Optimus: An Operator Fusion Framework for Deep Neural Networks

XUYI CAI, Institute of Computing Technology, Chinese Academy of Sciences; University of Chinese Academy of Sciences, China
YING WANG, Zhejiang Lab; State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, China
LEI ZHANG, Institute of Computing Technology, Chinese Academy of Sciences, China

The reduction of neural parameters and operations for the applications on embedded and IoT platforms in current deep neural network (DNN) architectures has received increasing attention. Relatively, the intermediate feature maps of such lightweight neural networks begin to grow and usually outsize the on-chip memory as the new bottleneck, which introduces considerable power-consuming off-chip memory accesses. To reduce the feature-induced memory accesses, operator fusion has been proposed to parallelize the execution of multiple convolutional layers and shown significant reduction of off-chip memory accesses. However, how to fuse the neural operators is still a challenging issue that heavily depends on both the neural network (NN) topology and the specific DNN accelerator configuration. In this work, we observed prior operator fusion approaches fail to guarantee memory-level optimality as they search in the constrained operator fusion design space. Considering the complexity of the NN topologies and the constrained resources of the DNN accelerators, we develop a novel operator fusion framework, Optimus. Optimus includes an accurate memory cost model dedicated to the scheduler to evaluate the potential operator-fusion schemes and a directed acyclic graph-based operator fusion algorithm for both off-line and on-line workload deployment scenarios, which altogether generates high-efficiency operator-fusion solutions for arbitrary network models running on DNN accelerators. The experimental results show that Optimus reduces 17–75% off-chip memory accesses and obtains 1.86×–3.66× energy efficiency on state-of-the-art DNN workloads when compared to the baselines and brings significant power-efficiency boost to the DNN accelerators of different architectures and dataflows.

CCS Concepts: • **Hardware → Emerging languages and compilers**; **Emerging tools and methodologies**; • **Computer systems organization → Neural networks**; **Embedded systems;**

Additional Key Words and Phrases: Neural network, embedded processor, memory, layer fusion

This work was supported by the National Natural Science Foundation of China (under Grants No. 61874124, No. 61876172 and No. 62090020), and the Strategic Priority Research Program of Chinese Academy of Science (under Grant No. XDC05030201).

Authors' addresses: X. Cai, Institute of Computing Technology, Chinese Academy of Sciences; University of Chinese Academy of Sciences, Beijing, China; email: caixuyi20b@ict.ac.cn; Y. Wang (corresponding author), Zhejiang Lab; State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; email: wangying2009@ict.ac.cn; L. Zhang, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; email: zlei@ict.ac.cn.

1539-9087/2022/10-ART1 $15.00
https://doi.org/10.1145/3520142

ACM Reference format:
Xuyi Cai, Ying Wang, and Lei Zhang. 2022. Optimus: An Operator Fusion Framework for Deep Neural Networks. *ACM Trans. Embedd. Comput. Syst.* 22, 1, Article 1 (October 2022), 26 pages.
https://doi.org/10.1145/3520142

1 INTRODUCTION

Deep neural networks (DNNs) are currently the most effective solution for many challenging problems, e.g., computer vision, natural language processing, and speech recognition, and so on. Their hardware accelerators also become an emerging need to enable embedded usage. Although computational optimization has been extensively explored, the energy efficiency of such accelerators remains limited by off-chip memory access.

State-of-the-art DNNs have millions of parameters and high-dimension **feature maps (fmaps)** that usually cannot fit in the on-chip memory of edge DNNs accelerators and hence induce a large amount of off-chip memory traffic. Since the energy cost of accessing the main memory is orders of magnitude higher than arithmetic operations, off-chip memory accesses will account for most of the energy consumption in the DNNs accelerators [2, 7, 9, 29, 40] and become the performance bottleneck. As shown in Figure 1, the main memory access consumes most of energy in EIE [16], DianNao [6], and Cambricon-X [46]. Although modern neural network processors, e.g., References [9, 16, 34, 46, 48, 49], put more emphasis on the compression of neural parameters to reduce the parameter-induced memory accesses of network inference, unfortunately, as presented in Figure 2, with the development of deep learning algorithms, state-of-the-art lightweight network architectures [18, 20, 37] exhibit a clear design trend that the size of intermediate feature maps generated by neural operators far exceeds the size of parameters. It means sometimes the intermediate feature maps become the main memory bottleneck for the edge DNN accelerators rather than parameters [2, 19, 28, 50].

However, the technology of operator fusion [2, 43, 50] focuses on reducing the feature-induced memory accesses. As illustrated in Figure 3, *operator fusion* technique partitions the network models, which are represented as **directed acyclic graphs (DAGs)**, into *fused operator-groups*, e.g., `(G1, G2, G3, G4)` in Figure 3(a) and `(G5, G6, G7)` in Figure 3(b). Within a fused operator-group, the intermediate feature maps could be consumed immediately after producing by scheduling in advance the corresponding operations of the consuming operators. Otherwise, they will be evicted to the off-chip memory and re-loaded again from the main memory to the compact on-chip memory before the next operator is started. Thus, it is viable to employ the operator fusion technique to improve the performance of many existing deep learning accelerators and reduce the memory access overhead.

Nevertheless, to achieve the optimal operator fusion for complicated NN topologies on the DNN accelerators remains an unaddressed and non-trivial problem. First, in the DNN model, the various partitions of fused operator groups, as shown in Figure 3, lead to different memory overheads. The search space is enormous due to the combinatorial explosion, and it is an non-deterministic polynomial(NP) hard problem [30] to explore the entire operator fusion space for a complicated NN model. The heuristic algorithms proposed in previous works cannot guarantee the global optimum. Second, previous works assumes that the depended parameters of the whole fused operator-group must be kept in the on-chip memory and they do not allow parameters-induced memory accesses when processing a fuse operator-group. However, it is very common that a given off-the-shelf DNN accelerator cannot accommodate all parameters of the fused operator-group. On the contrary, allowing parameter-induced memory access during the computation of a fused-operator

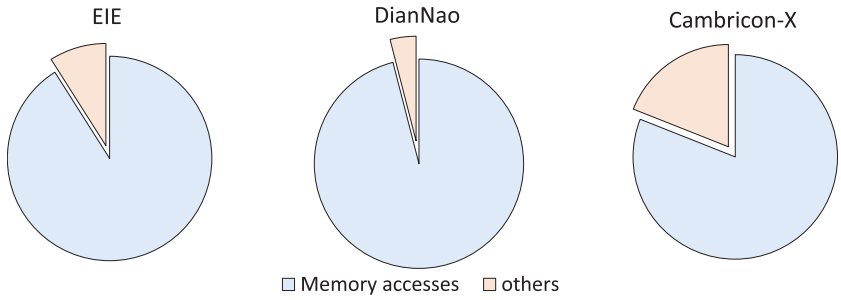

Fig. 1. Energy breakdown of state-of-the-art DNN accelerators, and the statistics are reported from EIE [16], DianNao [6], and Cambricon-X [46].

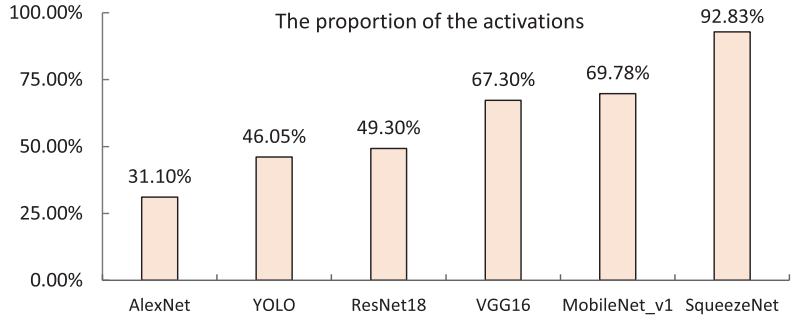

Fig. 2. The proportion of the intermediate feature maps of state-of-the-art DNN model.

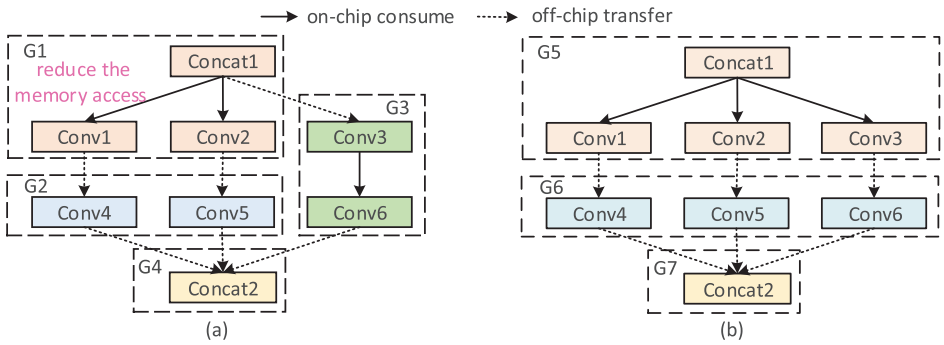

Fig. 3. Examples of DAG-based operator fusion.

group will gain benefits in terms of the total memory access overhead. Third, for the scenarios and platforms such as reconfigurable processor architecture [52], resource sharing accelerator-based INFerence-as-a-Service [11, 13, 23], consolidation, and FPGA virtualization for AI workloads [45], the underlying computing resources are dynamic, and a fixed fusion scheme does not work when the memory space or processing elements assigned to the network workload are changing, since the optimal solution depends on the resource assignment. In this case, searching for the optimal fusion must be performed online, and it must be fast enough and put negligible impacts on the end-to-end network inference latency. However, how to reach fast and high-performance online operator fusion is not discussed, and most of them employ expensive search polices to find premier fusion schemes offline.

None of the previous work on neural network fusion and accelerator design has considered the problem of how to reach optimal operator fusion implementation for an arbitrary network topology and the given deep learning accelerator underneath. In this work, we propose an optimal operator fusion framework, driven by the memory cost model, to search memory-optimal and energy-efficient operator fusion schemes for an arbitrary combination of networks and DNN accelerators. Specifically, our work makes the following contributions:

(1) The proposed optimal operator fusion algorithm, DAG-based hardware-aware operator fusion algorithm explores all the feasible operator fusion options to find the memory-optimal operator fusion schemes in a reasonable time.
(2) The proposed accurate memory-cost model contains a scheduler that effectively map the fused operator-group to the given accelerators, is useful in capturing the achievable minimum off-chip memory overhead for the fused operator-groups, considering both feature-map and parameters induced memory accesses.
(3) The proposed fast and efficient on-line variant of operator fusion algorithm can discover the near-optimal fusion scheme for the online network deployment scenario when the resources of processors are allocated dynamically.
(4) The experiments are conducted across a variety of neural network architectures and DNN accelerators. The results show that the proposed algorithms guarantee the memory optimality and bring significant benefits to DNN accelerators of different architectures and dataflows when compared to the baselines.

The rest of this article is organized as follows. Section 2 presents the background and motivational analysis. Section 3 provides an overview of the our end-to-end operator fusion framework. Section 4 elaborates the proposed DAG-based hardware-aware operator fuser. Section 5 presents the memory cost model with a scheduler for fused operator-groups. Section 6 proposes the online variant of operator fusion algorithm for the online scenario. In Section 7, the effectiveness of our framework is illustrated by experiments, and the insights are also presented. We also introduce the related work in Section 8. Finally, Section 9 concludes the article.

