Shortcut Mining: Exploiting Cross-layer Shortcut Reuse in DCNN Accelerators

Arash Azizimazreah, and Lizhong Chen
School of Electrical Engineering and Computer Science
Oregon State University, USA
{azizimaa, chenliz}@oregonstate.edu

ABSTRACT

Off-chip memory traffic has been a major performance bottleneck in deep learning accelerators. While reusing on-chip data is a promising way to reduce off-chip traffic, the opportunity on reusing shortcut connection data in deep networks (e.g., residual networks) have been largely neglected. Those shortcut data accounts for nearly 40% of the total feature map data. In this paper, we propose *Shortcut Mining*, a novel approach that "mines" the unexploited opportunity of on-chip data reusing. We introduce the abstraction of logical buffers to address the lack of flexibility in existing buffer architecture, and then propose a sequence of procedures which, collectively, can effectively reuse both shortcut and non-shortcut feature maps. The proposed procedures are also able to reuse shortcut data across any number of intermediate layers without using additional buffer resources. Experiment results from prototyping on FPGAs show that, the proposed Shortcut Mining achieves 53.3%, 58%, and 43% reduction in off-chip feature map traffic for SqueezeNet, ResNet-34, and ResNet-152, respectively and a 1.93X increase in throughput compared with a state-of-the-art accelerator.

1. INTRODUCTION

State-of-the-art deep learning models, especially deep convolutional neural networks (DCNNs), can achieve a surprisingly high accuracy in a wide range of tasks, thus are being deployed in many conventional and emerging fields. Due to the special computation and memory behaviors of DCNNs, hardware accelerators have become increasingly important to achieve the speedup and power efficiency that are not possible in today's general-purpose platforms. However, because of the memory intensive nature of DCNNs, a considerable amount of off-chip memory traffic can be generated by the accelerators to access feature maps and weights. These off-chip accesses are extremely costly in terms of both latency and energy [35]. Unfortunately, the situation is only getting worse as deep learning models have started to employ a large number of convolutional neural network (CNN) layers for higher accuracy, with some exceeding a thousand layers [9].

A very promising approach to mitigate this issue directly is to increase on-chip data reuse. While limited prior work has investigated some reuse scenarios (e.g., [2][4][20]), recent advancement in DCNNs and hardware platforms have led to changed behaviors and properties of the network models, which results in new opportunities for data reusing. First, the percentage of off-chip data from feature maps increases rapidly (e.g., up 72% in SqeezeNet; more detail in Section 2). This is mainly because of deeper networks and an outburst of optimizations in reducing weight data. Hence, there is a pressing need to reduce off-chip feature map traffic. Second,

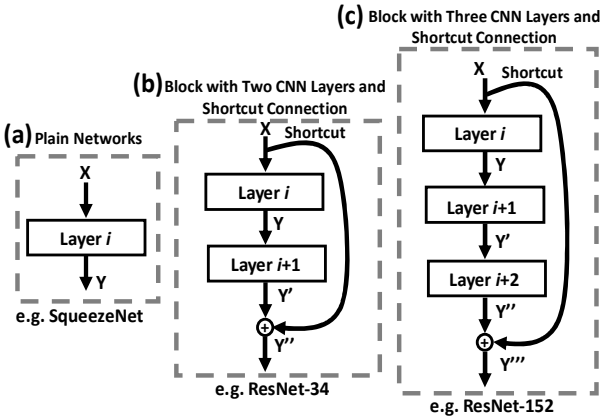

Fig. 1: Plain networks (a) and networks with shortcut (b)(c).

shortcut connections are being introduced to address the gradient "wash out" problem in deep structures, which is critical in developing highly accurate CNNs [9][13][34][38]. Fig. 1 shows the building blocks of a plain network (Squeeze-Net [14]) and two residual networks including ResNet-34 and ResNet-152 [9]. The ResNet-152, for example, includes 50 shortcut connections that account for nearly 40% of the feature map data. This presents a significant, unexplored opportunity for reusing the shortcut data. Third, data reusing is made more viable by the continuing advancement in manufacturing technology. A number of FPGA and other accelerator platforms (e.g., [36][41][15][40][11]) have on-chip buffers that can fit a considerable portion or even all of the data for a CNN layer. Therefore, a substantial amount of shortcut and non-shortcut feature map data remains on-chip at the end of a layer processing, which can and should be exploited for reuse. To-date, little work has been conducted to exploit the new opportunities discussed above.

In this work, we propose *Shortcut Mining (SCM)*, a novel approach that "mines" the largely unexplored opportunity of reusing shortcut and feature map data to reduce off-chip traffic. To achieve this objective, several major challenges must be addressed: (1) the lack of flexibility in existing buffer architecture prevents on-chip buffers from being allocated and utilized at the needed granularity; (2) reusing shortcut data requires the data to be somehow preserved across multiple layers, which is difficulty without incurring extra buffer overhead; and (3) it is challenging to provide a unified solution that works for different networks, as the building block could contain an arbitrary number of layers. Thus, the shortcut data needs to be preserved for any number of layers.

In this paper, we address the first challenge by introducing the use of logical buffers that are formed dynamically from a pool of physical buffer banks. This allows buffer operations

2378-203X/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCA.2019.00030

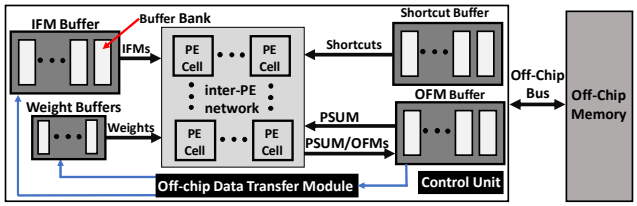

Fig. 2: Convolution layer processor (CLP) datapath.

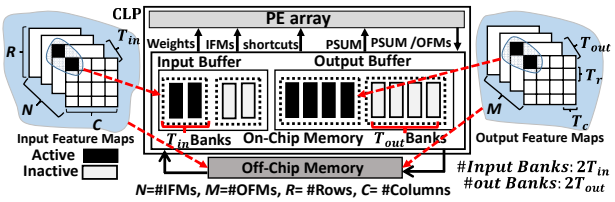

Fig. 3: Tiling and ping-pong buffers in a CLP.

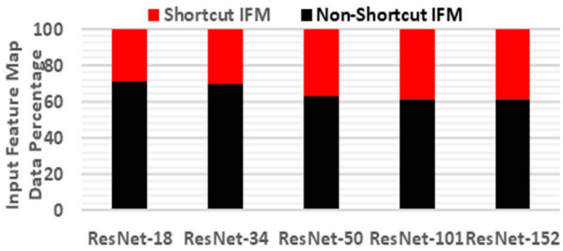

Fig. 5: Feature map data in various ResNets.

to be done at individual bank level, without physically moving the data. Enabled by this, we develop three procedures, namely Prolog, Kernel and Epilog, that work collaboratively to allow shortcut data to be reused across any number of layers in the building block of DCNNs. On the high level, the basic idea is for the Prolog procedure to save the shortcut data of a CNN layer in a certain region of the on-chip buffer called the *invariant buffer space*. Later after several layers, the Epilog procedure can restore the shortcut data from the invariant buffer space for the shortcut to be reused. In the meantime, the Kernel procedure is executed multiple times, one for each layer in a building block. At each layer, the Kernel reuses the non-shortcut feature maps of the previous layer as the input of the current layer and, at the same time, keep the invariant buffer space untouched, regardless of how many times the Kernel is executed. The collective result of the three procedures is that shortcut feature maps are preserved across layers while non-shortcut feature maps are reused immediately after each layer. All these operations are achieved without using additional buffer resources. Experiment results from prototyping on FPGAs show that, the proposed Shortcut Mining is able to achieve 53.3%, 58%, and 43% reduction in off-chip feature map traffic for SqueezeNet, ResNet-34, and ResNet-152, respectively and a 1.93X increase in performance (inference operation throughput) compared with a state-of-the-art DCNN accelerator. The main contributions of this paper are the following:

- Identifying new opportunities in reusing on-chip feature map and shortcut data in DCNN accelerators
- Developing a novel scheme to solve the difficult problem of reusing shortcut data across any number of layers
- Demonstrating the effectiveness of the proposed scheme by implementing on FPGAs

The rest of this paper is organized as follows. Section 2 provides more background and motivation on the reusing opportunity of on-chip data in DCNN accelerators. Section 3 illustrates the associated challenges. In Section 4, we describe the proposed scheme and architecture in detail. The accelerator implementation is explained in Section 5, and evaluation results and analysis are presented in Section 6. Finally, related work is summarized in Section 7, and Section 8 concludes this paper.

