish. The flexibility in search budget allocation is crucial for a co-search framework.

III. Evaluation

Our evaluation contains two parts. First, CASCO is compared with fixed HW configurations baselines [6], [24], [25]. This shows the advantage of conducting co-optimization by being aware of all three design aspects. Second, CASCO is compared with HW-SW co-optimization baselines. By this comparison, we aim to show the advantages of CASCO from two perspectives: 1) the inclusion of graph layer fusion into co-design, and 2) the use of advanced searching methods for effective design space exploration.

A. Evaluation Environment

Design Constraints: Three different spatial accelerators, i.e., EdgeTPU [25], Eyeriss [6], and Shidiannao [24], are selected in our first experiments. To compare with each baseline architecture, CASCO is conducted using the corresponding hardware configurations as the resource constraints for defining the design space. This includes the maximum number of PEs, the maximum on-chip memory capacity, and the maximum off-chip memory capacity. For all the comparisons, CASCO searches the number of PEs at a stride of 1, on-chip memory capacity at a stride of 8B, and off-chip memory capacity at a stride of 16KB. To demonstrate the end-to-end performance of our proposed approach, CASCO is evaluated on individual networks (including MobileNet, AlexNet, SqueezeNet, VggNet, ResNet18, GoogleNet) under edge (power $\leq$ 2W) constraints. Area constraint ($5e3\, um^2$) is also put into consideration. These two constraints result in the number of PEs $\leq$ 384 and L2 $\leq 4KB$. After the search is done, we pick a HW configuration with the min-Euclidean distance from the found Pareto-front set. We then compare the PPA (latency, power, area) for different hardware configurations.

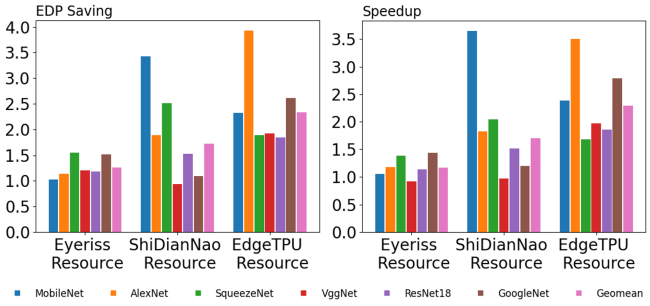

Fig. 3. Speedup and EDP savings when CASCO co-explores for several network benchmarks within the baseline hardware resources.

B. Comparison with Existing Design Architectures

Fig. 3 shows the speedup and energy savings when using CASCO for 6 network benchmarks (MobileNet, AlexNet, SequeezeNet, VggNet, ResNet18 and GoogleNet) within the baseline hardware resources, i.e., Eyeriss [6], ShiDianNao [24], and edgeTPU [25]. For each case, CASCO is conducted using the corresponding hardware configurations as the constraints for design space exploration. For this experiment, CASCO achieves 1.2×, 1.7×, and 2.3× speedup and 1.3×, 1.7×, and 2.3× EDP savings on average compared with Eyeriss, ShiDianNao, and edgeTPU, respectively. The results confirm the superiority of CASCO co-design optimization, which is capable of finding better accelerators that consume less energy with better performance than the well-known accelerators. It also demonstrates the portability of CASCO: as a three-level co-exploration architecture, it can be applied to different accelerators to reduce their energy consumption while improving performance by delivering a unique combination of hardware, software, and layer-fusion configurations.

C. Comparison with Co-Design Methods

We compare CASCO with one most recent co-design baseline HASCO [8]. To make HASCO and CASCO comparable, we let HASCO use the same software mapping search offered by Timeloop. To show the advantage of CASCO's adaptive

TABLE I

COMPARISONS BETWEEN CASCO AND TWO BASELINE: EACH NETWORK CO-OPTIMIZED SEPARATELY.

Methods	Networks	AlexNet	MobileNet	GoogleNet	SqueezeNet	ResNet18	ResNet50	UNet	SRGAN	VIT
Baseline 1 (HASCO)	Energy (mJ)	7.0	218.0	167.0	92.4	42.8	795.0	287.3	249.3	98.7
	Latency (Ms)	3.6	3.4	6.8	3.9	10.7	26.3	92.9	82.6	12.2
	Area ($um^2 * 10^3$)	**3.3**	5.0	5.0	4.9	**4.2**	**4.7**	7.5	**6.3**	3.1
Baseline 2 (HASCO + Optimus)	Energy (mJ)	1.4	**1.6**	3.8	1.4	4.7	10.9	**28.3**	21.8	**33.2**
	Latency (Ms)	2.2	2.2	6.9	3.5	7.6	17.7	232.8	36.4	**2.1**
	Area ($um^2 * 10^3$)	5.0	4.6	**4.2**	4.9	4.9	5.0	**5.5**	6.9	3.1
CASCO	Energy (mJ)	**1.3**	**1.6**	**3.5**	**1.3**	**4.6**	**10.8**	29.5	**20.9**	**33.2**
	Latency (Ms)	**2.1**	**2.0**	**6.1**	**2.1**	**7.0**	**12.6**	**90.1**	**35.2**	2.3
	Area ($um^2 * 10^3$)	4.2	**4.2**	**4.2**	**4.6**	4.7	5.0	**5.5**	6.6	**2.2**