2 BACKGROUND AND MOTIVATION

2.1 DNN Accelerators

The rise of DNNs [17, 20, 24] has stimulated intensive research on DNN accelerators [7, 9, 10, 12] to address the high compute and memory requirements. Figure 4 exemplifies a typical architecture of state-of-the-art DNN accelerators. These accelerators include a number of **processing elements (PEs)** organized in a two-dimensional (2D) array. Each PE contains an ALU for **multiply-accumulate (MAC)** operations and a small **register file (RF)**. A larger SRAM buffer is shared by all PEs to cache the reusable on-chip data to reduce off-chip memory (e.g., DRAM) accesses. Since the neural parameters and activations of the DNN models are too large to fit entirely in the compact on-chip memory, the data exchange between off-chip memory and on-chip buffer is frequent. Furthermore, the energy cost of DRAM access is orders of magnitude higher than that of the requests hitting other levels [9], thereby dominating the system energy consumption of typical neural network chips. Consequently, the efficiency and performance of the DNN accelerators depend on how the data are scheduled between on-chip and off-chip. In this article, we focus on the operator fusion scheduling that reduces the most expensive memory accesses through avoiding the frequent intermediate feature map traffic between off-chip DRAM and the on-chip buffer.

The high-performance scheduling solution changes along with the hardware resources of DNN accelerators such as the on-chip buffer space and processing elements. In the multi-tenant use

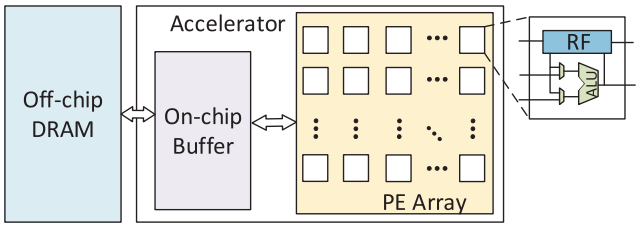

Fig. 4. A typical DNN processor architecture design.

cases, including the scenarios of resource-sharing accelerator-based INFerence-as-a-Service [11, 13, 23], reconfigurable processors and accelerators [52], FPGA virtualization and consolidation for DL [45], the resources for target workloads are allocated on-demand dynamically. Scheduling schemes need to be adapted in time to achieve high resource utilization. Consequently, fast and efficient online redeployment of neural models is also necessary to ensure that task progress will not be affected even in the case of frequent dynamic resource reallocation [45].

2.2 Operator Fusion Space Exploration

In general, DNN algorithms can be represented as DAGs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which provide a global view of the interconnected operators as shown in Figure 3. In DAGs, vertices $\mathcal{V}$ represent tensor operators (e.g., Conv operator), and edges $\mathcal{E}$ represent data dependencies between operators. Since simple element-wise operators, e.g., **batch-normalization (BN)** and ReLU, can be directly fused, e.g., Conv+BN+ReLU, they are ignored in the figure for simplicity. In DAGs, vertices (operators) can be partitioned into different groups; for example, Figure 3 shows two different types of partitioning. The various partitions, which constitute the operator fusion search space, lead to different memory overhead. The search space is enormous, and it has been proven to be a NP-hard problem [30] to explore the entire operator fusion space for a complicated NN model.

Prior operator fusion works are far from optimal, since their approaches did not consider sophisticated DNN model structures, or they did not fully explore the search space of fused operator-groups. Fused-layer [2] find the optimal operator (layer) fusion scheme for simple DNN models without branches by exhaustively evaluating the whole operator fusion schemes. Unfortunately, as the network topologies become more complex with branch, the time complexity of enumeration policy is exponentially explosive, and it is impractical for the state-of-the-art DNN models. To address the challenges posed by branches in the DNN topology, DNNVM [43] set the operators that are dependent on more than one operator or by different operators as barriers and assume that the fused operator-groups will never contain the barriers. For example, the operator-group `{Concat1, Conv1, Conv2, Conv3}` as shown in Figure 5(a) is invalid in DNNVM. However, for the network with many branches, e.g., SqueezeNet [20], GoogleNet [38], and NasNet [53], they approach will miss a great amount of the potential fusion opportunities. Reference [50] eliminates the branches in the DNN topology by transforming the complicated model into a linear proxy model without violating the data dependencies, for example, converting the models in Figure 5(a) to the linear topology in Figure 5(b). Then they applied dynamic programming algorithm on the linear topology to find the optimal operator fusion result. However, the result may not be optimal for the original complicated network model, for example, the fused operator-groups `{Conv1, Conv4}`, `{Conv2, Conv5}`, and `{Conv3, Conv6}` will never be identified as valid fused operator-groups by this method. Besides, building the linear structure also requires a non-negligible time overhead.

Conversely, in this work, we put forward a hardware-aware operator fusion algorithm upon the original complicated DAGs of DNNs, which searches through the entire operator fusion space, and

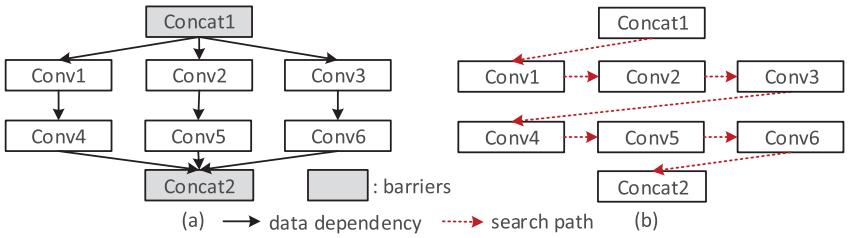

Fig. 5. A typical operator fusion search process in prior works.

Table 1. Notation Used in Fused Operator-Groups

Notations	Explanation
$O_h^{(l)}, O_w^{(l)}$	The height and width of the output feature map (ofmap).
$K_h^{(l)}, K_w^{(l)}$	The height and width of the weight kernel.
$C_{in}^{(l)}, C_{out}^{(l)}$	The channel numbers of input and output feature map.
$S_h^{(l)}, S_w^{(l)}$	The convolution strides in the h and w directions.
$t_h^{(l)}, t_w^{(l)}$	The height and width of the ofmap tile.

*The superscript (l) represents the operator number in a certain fused operator-group.

it is even faster than in previous works [43, 50]. Furthermore, we present a fast on-line variant of operator fusion algorithm for the online scenario while achieving near-optimal fusion scheme.

2.3 Fused Operator-Groups

Figure 6 illustrates the execution process of a fused operator-group with an example that fuses two convolution (Conv) operators, e.g., G3 in Figure 3, where the notations of the lth operator in a certain fused operator-group are listed in Table 1.[1]

A fmap tile $t_h^{(l)} \cdot t_w^{(l)}$ is a partition from one fmap, and it is the basic unit received and processed by the DNN accelerators. In the fused operator-group, the fmap tiles are directly consumed by the subsequent operators rather than being evicted to the off-chip memory. For example, the Conv kernel in Operator-1 operates on the $7 \cdot 7$ tiles of it **input feature maps (ifmaps)**, consisting of $7 \cdot 7 \cdot C_{out}^{(0)}$ input pixels, and produces $5 \cdot 5 \cdot C_{out}^{(1)}$ pixels. After that, the Operator-2 use these $5 \cdot 5 \cdot C_{out}^{(1)}$ region to produce $3 \cdot 3 \cdot C_{out}^{(2)}$ outputs in the **output feature maps (ofmaps)**. It then continues with the next tile until all outputs are produced.

The height/width of the ofmap tiles between two operators satisfy the relationship $t^{(l-1)} = t^{(l)} \times S^{(l)} + K^{(l)} - S^{(l)}$ [2], wherein operator-$(l-1)$ is the input producer of operator-l. Thus, according to the producer-and-consumer relationship, the **r**equired minimum size of the ofmap tile for operator-l in the fused-group, which is referred to as $\mathcal{R}(l,\ t^{(n)})(\mathcal{R}(n,\ t^{(n)}) = t^{(n)})$, can be deduced and determined backwardly from the last operator (operator-n) in the group. For example, as shown in in Figure 6, to obtain a $3 \cdot 3$ ofmap tile of Operator-2 depends on the $5 \cdot 5$ ofmap tiles from Operator-1, which further relies on the $7 \cdot 7$ ofmap tiles from Operator-0 (input data).

Previous works [2, 43, 50] on operator fusion put some constraints on fused operator groups that the feature pixels with the size of at least $\sum_{l=0}^{n} C_{out}^{(l)} \cdot \mathcal{R}_h(l, t_h^{(n)}) \cdot \mathcal{R}_w(l, t_w^{(n)})$ in total and all the required parameters must be kept in the on-chip buffer at the same time. Otherwise, this

[1]Besides the Conv operator, other operators can also be expressed by these factors. For example, a fully connected operator can be considered as a special Conv operator by setting $S_h^{(l)}$, $S_w^{(l)}$, $O_w^{(l)}$, $O_h^{(l)}$, $K_w^{(l)}$, and $K_h^{(l)}$ to 1.

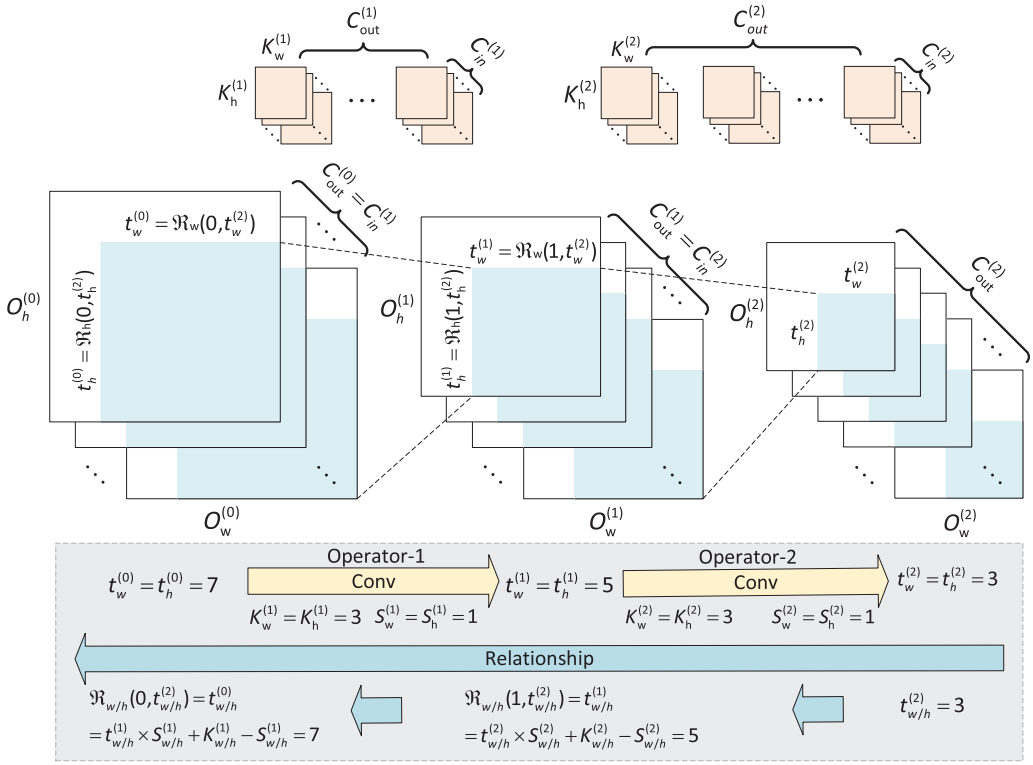

Fig. 6. Processing of a fused operator-group. The ofmap tiles $t_h^{(l)} \cdot t_w^{(l)}$ of operator-l are directly consumed by the subsequent operators rather than being evicted to the off-chip memory.

fused-group is deemed invalid due to the lack of on-chip memory space. With these constraints, the parameters can be loaded at the start of computation and remained on-chip until the entire fused operator group is completed. In this case, the off-chip memory access volume equals to the sum of the ifmaps' size of operator-1, the ofmaps' size of operator-n and the parameters' size of the whole operator-group, i.e., $|ifmap^{(1)}| + |ofmap^{(n)}| + \sum_{l=1}^{n} |param^{(l)}|$.