2. BACKGROUND AND MOTIVATION

2.1 DCNN Accelerator Architecture

Current state-of-the-art DCNN accelerators use a *convolution layer processor* (CLP) to evaluate each CNN layer sequentially [15]. A general CLP has an array of processing elements (PEs) and various on-chip buffers. As shown in Fig. 2, the *processing element (PE)* array fetches *input feature maps (IFMs)* from the input buffer, *weights* from the weight buffer, *shortcut connection data* from the shortcut buffer, and then calculates *output feature maps (OFMs)* that are put into the output buffer. If multiple iterations are needed to generate the *OFMs*, the PE array also loads the *partial sum (PSUM)* that contains the partial OFMs accumulated so far, and then outputs the updated PSUM or the final OFMs to the output buffer. The granularity of a PE cell can vary from a simple multiplier-and-adder (fine-grained) such as in Google TPUs [15] where PE cells are arranged in a two-dimensional systolic array, to a vector-dot-product engine (coarse-grained) such as in Microsoft's BrainWave DPUs [12]. In either case, a critical design consideration is to reduce off-chip memory accesses that have long latency and large power consumption. Consequently, several techniques are commonly adopted in designing the on-chip buffers in DCNN accelerators, as described below.

Banked On-chip Buffer Designs: Recent DCNN accelerators, both on ASIC and FPGA based platforms, have started to provide large on-chip buffers (e.g., [22][36][40][41], and TPU has 28MB of on-chip buffer capacity [15]). Like off-chip memory designs, a large on-chip buffer can be partitioned into smaller banks in order to avoid long word-lines and bit-lines that have high capacitance [8]. Moreover, banked buffers provide more read and write ports, allowing multiple simultaneous accesses to IFMs, weights, shortcut connections, and OFMs. Therefore, the banked organization is essential for low-latency, energy-efficient, and high-bandwidth data transfer between the PE array and on-chip buffers [4][5].

Tiling Techniques: As feature maps can be larger than the on-chip input and output buffers, tiling techniques are often used to reduce the requirement of on-chip buffer sizes [23][27]. As depicted in Fig. 3, IFMs ($N \times R \times C$) and OFMs ($M \times R \times C$) are organized in three dimensional structures. With tiling factors of T_{in}, T_r and T_c for IFMs, the IFMs can be divided into $\lceil \frac{N}{T_{in}} \rceil \times \lceil \frac{R}{T_r} \rceil \times \lceil \frac{C}{T_c} \rceil$ tiles; in each iteration, T_{in} IFM tiles are loaded into the input buffer. Similarly, the OFMs are divided into $\lceil \frac{M}{T_{out}} \rceil \times \lceil \frac{R}{T_r} \rceil \times \lceil \frac{C}{T_c} \rceil$ tiles. As the T_{in} IFM tiles may not

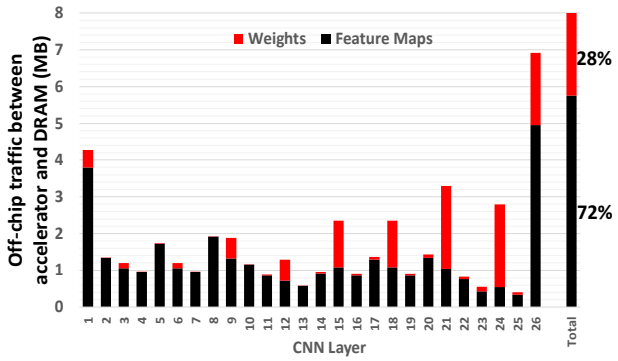

Fig. 4: Off-chip traffic breakdown for SqueezeNet.

include all the IFM tiles that are required to calculate an OFM tile, multiple iterations of input loading and calculation may be needed, and partial sum is generated and accumulated in each iteration[1].

Ping-Pong Buffers: Due to large off-chip latency, many DCNN accelerators (e.g., [29][30][23][42][2][15][17][35]) use the decoupled access/execute architecture [31] to overlap communication latency with computation time. This is referred to as the ping-pong buffer technique in a CLP. As shown in Fig. 3, the input and output buffers (blacked rectangles) that are being used by the PEs during the current iteration are called *active* input and *active* output buffers, respectively. The other set of input and output buffers are called *inactive* input and output buffers. The inactive input buffer is used to preload the T_{in} IFM tiles that are needed by the next iteration; whereas the inactive output buffer temporarily holds the T_{out} OFM tiles from the last iteration while those tiles are being written to the off-chip memory. The ping-pong buffer hides off-chip memory access latency by overlapping loading/storing data with computation. The weight and shortcut buffers work in a similar manner but are omitted in Fig. 3 for better clarity.

2.2 Need for Reducing Off-chip Feature Map Traffic

While the above optimizations have helped to improve the memory system of DCNN accelerators, there remains a significant need to reduce off-chip memory traffic, particularly from loading and storing feature maps. First, it is imperative and beneficial to minimize all types of off-chip traffic due to its large delay and energy consumption, e.g., only 1pJ for a 32-bit floating point addition but 640pJ for accessing a 32-bit word in DRAM under 45nm CMOS technology [7][10]. Second, there is an emerging need to reduce off-chip traffic that is caused by feature maps. This is due to the fact that, as deep learning models evolve to deeper structures, the percentage of data from feature maps increases rapidly. For instance, feature maps account for 25% data in earlier models such as AlexNet, but become more than 50% in recent models such as VGG and GoogLeNet [2].

Moreover, the phenomenon is becoming more and more evident as existing techniques in reducing data from weights and from feature maps are not equally effective. A great deal of effort has been made to reduce the size of weights such as pruning [14][18][7] and re-architecting CNN networks [14], most of which have been quite effective. For example, by employing filter (weight) pruning, the parameters of ResNet-101 and VGG-16 are reduced by 32.4% and 64%, respectively [18], and SqueezeNet is able to reach the accuracy of AlexNet but with 50X smaller weight data [14]. Also, by using batch processing, weights can be further reduced as they are reused for multiple inputs. In comparison, it is more difficult to reduce feature map data. Even for well-designed feature map compression techniques, the effectiveness is still limited and depends on input data (e.g., less 2X reduction for AlexNet [4]). In Fig. 4, we have examined the off-chip traffic of a SqueezeNet with 26 layers (2 CNN layers, plus 8 fire modules each having 3 layers). As can be seen, feature map traffic is predominant in 22 out of the 26 layers. On average, feature maps account for nearly 72% of the total off-chip traffic, exemplifying a pressing need to reduce this type of traffic.

Table 1: On-chip buffers in several modern platforms.

Chips	On-Chip Buffers	Process Technology
Xilinx Virtex UltraScale+ VU37P [36][41][40]	47.2 MB	16nm
Google's TPU [15]	28MB	28nm
Intel Stratix 10 TX 2800 [11]	30.5MB	14nm
Xilinx Virtex-7 690T [22]	7.79MB	28nm

2.3 Opportunity for Reusing Feature Maps

As the output feature maps of the previous layer are the input feature maps of the next layer, a promising way to reduce off-chip feature map traffic is to increase on-chip data reuse. As shown in Fig. 3, when a layer completes the computation of the last iteration, T_{out} OFM tiles remain in the on-chip active output buffer. With larger on-chip buffers, more OFM tiles can be kept on-chip and reused without having to fetch from the off-chip memory. In the ideal case, if the output buffer is large enough to hold all the OFMs of layer i, no IFMs of layer i+1 need to be fetched. While this might seem as a wishful thinking in the past, recent process technologies have made this a reality for the most part.

To understand this better, Table 1 lists the available on-chip buffers and process technology for several modern accelerator platforms including Xilinx UltraScale+ VU37P FPGA, Google's TPU chip, Intel Stratix 10 FPGA which is used to implement Microsoft BrainWave DPU, and Xilinx Virtex-7 690T. These platforms all provide megabytes of on-chip buffers. To put the numbers in Table 1 in perspective, the four largest CNN layers in ResNet-152 require 9.19MB, 6.95MB, 3.7MB, and 2.27MB of data, respectively, in 32-bit floating point. These examples demonstrate that modern acceleration platforms, in theory, have on-chip buffers that are large enough to hold a considerable portion or even all of the data for a CNN layer. This offers a new opportunity in reusing feature map data and should be exploited.

