Flexible On-chip Memory Architecture for DCNN Accelerators

Arash Azizimazreah Lizhong Chen
Oregon State University, OR, 97331, USA
{azizimaa, chenliz}@oregonstate.edu

ABSTRACT

Recent studies show that as the depth of Convolution Neural Networks (CNNs) increases for higher performance in different machine learning tasks, a major bottleneck in the improvement of deep CNNs (DCNNs) processing is the traffic between the accelerator and off-chip memory. However, current state-of-the-art accelerators cannot effectively reduce off-chip feature map traffic due to the limited capability in reusing output feature maps from the previous CNN layer to the next. In this paper, we propose a flexible on-chip memory architecture with a memory bank management scheme to reduce feature map data movement during the computational transition between CNN layers. Moreover, the proposed scheme can also be used as a complementing approach to existing CNN dataflows to further improve the energy efficiency of DCNN accelerators. Experiment results show that the proposed scheme can reduce the off-chip feature map traffic of ResNet-152 and VGGNet-E by 40% and 50% in 32-bit floating point and 16-bit fixed-point arithmetic, respectively, on Xilinx Virtex FPGAs.

1 INTRODUCTION

To achieve higher accuracy, current deep convolution neural networks (DCNNs) employ many CNN layers, with some exceeding a thousand layers [1]. However, DCNNs become very computation and memory intensive as they evolve to deeper structures. This rapidly increased demand on computation and memory resources has caused conventional CPUs to be a bottleneck in processing DCNNs during both training and inference [2]. Alternatively, while GPUs have also been used to process DCNNs by exploring their massive parallelism, the large power consumption of GPUs has limited their extensive uses in data-center applications and embedded systems where power is critical. Meanwhile, FPGAs as reconfigurable computing platforms are able to provide an energy-efficient and massively parallel processing capability at the same time, thus surging as an attractive candidate for energy-efficient DCNNs acceleration in recent years.

The state-of-the-art FPGA-based accelerators process DCNNs layer by layer [2-7], but they all suffer from a problem which we refer to as the *static bank assignment* problem. That is, the assignment of banks to the input feature map (IFM) buffer and output feature map (OFM) buffer are statically determined at the design time and cannot be changed after implementation. This static bank assignment forces accelerators to write back on-chip OFMs during the transition from processing one layer to the next. This policy makes sense previously where the size of OFM buffer was small and there were negligible performance and energy benefits in the reusing the data in OFM buffer due to the small size of on-chip memory on FPGA chips. However, modern FPGAs typically contain a considerable amount of on-chip memory, e.g., the Xilinx Virtex UlteraScale+ FPGAs have up to 56.81MB on-chip memory organized as BRAMs and URAMs [8]. Moreover, as DCNNs evolve to deeper structures, the contribution of feature map data movement across layers shows an increasing trend [9]. Consequently, the static bank assignment policy leads to a poor utilization of on-chip memory and squanders the opportunity in reusing OFMs as the IFMs for the next layer.

Nevertheless, reusing OFMs for the next layer processing faces several major implementation challenges including how to keep track of individual memory banks as each bank is used in both input and output buffer to reuse OFMs. Therefore, the current architecture and management of the on-chip memory sub-system in FPGA-based DCNN accelerators need to be redesigned for a more efficient utilization of the large on-chip memory in modern FPGAs.

To address the limitation of static bank assignment and enable efficient OFM reuse, in this paper, we propose a flexible on-chip memory architecture with a memory bank management scheme to maximize OFM reuse during the computational transition between two consecutive CNN layers. The proposed flexible memory sub-system can be used in existing CNN dataflows to improve their ability in reducing off-chip traffic. Experiment results show that the proposed architecture is able to reduce the feature map traffic by 40% for a deep network ResNet-152 in 32-bit floating point on Xilinx VU13P FPGAs, and reduce the traffic by 50% in a more compact 16-bit fixed-point data type on Xilinx VU9P FPGA.