However, as presented in Figure 7, it is very common that the buffer space of a given DNN accelerator cannot fit all required data of the fused operator-groups. In this work, we point out that the fused operator-groups will not be limited by the amount of the parameters through allowing parameter-induced memory access when computing the fused-operator groups, which enlarges the design space of operator fusion and brings benefits to the total memory access overhead. In addition, allowing parameter refill during the inference process of the fused operator-group exposes the potential benefit of reducing the on-chip buffer space required by the fmap pixels, which can further reduce off-chip memory accesses. Consequently, we propose a memory-cost model (Section 5) to precisely evaluate the achievable minimum memory-traffic achieved by the presented fused operator-group scheduler, which takes into account the parameter-induced memory accesses.

3 OVERVIEW

Operator fusion is a general technique to reduce the feature-map induced memory access for DNN accelerators. In fact, operator fusion only changes the order of operations in the DNN model, and it

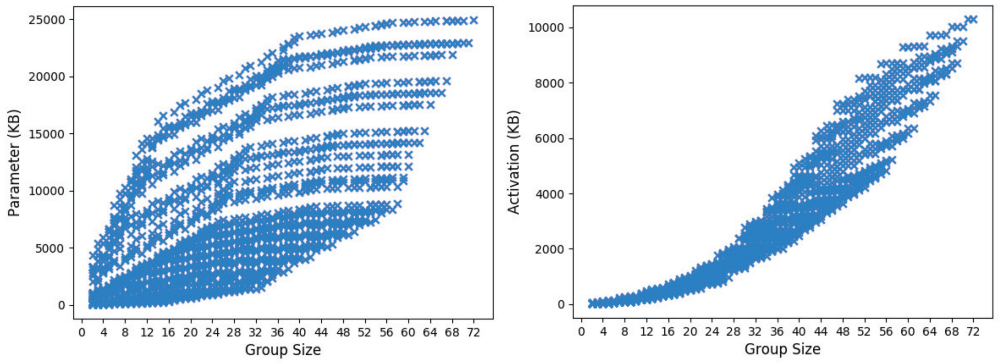

Fig. 7. The on-chip memory footprint of the required data for the fused operator-groups.

can be practiced and applied to most of the general DNN accelerators [6–10, 23] in the compilation or network-mapping stage to generate the according execution bitstreams or instructions, which will control the hardware to run the network operators in the corresponding dataflow. We propose the optimal operator fusion framework, Optimus, which search memory-optimal and energy-efficient operator fusion scheme for DNN workloads to run on the most of state-of-the-art DNN accelerators.

As shown in Figure 8, the proposed operator fusion framework starts with importing the DNN models from the mainstream deep learning frameworks, such as Pytorch [35], TensorFlow [1], and so on. Then, we parse the DNN models to obtain the DAG topologies used in Optimus and merge some simple operators, such as convolution+BN+Scale, which can be pre-calculated or statically determined; convolution+activation, where the element-wise operations can execute in situ; and operator+flatten/split/reshape+operator, where the flatten/split/reshape operators can naturally merge into the save process of the previous operator or the load process of the subsequent operator. After that, the DNN topologies are partitioned by the DAG-based operator fusion algorithm to iteratively generate fused operator-groups and estimate them.

DAG-based operator fuser explores the optimal operator fusion scheme for the DAG topology of DNN model running on the given accelerator. In each iteration, it generates a operator group and analyzes its validity, and the valid fused operator-groups are passed to the memory cost model to obtain the memory cost feedback. The optimal operator fusion scheme could be determined after the exploration of the whole operator fusion space driven by the memory cost.

Memory cost model first generates the schedulings of the fused operator-groups for the given accelerators through the memory-efficient scheduler. Then, the schedulings that parameterize the loop-nest of the fused operator-groups are feed to the analyzer and estimator to collect the metrics. The loop-nest can be directly transformed to the instruction. In this article, we analyze the off-chip memory access as detailed in Section 5. This analyzer can quickly analyze the memory access for the enormous operator fusion space, and the differences between it and the measure result of the real system are less than 5%. Optimus also allow to evaluate other metrics through the external mathematical models [25, 44] and runtime profiling.

On-line variant filters the low-efficiency fused operator-groups and limits the size of the fused operator-groups. The fused operator-groups that do not satisfy its rules are deemed as invalid and affect the DAG-based operator fuser, which helps to run fast and find the near-optimal fusion scheme for the online scenario.

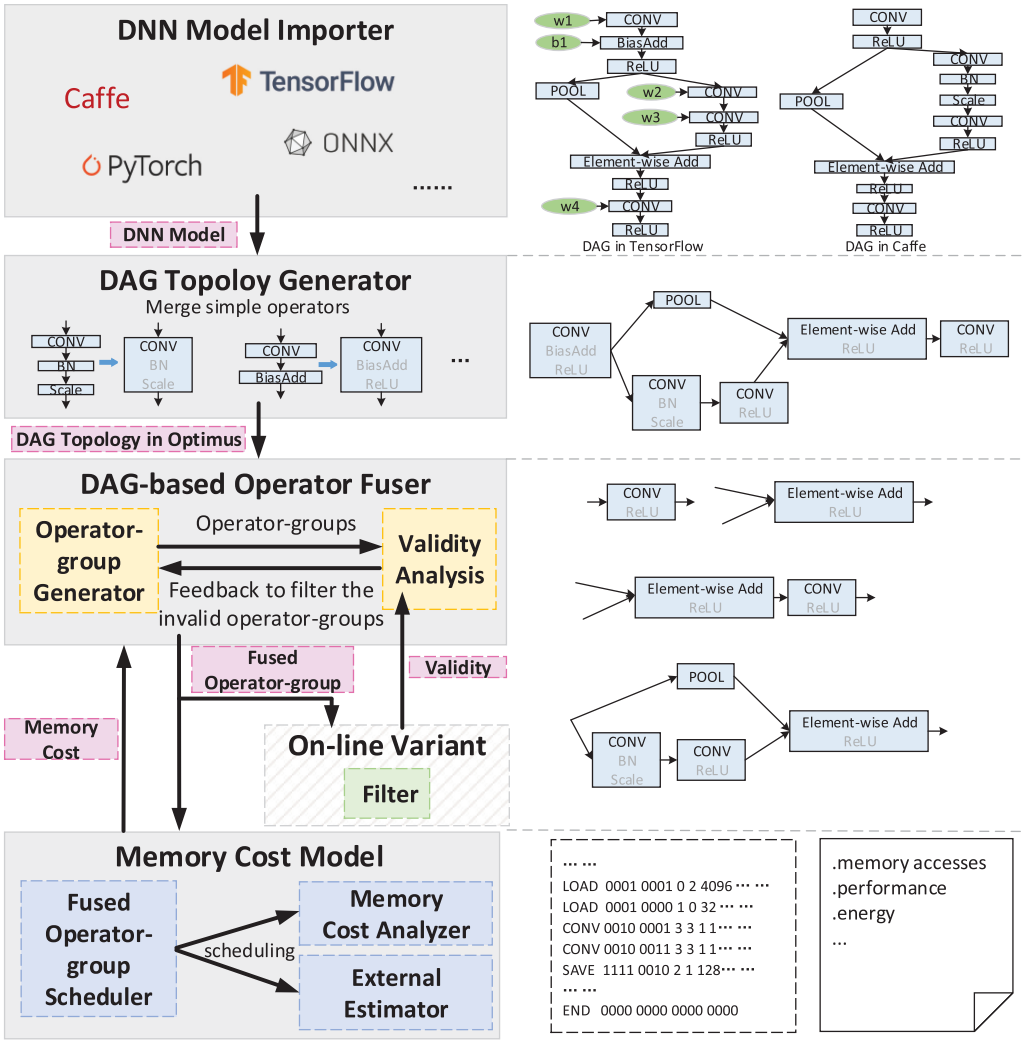

Fig. 8. An overview of our optimal operator fusion framework for DNN workloads to run on state-of-the-art DNN accelerators.

4 DAG-BASED OPERATOR FUSER

In this section, we elaborate our DAG-based hardware-aware operator fusion algorithm. First, we formalize the network fusion problem. Next, we introduce the optimal sub-structure for operator fusion with our observation. Then we give the implementation details of the algorithm. Finally, we analyze the time complexity.

4.1 Operator-Fusion Problem

The formulation of the operator fusion problem will guide us to exhaustively explore the entire schedule space and evaluate the best solution. However, it is not formally or comprehensively defined in prior works. In this article, we formalize that the optimal operator fusion problem is to find an optimal operator fusion scheme that has the minimum off-chip memory accesses.

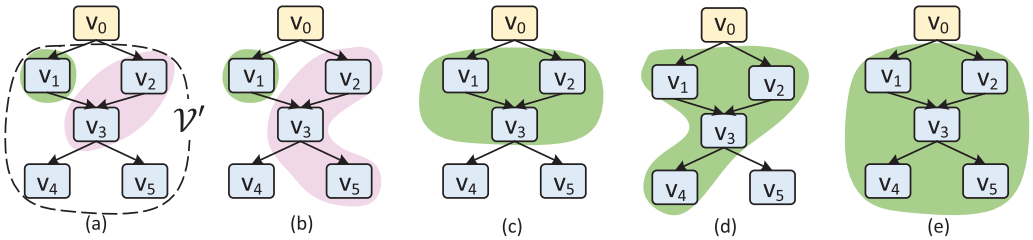

Fig. 9. The illustration of adding a new input vertex v_0 into the graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$. Panels (a)–(e) show different operator-groups of $\mathcal{V}'$ that have data dependencies with v_0. Operator-groups $\{v_0, v_2, v_3\}$ in (a) and $\{v_0, v_2, v_3, v_5\}$ in (b) cause cyclic data dependency when fusing with v_0.

For a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of a DNN model, let $L = \{G_1, G_2, \ldots\}$ ($L \in \mathcal{L}$, $\mathcal{L}$ is the operator fusion space) be a operator fusion scheme that patitions the network $\mathcal{V}$ into *disjoint* fused operator-groups G_i ($\mathcal{V} = \bigcup G_i$) that *have no cyclic dependencies* in them, as shown in Figure 3. Thus, the optimal operator fusion is as follows:

$$\min_{L \in \mathcal{L}} \sum_{G_i \in L} MemoryCost\,(G_i), \tag{1}$$

where $MemoryCost\,(\cdot)$ is the cost function that models the memory overhead of the fused operator-groups, which will be detailed in Section 5.