2.4 Opportunity for Reusing Shortcut Connection Data

Another largely unexplored opportunity that can greatly reduce off-chip traffic is to reuse shortcut connection data. For

[1] If the size of the input PE array (T_n) is smaller than input buffer size T_{in}, another level of tiling can be performed to use the PE array multiple times to consume T_{in} IFM tiles.

plain networks such as the SqueezeNet example in Fig. 4, the input feature maps of a layer are all from the output feature maps of the previous layer. However, it has been shown that it is essential to have shortcut connections for very deep networks to achieve high accuracy [9][13]. This results in the so-called residual networks, such as ResNets [9], Wide Residual Networks[44], ResNeXt[38]. In these networks, the input feature maps of a layer may include the feature maps of one or multiple previous layers. In other words, there are shortcut connections that forward the IFMs of layer *i* to subsequent convolutional layers that are not directly connected (e.g. layer *i*+2, layer *i*+3, etc.). These shortcut connections change the composition of feature map data. Fig. 5 plots the breakdown of input feature maps for different residual DCNNs. As can be seen, 30% to 40% of all the IFMs are shortcut connections. In particular, ResNet-152 has 50 shortcut connections, which accounts for around 40% of the IFMs that could be reused.

Unfortunately, despite the considerable amount of shortcut connection data, little research has been conducted to explore this opportunity. While related work is discussed in detail in Section 7, most existing works (e.g., [42][3][4][24][19]) focus on optimization within a single CNN layer, such as loop operations. However, at the end of a layer processing, the OFMs in the on-chip output buffer still need to be written to off-chip memory (see Fig. 3), and the IFMs of the next layer need to be fetched from the memory again. Both shortcut and non-short IFMs are not reused in this case. Another interesting approach, fused-CNN [2], cascades multiple layers together to avoid off-chip accesses. All the intermediate tiles for the fused layers are held on-chip which results in considerable on-chip memory (e.g., 5 layers of VGGNet-E already consumes 85% of the BRAMs in Xilinx Virtex-7 [2]). Thus, while the fused-CNN works very well for shallow networks such as AlexNet and VGGNets, it is less effective for recent deeper networks (where shortcut connections are prevalent) due to its relatively large on-chip memory requirement.

3. CHALLENGES FOR CROSS-LAYER REUSING

With the need for reusing shortcut and non-shortcut feature maps, and the physical feasibility from large on-chip buffers, this section analyzes several main limiting factors and challenges that must be addressed to realize effective cross-layer data reusing.

3.1 Providing Flexible Buffer Architecture

The current buffer architecture shown in Fig. 3 is static in the sense that input buffers are always used for PE input and output buffers are always used for PE output. This would be very inefficient for cross-layer data reuse where the role of a given feature map may be switched dynamically between input and output, as well as between active and inactive.

Additionally, the granularity of buffer allocation is at the set level rather than at the bank level. That is, multiple banks are grouped together as a set (e.g., the inactive output buffer). As data in this buffer is being written to off-chip memory, more and more banks in this output buffer become available. However, the newly available banks remain unused (locked) until the entire inactive output buffer finishes writing. We have derived that the best bank utilization at the set-level is $(1/2+1/T_{out})$, which is merely 0.51 for an AlexNet implementation on Xilinx Virtex-7 485T.

These limitations call for a more flexible buffer architecture where buffer banks can be allocated at a finer granularity and their status can be switched dynamically without physically moving/copying data.

3.2 Finding Buffer Space for Shortcuts

By definition, shortcut connection data are not consumed immediately in the next layer. To reuse them, we need to keep those data on-chip until multiple layers have been processed. During this period, some buffer space is needed to hold the shortcut connections. However, budgeting a separate buffer just for storing shortcut data may not be the most cost-effective solution, as the extra buffer resources could likely be utilized better for other purposes. Meanwhile, it is challenging to find space to hold the shortcut data with only existing buffers, as those buffers are being used continuously as input and output. Therefore, a clever solution to solve this dilemma is much needed.

3.3 Devising Unified Solution for Various Networks

Perhaps the most challenging part is to devise a unified architecture that can reuse data for various neural network models. As depicted in Fig. 1, neural networks come in different forms. Plain networks have no shortcut at all; whereas residual networks have shortcut connections that may cross a varying number of layers. A special procedure may be developed to find extra buffer space and reuse shortcut data for residual networks with 2-layer building blocks (i.e., depth of 2). Such procedure may *not* work for residual networks with building blocks of depth of 3, since the shortcut data needs to be kept on-chip for a longer time. To make things worse, given the trend for deeper neural networks and the pursuit for higher accuracy, future networks will likely have shortcut connections that span more layers, or even have building blocks of different depths in a single network. Therefore, it is important to develop a unified scheme that can handle different building blocks and thus have wide applicability. This is a main objective of the proposed work.

4. PROPOSED APPROACH

4.1 Basic Idea

In this work, we propose *Shortcut Mining (SCM)*, a novel yet effective scheme that "mines" the largely unexplored opportunity of reusing shortcut and feature map data to reduce off-chip traffic. To allow flexible and fine-grained allocation of on-chip buffers, we first introduce the abstraction of logical buffers. Those logical buffers are dynamically formed from a pool of individual physical buffer banks, allowing reusable data to be switched easily between input and output and be used at the per bank level. This decoupled physical-logical buffer structure provides the basis for the main shortcut mining operations.

The key part of the proposed scheme is a sequence of operations that allow shortcut data to be reused across *any* number of layers in the building block of DCNNs. To achieve this ambitious goal, we develop three procedures, namely Prolog, Kernel and Epilog, that work collectively. The Prolog procedure saves the shortcut data of a CNN layer in certain region of the on-chip buffer. The shortcut data can be restored by the Epilog procedure several layers later when the shortcut data

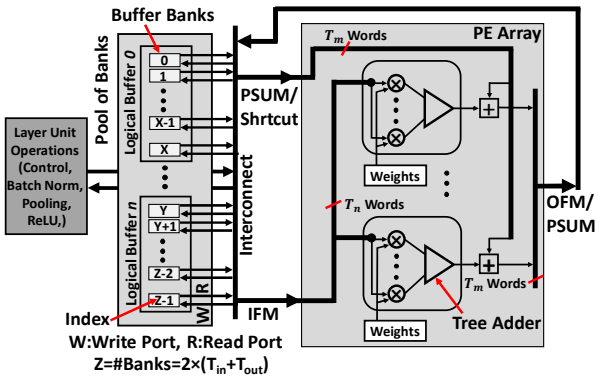

Fig. 6: Accelerator with the proposed Shortcut Mining.

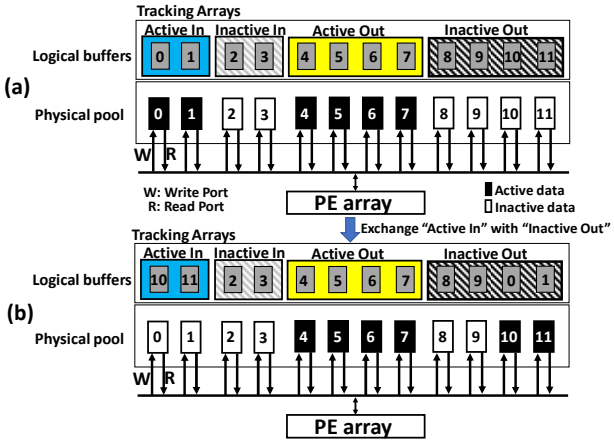

Fig. 7: Logical buffers are formed by using tracking arrays.

is supposed to be consumed. During this time, the Kernel procedure is executed multiple times, one for each layer in a building block. At each Kernel execution, the Kernel reuses the non-shortcut OFMs of the previous layer as the IFMs of the current layer and, more importantly, does not touch the region of the buffer where the shortcut data is saved by the Prolog. Thus, shortcut feature maps are preserved across layers while non-shortcut feature maps are reused immediately after each layer. It is important to note that all these operations are achieved without using additional buffer resources other than what are already provided. This requires clever designs and uses of the on-chip buffers, as the existing input and output buffers have their assigned functionalities.

Fig. 6 depicts the overall structure of the proposed accelerator. The structure is similar to existing ones by the looks of it, as the PE array still reads various inputs from the buffer, computes the output of a layer, and stores the results back to the buffer. The main differences, however, are the addition of logical buffers and, more importantly, the procedures to reuse feature maps as much as possible. The latter is included as part of the "Layer Unit Operation" box on the left side, so the procedures are not directly visible in Fig. 6 but are explained in detail in the following subsections.

4.2 Preliminary: Decoupled Physical-Logical Buffers

As the basis for enabling various buffer operations of the proposed three procedures, we first present the flexible buffer architecture. The main idea is to introduce a level of abstraction called *logical buffers* between the PE array and physical buffer banks. During runtime, the logical buffers are created and constructed dynamically from a pool of physical banks.

Fig. 7a demonstrates how this is done. Each physical bank in the pool is uniquely identified by an index. A logical buffer is formed by having a tracking array that holds the indices of the physical banks that currently belong to the logic buffer. Four logical buffers are formed in this way, namely active input buffer, inactive input buffer, active output buffer, and inactive output buffer. For example, the logical inactive input buffer in Fig. 7a is constructed from physical buffer banks 2 and 3. A logical buffer can be reconstructed dynamically with a new set of physical banks by updating the tracking array.