The rest of this paper is organized as follows. Section 2 provides more background on the architecture of state-of-the-art FPGA-based DCNN accelerators. Section 3 discusses the motivation for this work. In Section 4, we describe the proposed memory sub-system architecture, its management scheme, and the implementation in detail. Evaluation results and analysis are presented in Section 5. Finally, related work is summarized in Section 6, and Section 7 concludes this paper.

2 MODERN DCNN ACCELERATORS

The original core of the state-of-the-art DCNN accelerators is a processing fabric based on the spatial architecture [10]. A DCNN accelerator based on the spatial architecture can be broken down into two main components: an array of processing elements (PEs) and on-chip buffers [10]. The PE array is responsible for carrying out the computation of CNN layers. The on-chip buffers cache the required data during the acceleration. The PE array and the on-chip buffers are connected via a simple network-on-chip.

The granularity of PEs can vary from a simple multiplier and adder (fine-grained) to a vector-dot-product (coarse-grained). The state-of-the-art FPGA-based DCNN accelerators such as [2, 4, 6, 9] use a convolution layer processor (CLP) to evaluate the CNN layers sequentially [2]. Fig. 1 shows the datapath of a convolution layer processor. The PE array fetches T_n IFM pixels from the input buffer, $T_n \times T_m$ weights from the weight buffer and T_m OFM partial sums (psum) from the output buffer, and then calculates T_m OFM psums or pixels simultaneously. We define the T_n and T_m in more detail later in subsection 2.2.

2.1 CNN Dataflow

CNN dataflows can generally be classified into two categories. The first category of CNN dataflows controls the accelerator to evaluate a CNN in a layer-by-layer fashion based on the network's natural structure. This category of dataflows (single-layer dataflows) assumes that at the beginning of each CNN layer processing, the input feature maps are stored in the off-chip memory. Then, after finishing processing the CNN layer, the computed output feature maps are written back into the off-chip memory. This assumption causes a great number of feature maps being written/read back and forth between the off-chip memory and the accelerator during the CNN processing. Examples of dataflows in the first category are reported in [6, 10, 11, 12].

The second category of CNN dataflows focuses on the data movement between CNN layers rather than the data movement in a single CNN layer. To our knowledge, only one dataflow reported in [9] (fused-CNN dataflow) is in this latter category. This dataflow fuses the computation of several CNN layers together to eliminate the off-chip memory traffic (for the fused layers only) by avoiding fetching the intermediate feature maps. The fused-CNN dataflow is most effective for the first few CNN layers where feature maps have large grids (dimensions) requiring extra on-chip storage or re-computing energy. Applying this dataflow results in the cascading of multiple CLPs in the pipeline fashion. Once a CLP computes a CNN layer instead of writing back the computed OFM pixels, the pixels are used by the next CLP in the pipeline to eliminate the off-chip memory traffic due to the write-back of the intermediate feature maps. The problem with this approach, however, is that the power efficient fusion of multiple CNN layers costs prohibitive on-chip storage. For a given FPGA with certain on-chip memory budget, only very few layers can be fused (e.g., 5 layers). In the shallow networks, such as AlexNet and VGGNets with a maximum number of sixteen CNN layers, because the first few CNN layers of the network have the largest feature map traffic in the network, only the first few CNN layers of these shallow networks need to be fused together to reduce feature map traffic. However, modern DCNNs have a large number of layers (easily exceeding hundreds), and the feature map traffic is distributed across the network rather than on the first few CNN layers. Thus, the effectiveness of this approach is very limited.

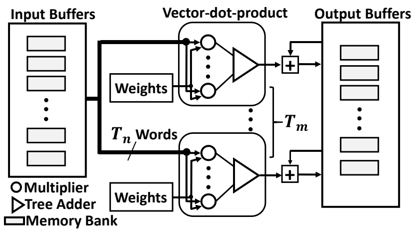

Fig. 1: Convolution layer processor (CLP) datapath.