4.2 Optimal Sub-Structure for Operator Fusion

As shown in Figure 9, for a DAG of NN model $\mathcal{G}' = (\mathcal{V}', \mathcal{E}'), \mathcal{V}' = \{v_1, \ldots, v_N\}$, we observe that a newly added input vertex v_0 in the DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E}), \mathcal{V} = \{v_0, v_1, \ldots, v_N\}$ can be fused with one of the operator-groups that have data dependencies with it or can remain separate. These two choices are mutually exclusive, and the decisions can be made by comparing the cost of them. However, for complicated NN models, it will make the decision space grow exponentially, because there are $2^{|succ(v_i)|}$ combination choices for each multi-output vertex,[2] so the total combinative operator-groups for v_0 is $O(\prod_i 2^{|succ(v_i)|})$. Fortunately, in operator fusion, operator-groups that result in cyclic data dependencies, e.g., operator-group $\{v_0, v_2, v_3\}$ in Figure 9(a) and $\{v_0, v_2, v_3, v_5\}$ in Figure 9(b), are invalid; otherwise, the operation on fused operator-groups must be interrupted, e.g., the interruption request occurs after v_0 (or v_2) to calculate the output data of v_1 that v_3 needs, which stalls the continuous execution of the fused operator-groups. Under this constraint, the total number of available combinations of operator-groups becomes $O(\sum_i 2^{|succ(v_i)|})$, i.e., $O(2^{\max_i |succ(v_i)|}\,|\mathcal{V}|)$.

Based on this observation, we present the DAG-based operator fusion algorithm to search for the optimal fusion scheme. Assuming we have already worked out the optimal operator fusion schemes of $\mathcal{G}'$ and its subgraphs, there are two choices for a newly added input vertex v_0 in $\mathcal{G}$: (1) if there is no cycle dependencies after fusing, then fuse it with one of fused operator-groups that have data dependencies with it; (2) let it remain separate. In both cases, the algorithm works recursively, and thus the original problem is reduced into smaller sub-problems. By summarizing over these two cases, we can have the general form of sub-problems as minimizing the memory cost of the graph $\mathcal{V}'$. And we can represent the optimal value (minimum memory access number) of the sub-problems as $cost\,(\mathcal{V}')$ ($cost(\varnothing) = 0$). Therefore, we obtain the optimal sub-structure

[2] $|succ(v_i)|$ is the number of successor vertices of v_i.

property:

$$cost(\mathcal{V}) = \min_{G'}\{cost(\mathcal{V}' - G') + MemoryCost(v_0 + G')\}, \quad (2)$$

where G' is the fused operator-group that have data dependencies to v_0 or the empty set $\emptyset$. Finally, $cost(\mathcal{V})$ is the minimum memory access of an optimal operator fusion scheme for the entire DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

4.3 Algorithm Implementation

With this general idea, we implement DAG-based operator fusion as shown in Algorithm 1. `Fuser` (`L4–23`) takes the DAG topology of a DNN model and the description of the DNN accelerator as inputs and returns the optimal operator fusion scheme and the minimum memory cost of the entire network. `ISValid` (`L24–30`) judges the validity of the operator-groups.

ALGORITHM 1: DAG-based Operator Fusion

Input: the network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the accelerator description *acc*
Output: the optimal operator fusion scheme and the minimum memory cost of the entire network

```
1  Let cost[V'] = ∞ for all V' ⊂ V but cost[∅] = 0;
2  Let S_opt[V'] = ∅ for all V' ⊂ V;
3  Let operatorGroup[v_0] = ∅ for all v_0 ∈ V;
4  Function Fuser(G, acc):
5      visited ← ∅;
6      for v_0 ← reversed topological order of G do
7          visited.add(v_0);
8          for G' ∈ operatorGroup[v_0] do
9              if ISValid(G', v_0) then
10                 M', S' ← MemoryCost(G' + v_0, acc);
11                 M ← cost[visited − G'] + M';
12                 if M < cost[v_0 + visited] then
13                     cost[v_0 + visited] ← M;
14                     S_opt[v_0 + visited] ← (G' + v_0, S');
15                 for v' ∉ visited and v' has data dependencies with operator-group G' + v_0 do
16                     operatorGroup[v'].add(G' + v_0);
17     Q ← empty list;
18     V' ← V;
19     while V' ≠ ∅ do
20         G', S' = S_opt[V'];
21         add the schedule (G', S') into Q;
22         V' ← V' − G';
23     return the optimal operator fusion scheme Q and the minimum memory cost cost[V];
24 Function ISValid(G', v_0):
25     isValid ← true;
26     for c ∈ succ(v_0) do
27         if c ∉ G' and c can reach G' then
28             isValid ← false;
29             break;
30     return isValid;
```

Fuser (L4–23), the core part of our algorithm, is a function implementing the bottom-up dynamic programming algorithm. We visit the graph in reversed topological order (L6–16). For the vertex v_0 being visited, we traverse the operator-groups $operatorGroup[v_0]$ with which it has data dependencies (L8–16). We use ISValid to judge whether it is valid to fuse v_0 with the operator-group G' (L9). For the valid fused operator-groups, we pass them to the memory cost model to obtain the their memory cost feedback M' and the corresponding schedule S' as detailed in Section 5 (L10) and update the $cost[v_0 + visited]$ and $S_{opt}[v_0 + visited]$ when the fusing is profitable (L11–14). The operator-group generator $operatorGroup[\cdot]$ is updated for the vertices that have data dependencies with the new fused operator-group $G' + v_0$ (L15–16). We use table $cost[\cdot]$ and $S_{opt}[\cdot]$ to implement the "tabulation" mechanism of dynamic programming algorithm, taking the advantage of the common sub-problems among different partitions. After visiting the vertices of $\mathcal{V}'$, the minimum memory cost of an optimal operator fusion scheme for $\mathcal{V}'$ is stored in $cost[\mathcal{V}']$, while the last fused operator-group along with its schedule in the corresponding optimal operator fusion scheme is stored in $S_{opt}[\mathcal{V}']$. And we construct the entire optimal operator fusion scheme of the whole DNN network using the table $S_{opt}[\cdot]$ (L17–22). We start with an empty list as the initial state of our optimal operator fusion scheme (L17) and let $\mathcal{V}'$ be all the vertices in $\mathcal{G}$ (L18). We inquire about the last fused operator-group (G', S') of $\mathcal{V}'$ by $S_{opt}[\mathcal{V}']$ and add it into the operator fusion scheme Q and repeat this process by letting $\mathcal{V}' = \mathcal{V}' - G'$ to get the optimal operator fusion scheme of the remaining sub-graph (L19–22), until $\mathcal{V}' = \emptyset$. Finally, the algorithm returns an optimized optimal operator fusion scheme Q for $\mathcal{G}$ and the minimum off-chip memory access volume $cost[\mathcal{V}]$.

ISValid (L24–30) judge the validity of fusing the given operator v_0 with the operator-group G'. The new operator-group is invalid if it has data dependency conflict that is reflected in forming a cycle in the DAG after fusing, for example, $\{v_0, v_2, v_3\}$ in Figure 9(a) and $\{v_0, v_2, v_3, v_5\}$ in Figure 9(b). If any successor of v_0 is not belong to the operator-group but has a descendant in operator-group, then the fused operator-group $v_0 + G'$ creates a cyclic data dependency (L27–29).

4.4 Complexity Analysis

When applied to the neural network without branches, there are $|\mathcal{V}|$ vertices, and each has $O(|\mathcal{V}|)$ operator-groups to fuse. Thus the complexity of the algorithm is $O(|\mathcal{V}|^2)$. For those complicated DNN models with branches, there are $O(2^{\max_i |succ(v_i)|}\,|\mathcal{V}|)$ operator-groups. So, the complexity of the algorithm is $O(2^{\max_i |succ(v_i)|}\,|\mathcal{V}|^2)$. As we can see, $\max_i |succ(v_i)|$ is often small in the DNN models, and thus the time complexity is within a reasonable range.

5 MEMORY COST MODEL OF FUSED OPERATOR-GROUPS

In this section, we present the memory cost model of the fused operator-groups. First, we eliminate the constraints that put on the fused operator-groups by the prior works and remodel the memory cost. Then, we introduce the scheduler for the fused operator-groups to reduce its minimum memory access volume.

5.1 Memory Cost Remodeling

Unlike prior works, we discard their assumption that all the parameters associated to the fused operators must be kept in the on-chip memory. As illustrated in Figure 10(b), when the on-chip buffer of the accelerator is large enough to keep all the parameters of the fused operator-group, we load the parameters at the start of computation, i.e., load the parameters outside the loop nest, and reuse them until the entire fused operator group is completed. However, when the parameters

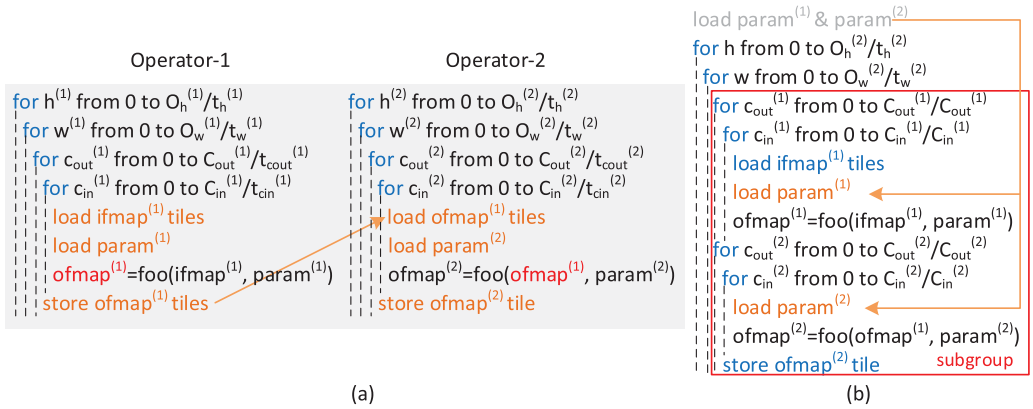

Fig. 10. Pseudo code of the two operators (a) before and (b) after fusing as Figure 6 (foo represents the execution of the target DNN accelerator).

of the fused operator-group cannot fit into the on-chip buffer of the target accelerators, all parameters are loaded in each sub-group iteration. In this case, the parameters in the loop nest do not need to be loaded onto the chip all at once. Instead, only the part of parameters required by the operations are loaded according to the dataflow of the target DNN accelerators, and the loaded parameters are reused as much as possible by the fmap tiles before releasing the buffer space they occupy, which means the on-chip buffer footprint required by the fused operators are overestimated in prior works. In our cost model, such a reuse mechanism in the sub-groups are faithfully accounted for. Therefore, we have that the total number of subgroups is $\lceil O_h^{(n)}/t_h^{(n)} \rceil \cdot \lceil O_w^{(n)}/t_w^{(n)} \rceil$ for a fused operator-group G_i with n operators. Accordingly, in our cost model, the involved off-chip parameter-induced memory traffic for the operator-group is measured as $\lceil O_h^{(n)}/t_h^{(n)} \rceil \cdot \lceil O_w^{(n)}/t_w^{(n)} \rceil \cdot \sum_{l=1}^{n} |param^{(l)}|$. Thus, the total off-chip memory access volume is[3] as follows:

$$MemoryCost(G_i) = |ifmap^{(1)}| + |ofmap^{(n)}| + \left\lceil \frac{O_h^{(n)}}{t_h^{(n)}} \right\rceil \left\lceil \frac{O_w^{(n)}}{t_w^{(n)}} \right\rceil \sum_{l=1}^{n} |param^{(l)}|. \tag{3}$$

5.2 Fused Operator-Group Scheduler

As analyzed above, minimizing the memory cost is equivalent to maximizing $t_h^{(n)} \cdot t_w^{(n)}$, which is limited by the on-chip buffer space of a given accelerators. However, the on-chip buffer space is compact to save area and power overhead. Consequently, we present how to schedule to maximize $t_h^{(n)} \cdot t_w^{(n)}$ for each fused operator-group under a fixed buffer capacity.