Fig. 7b illustrates how this flexibility can be used for reusing feature maps. Assume that at this clock cycle four OFMs of a CNN layer *i*-1 are stored in the logical inactive output buffer (assuming one OFM in one physical bank for simplicity). Instead of writing the OFMs to off-chip memory, two banks of the logical inactive output buffer can be exchanged with the two banks of the logical active input buffer, thereby reusing the OFMs of layer *i*-1 as the IFMs of layer *i*. The bank exchange is achieved simply by exchanging the indices in the tracking arrays, rather than copying data across banks physically.

While the idea of decoupled physical-logical buffer seems straightforward, it addresses the static bank assignment and the coarse-grained allocation issue nicely. First, each physical bank can be used dynamically as input or output, thus no longer being statically assigned. Second, physical resources are allocated at the per bank level, thus allowing individual banks to be utilized as soon as they become free. This greatly increases the utilization of on-chip buffers.

4.3 Kernel Procedure

The Kernel procedure has two goals: (1) reusing the available on-chip OFMs of the previous CNN layer (layer *i*-1) as the IFMs of layer *i* to produce the OFMs of layer *i*, as indicated by the red arrow in Fig. 8a. That is, instead of writing those reusable OFM tiles to off-chip and fetching them again, the Kernel directly reuses those OFM data; (2) keeping a certain region of the buffer space, referred to as the *invariant buffer space*, untouched throughout the operations of the Kernel. The invariant buffer space is used to preserve shortcut data (saved by the Prolog) that is not meant for layer *i*. Note that plain networks do not have shortcut connections, so applying the Kernel is sufficient for reusing feature maps for plain networks, without the need for the Prolog and Epilog.

For easier explanation, we divide the Kernel procedure into two phrases: *Reuse* and *Preload/Write Back*. The *Reuse* phase reuses the OFMs of layer *i*-1 that are still in the active output buffer at the end of layer *i*-1 processing. Due to the limited size of output buffer, the above OFMs may not include all the OFMs of layer *i*-1, so additional loading may be needed to fetch the rest OFMs from the off-chip memory. This is handled by the *Preload/Write Back* phase, which preloads the remaining OFMs of layer *i*-1 (i.e., the IFMs of layer *i*) to the inactive buffer, in parallel with computation. This phase also writes the computed OFMs back to off-chip when needed.

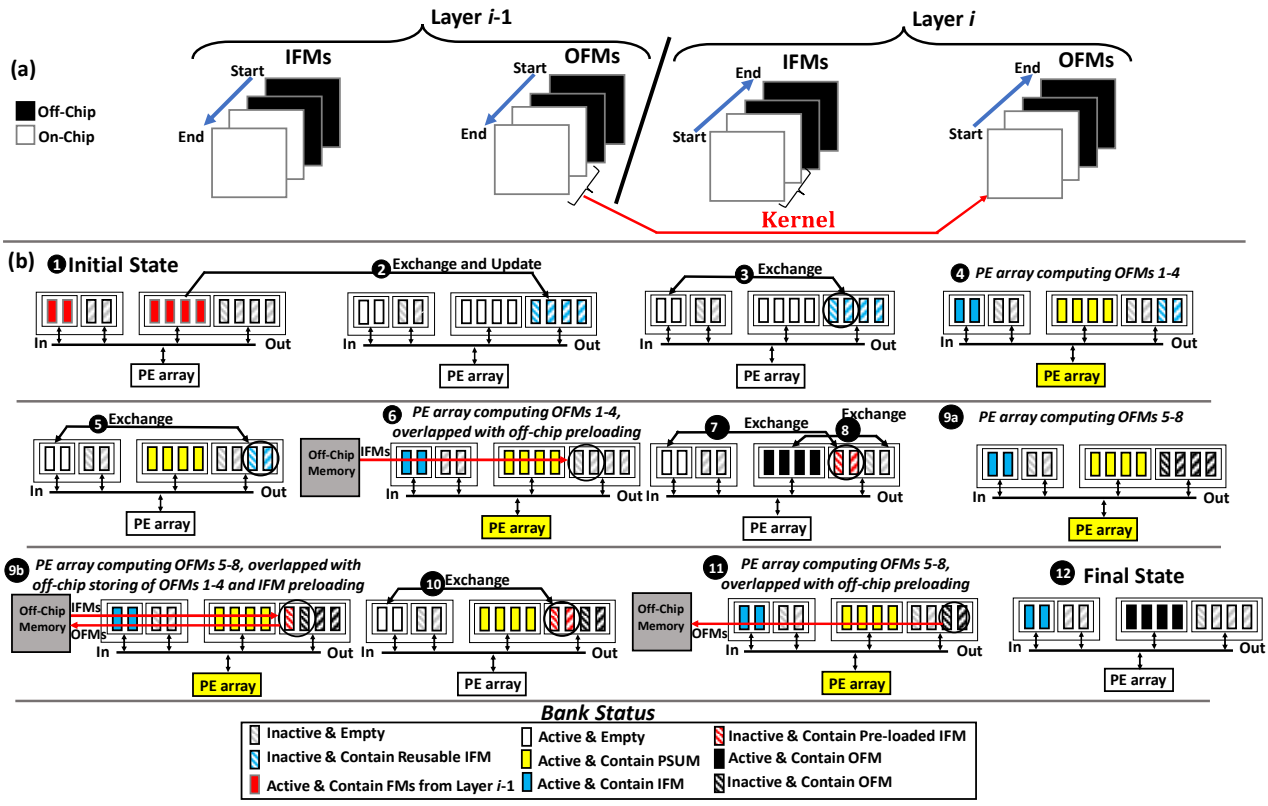

Fig. 8: Illustration of the Kernel procedure for reusing the OFMs of layer *i*-1 as the IFMs of layer *i*.

Fig. 8b demonstrates how the Kernel achieves its two goals during its execution for layer *i* with a walk-through example. In Fig. 8b, each small vertical rectangle is a buffer bank. In this example, layer *i*-1 produces 4 OFMs which become the 4 IFMs of layer *i*. Layer *i* then produces 8 OFMs. Also assume that the accelerator has T_{in} =2 and T_{out} =4 (i.e., 2 banks in the active input buffer and 4 banks in the active output buffer). If each buffer bank can store one feature map, this setting means that, in one iteration, the accelerator can read in 2 IFMs and computes the convolution results for 4 partial OFMs. Two iterations are needed, in order to load all the 4 IFMs that are needed to compute the four final OFMs (OFMs 1 to 4) of layer *i*. Similarly, another two iterations are needed to produce OFMs 5 to 8.

Reuse Phase: At the end of layer *i*-1, two IFMs and four OFMs of this layer remain on-chip as shown in Fig. 8b-Label❶ by the red rectangles. This is the initial state of the Kernel. In order to reuse the four OFMs, the active output banks are exchanged with the inactive output banks as shown in Label❷. Again, this is done through updating the tracking arrays, not moving data physically. Then, the inactive output banks (which now contains the four OFMs) are marked as the *reusable IFMs* (striped blue rectangles). Also, the current IFMs in the active input buffer are discarded because they are the old IFMs of layer *i*-1. This makes the active input banks empty, represented as white rectangles. The reuse process is started by exchanging the active input banks with two of the inactive output banks as shown in Label❸, so the active input banks contain the reusable IFMs that can be fed to the PE array for computation. Label❹ shows the status of the accelerator after performing the previous exchange. The active input banks contain two IFMs (blue rectangles). The PE array is highlighted in yellow to indicate that it has started the computation to produce part of OFMs 1-4 in the active output buffer (indicated by yellow rectangles). After finishing the computation, the Kernel can mark the active input banks as empty banks again. By another round of exchange in Label❺, the rest of the reused IFMs are consumed by the PE array in Label❻. After ❻, the final OFMs 1-4 are computed. Note that those OFMs could be all the OFMs that need to be computed for layer *i*, if the PE array and the output buffer are large enough. In this example, however, OFMs have to be computed in two batches, so additional preloading is needed for OFMs 5-8, as explained below.