TABLE II

COMPARISON BETWEEN CASCO AND TWO BASELINES: CO-OPTIMIZATION ON TWO NETWORKS, GENERATED HW VALIDATED ON FOUR NETWORKS

Methods	Evaluation Metrics	Training Networks	Validation Networks			
		ResNet18, SRGAN	MobileNet	ResNet50	UNet	VIT
Baseline 1 (HASCO)	Area ($um^2 * 10^3$)	5.5				
	Energy (mJ)	292.7	218.0	795.6	300.2	98.8
	Latency (Ms)	105.4	**3.6**	**23.8**	430.1	2.6
	EDP	30850.6	784.8	18935.3	129116.0	256.9
Baseline 2 (HASCO + Optimus)	Area ($um^2 * 10^3$)	**3.2**				
	Energy (mJ)	**25.9**	**1.7**	**10.7**	**26.9**	**33.2**
	Latency (Ms)	260.5	14.7	90.9	209.6	2.8
	EDP	6747.0	25.0	972.6	5638.2	93.0
CASCO	Area ($um^2 * 10^3$)	**3.2**				
	Energy (mJ)	26.8	**1.7**	11.0	31.9	**33.2**
	Latency (Ms)	**91.7**	4.5	33.0	**123.1**	**1.6**
	EDP (reduction w.r.t Baseline 2)	**2457.6** (2.74×)	**7.7** (3.24×)	**363.0** (2.67×)	**3926.9** (1.43×)	**53.1** (1.75×)

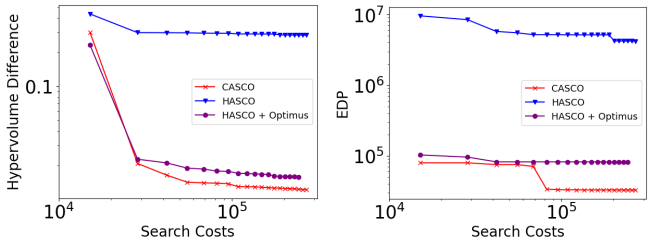

Fig. 4. Comparisons of CASCO and baseline co-exploration methods regarding Hypervolume Difference and EDP convergence.

search, we then create another new and challenging baseline by stacking layer-fusion engine *Optimus* [18] and HASCO.

1) Improvement Due to Adding Layer-Fusion into Co-Design: As shown in Table. I, compared to the HASCO, CASCO reduces the energy consumption by 5.8× and achieves speedups by 2.2× on average for all the networks. Since CASCO includes the layer fusion search, each fused layer group is loaded to the on-chip memory for mapping search leading to energy/latency reduction. Without layer fusion, every network layer is loaded to the on-chip memory for mapping search, as in HASCO. Therefore, HASCO's design often needs a smaller on-chip memory along a smaller area. Overall, CASCO outperforms HASCO by orders of magnitude with respect to energy and latency altogether.

2) Improvement Due to CASCO's Adaptive Search: This HASCO[8] + Optimus[18] baseline represents an aggressive rival for CASCO since it also contains three optimization levels, i.e., HW, layer fusion, and operator scheduling. As in Table I, CASCO still outperforms this baseline by 3.8× on average among all application workloads. This result shows that CASCO is not a simple stacking of existing frameworks. Instead, it employs a novel multi-level co-exploration algorithm consisting of a more effective Multi-objective Bayesian optimization and its conjunctive use of successive-halving for adaptive search. To illustrate how CASCO's search behavior, we report in Fig. 4 the hyper-volume convergence for each co-optimization method. The comparison indicates that, because of a more intelligent search, CASCO achieves not only better PPAs but also yields faster convergence.