Invoking subsequent operator in advance (ISOA): In a fused operator-group, we allow the subsequent operator to be invoked immediately once a subset of fmap tiles produced by the predecessor operator are ready. For example, in Figure 11, operator-1 starts processing T_c[4] ifmap tiles as

[3]For simple illustration, we assume that there is only one input operator and only one output operator in the operator-group. In fact, our derivation is also applicable to more complicated operator-group with multiple input and output operators, e.g., {v1,v2,v3,v4} in Figure 9.

[4]Herein, T_* is the **t**hroughput of the accelerators determined by the PE-array size and the dataflow [9, 44], since we realized that there are impacts of tile size on the on-chip resources utilization of the target accelerator (Section 7.3). It implies that the tile size needs to be greater than the throughput determined by the dataflow and PE-array of a given DNN accelerator; otherwise, the accelerator datapath and the corresponding on-chip resources cannot be fully utilized.

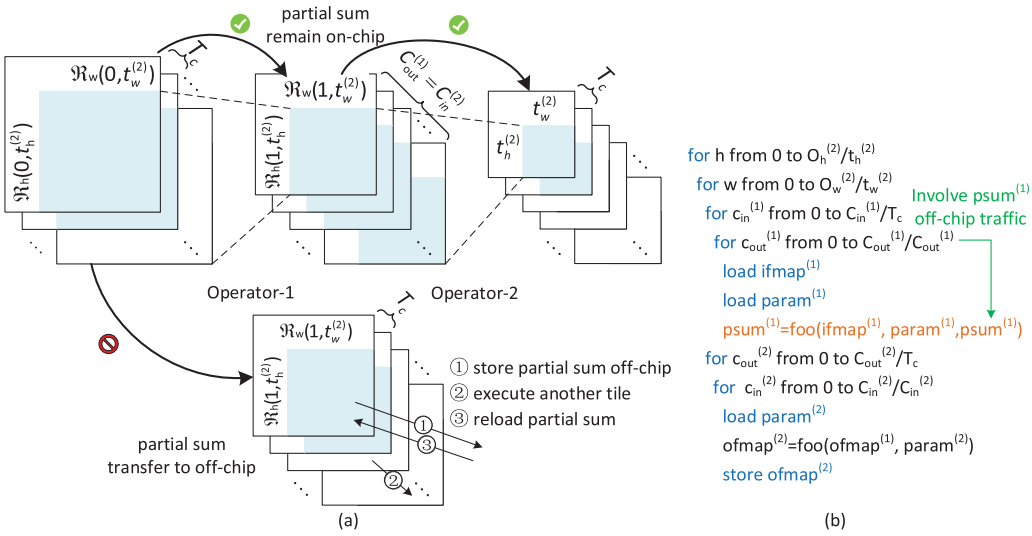

Fig. 11. Reducing the on-chip buffer space requirement of the fused operator-group in Figure 6.

soon as they are provided, which means a buffer space of T_c fmap tiles is sufficient for processing the tiles smoothly.

Through ISOA, the on-chip buffer space requirement for the fmap pixels can be reduced significantly. Thus, the fmap tile $t_h^{(n)} \cdot t_w^{(n)}$ can be enlarged under the same buffer capacity. However, because all channels of ifmap must be reduced to produce one channel of ofmap, the on-chip buffer space requirements of two successive operators cannot be reduced simultaneously. As the example illustrated in Figure 11, buffering only T_c ifmap and ofmap tiles of operator-1 causes the partial sums to be evicted to the off-chip memory and then reloaded to the on-chip buffer again as required by the next accumulation operation, which violates the principle that intermediate data in the fused operator-group is not saved off-chip.

To deduce the minimum on-chip buffer requirement of the fmap pixels for the fused operator-group with n operators, we present an algorithm to determine which operators in the fused operator-group are compatible to ISOA. As formulated in Algorithm 2, the operator group is traversed in reversed topological order (L1), and we define $dp[l][1]$ as the case when operator-l adopts ISOA (L2) and $dp[l][0]$ as the case when ISOA is not applied (L3). If the operator-l adopts ISOA, then its successor operators must not apply ISOA (L5); otherwise, its successor operators are allowed to use ISOA method (L6). The cost is the corresponding buffer requirement of the operator when the $t_h^{(n)} \cdot t_w^{(n)} = 1 \cdot 1$. We minimize the global cost and have the decision $\phi(\cdot)$ for each operator (L7--8).

Maximizing fmap tile size: Since convolution performs $K_h \times K_w (K_h, K_w \geq 1)$ kernel size, adjacent fmap tiles are usually overlapped with each other. Overlapped elements can be recalculated [19] or stored on-chip [2, 42, 50] without introducing significant overhead. In this article, for operators that adopt ISOA (i.e., $\phi(l) = 1$), the overlapped elements cannot be stored on-chip and they are recalculated. For operators that do not adopt ISOA (i.e., $\phi(l) = 0$), overlapped elements are stored on-chip using line-buffer technique ($t_w^{(n)} = O_w^{(n)}$) as in Reference [42] to avoid introducing additional buffer resource consumption. Therefore, we can maximize $t_h^{(n)}$ under the premise that the buffer requirement of fmaps cannot exceed the on-chip buffer capacity B_c as shown in Equation (4). When constraints cannot be met, Equation (4) has no solution, and the

ALGORITHM 2: ISOA Applying Decision

Input: fused operator-group G_i
Output: Binary indicates whether a operator adopts ISOA ϕ

for $l \in$ *reversed topological order of* G_i **do**
  $dp[l][1] = T_c \cdot \mathcal{R}_h(l,1) \cdot \mathcal{R}_w(l,1)$;
  $dp[l][0] = C_{out}^{(l)} \cdot \mathcal{R}_h(l,1) \cdot \mathcal{R}_w(l,1)$;
  for $v \in \mathtt{succ}(l)$ **do**
    $dp[l][1] += dp[v][0]$;
    $dp[l][0] += \min(dp[v][1], dp[v][0])$;
$minCost = \min(dp[0][1], dp[0][0])$;
$\phi = \mathtt{Backtracking}(dp, minCost)$;
return ϕ

fused operator-group is regarded as invalid,

$$\begin{aligned} &\max\ t_h^{(n)} \\ &s.t.\ \sum_{l=0}^{n} t_c^{(l)} \cdot \mathcal{R}_h\left(l, t_h^{(n)}\right) \cdot \mathcal{R}_w\left(l, O_w^{(n)}\right) \le B_c \\ &\quad t_c^{(l)} = \begin{cases} \min(T_c, C_{out}^{(l)}), & \phi(l) = 1 \\ C_{out}^{(l)}, & \phi(l) = 0 \end{cases} \\ &\quad min\left(T_h, O_h^{(n)}\right) \le t_h^{(n)} \le O_h^{(n)}, \end{aligned} \tag{4}$$

where $t_c^{(l)}$ is the channel tile of the ofmap and $\phi(l)$ is the ISAL applying decision from Algorithm 2. Finally, we can obtain the achievable minimum off-chip memory access volume for fused operator-groups according to the memory cost model in Equation (3).

6 ON-LINE VARIANT OF OPERATOR FUSION

When the chip resources such as on-chip buffer capacity are allocated on-line, the decision of operator fusion must be made at a reasonable time overhead in case of penalizing the network inference performance. The complexity of DAG-based operator fusion algorithm in Section 4 may add a significant amount of time overhead to the milliseconds-level network inference latency when being executed on-line. Therefore, we need a fast and efficient on-line variant of operator fusion to deploy fused neural networks at runtime. Intuitively, shrinking the search space of operator fusion schemes can dramatically save the algorithm runtime overhead. Fortunately, we found some empirical rules from experimental results with the implementation in Section 4.

Removing the low-efficiency fused operator-groups: On the one hand, there are very limited benefits gained to fuse a operator and only part of its fan-out successor operators into the same group, because the intermediate data still need to be written back to the off-chip memory for the remaining consumers. For example, in the operator-group $\{v_0, v_1\}$ of Figure 9, the intermediate data of operator v_0 will be transfer to the off-chip memory for the operator v_2 to use. Thus, a operator should be either fused with, or separated from, all of its fan-out successor operators. On the other hand, it is not rewarding to fuse the operators into the same group if they share no data dependencies. The data dependencies to exploit can be either due to sharing the same input with another operator in the operator-group, or producing the input data of a operator in the operator-group. For example, v_1 shares input data with v_2 and thus the operator-group $\{v_1, v_2\}$ is profitable.

Thus, we see that a operator can be added into a temporary operator-group only if it shares some data dependencies with other operators in the operator-group.

Limiting the size of the fused operator-group: According to Equations (3) and (4), the more operators are fused, the more feature-induced memory accesses are reduced, but it will introduce excessive parameters-induced memory accesses. Therefore, we can reduce the search space by limiting the size of the operator-group and avoiding too many parameter-induced memory accesses. However, the proper operator-group size limitation varies with the on-chip memory size. Consequently, we devise a look-up table mechanism to quickly determine the operator-group size limitation for each dynamically allocated on-chip memory size.

The look-up table is pre-built in the Algorithm 3. Each table entry corresponds to the on-chip memory size $memo[n]$, and the infeasible operator-group size n associated to this entry, wherein $memo[n] = \min_{|G_i|=n} \sum_{l=0}^{n} t_c^{(l)} \mathcal{R}(l, t_h^{(n)}) \mathcal{R}(l, t_w^{(n)})$ and $t_c^{(l)}$ determined by Algorithm 2. And $t_h^{(n)}$ and $t_w^{(n)}$ cannot be too small, otherwise the memory access induced by the parameter $\lceil O_h^{(n)}/t_h^{(n)} \rceil \cdot \lceil O_w^{(n)}/t_w^{(n)} \rceil \cdot \sum_{l=1}^{n} |param^{(l)}|$ is too large. According to a large number of experiments and analyses, we conclude that if the volume of the access is 16 times the number of parameters, then the loss is possibly greater than the gain, and thus the $t_h^{(n)}$ and $t_w^{(n)}$ are set to $O_h^{(n)}/4$ and $O_w^{(n)}/4$ (almost all networks have $O_h^{(n)} = O_w^{(n)}$). Therefore, $memo[n]$ is a minimum memory size to ensure the benefit of the operator-group whose size does not exceed n, and $memo$ can be proved to be increasing.

ALGORITHM 3: Building Look-up Table

Input: network graph $\mathcal{V}$
Output: look-up table $memo$

```
memo[1, ..., |V|] ← ∞;
for each G_i ∈ V do
    n ← sizeof(G_i);
    M ← Σ_{l=0}^{n} t_c^{(l)} R(l, O_h^{(n)}/4) R(l, O_w^{(n)}/4);
    if memo[n] > M then
        memo[n] ← M;
return memo
```

With this compact look-up table, when given the buffer capacity B_c online, we can quickly estimate the maximum group size it allows and use it to filter many of the fusion options. That is, if the target network is allocated with an on-chip memory capacity B_c in between $memo[n]$ and $memo[n+1]$ in the table, i.e., $memo[n] < B_c < memo[n+1]$, then the group-operators with size larger than n can be rejected early without being evaluated.

By imposing the above rules we observed from operator fusion scheme exploration, the search space of the on-line variant is significantly pruned, and it is a heuristic algorithm designed based on Algorithm 1. If on-line mode is set to true, then the fused operator-groups that do not satisfy these rules are deemed as invalid.

7 EVALUATION

In this section, we compare the Optimus with latest network fusers and analyze the effectiveness of Optimus over a variety of neural networks and different DNN accelerators. The experiment results demonstrate the consistent effectiveness of Optimus.