Preload and Write Back Phase: To compute OFMs 5-8, two IFMs needed to be fetched from off-chip. This is done as a preloading operation in Label❻, while the PE array is computing OFMs 1-4. The preloaded IFMs are stored in the inactive output buffer (which has been emptied in ❸), so no additional buffer space is needed. Then, the two banks in the inactive output buffer that contain the preloaded IFMs are exchanged with the active input banks, as shown in Label❼. Note that, at this time, OFMs 1-4 have been computed and

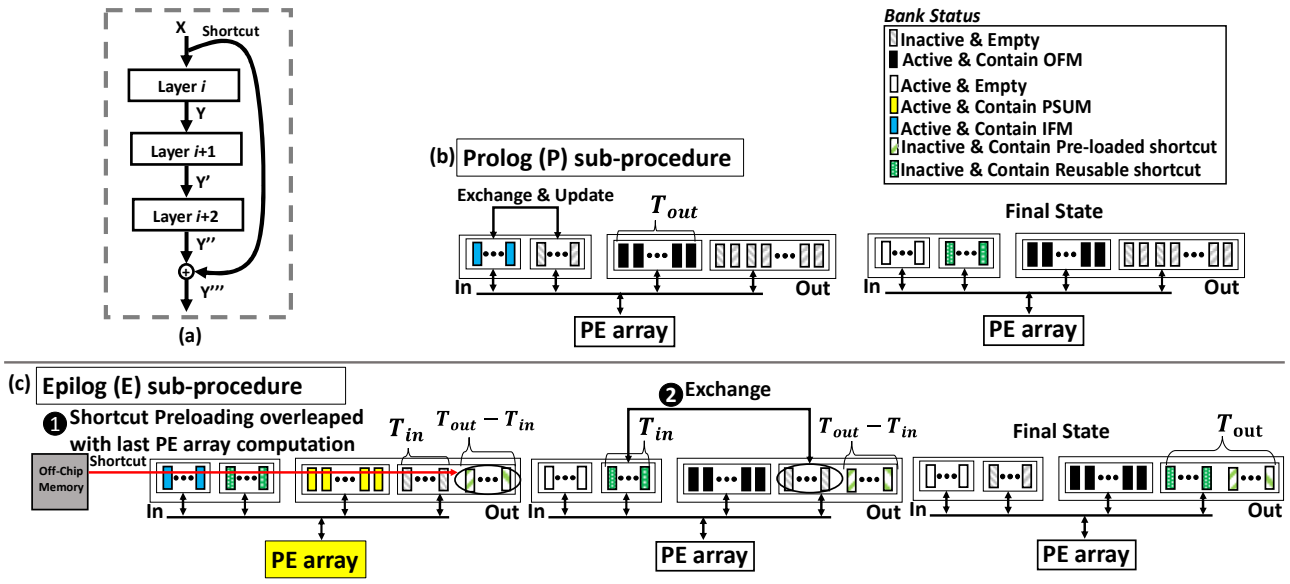

Fig. 9: Prolog and Epilog procedures.

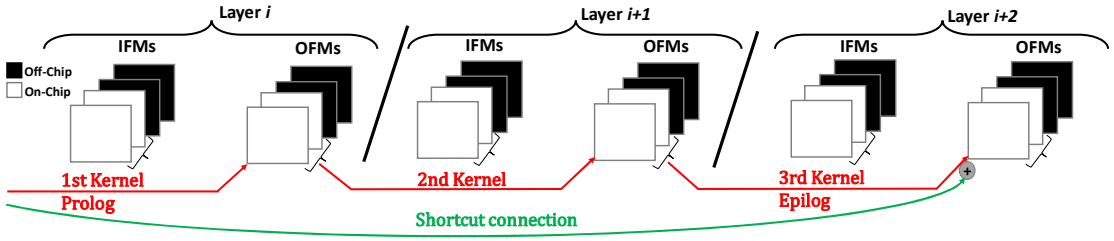

Fig. 10: Illustration of Shortcut Mining for a residual building block with three CNN layers.

stored in the active output banks (black rectangles). Therefore, following the exchange in ❼, all the active output banks are exchanged with the inactive output banks as shown in Label❽, so OFMs 1-4 can start to be written back to the off-chip memory[2]. After ❼ and ❽, Label❾a shows the instantaneous state that there is no empty bank on the output side, since the PE array is producing part of OFMs 5-8 to the active output buffer; whereas the inactive output banks contain OFMs 1-4. In order to preload the next two IFMs to complete the computation OFMs 5-8, three operations are taking place in Label❾b: (1) the PE array is computing part of OFMs 5-8, (2) OFMs 1-4 are being stored to off-chip, and (3) as soon as one of the OFMs 1-4 finishes storing, an IFM can be preloaded to the newly freed bank in the inactive output buffer. Here, the bank-level granularity of the proposed buffer architecture is exploited. Once two IFMs are preloaded in the inactive output banks, they are exchanged with the active input banks as shown in Label❿. In Label ⓫, the PE array is finishing up the computation of OFMs 5-8, and the remaining OFMs 1-4 in the inactive output banks are also finishing up the writing to off-chip memory. The final state in Label⓬ has the similar "format" as the initial state, containing only the IFMs and OFMs of layer *i* in the active input and output buffers, respectively. In this way, if this is a multi-layer building block, the next Kernel execution can be readily applied after ⓬. Note that OFMs 5-8 can be written down to off-chip memory through reusing steps (❸ to ❻) of next kernel if they are needed later in the layer processing after reuse.

Note that, throughout the steps of the above example, we deliberately avoid using the inactive input buffer. Even if the Kernel is executed multiple times, this buffer space would still be untouched. Therefore, without using additional buffer resources, we have obtained the invariant buffer space that is needed to preserve the cross-layer shortcut data.

4.4 Prolog and Epilog for Shortcut Reuse

As mentioned, plain networks can apply the Kernel procedure once to reuse feature maps. For networks with shortcut connections, Prolog and Epilog are needed, along with multiple executions of Kernel. Without loss of generality, we use a residual network with 3-layer building blocks as an example to explain the operations. As shown in Fig. 9a, reusing shortcut data means to save the IFMs of layer *i*, preserve the data while layer *i*, layer *i*+1 and layer *i*+2 are processed, and then restores the data and add to the OFMs of layer *i*+2.

We first describe the Prolog procedure, which is simply to

[2] In this example, the preloaded two IFMs from Label ❼ are used to compute OFMs 5-8. However, if more IFMs are needed to compute OFMs 1-4, we can simply repeat Labels ❻ and ❼ until OFMs 1-4 are computed.

exchange the active input banks that contains the IFMs with the inactive input banks, and mark it as the reusable shortcut data, as shown in Fig. 9b. Although the procedure is simple, care needs to be taken on when the Prolog is applied. Recall that in Fig. 8 Label❶, the IFMs in the active input buffer are actually the IFMs of layer *i*-1. Therefore, at the beginning of the first Kernel in Fig. 10, the IFMs in the input buffer are not the intended shortcut in Fig. 9a, and the Prolog should not be applied at that step. Instead, Fig. 8 shows that the actual IFMs of layer *i* are available at Label❹ at the earliest time, and will no longer be used after Label❺. Based on this, the Prolog is executed at the end of Label❹, to exchange and then save the shortcut data in the inactive input buffer (i.e., the invariant buffer).

Fig. 9c shows the Epilog procedure. A direct objective is to restore the saved shortcut data, which is achieved by the exchange in Label❷. While we can define the Epilog just like that, it is possible that the output buffer size T_{out} is larger than the input buffer size T_{in}, so additional empty banks are available in the inactive output buffer. To make full use of the buffer space, we add a preloading step to the Epilog, as shown in Label ❶, to preload ($T_{out} - T_{in}$) shortcut data from off-chip. This step is done prior to the above exchange step, so as to overlap the preloading with PE array computation. The final state of the Epilog has T_{out} shortcut data ready to be used.

Similar to the Prolog which is executed in parallel with the first Kernel, the Epilog is executed in parallel with the third Kernel, as shown in Fig. 10. Specially, the Epilog can be applied at Labels ⓫ and ⓬ in Fig. 8 (imagine that the figure now represents the Kernel for layer *i*+2) to preload and assemble T_{out} shortcut data in the inactive output buffer. However, additional time is needed to add the shortcut in the inactive output buffer with the data in the active output buffer (OFMs of layer *i*+2), because the summation is what is actually needed to the next building block, as shown in Fig. 9a. Alternatively, to avoid the additional time, the Epilog can be applied in parallel with Label❶ of the third Kernel, so the T_{out} shortcut data is preloaded into the inactive output buffer at Label❶ (the preloading itself is overlapped with the PE array computation of the previous Kernel). This shortcut data is exchanged to the active output buffer in Label❷ and accumulated with the PSUM of OFMs 1-4 in Label❹ automatically, thanks to existing steps in the third Kernel. In either way of applying Epilog, the shortcut data is correctly added with the normal OFMs of layer *i*+2, thereby generating the input data that is needed by the next building block.