2.2 Tiling

The tiling technique is often used on feature maps to reduce the requirement for on-chip memory [5]. Each feature map is tiled by a factor of T_r and T_c in the row and the column, respectively. Also, T_n is the tiling factor on the N IFMs, and T_m is the tiling factor on M OFMs. Fig. 2a shows the tiling of feature maps of a convolutional layer in a CLP. Based on the available computational resources of the target FPGA and the dimensions of the CNN layer, the optimal values for T_n and T_m can be found for the minimal execution cycles. However, the optimal values for these parameters (T_n, T_m) can be different for different CNN layers. As described in [6], the global optimal values of T_n and T_m for the maximum performance can be selected by enumerating of all possible combinations of T_n and T_m. The optimal values of T_n and T_m across all CNN layers are called *cross-layer unroll factors*. Fig. 2b shows an example pseudo code of the tiling technique in the CLP. The main modules are the Load_IFM_Tile, Load_Weight, Conv and Store_OFM_Tile. The Load and Store modules interact with external memory. "DATAFLOW" directive applies the ping-pong operation to overlap the computation and the communication latency, as explained in the next sub-section. We define the direction of processing as the counting direction of *r*, *c*, *m*, and *n* loops in Fig. 2b, e.g., up-counting signifies an increasing value of the counting variable when processing (like Fig 2b). Therefore, if for a given layer *i* the direction of processing is up-counting, *reversing* the direction of processing for the next layer means that layer *i*+1 is processed in the down-counting direction.

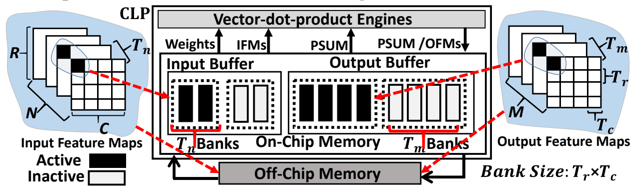

Fig. 2a: Tiling of feature maps of a CNN layer in CLP.

```
for (r =0; r<R; r+= T_r)
    for(c =0; c<C; c+= T_c)
        for (m =0; m<M; m+= T_m)
            for (n =0; n<N; n+= T_n)
#pragma DATAFLOW {
    Load_IFM_Tile(n,r,c);
    Load_Weights (n,m);
    Conv();
    Store_OFM_Tile(m,r,c); }
```

Fig. 2b: Pseudo code example for tiling technique in CLP.

2.3 On-chip Memory Sub-System

The on-chip memory sub-system is a vital part of DCNN accelerators and has a considerable impact on the performance and energy of CNN accelerators [11]. The CLP organizes its on-chip memory as double input and output buffers in the ping-pong manner, so as to hide the communication latency of the off-chip memory by overlapping loading/storing time with computation time [2, 4, 6, 9]. As shown in Fig. 2a, the input and output buffers that are used by the PEs during convolutional layer processing are *active* input and output buffers; whereas the other set of input and output buffers that are used by the data transfer manager for preloading and storing are *inactive* buffers.

The on-chip memory of FPGA-based DCNN accelerators is organized as a banked memory rather than a unified memory [11] (as well as all the current state-of-the-art designs [2, 4, 6]), because the banked memory organization provides multiple simultaneous accesses to IFM pixels, weights, and partial sums (intermediate results). The IFM, OFM, and weight buffers (both active and inactive) of the CLP are implemented as $2 \times T_n$, $2 \times T_m$, and $2 \times (T_n \times T_m)$independent memory banks, respectively. It means that in every cycle, the CLP can read T_n pixels from the input feature map buffer and write T_m pixels (or partial sums) into the output feature maps buffer (see Fig. 1). Meanwhile, the accelerator can load IFMs from the off-chip memory into T_n inactive input banks, and it can store OFMs from T_m inactive output banks into the off-chip memory to hide the off-chip memory latency. Each input buffer bank has the size of $[S \times T_r + K - S] \times [S \times T_c + K - S]$ words. Where, K and S are the size of the kernel and stride, respectively. Also, the size of each output and weight buffer bank is $T_r \times T_c$ and $K \times K$ words, respectively [2, 4]. Different layers may have different parameters in a DCNN. Therefore, each input, output and weight buffer bank should be sized properly to support all layers [4].