Table 2. DNN Accelerator Configurations

on-chip buffer	128 KB
PE-array	16×16
RFs/PE	64 B
dataflow	$O_w \vert C^*_{out}$
arithmetic units	16-bit fixed-point
core frequency	800 MHz

*Parallel the loop O_w and C_{out} [44].

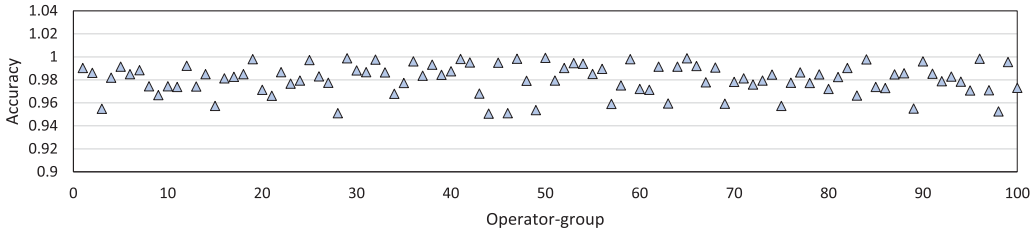

Fig. 12. Accuracy of the memory cost model results against measure results.

7.1 Experiment Setup

Workloads: We evaluate *Optimus* using several state-of-the-art CNN models, including the classic AlexNet [24], VGG16 [37], GoogleNet [38], SqueezeNet [20], MobileNet [18], ResNet [17], and the latest NasNet designed via NAS [53]. Unless otherwise specified, the batch size of these workloads is set to 4.

Baselines: To fairly evaluate *Optimus*, we use the latest network fusers *DNNVM* [43] and *Efficient-S* [50] as the baselines. To evaluate the effectiveness of our approach for DNN accelerators of different dataflow and architectures, we apply it to the design of ShiDianNao [10], Eyeriss [9], and [7].

Hardware setup: In evaluation, *Optimus* and the baselines are applied to the DNN accelerators as in Figure 4, which are implemented and synthesized with Design Compiler using 65-nm process technology. Unless otherwise specified, the DNN accelerator configurations are shown in Table 2. We also use the analysis framework as in Reference [9] to estimate the performance and the energy efficiency of the DNN accelerators for rapid performance analysis, including the energy cost for data accesses from DRAM, on-chip buffer (SRAM), PE-array (inter-PE communication), and RFs (register files) and the energy cost of MACs. CACTI 7.0 [3] is used to simulate both the SRAM buffer and the DRAM (DDR3-800).

7.2 Memory Cost Model Validation

We thoroughly validate the accuracy of the memory cost model by comparing its results to our complete DNN accelerator design generated by the synthesis toolchain. Figure 12 shows the DRAM access validation results for our memory cost model against the measure results. The accuracy is measured as in Equation (5), and it ranges from 95.05% to 99.90% with an average 98.06% for all 100 operator-groups,

$$Accuracy = 1 - \frac{|memory\ cost\ model\ result - measure\ result|}{measure\ result}. \tag{5}$$

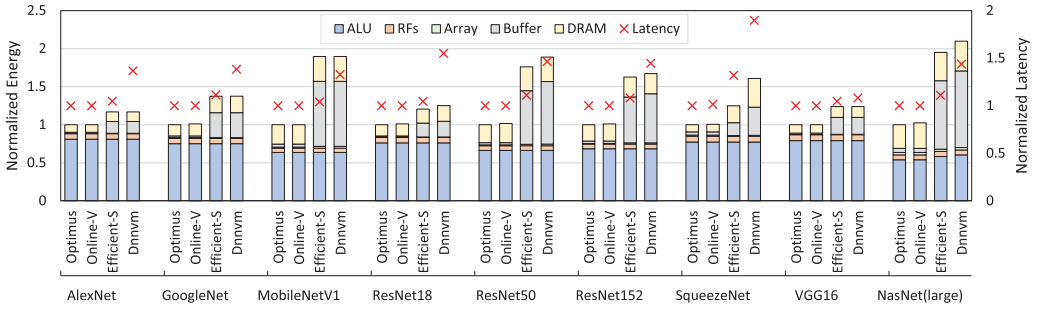

Fig. 13. Performance (right axis) and energy (left axis) comparison between baselines and *Optimus* (the results are normalized relative to *Optimus*).

7.3 Performance and Energy Comparison

We present the energy breakdown and the latency achieved by *Optimus* and the baselines. The result indicates that *Optimus* is capable of reducing the DRAM accesses for the neural networks running on the accelerators that ensures high energy efficiency and performance.

Figure 13 shows the energy comparison over all workloads between the baselines and *Optimus* (left axis). Compared to *DNNVM* and *Efficient-S*, *Optimus* obtains 1.89×–3.66× and 1.86×–3.47× energy efficiency (except ALU energy), respectively, and there is only a slight drop in results for the on-line variant (*Online-V*). That is because our operator fusion framework makes greater use of the locality of intermediate feature maps to reduce the memory access, even for on-line variant with the shrunken search space. Meanwhile, the baselines do not consider the influence of accelerator architectures, such as the PE-array size and the dataflow type, on the efficacy of operator fusion options [2, 43, 50]. Consequently, they cannot well exploit the reusability of data in the low-cost PE local RFs and inter-PE communication when fusing the operators, so that they induce relatively more frequent accesses to the on-chip buffer and higher on-chip buffer energy cost. In contrast, *Optimus* is fully aware of the accelerators architectures in Section 5, and thus it will not deteriorate the energy efficiency of other components while reducing the energy cost of DRAM.

Figure 13 also shows the end-to-end latency comparison between *Optimus* and the baselines (right axis). *Optimus* achieves significant performance gains when the DRAM cost dominates the network inference latency. The reduction in DRAM access reduces the DRAM access delay, which in turn reduces the end-to-end latency of the neural networks running on the accelerators. Besides, our methods consider the influence of the PE-array size and the dataflow type, thus *Optimus* and *Online-V* maintain high utilization of the processors' PE-array while reducing the memory access. The high utilization of the PE array can also reduce the end-to-end latency. Compared with *DNNVM* and *Efficient-S*, it shows an average improvement of 1.44× and 1.10×, respectively.

7.4 Memory-level Analysis

In this section, we examine in detail how *Optimus* reduces memory access. As shown in Figure 14, compared to *DNNVM* and *Efficient-S*, *Optimus* reduces reduces 19–75% and 17–58% DRAM accesses, respectively. The improved results vary to the specific benchmarks, because the achievable minimum memory overhead and the optimal operator fusion scheme heavily depend on the workloads. In some network architectures, especially those with a similar structure to SqueezeNet, i.e., the amount of calculation is small and the feature maps occupies memory access, the performance of prior works on operator fusion is far from optimal. To further analyze the effectiveness of the proposed optimization techniques, Figure 15 compares the memory cost of *Optimus* against that of the four alternative implementations that selectively disable one of the optimization options.

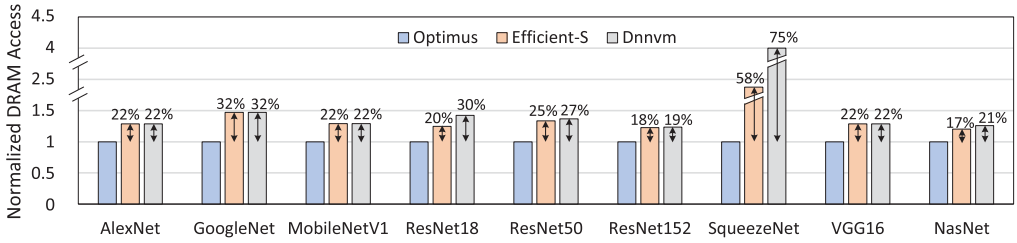

Fig. 14. Statistic on DRAM accesses of Optimus and the baselines.

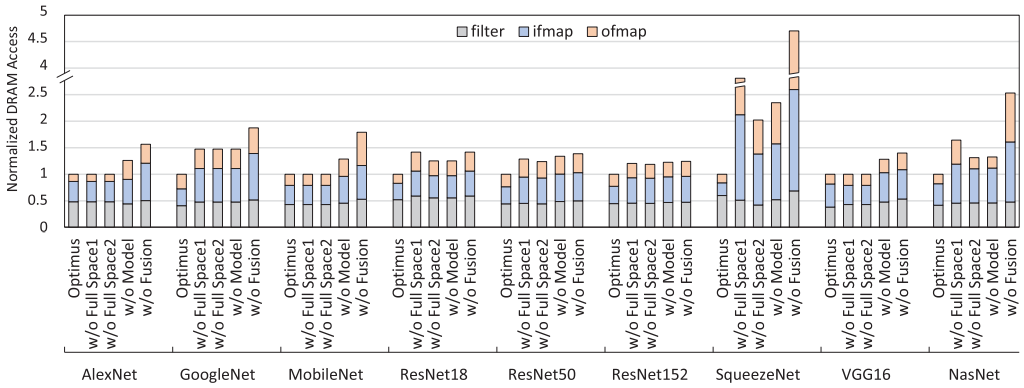

Fig. 15. Memory-level analysis.

When comparing *Optimus* to *w/o Fusion*, i.e., only merging the simple element-wise operations, e.g., CONV+BN+ReLU, operator fusion technology reduces the DRAM access significantly for all workloads, which indicates that operator fusion is crucial for the DNN accelerators to improve their efficiency when running the neural networks. In particular, networks where intermediate feature maps occupy a large amount of memory space can reduce memory access significantly through operator fusion, such as SqueezeNet. Comparing with *w/o Model* (use the memory cost function in Reference [50]), *Optimus* benefits from the memory cost model in Section 5, because it takes into consideration that the parameters of fused operator-groups may not fit into the on-chip buffer and captures the minimum overall off-chip memory access accurately using a scheduler that makes sure the optimal network fuser goes toward the correct search direction. Compared to *w/o Full Space1*, i.e., explore the restricted space of *DNNVM* in Figure 5(a), and *w/o Full Space2*, i.e., explore the restricted space of *Efficient-S* in Figure 5(b), the DNN models with complicated architecture, such as GoogleNet, ResNet, SqueezeNet, and NasNet, benefit from the full exploration of the operator fusion spaces. *Efficient-S* explores a larger space, since it allows the operator fusion across the multi-input and multi output operators and achieves better results than *DNNVM*. However, its restricted space is still not enough for the complex structures that are fully explored in *Optimus*.

Comprehensive appeal analysis, the advantage of our method is that(1) optimus searches the optimal operator fusion scheme through the entire operator fusion space, (2) optimus removes the prior memory restrictions and allow parameter-induced memory access, (3) optimus optimizes the scheduling of the fused operator groups, and (4) optimus presents a look-up table mechanism to greatly reduce the search space for online scenario with a small loss of efficiency.

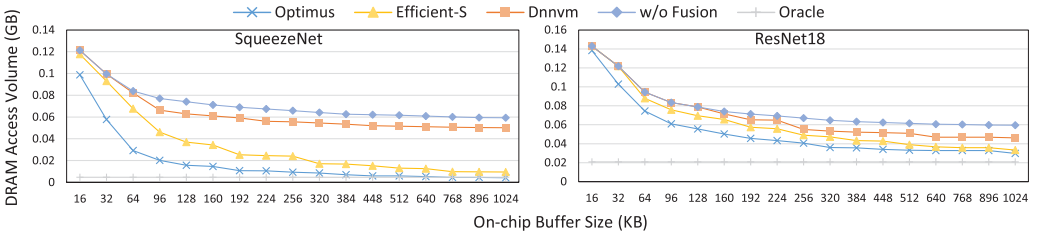

Fig. 16. Different on-chip buffer capacities.