To summarize, in the three-layer building block example, the Prolog is executed in parallel with the first Kernel to save the shortcut data. This is followed by the second Kernel. Then the Epilog is executed in parallel with the third Kernel to restore and reuse the shortcut data. During this time, the three Kernel executions reuse the OFMs of layer *i*-1, layer *i* and layer *i*+1 as the IFMs of layer *i*, layer *i*+1 and layer *i*+2, respectively. In general, assuming K_i is the Kernel for layer *i*, for a residual network that is constructed with building blocks of depth of *D*, the sequence of operations for reusing both shortcut and non-shortcut feature maps is:

$$(P\&K_1) + \sum_{2}^{D-1} K_i + (K_D\&E)$$

Following this sequence, the proposed Shortcut Mining can work for building blocks with any number of layers.

5. ACCELERATOR IMPLEMENTATION

We demonstrate the effectiveness of the proposed Shortcut Mining (SCM) scheme by prototyping on a FPGA. Besides common benefits of FPGA-prototyping, this also enables us to track all the transactions between the accelerator and the off-chip memory (DRAM) byte-by-byte at the cycle level.

To determine the optimal parameters for the configurations of the accelerators for a baseline (referred to as BL hereafter) and the proposed SCM, we have developed an optimization program tool, following previous methodologies in [29][42] [26]. The tool takes as input a file containing the description of each CNN layer, the target FPGA resource budget profile, target frequency, data type format and the maximum memory bandwidth. The loop orders in [42] are used for both BL and SCM. The tool then calculates the estimated execution cycles, memory bandwidth requirement, as well as the usage for DSP slices, BRAMs and URAMs for each set of the parameters. The optimal set of parameters are then selected to implement BL and SCM on the FPGA.

One of the important parameters is bank size. Large banks result in more data reuse in earlier layers of a network that have large grid sizes. On the other hand, later layers have smaller grid sizes due to down-sampling, thus smaller banks are more favorable for reusing data and increasing buffer utilization. Additionally, smaller bank size has better flexibility but longer access latency. To determine the bank size, the tool generates all possible combinations of T_r and T_c (as defined in Section 2.1). For each pair, the tool estimates the off-chip traffic, bank access delay, and memory utilization based on [29][42][26] and the physical characteristics of the tracking arrays. The pair that has the lowest traffic and computation cycles, averaged over all the layers of a given network, is selected. The final bank size is 4,067 for SqueezeNet, 1581 for ResNet-34 and 1681 ResNet-152, all in the unit of words. It is worth mentioning that, bank access latency is small (e.g., 3ns) relative to the latency of PE array (i.e., MAC units, 10ns), so the impact of using smaller and more flexible banks has very limited negative impact on the overall performance.

To increase the generality of our implementation, we have developed a parameterized approach where the values of the parameters can be set to the optimal ones obtained above, or other values if needed. We have carefully designed and implemented the control logic that is needed to support the various operations in Prolog, Kernel and Epilog. The controller supports an arbitrary number of layers in a building block. The accelerators are implemented in the high-level-synthesis (HLS). Xilinx Vivado HLS 2017.4 is used to compile the HLS code with the selected parameters to synthesizable Verilog. AXI4 ports are used to access buffer banks, and the AXI interconnect is used to access the memory interface controller. From the Xilinx tools, we are able to obtain the FPGA resource utilization, feature map traffic between the accelerator and memory for each inference operation, as well as other results shown in the evaluation section. While we evaluate accelerators with SCM on FPGA in this paper, a similar implementation flow can be used for ASIC. For ASIC that communicates with off-chip memory (DRAM), the ASIC has

Table 2: Baseline (BL) vs. Shortcut Mining (SCM).

	SqueezeNet		ResNet-34		ResNet-152	
Approach	BL	SCM	BL	SCM	BL	SCM
B/w (GB/s)	10.4	10.4	20.5	20.5	23	23
Off-chip FMs traffic (MB)	30	14	56.23	23.58	240.3	136.9
URAMs	0	0	256	0	512	0
BRAM 18K	2,737	2,354	3,078	3,198	3,145	3,210

its own on-chip memory where asynchronous FIFOs are typically added between the accelerator's core and DRAM. The main design of SCM would be the same while a Verilog version of SCM can be developed, synthesized and then placed & routed for ASIC implementation.

6. EVALUATION

6.1 Off-chip Feature Map Traffic

To evaluate the effectiveness of the proposed SCM in reducing off-chip feature map traffic, we compare BL and SCM for DCNNs constructed with three depths of building blocks: SqueezeNet as a representative example for plain networks, ResNet-34 for networks with two-layer building blocks, and ResNet-152 for networks with a three-layer building blocks.

Table 2 summarizes the comparison on the provisioned off-chip memory bandwidth, off-chip feature map traffic, and FPGA on-chip memory resource utilization for BL and SCM. Each BRAM is 18Kb and each URAM is 288Kb, and the total number of BRAMs and URAMs that are utilized for each accelerator are shown in the last two rows of Table 2. The off-chip feature map traffic is measured as the amount of data that is transferred between the accelerator and the DRAM for each inference of the CNN processing. The main difference between BL and SCM is the buffer architecture and how feature map data are managed. To have a competitive baseline, the BL includes all the optimization techniques mentioned in Section 2.1 that are common to DCNN accelerators. Also, we implement operations in BL such as pooling and activation function in a pipeline manner to overlap with PE array computation. Furthermore, shortcut connections are treated as partial sums and their buffer in BL is implemented as a ping-pong buffer to overlap the preloading with layer computation. The buffers in SCM are logical buffers instead, and the feature map operations follow the procedures described in Section 4.

As shown in Table 2, an inference operation in SqueezeNet has 30MB of feature map data that needs to be transferred between the accelerator and the DRAM for the baseline. In contrast, for the proposed SCM, the off-chip feature map data is reduced to 14MB, which is a 53.3% reduction from the BL. Moreover, SCM uses smaller 14% on-chip buffers as part of the inactive output buffer is saved by using some of the idle inactive input buffer in SCM.

For ResNet-34 and ResNet-152, similar trends can be observed with one difference. Compared with SqueezeNet, the ResNets have considerable numbers of shortcut connections, so the BL accelerator needs a dedicated buffer to preload the shortcut connections. Therefore, BL has four buffers including the IFM buffer, weight buffer, OFM buffer, and shortcut buffer. Based on the characteristics of the buffer components in FPGA, it is more efficient to implement the first three in

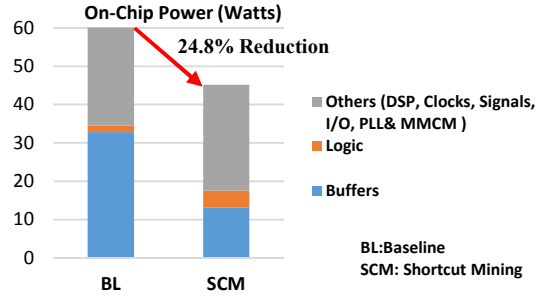

Fig. 11: Power breakdown of ResNet-152.

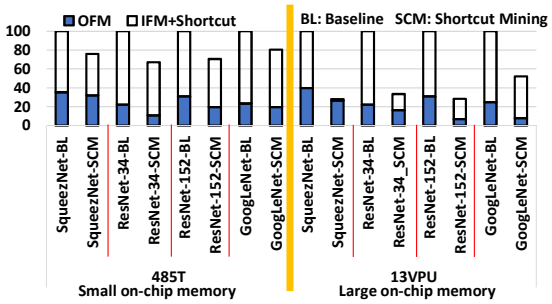

Fig. 12: Off-chip traffic comparison for scalability.

BRAMs and the last one in URAMs. The SCM does not need the shortcut buffer owing to the shortcut reuse procedure, as well as the reuse of the inactive output buffer for preloading the remaining shortcuts, as explained in section 4. Overall, compared with BL, the proposed SCM reduces the off-chip feature map traffic for an inference operation in ResNet-34 from 56.23MB to 23.58MB, which is a large 58% reduction. Meanwhile, SCM does not incur the 256 URAMs overhead for the shortcut buffer in BL. Likewise, for ResNet-152, the SCM reduces the off-chip feature map traffic by 43% (from 240.3MB to 136.93MB), without using 512 URAMs compared with the BL. There is a slightly increase in the number of BRAMs for ResNets in SCM (2-3%), which is due to the internal fragmentation when mapping buffers to BRAMs.

6.2 On-chip Power

Fig. 11 compares the on-chip power consumption of the BL and SCM accelerators for the case of ResNet-152. We present the ResNet-152 results here because SCM has more complex control flow (and consequently more logic overhead) for this network compared with the other two networks. This gives slightly more advantage to the BL. Also, ResNet-152 has a deeper structure which might reflect future networks better. All the power consumption in this figure is from Vivado by using the Xilinx Virtex UltraScale+ FPGA VCU118 evaluation platform. In total, the BL consumes 60.1 Watts and the SCM design consumes 45.1 Watts.