The modern FPGAs may have enough on-chip memory to accommodate a considerable portion of a CNN layer or, in some cases, the entire CNN layer. Table 1 shows the available on-chip memory in modern Xilinx FPGAs. FPGAs from the Virtex UltraScale+ family can have an on-chip memory up to 56.81MB (45MB UltraRAM+11.81 BRAMs). In comparison, the four largest CNN layers in ResNet with 151 CNN layers (ResNet-152) required 9.19MB, 6.95MB, 3.7MB, and 2.27MB in 32-bit floating point data type. Thus, it can be seen from Table 1 that a modern FPGA such as an FPGA from the Xilinx Virtex UlteraScale+ family can store each of these four largest CNN layers of ResNet-152 on-chip.

Table 1: On-chip memory in modern Xilinx FPGAs

Xilinx FPGA Family [8]	On-Chip Memory	
	Block Memory(MB)	UltraRAM(MB)
Virtex UltraScale+	3.16 – 11.81	11.25 – 45
Kintex UltraScale+	1.59 – 4.32	4.5

3 MOTIVATIONS AND CHALLENGES

As CNNs evolve to the deeper structures, the off-chip feature map traffic increases rapidly [9]. Our goal is to reuse the on-chip OFMs at the end of each CNN layer processing as part of the IFMs for the next layer to reduce the off-chip feature map traffic. The key motivation behind the on-chip OFMs reuse is that a modern FPGA has a large amount of on-chip memory and reusing the large amount of on-chip OFMs can provide a unique opportunity to reduce the off-chip feature map traffic. However, the OFMs reuse faces the following challenges in the start-of-the-art FPGA-based CNN accelerators.

Static Memory Bank Assignment. As explained previously, the on-chip memory sub-system of the CLP includes two sets of input buffer (which includes the weights buffer) and output buffer to support the ping-pong operation for overlapping the computation with the communication. While the input and the output sets change their status between the active and the inactive modes dynamically, the type of the sets is static and this assignment is performed during the design time. This means that the memory banks in the input buffer are always used at the input of the PEs and memory banks in the output buffer always are used at the output of the PEs. This kind of the static bank assignment limits the reuse opportunity of the OFMs of the current layer as the IFMs for next layer processing. The static assignment forces the CLP with a single-layer dataflow to write back the OFMs during the processing transition from one layer to the next layer. Even when multi-CLP are cascaded in the pipeline manner to implement the fused-CNN dataflow, the accelerator still suffers from the static memory bank assignment problem. This is because, when the accelerator finishes the computation of the fused CNN layers, the OFMs of the last fused layer must be stored into the DRAM by the accelerator [9]. That is, the first layer's input feature maps (the starting point of the fusion) are loaded into the on-chip memory and after the fusion evaluation, the last fused layer's output feature maps (the stopping point of the fusion) in the output buffer are written back to the DRAM [9].

OFM Reuse Challenges. There are two major challenges in the on-chip OFMs reuse implementation:

1)-Padding around the OFMs: In order to reuse layer *i*'s OFMs (which are on-chip) as the IFMs of the layer *i*+1, a padding should be performed around the OFMs, to prepare them for further processing. Depending on the position of the remaining tile of OFMs (the upper left corner or the lower right corner) in the output buffer at the end of layer *i* processing, different areas of the OFMs should be padded to make the OFMs ready for reusing in the layer *i*+1 as is illustrated in Fig. 3a. However, performing the padding on the on-chip OFMs can cause the position of the pixels in the output buffer banks changes. Fig. 3b illustrates an example of the pixel relocation issue where padding is performed on a 3×3 OFM, with kernel size of 3 and stride of 1. The relocation of the on-chip OFM pixels causes on-chip data movement and energy consumption. The situation becomes worse in the modern FPGAs with a large on-chip memory where a considerable amount of OFMs can be reused. Therefore, the computed OFM pixels should be written in the locations of the output buffer where performing the padding doesn't need any further pixel relocation.