7.5 Impact of On-Chip Memory Space

The analysis in Section 5 indicates that the on-chip buffer capacity is an important factor that influences the DRAM access traffic. We test the DRAM access volume of *Optimus* under different on-chip buffer capacities, as shown in Figure 16. *Oracle* refers to the ideal case when all the weights and feature maps of the network are accessed once without any intermediate feature-induced off-chip traffic. It is seen that *Optimus* outperforms *DNNVM* and *Efficient-S* in all cases with different on-chip buffer size. All these solutions under comparison achieve relatively more DRAM access reduction on larger on-chip buffers.

When the buffer is too small, *DNNVM* and *Efficient-S* cannot achieve better result than *w/o Fusion*, since the buffer space cannot meet the buffer occupancy requirements of the fused operator-groups as shown in Figure 7. Conversely, in our work, the marginal benefits brought by buffer increases are higher when the on-chip buffer size is small. On the one hand, we relaxed the requirements of the buffer space so that there are many profitable fused operator-groups on small buffer space, on the other hand, we made it possible to achieve less memory access for the fused operator-groups through the scheduler.

When the on-chip buffer size is larger, the restricted operator fusion space stops *DNNVM* from further reducing DRAM accesses, since there is no more operator-fusion potential available from *DNNVM* to exploit. On the contrary, when buffer size increases, *Optimus* and *Efficient-S* still produce performance gains even though at a much lower rate than that in smaller buffers, until all intermediate feature maps can be consumed on-chip and no longer evicted out to the memory. However, *Efficient-S* does not perform as well as *Optimus*, because its simplified search strategy will omit some profitable operator-groups by default, and it does not have an accurate cost estimation to drive their search algorithm.

7.6 Impact of Accelerator Architecture

As mentioned in Section 5, the throughput of the DNN accelerators determined by the PE-array size and the dataflow are described as T_* in Equation (4), which may have effects on operator fusion results.

In Figure 17, we can see *Optimus* helps DNN accelerators of different dataflow achieve similar DRAM access and energy efficiency on the same hardware configuration. One reason is that the on-chip communication is generally only a small portion of the total energy when well exploiting the on-chip resources, and the other reason is that *Optimus* is hardware-aware and able to locate the optimal operator fusion solutions for different dataflows to obtain the close DRAM access energy under the same on-chip buffer space. Figure 17 also shows that smaller RF can decrease the energy consumption due to the much lower energy cost per access of the smaller RF but almost no impact on the DRAM energy.

Figure 18 shows the DRAM access number and the energy efficiency of *Optimus* on different configuration as shown in Table 3. According to the Equation (4), enlarging the PE-array, i.e.,

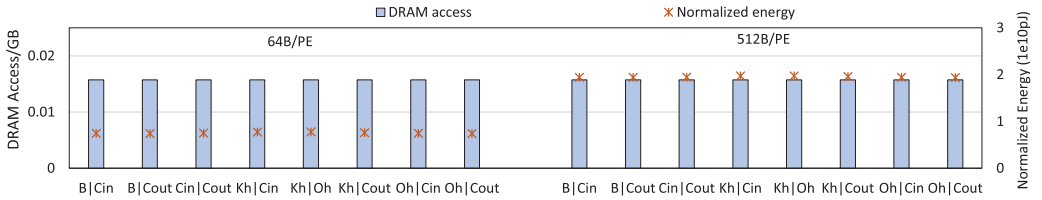

Fig. 17. Different dataflow (SqueezeNet).

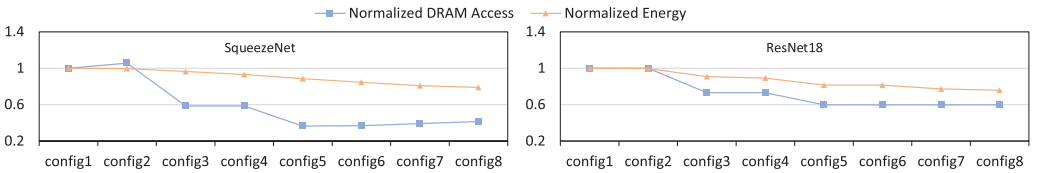

Fig. 18. The impact of PE-array size.

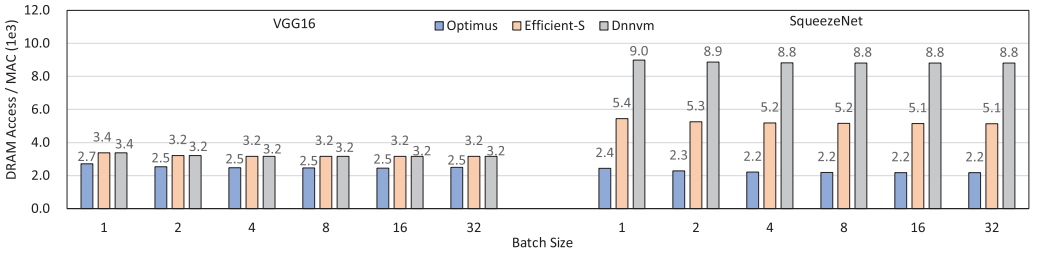

Fig. 19. Impact of batch size.

Table 3. Different Configurations of Accelerator Architectures

Components	config1	config2	config3	config4	config5	config6	config7	config8
PE array	16 × 16	32 × 32	16 × 16	32 × 32	8 × 8	16 × 16	32 × 32	64 × 64
RFs/PE	64 B	64 B	64 B	64 B	64 B	64 B	32 B	32 B
On-chip buffer	128 KB	128 KB	256 KB	256 KB	512 KB	512 KB	512 KB	512 KB

enlarging T_*, will result in a slight increase in the number of memory requests, but the total energy consumption will decrease due to higher computation efficiency and shortened latency. In general, the optimal operator fusion solution is varying to the accelerator configurations, and the *Optimus* is generalizable to different accelerator architectures.

7.7 Impact of Batch Size

Figure 19 shows the impact of batch size on the memory-level performance of VGG16 and SqueezeNet. *Optimus* is significantly superior to *DNNVM* and *Efficient-S* under a variety of batch sizes. When the batch size of VGG16 and SqueezeNet is greater than 4, the DRAM access number stops dropping, because increasing the batch size can improve the reuse opportunities of parameters only when the batch size is small. However, when the batch size becomes larger, the limited on-chip memory space prevents it from taking advantage of the data reuse opportunities.

7.8 Performance on Different Accelerators

Optimus can easily be applied to most of the deep learning accelerators and minimize their off-chip memory accesses. As a case study, we integrate *Optimus* into three DNN accelerators, ShiDianNao,

Table 4. Reduction in Memory Access (%) Compared to the Accelerator Baselines

	AlexNet	VGG16	GoogleNet	ResNet18	ResNet50	ResNet152	MobileNet	SqueezeNet
Eyeriss	33.25	31.75	49.56	34.44	35.09	23.77	55.05	87.59
ShiDianNao	37.86	52.45	49.45	24.39	34.43	23.88	57.28	85.26
[7]	32.43	27.41	47.31	16.86	33.82	25.32	57.08	84.29

Eyeriss, and Reference [7], as shown in Table 4. Note that the buffer of ShiDianNao is split into three units: an input buffer, an output buffer, and a synapse buffer. Reference [7] is designed to achieve the communication lower bound for the Conv operators but do not consider the optimization of operator fusion. *Optimus* even achieves better results on Eyeriss without using data compression when compared to baseline Eyeriss with sparse compression, and it outperforms the baseline ShiDianNao when applied to the ShiDianNao accelerator. *Optimus* also further drops the off-chip communication lower bound for the convolutional accelerator proposed in Reference [7], because *Optimus* can eliminate the unnecessary off-chip traffic induced by intermediate feature maps.

7.9 Algorithm Overhead Analysis

In this section, we analyze the overhead of our algorithm that is implemented using two different methods: top-down with memorization in Reference [5] and bottom-up with tabulation in this article. The top-down method makes it easy to think of optimal sub-structures, and it can be implemented directly through recursive calls with memorization. But it runs slowly due to lot of recursive calls and return statements. The bottom-up method is difficult to consider state transition relations, but it runs fast as we can directly access the sub-problem states from the table.

Table 5 presents the number of evaluated operator-group options and the time overhead it takes by the algorithms to search for the final fusion solutions. The experiment is conducted on Inter Core i7-6700 CPU @ 3.40 GHz, and the OS is Ubuntu 16.04.4 LTS. It shows that the execution time overhead of *Optimus* is within a reasonable range, especially with the bottom-up method. For example, the *Optimus* program takes 0.48–8713.16 ms to generate the optimal operator fusion solutions that is also much faster than what has been reported in previous work [21, 43, 50]. Although ResNet1202 has the most operators, it has less execution time overhead than NasNet, because it has fewer successor vertices in the DAG and the complexity of the algorithm is $O(2^{\max_i |succ(v_i)|} |\mathcal{V}|^2)$. Furthermore, the online variant costs only 0.11–274.10 ms to achieve near-optimal fusion schemes, since we shrink the search space according to the empirical rules in Section 6. Since the ISOA Applying Decision in Algorithm 2 can be implemented along with Algorithm 1, the memory cost model only needs to search $t_h^{(n)}$ and $t_w^{(n)}$, which is fast, typically shorter than 1 ms.

After the optimization of Optimus, the numbers of fused operator-groups to VGG, ResNet50, ResNet152, ResNet1202, GoogleNet, SqueezeNet, and NasNet are 8, 31, 99, 841, 21, 8, and 328, respectively.

8 RELATED WORK

State-of-the-art DNN processors: To overcome the computing challenge of DNNs, lots of specialized ASIC-based [6, 8–10] and FPGA-based [2, 15, 36, 41, 42, 45, 47] DNN processors have been proposed for better performance or energy efficiency. References [7, 9, 10, 15, 36], and so on, process networks layer by layer, targeting at reduce the data transfer incurred for each layer individually. References [2, 39, 42] instantiate a fusion design for dedicated NN models on FPGA; however, it is difficult to scale these designs for deeper and more complicated networks, and

Table 5. Execution Time of Operator Fusion Algorithm

workload	max#|*succ*|	#operators	#groups		latency(ms)			
			Optimus	Online-V	Optimus		Online-V	
					Top-down	Bottom-up	Top-down	Bottom-up
VGG	1	18	171	80	0.36	0.48	0.18	0.11
ResNet50	2	52	3510	195	14.59	2.73	0.64	0.29
ResNet152	2	154	24250	467	173.43	5.01	3.22	0.23
ResNet1202	2	1204	80825	8085	642.65	383.95	66.29	5.26
GoogleNet	4	72	50436	423	514.53	27.89	3.43	0.64
SqueezeNet	2	38	1057	216	3.39	2.19	0.47	0.24
NasNet	10	456	2.9e+07	1627	3.28e+05	8713.16	45.38	274.10

re-configuration overheads are introduced when switching to other models. Sparse DNN accelerators, e.g., References [14–16, 34, 46, 48, 49], improved NN efficiency by avoiding redundant operations and reducing memory footprints, which are orthogonal to our exploration.