The SCM design has a more complex interconnect between the buffer banks and the PE array's I/O and also more complex control unit. This overhead leads to larger logic power consumption as shown in Fig. 11. However, SCM requires smaller on-chip buffers due to its efficient utilization of various buffers. Moreover, a considerable number of buffer read and write operations are eliminated in SCM due to shortcut and non-shortcut feature map reuse. These improvements

Table 3: Performance comparison with state-of-the-art single-layer CNN accelerator on equivalent FPGAs.

	Design in [21]	SCM
FPGA	Arria-10 GX 1150	Virtex-7 485T
Frequency (MHz)	150	150
Network	ResNet-152	ResNet-152
#Operations (GOP)	22.62	22.63
DSP Utilization	100%	100%
BRAM Utilization	93%	99%
Logic Utilization	33%	86%
Data Format	16-bit	16-bit
Latency (ms)	71.71	35.24
Throughput (GOPS)	315.48	608.28
Power	Not Reported	21.64
Efficiency (GOP/J)	Not Reported	28.1
Off-chip FMs (MB)	Not Reported	62.93

translate into 24.9% reduction in the total on-chip power for ResNet-152.

6.3 Performance Comparison with State-of-the-Art

Besides the baseline, the proposed SCM is also compared with two state-of-the-art designs. The first one is a residual CNN accelerator that is proposed recently to optimize operations within a single layer [20][21]. To our knowledge, that work reports the best results so far among the existing schemes that target single-layer dataflow. The design reduces off-chip accesses and on-chip data movement, and increases the PE array utilization during each CNN layer processing. However, no shortcut connection data is exploited for reusing.

Table 3 lists the comparison. The residual CNN accelerator is implemented on Altera Arria-10 GX 1150 FPGA. For fair comparison, we evaluate SCM on an equivalent Xilinx FPGA in terms of the available number of DSPs and on-chip buffer, using the same data type and frequency (our timing analysis shows that the changed interconnect and controller in SCM has a negligible impact on frequency since the critical path is mainly determined by the data type and PEs). Compared with the 315.48 GOPS throughput in the residual CNN accelerator, SCM achieves 608.28 GOPS, which is a 1.93X improvement in performance. The main reason for the large improvement is that the PE array in SCM has more accesses that are serviced by on-chip buffers, during each CNN layer processing as well as during the transition to the next layer due to feature map reuse.

We also compare with fused-CNN [2], a recent work on optimizing cross-layer dataflow. Fused-CNN can reduce the off-chip feature map traffic for five fused layers of VGGNet-E by 95%, compared with a single-layer CNN dataflow accelerator[2]. Meanwhile, fused-CNN is also slightly slower as it needs 6.5% more cycles for each inference operation of the fused layers[2]. However, as discussed previously, only a limited number of layers can be fused on a given FPGA. In our experiments, we found that 32 CNN layers of the ResNet-152 can be fused on Xilinx Virtex UltraScale+ VU9P with 32-bit floating point format. As a result, fused-CNN reduces the off-chip the off-chip feature map traffic by 26%. In comparison, the proposed SCM can reduce this traffic by 43% for ResNet-152 on the same platform and precision.

6.4 Scalability

Fundamentally, the effectiveness of SCM in reducing off-chip feature map traffic depends on two key factors. First, the amount of feature map data that remains on-chip at the end of each layer processing. Second, the amount of OFMs of the next layer that can be computed by using the above on-chip feature maps. Both factors greatly relate to the available on-chip buffer in the accelerator. To investigate the impact of on-chip buffer on the feature map reduction capability of SCM, we consider two extreme cases. In the first case, we examine an FPGA with a small on-chip buffer (Xilinx Virtex 7 485T with 4.5MB on-chip buffer size [39]), and in the second case an FPGA with a large on-chip buffer (Virtex UltraScale+ VU13P with 56MB [40]). Due to the lack of physical access to advance FPGAs, we project the off-chip feature map reduction using the estimation tool that we developed following models in [29][42][26], as discussed in Section 5. The results are presented in Fig. 12, all based on 32-bit floating point precision. On the FPGA with a small on-chip buffer, for SqueezeNet, ResNet-34, ResNet-152 and GoogLeNet, the off-chip feature map traffic is reduced by 23.9%, 32.8%, 29.5% and 19.6%, respectively. On the FPGA with a large on-chip buffer, the off-chip feature map traffic is reduced by 71.8%, 66.6%, 71.7%, and 47.8% for SqueezeNet, ResNet-34, ResNet-152, and GoogLeNet, respectively. It is expected that the VU13P FPGA, which has a larger on-chip buffer, has more reduction in feature maps due to more reusable on-chip data. However, it is worth noting that, even for the 485T FPGA with a small buffer, the reduction is still quite substantial.

6.5 Compact Data Type

There is an increasing trend to use compact data representations in DCNNs to improve computation efficiency [22]. To investigate the impact of this on the effectiveness of SCM, we conduct repeat the experiment in the previous subsection but for 16-bit fixed-point precision. For SqueezeNet, evaluation results show that SCM reduces the off-chip feature map traffic by 67% on the 485T. This is much larger reduction compared with the 23.9% in the 32-bit floating point case. Similarly, with 16-bit fixed-point, SCM achieves 57.3% and 47.6% off-chip feature map reduction for ResNet-34 and ResNet-152, respectively, both of which exceed the 32.8% and 29.5% reduction under 32-bit floating point. These results are expected as with compact data representations, more feature maps and shortcut connection data can be fit in the same on-chip buffer. Therefore, the proposed SCM will work well for future lower precision representations.

6.6 Layer-by-layer Analysis

We conduct further evaluation to analyze the effectiveness of the proposed approach in off-chip energy consumption and latency reduction as a function of CNN layers, available on-chip memory, and data type. Fig. 13 shows the normalized off-chip energy consumption and latency for each CNN layer of VGGNet-E in two cases: first, a small on-chip memory with 32-bit floating point data type, and second, a large on-chip memory with 16-bit fixed-point data type. As it can be seen from Fig. 13(a), the large on-chip memory with compact data type has more energy reduction as more feature maps

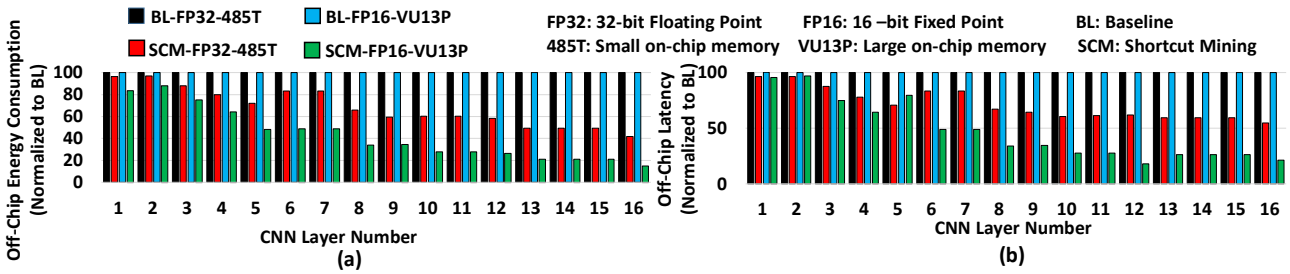

Fig. 13: VGGNet-E off-chip energy consumption (a) and latency (b).

remain on-chip at the end of each CNN layer for reuse. However, on average, the off-chip energy consumption is reduced substantially in both cases, by 25.2% and 50% for the first and second case, respectively. A similar trend can be observed for the off-chip latency as shown in Fig. 13(b). Overall, off-chip latency is reduced by 23% and 45.7% for the first and second case, respectively. For residual networks (not shown), we have observed more aggressive savings in off-chip energy consumption and latency due to shortcut reuse in addition to feature maps. For example on the FPGA with a large memory and 16-bit fixed point (second case), the off-chip energy consumption and latency for ResNet-152 are reduced by 81% and 79%, respectively.

7. RELATED WORK

There is an increasing number of works focusing on the design of hardware accelerators for DCNNs, and it has been an active research topic in the computer architecture community in recent years. Below, we discuss the most relevant works to ours in terms of single-layer and cross-layer CNN dataflow. This is followed by a brief discussion on other related work.

In the past few years, many schemes based on single-layer dataflow are proposed (e.g., [23][42][32][4][5][20][19][35][33][15]). These works focus on the efficient processing of a single CNN layer mostly through the optimized loop operations and the computation order to reduce off-chip and on-chip data movement and increase PE array utilization. Feature map reuse, particularly shortcut data reuse, has not been explored well. Feature maps can be compressed to reduce off-chip traffic. Its effectiveness is limited as it depends on the number of zeros and their distribution. Even under ReLU, which is a benign activation function for compression, only 1.2X to 1.9X reduction is achieved [4]. Compression is orthogonal to data reuse. When compression is used, the proposed SCM would achieve similar reduction percentage for off-chip traffic that is caused by feature maps. However, the overall reduction percentage may be lower as the relative off-chip traffic from feature maps (vs. weights) is smaller due to compression.