2)-Memory Bank Tracking: State-of-the-art implementations of the CLP use High-Level Synthesis (HLS) [2, 4, 6, 9]. For example, the most recent implementation of the CLP uses C++ HLS. The "DATAFLOW" directive is used for implementing the ping-pong operation [2]. If a flexible on-chip memory is employed to solve the static memory bank assignment problem, each bank needs to have the ability to be used as an active input bank, an inactive input bank, an active output bank, or an inactive output bank. For efficient reuse of on-chip OFMs, the CLP should be able to keep track of individual banks during the ping-pong operation. Consequently, in the implementation of the CLP with the flexible on-chip memory instead of relying on the "DATAFLOW" directive for the ping-pong operation, a new tracking scheme of individual memory banks should be used.

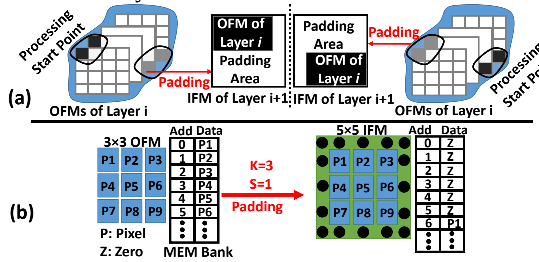

Fig. 3: Padding on the OFMs for reusing.

4 FLEXIBLE ON-CHIP MEMORY

The static memory bank assignment limits the OFM reuse during the computational transition between adjacent layers. To address this limitation in the CLP, we proposed a flexible on-chip memory architecture as shown in Fig. 4. In the proposed design, each memory bank can be used either as an input bank or an output bank by the CLP during the CNN layer processing. This facilitates the on-chip OFM reuse for the next layer processing without additional on-chip data movement. As each bank is used to store a tile of an IFM or a tile of an OFM, the size of a bank should be large enough to support the storage requirements across the CNN layers. This means that the size of each bank should be $[S \times T_r + K - S] \times [S \times T_c + K - S]$ words and the T_r, T_c, S, and K should be set to provide enough space for the CNN layer that needs the largest on-chip memory. Note that the proposed on-chip memory architecture can be configured to work with different on-chip memory sizes. This is because the row and the column tiling factors (T_r,T_c) can be adjusted separately in the flexible memory architecture for any given memory size constraint. For example, T_r, and T_c can be selected to use the same amount of on-chip memory that is used for the baseline memory sub-system. In order to keep track of the status of each individual bank, the flexible memory sub-system holds four arrays for active input banks (with T_n entries), inactive input banks (with T_n entries), active output banks (with T_m entries), inactive output banks (with T_m entries). We call these arrays as *bank tracking arrays*. Each element of the arrays stores the index of a memory bank. For example, the array for the active input banks has T_n entries and this array stores the index of all the banks that can be used as the active input banks. This data structure allows the implementation of a tracking scheme to track individual banks during the ping-pong operation.

4.1 On-chip Memory Bank Management

To reduce off-chip feature map traffic and increase on-chip buffers utilization, we develop an on-chip memory bank management scheme for the proposed flexible on-chip memory architecture. This scheme is based on maximizing the feature map reuse during the computational transition from one CNN layer to the next CNN layer. In the following, to keep the generality of the proposed approach, we assume that the active output buffers contain T_m tiles of OFMs at the end of each CNN layer processing, and each output tile contains $T_r \times T_c$ OFM pixels (see Figure 2). However, in practice, depending on the FPGA chip, it is possible to hold an entire OFM on-chip. Our experiment results on different DCNNs and FPGAs show that the optimal value of the cross-layer unroll factor T_m is greater than T_n. The current implementations of the CLP including [2, 6] also confirm this observation.