Operator fusion technique: References [4, 21, 22, 26, 27, 31–33], and so on, introduced fusion methods to various computer vision and computational photography algorithms for general-purpose processors, e.g., CPU and GPU. The target algorithms of References [21, 31, 32] perform 1D or 2D convolution, rather than the 3D convolutions needed for DNNs. Reference [21] directly restricted the number of nodes in a single group to reduce the fusion searching efforts, exploring more potentially profitable fusion opportunities compared with References [31, 32]. References [4, 22, 26, 27, 33] take as input the fusion specifications and generates candidate basic operators using the given specifications. They do not allow two complex operators (e.g., Conv + Conv) to fuse, because attempting a combined execution of two complex operators will be too complicated and will likely negatively impact register and cache usage in general-purpose processors. However, focusing on specialized processor allows us to simply change the execution order of each layer in the fused operator-groups to consume the intermediate data immediately and to use a large amount of information to model the achievable minimum memory overhead of the fused operator-groups.

On DNN processors, References [42, 51] use DP algorithm to find the optimal operator fusion option for simple DNN models without branches unlike the straightforward enumeration-and-evaluate strategy as in Reference [2]. References [43, 50] attempt to explore the restricted operator fusion space for the complicated networks. Reference [50] proposed a backtracking-based approach to determine the execution order within a fused operator-group, releasing the buffer occupation as soon as possible, and Reference [42] proposed line buffer technique to avoid introducing additional buffer resource consumption or computational redundancy for overlapping input data in operator fusion, which are orthogonal to our work. Reference [19] designed an accelerator for computational imaging to fused all operators into a fused operator-group. To the best of our knowledge, we are the first work to give the achievable minimum memory-cost model for fused operator-groups and also the first work to discuss how to reach fast and high-performance online operator fusion.

9 CONCLUSION

In this article, we redefine the operator-fusion problem for neural network applications and propose *Optimus*, a novel neural network fusion framework to achieve the optimal operator fusion for arbitrary network architectures and different state-of-the-art DNN processors. *Optimus* includes an off-line and an on-line operator fusion algorithm and an accurate memory cost model with a scheduler for fused operator-groups that directs the search procedure toward the memory-optimal operator fusion solution. In evaluation with state-of-the-art workloads and DNN accelerator

implementations, the experiment results demonstrate that *Optimus* reduces 17–75% off-chip memory accesses and obtains up to 3.66× energy efficiency over the baselines on processors of different architectures and dataflows.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*. 265–283.

[2] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. IEEE, 1–12. https://doi.org/10.1109/MICRO.2016.7783725

[3] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Trans. Archit. Code Optim.* 14, 2 (2017), 14:1–14:25. https://doi.org/10.1145/3085572

[4] Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Alexandre V. Evfimievski, and Prithviraj Sen. 2018. On optimizing operator fusion plans for large-scale machine learning in systemml. arXiv:1801.00829. Retrieved from https://arxiv.org/abs/1801.00829.

[5] Xuyi Cai, Ying Wang, and Lei Zhang. 2021. Optimus: Towards optimal layer-fusion on deep learning processors. In *Proceedings of the 22nd ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. 67–79. https://doi.org/10.1145/3461648.3463848

[6] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the International Conference on Architectural Cupport for Programming Languages and Operating Systems (ASPLOS'14)*. 269–284. https://doi.org/10.1145/2541940.2541967

[7] Xiaoming Chen, Yinhe Han, and Yu Wang. 2020. Communication lower bound in convolution accelerators. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*. IEEE, 529–541. https://doi.org/10.1109/HPCA47549.2020.00050

[8] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'14)*. IEEE, 609–622. https://doi.org/10.1109/MICRO.2014.58

[9] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA'16)*. 367–379. https://doi.org/10.1145/3007787.3001177

[10] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA'15)*. 92–104. https://doi.org/10.1145/2749469.2750389

[11] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. 2018. A configurable cloud-scale DNN processor for real-time AI. In *Proceedings of the ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18)*. IEEE, 1–14. https://doi.org/10.1109/ISCA.2018.00012

[12] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. Tangram: Optimized coarse-grained dataflow for scalable NN accelerators. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*. 807–820. https://doi.org/10.1145/3297858.3304014

[13] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, et al. 2020. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. IEEE, 681–697. https://doi.org/10.1109/MICRO50266.2020.00062

[14] Ashish Gondimalla, Noah Chesnut, Mithuna Thottethodi, and T. N. Vijaykumar. 2019. SparTen: A sparse tensor accelerator for convolutional neural networks. In *Proceedings of the International Symposium on Microarchitecture (MICRO'19)*. 151–165. https://doi.org/10.1145/3352460.3358291

[15] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. 2017. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. 75–84. https://doi.org/10.1145/3020078.3021745

[16] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the International Symposium on Computer Architecture (ISCA'16)*. 243–254. https://doi.org/10.1109/ISCA.2016.30

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861 (2017). Retrieved from https://arxiv.org/abs/1704.04861.

[19] Chao-Tsung Huang, Yu-Chun Ding, Huan-Ching Wang, Chi-Wen Weng, Kai-Ping Lin, Li-Wei Wang, and Li-De Chen. 2019. ecnn: A block-based and highly-parallel cnn accelerator for edge inference. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 182–195. https://doi.org/10.1145/3352460.3358263

[20] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. arXiv:1602.07360 (2016). Retrieved from https://arxiv.org/abs/1602.07360.

[21] Abhinav Jangda and Uday Bondhugula. 2018. An effective fusion and tile size model for optimizing image processing pipelines. *ACM SIGPLAN Not.* 53, 1 (2018), 261–275. https://doi.org/10.1145/3200691.3178507

[22] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 47–62. https://doi.org/10.1145/3341301.3359630

[23] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 1–12. https://doi.org/10.1145/3079856.3080246

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90. https://doi.org/10.1145/3065386

[25] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. 2020. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *IEEE Micro* 40, 3 (2020), 20–29. https://doi.org/10.1109/MM.2020.2985963

[26] Rasmus Munk Larsen and Tatiana Shpeisman. 2019. TensorFlow Graph Optimizations. Retrieved from https://www.tensorflow.org/guide/graph_optimization.

[27] Chris Leary and Todd Wang. 2017. XLA: TensorFlow, compiled. Retrieved from https://www.tensorflow.org/xla.

[28] Gang Li, Zejian Liu, Fanrong Li, and Jian Cheng. 2021. Block convolution: Towards memory-efficient inference of large-scale CNNs on FPGA. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 41, 5 (2021), 1436–1447. https://doi.org/10.1109/TCAD.2021.3082868

[29] Jiajun Li, Guihai Yan, Wenyan Lu, Shuhao Jiang, Shijun Gong, Jingya Wu, and Xiaowei Li. 2018. SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*. IEEE, 343–348. https://doi.org/10.23919/DATE.2018.8342033

[30] Orlando Moreira, Merten Popp, and Christian Schulz. 2017. Graph partitioning with acyclicity constraints. arXiv:1704.00705. Retrieved from https://doi.org/10.4230/LIPIcs.SEA.2017.30

[31] Ravi Teja Mullapudi, Andrew Adams, Dillon Sharlet, Jonathan Ragan-Kelley, and Kayvon Fatahalian. 2016. Automatically scheduling halide image processing pipelines. *ACM Trans. Graph.* 35, 4 (2016), 1–11. https://doi.org/10.1145/2897824.2925952

[32] Ravi Teja Mullapudi, Vinay Vasista, and Uday Bondhugula. 2015. Polymage: Automatic optimization for image processing pipelines. *ACM SIGARCH Comput. Arch. News* 43, 1 (2015), 429–443. https://doi.org/10.1145/2694344.2694364

[33] Wei Niu, Jiexiong Guan, Yanzhi Wang, Gagan Agrawal, and Bin Ren. 2021. 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*. 883–898. https://doi.org/10.1145/3453483.3454083

[34] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA'17)*. IEEE, 27–40. https://doi.org/10.1145/3079856.3080254

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* 32 (2019), 8026–8037.

[36] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. 2016. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'16)*. 26–35. https://doi.org/10.1145/2847263.2847265

[37] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR'15)*, Yoshua Bengio and Yann LeCun (Eds.).

[38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9. https://doi.org/10.1109/CVPR.2015.7298594

[39] Stylianos I. Venieris and Christos-Savvas Bouganis. 2017. fpgaConvNet: Automated mapping of convolutional neural networks on FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 291–292. https://doi.org/10.1145/3020078.3021791

[40] Siqi Wang, Anuj Pathania, and Tulika Mitra. 2020. Neural network inference on mobile socs. *IEEE Des. Test* 37, 5 (2020), 50–57. https://doi.org/10.1109/MDAT.2020.2968258

[41] Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li. 2016. DeepBurning: Automatic generation of FPGA-based learning accelerators for the neural network family. In *Proceedings of the 53nd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. IEEE, 1–6. https://doi.org/10.1145/2897937.2898002

[42] Qingcheng Xiao, Yun Liang, Liqiang Lu, Shengen Yan, and Yu-Wing Tai. 2017. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference*. 1–6. https://doi.org/10.1145/3061639.3062244

[43] Yu Xing, Shuang Liang, Lingzhi Sui, Zhen Zhang, Jiantao Qiu, Xijie Jia, Xin Liu, Yushun Wang, Yi Shan, and Yu Wang. 2019. DNNVM: End-to-End compiler leveraging operation fusion on FPGA-based CNN accelerators. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 187–188. https://doi.org/10.1145/3289602.3293972

[44] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. 2020. Interstellar: Using halide's scheduling language to analyze DNN accelerators. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*. 369–383. https://doi.org/10.1145/3373376.3378514

[45] Shulin Zeng, Guohao Dai, Hanbo Sun, Kai Zhong, Guangjun Ge, Kaiyuan Guo, Yu Wang, and Huazhong Yang. 2020. Enabling efficient and flexible FPGA virtualization for deep learning in the cloud. In *Proceedings of the IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'20)*. IEEE, 102–110. https://doi.org/10.1109/FCCM48280.2020.00023

[46] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-x: An accelerator for sparse neural networks. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. IEEE, 1–12. https://doi.org/10.1109/MICRO.2016.7783723

[47] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2018. DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'18)*. IEEE, 1–8. https://doi.org/10.1145/3240765.3240801

[48] Xiandong Zhao, Ying Wang, Xuyi Cai, Cheng Liu, and Lei Zhang. 2019. Linear symmetric quantization of neural networks for low-precision integer hardware. In *Proceedings of the International Conference on Learning Representations*.

[49] Xiandong Zhao, Ying Wang, Cheng Liu, Cong Shi, Kaijie Tu, and Lei Zhang. 2020. BitPruner: Network pruning for bit-serial accelerators. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC'20)*. IEEE, 1–6. https://doi.org/10.1109/DAC18072.2020.9218534

[50] Shixuan Zheng, Xianjue Zhang, Daoli Ou, Shibin Tang, Leibo Liu, Shaojun Wei, and Shouyi Yin. 2020. Efficient scheduling of irregular network structures on CNN accelerators. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 39, 11 (2020), 3408–3419. https://doi.org/10.1109/TCAD.2020.3012215

[51] Li Zhou, Hao Wen, Radu Teodorescu, and David HC Du. 2019. Distributing deep neural networks with containerized partitions at the edge. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge'19)*.

[52] Yanqi Zhou and David Wentzlaff. 2014. The sharing architecture: Sub-core configurability for IaaS clouds. *ACM SIGPLAN Not.* 49, 4 (2014), 559–574. https://doi.org/10.1145/2654822.2541950

[53] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*. 8697–8710. https://doi.org/10.1109/CVPR.2018.00907

Received 14 October 2021; revised 10 January 2022; accepted 21 February 2022