On the cross-layer dataflow side, fused-layer CNN [2] cascades multiple CNN layers in a pyramid structure and reuses intermediate OFM tiles. Off-chip accesses are greatly reduced, although on-chip data copying between the output buffer and input buffer is still needed. This cross-layer dataflow is also supported by other recent works [17][28][37]. A main limitation of fusing layers is the need for large on-chip buffers to hold all the intermediate data between layers. This makes it less effective to process deeper networks. In comparison, the proposed Shortcut Mining in this paper can reuse shortcut data and feature maps very effectively for deep networks. The proposed scheme also works well for various types of networks, constructed with building blocks of different depths.

In addition to the above, one work is presented to minimize the bandwidth requirement for loading data into local buffers during each CNN layer processing [25], but not during the transition between layers. CirCNN is introduced to reduce the cost of weights in deep neural networks [6], which further increases the percentage of feature map data and makes our proposed SCM more important. Another accelerator [1] is designed based on the observation that majority of the computations performed by DNNs contain multiplications where one of the operands is zero. Thus, these computations can be eliminated for less energy consumption. Also, an accelerator based on a "bit-serial" PE array is designed to provide an energy efficient DNNs processing at the cost of increased area overhead [16]. The above two papers have very different approaches to our work. A very recent work [43] observes that, even by eliminating computation for zero inputs in neurons, many neurons still cannot be passed to the next layer. Thus, a two-stage DNN accelerator is proposed to first predict and then skip the computation for those ineffectual neurons. The input and filter sharing proposed in that work are among PEs in a single layer, which is different from the opportunity explored in this work that moves and reuses data across layers. Other earlier works such as [3][24] focus on 2D convolution engine, including the order of data fetching and data caching. These schemes are not for 3D convolution structures in deep networks.

8. CONCLUSION

Current and future deep learning networks employ deep structures for a higher accuracy in different machine learning tasks. However, off-chip memory accesses become a major issue in energy-efficient and high-performance processing. In this paper, we analyze the composition of off-chip accesses and different feature maps reuse opportunities in the modern DCNNs. We propose a novel approach to exploit the reuse of the shortcut connections and feature maps in different networks in order to reduce the off-chip traffic. Experiment results demonstrate that the proposed approach offers significant advantage in reducing up to 71.8% off-chip feature map traffic and increasing performance by 1.93X on modern accelerator platforms.

Acknowledgments

This research was supported, in part, by the National Science Foundation grants #1566637, #1619456 and #1750047.

REFERENCES

[1] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, A. Moshovos. 2016. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. *In Proceedings of the 44th International Symposium on Computer Architecture (ISCA '16).*

[2] M. Alwani, H. Chen, M. Ferdman, and P. Milder. 2016. Fused-layer CNN accelerators. In Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '16).

[3] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi. 2010. A Dynamically Configurable Coprocessor for Convolutional Neural Networks. *In Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)*.

[4] Y. Chen, T. Krishna, J. Emer, and V. Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits 52, 1.*

[5] Y. Chen, J. Emer and V. Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. *In Proceedings of 43rd Annual International Symposium on Computer Architecture (ISCA'16).*

[6] C. Ding, S. Liao, Y. Wang, Z. Li1, N. Liu1, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan, X. Ma, Y. Zhang, J. Tang, Q. Qiu, X. Lin, and B. Yuan. 2017. CirCNN: *Accelerating and Compressing Deep Neural Networks Using Block-Circulant Weight Matrices. In the Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture. (MICRO '17).*

[7] S. Han, J. Pool, J. Tran, and W. J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. *In Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)*. 2015.

[8] D. Harris and N. Weste, CMOS VLSI Design: A Circuits and Systems Perspective. Pearson/Addison-Wesley, 2005.

[9] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. *In the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[10] M. Horowitz. Energy table for 45nm process, Stanford VLSI wiki

[11] https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/ug-s10-memory.pdf

[12] https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/

[13] G. Huang, Z. Liu, L. van der Maaten, K. Q. Weinberger. 2017. Densely Connected Convolutional Networks. . *In the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[14] F. N. Iandola, M.W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. CoRR abs/1602.07360.

[15] N. P. Jouppi, , C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, *et al.* 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. *In Proceedings of the 44th International Symposium on Computer Architecture (ISCA '17).*

[16] P. Judd, J. Albericio, T. Hetheringtony, T. M. Aamodty, A. Moshovos. 2016. Stripes: Bit-Serial Deep Neural Network Computing. In Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '16).

[17] H. Kwon, A. Samajdar, T. Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. *In Proceeding of the 23rd ACM International Conference on Architectural Support for Programming Languages and Operating Systems. (ASPLOS'18)*

[18] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Peter Graf. 2016. Pruning filters for efficient convnets. CoRR abs/ 1608.08710.

[19] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li. 2017. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks. *In Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'17).*

[20] Y. Ma, Y. Cao, S. Vrudhula, J. Seo. 2017. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. *In Proceedings of the 25rd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17).*

[21] Y. Ma, M. Kim, Yu Cao, S. Vrudhula, J. Seo. 2017. End-to-End Scalable FPGA Accelerator for Deep Residual Networks. 2017. *IEEE International Symposium on Circuits and Systems (ISCAS'17).*

[22] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Gee Hock Ong, Y. Tat Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh. 2017. Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?. *In Proceedings of the 25th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17).*

[23] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. *In Proceedings of the 24th ACM/SIGDA International Symposium on Field- Programmable Gate Arrays (FPGA '16).*

[24] M. Peemen, A. AA Setio, B. Mesman, and H. Corporaal. 2013. Memory-centric accelerator design for Convolutional Neural Networks. *In Proceedings of the 31st IEEE International Conference on Computer Design (ICCD '13).*

[25] M. Peemen, B. Mesman, and H. Corporaal. 2015. Inter-Tile Reuse Optimization Applied to Bandwidth Constrained Embedded Accelerators. *In Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE '15).*

[26] A. Rahman, S. Oh, J. Lee, and K. Choi. 2017. Design Space Exploration of FPGA Accelerators for Convolutional Neural Networks. *In Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*.

[27] J. Ross, N. Jouppi, A. Phelps, C. Young, T. Norrie, G. Thorson, D. Luu, 2015. Neural Network Processor, Patent Application No. 62/164,931.

[28] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh. 2016. From High-Level Deep Neural Models to FPGAs. *In the Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture. (MICRO '16).*

[29] Y. Shen, M. Ferdman, and P. Milder. 2017. Maximizing CNN Accelerator Efficiency Through Resource Partitioning. *In Proceedings of the 44th International Symposium on Computer Architecture (ISCA '17)*.

[30] Y. Shen, M. Ferdman, and P. Milder. 2017. Escher: A CNN Accelerator with Flexible Buffering to Minimize Off-Chip Transfer. *In Proceedings of the 25th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM '17)*.

[31] J. E. Smith, 1982, April. Decoupled access/execute computer architectures. *Proc. Int'l Symp. on Computer Architecture*.

[32] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li. 2016. C-brain: A Deep Learning Accelerator That Tames the Diversity of CNNs Through Adaptive Data-level Parallelization. *In Proceedings of the 53rd Annual Design Automation Conference (DAC '16)*.

[33] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J. Seo, Y. Cao. 2016. Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks. *In Proceedings of the 24rd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16)*.

[34] C. Szegedy, S. Ioffe, and V. Vanhoucke. 2016. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. CoRR abs/1602.07261.

[35] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei. 2017. Deep Convolutional Neural Network Architecture with Reconfigurable Computation Patterns. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 8.*

[36] UltraScale Architecture and Product Data Sheet: Overview, DS890 (v2.11) Februarys 15, 2017.

[37] Q. Xiao, Y. Liang, L. Lu, S. Yan and Y. Tai. 2017. Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs. *In Proceedings of the 53rd Annual Design Automation Conference (DAC '17).*

[38] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. 2017. Aggregated Residual Transformations for Deep Neural Networks. CoRR abs/1611.05431.

[39] Xilinx. 2017. 7 Series FPGAs Data Sheet: Overview.

[40] Xilinx. 2017. UltraScale FPGA Product Tables and Product Selection Guide.

[41] Xilinx. 2017. UltraScale Architecture Memory Resources.

[42] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. *In Proceedings of the 23rd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15)*.

[43] M. Song, J. Zhao, Y. Hu, J. Zhang, and T. Li. 2018. Prediction Based Execution on Deep Neural Networks. *In Proceedings of the 45th International Symposium on Computer Architecture (ISCA '18).*

[44] S. Zagoruyko and N. Komodakis. Wide residual networks*. In BMVC, 2016.*