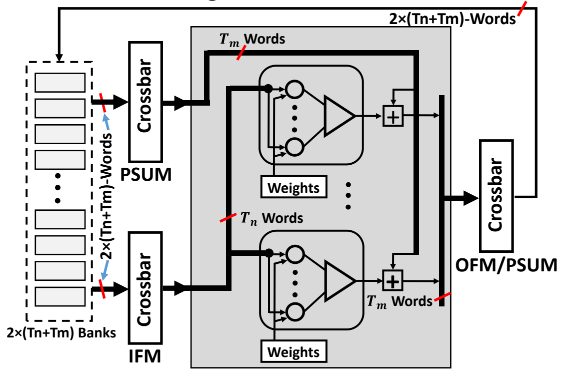

Fig. 4: CLP datapath based on the flexible on-chip memory.

After finishing the current convolutional layer under processing (layer *i*), the active output buffer contains T_m tiles of the OFMs. In the proposed approach, instead of writing these T_m tiles of the OFMs back into the off-chip memory and reading them again, these on-chip tiles can be used for the next layer processing (layer *i*+1) as the IFMs as shown in Fig. 5a. By reversing the direction of processing and switching between input and output buffers during the layer *i*+1's evaluation alternatively, the CLP can reuse the T_m tiles of the layer *i*'s OFMs as the IFMs for the layer *i*+1 (see Fig. 5a). Since during the layer *i+1* processing the CLP reuses the feature maps of the layer *i*, there is no need to load the feature maps from the off-chip memory. Therefore, the CLP does not use the inactive buffers to hide the off-chip memory latency. Instead, the accelerator preserves the OFMs from the previous layer (layer *i*) in the inactive output buffer for the future uses in the layer *i*+1 processing (referred to as the inactive output banks which preserve OFMs from the previous layer as inactive reserved banks). After consuming all the on-chip OFMs, the accelerator should access the off-chip memory for the continuation of the layer *i*+1 processing. In this scenario, the accelerator always skips the loading T_m tiles of the layer *i*+1's IFMs and if $M_{i+1} \leq T_m$ it can also skips the writing of T_m tiles of the layer *i*'s OFMs.

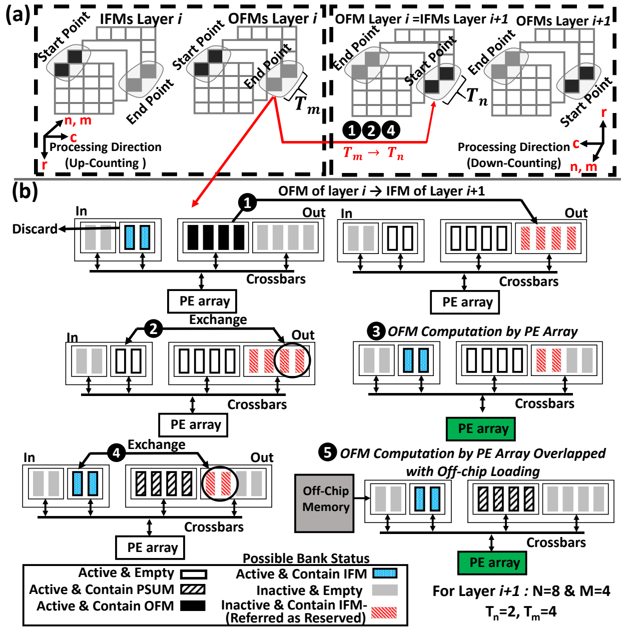

Fig. 5: Illustration of reusing the OFMs of layer *i* as IFMs of layer *i*+1 at the memory bank level.