D. Generalization Abilities

In previous results, we conducted co-optimization for each network separately. In practice, it is more preferred that one *HW* should support *multiple* workload networks. To see if CASCO can support this scenario, we conduct another set of experiments by doing co-optimization on two training networks and performing generalization experiments on 4 new networks. Table II summarizes the results. Consistent with Table I, these results show that CASCO produces a generalizable HW design, again outperforming the other two co-design methods. Indeed, including graph-level optimization into co-design optimization does not fundamentally change the HW being generated, since the HW design template remains the same as the co-design method HASCO. In other words, graph-level optimization serves as a high-level *regularization* that drives the co-optimization to produce HW also friendly for on/off-chip memory communication cost. This is an important aspect that is missing from previous co-design optimization architectures. We also see from Table II that simply stacking existing graph-level optimization and co-design method did not produce satisfactory results. This confirms that the advanced search algorithm by CASCO is another necessary component for such three-level co-design space exploration.

IV. Conclusion

We have presented CASCO, a cascaded optimization framework that can achieve holistic optimization for given neural networks by jointly optimizing layer fusion, operator scheduling, and HW parameters. Experiments showed that CASCO can help to find better HW design points compared to other co-design frameworks. CASCO achieved up to $2.3\times$ EDP saving compared to common DNN accelerators, and it performs up to $5.8\times$ better in terms of energy consumption compared to SoTA HW/SW co-design results. Different experiment settings also confirm that CASCO's superiority is not merely a result of stacking layer fusion into the co-design framework. The algorithm for exploring the vast joint-design space adaptively is another reason for its strong empirical results. In the future, one interesting direction is integrating CASCO with other platforms, e.g., DeFiNES [17], where more comprehensive layer fusion search space can be extracted with advanced analytical estimation of different fusion strategies.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 684–10 695.

[3] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[4] T. Norrie, N. Patil, D. H. Yoon, *et al.*, "Google's training chips revealed: Tpuv2 and tpuv3," in *2020 IEEE Hot Chips 32 Symposium (HCS)*, Oct. 2020. DOI: 10.1109/hcs49909.2020.9220735. [Online]. Available: http://dx.doi.org/10.1109/hcs49909.2020.9220735.

[5] T. Chen, Z. Du, N. Sun, *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, 2014.

[6] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[7] H. Liao, J. Tu, J. Xia, *et al.*, "Ascend: A scalable and unified architecture for ubiquitous deep neural network computing: Industry track paper," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2021, pp. 789–801.

[8] Q. Xiao, S. Zheng, B. Wu, P. Xu, X. Qian, and Y. Liang, "Hasco: Towards agile hardware and software co-design for tensor computation," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2021, pp. 1055–1068.

[9] D. Zhang, S. Huda, E. Songhori, *et al.*, "A full-stack search technique for domain optimized deep learning accelerators," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 27–42.

[10] S.-C. Kao, M. Pellauer, A. Parashar, and T. Krishna, "Digamma: Domain-aware genetic algorithm for hw-mapping co-optimization for dnn accelerators," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 232–237.

[11] S. Zheng, Y. Liang, S. Wang, R. Chen, and K. Sheng, "Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 859–873.

[12] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 1487–1495.

[13] A. Parashar, P. Raina, Y. S. Shao, *et al.*, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, IEEE, 2019, pp. 304–315.

[14] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, 2020.

[15] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–12.

[16] X. Yang, M. Gao, Q. Liu, *et al.*, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 369–383.

[17] L. Mei, K. Goetschalckx, A. Symons, and M. Verhelst, "Defines: Enabling fast exploration of the depth-first scheduling space for dnn accelerators through analytical modeling," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2023, pp. 570–583.

[18] X. Cai, Y. Wang, and L. Zhang, "Optimus: An operator fusion framework for deep neural networks," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 1, pp. 1–26, 2022.

[19] X. Cai, Y. Wang, K. Tu, C. Gao, and L. Zhang, "Olympus: Reaching memory-optimality on dnn processors," *IEEE Transactions on Computers*, vol. 71, no. 8, pp. 1939–1951, 2021.

[20] X. Chen, Y. Han, and Y. Wang, "Communication lower bound in convolution accelerators," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2020, pp. 529–541.

[21] Y. Xing, S. Liang, L. Sui, *et al.*, "Dnnvm: End-to-end compiler leveraging heterogeneous optimizations on fpga-based cnn accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2668–2681, 2019.

[22] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren, "Dnnfusion: Accelerating deep neural networks execution with advanced operator fusion," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 883–898.

[23] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Artificial intelligence and statistics*, PMLR, 2016, pp. 240–248.

[24] Z. Du, R. Fasthuber, T. Chen, *et al.*, "Shidiannao: Shifting vision processing closer to the sensor," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 92–104.

[25] A. Yazdanbakhsh, K. Seshadri, B. Akin, J. Laudon, and R. Narayanaswami, "An evaluation of edge tpu accelerators for convolutional neural networks," *arXiv e-prints*, arXiv–2102, 2021.