Fig. 5b illustrates an example of using the proposed memory bank management scheme in a CLP based on the flexible on-chip memory. For simplicity, we assume that each memory bank of the CLP can store a complete feature map (which is highly practical in modern FPGAs). Note that each bank of the CLP can be implemented by several BRAMs or UltraRAMs. In this example, we assume 2 and 4 values for T_n and T_m, respectively. Also, layer *i*+1 has eight IFMs (*N*=8) and four OFMs (*M*=4). Therefore, at the end of layer *i* processing four OFMs (in general T_m OFMs) of this layer remain on-chip which they can be reused as the IFMs for layer *i*+1 ($M_i = N_{i+1}$). The rest IFMs (in this example four IFMs) to complete the processing of layer *i*+1 are already computed during the processing of layer *i* but are currently stored in the off-chip memory. The following steps explain the reuse of on-chip OFMs of layer *i* as well as pre-loading the rest of IFMs for the next CNN layer *i*+1 processing. 1) At the end of the layer *i* processing, the CLP reverses the processing direction. In this example, the direction of processing changes from up-counting to down-counting. 2) The CLP keeps the OFMs of layer *i* on-chip for reusing as IFMs in the layer *i*+1 processing by switching the status of the active output banks to the inactive output banks (Label 1 in Fig. 5b). 3) The CLP exchanges T_n active input banks with T_n inactive output banks (reserved banks) which contain IFMs (Label 2 in Fig.5b, here T_n is 2). The exchange is done through index array manipulation rather than moving data across banks physically. 4) After the exchange, computation can be started by the PE array (Label 3 in Fig. 5b). 4) Again, the CLP exchanges T_n active input banks with the rest T_n inactive output banks which contain another T_n IFMs. 5) After the exchange, computation can be continued by the PE array. At this point, all the T_mOFMs which are remained on-chip at the end of layer *i* processing are reused as the IFMs for layer *i*+1 processing by the PE array. In addition, at this step, the CLP starts to pre-load T_n IFMs into the inactive input banks to overlap the communication latency with the computation, similar to the baseline CLP (Label 5 in Fig. 5b, here T_n is 2). The weights prefetching is ignored for the simplicity in this example, but are also preloaded by CLP in implementation.

In all the above steps there is no data movement associated with the buffer exchanges and updates since any bank in the flexible memory architecture can be selected as an active input bank, an inactive input bank, an active output bank, or an inactive output bank. The exchanges and updates only cause modification on the bank tracking arrays. Also, in order to avoid the data movement due to the padding when the remaining on-chip OFMs of layer i are reused as IFMs for layer $i+1$ (Label 1 in Fig.5b), the output pixels should be written into the locations of the active output buffer (during the computation) that is compatible with the next layer padding dimensions and format. This means that the generated write addresses by the CLP should be in a range (the gray area of the IFM in Fig. 3) that no further relocations of OFM's pixels are needed for the padding. We consider this write address generation pattern in our accelerator implementation to avoid any padding data movement. Another advantage of the proposed design is that the CLP has a full control on individual memory banks rather than having a control on a set of memory banks (the baseline memory sub-system). This flexibility leads to a better utilization of memory banks as it is demonstrated in the above example. Note that, in practice, modifications on the bank tracking arrays for buffer status updating and exchanging (labels 1, 2, and 4 in Fig. 5b) can take several hundreds of cycles (e.g., 572 cycles), which is negligible as each CNN layer processing takes tens of thousands of cycles (e.g., 83,000 cycles). Furthermore, it can be overlapped with computation if needed, e.g., modification in the bank tracking arrays of Label 4 in Fig. 5b can be overlapped with the computation in the Label 3 in Fig. 5b. The proposed flexible on-chip memory architecture achieves on-chip OFMs reuse and can be employed in the both categories of dataflow for CNN layer evaluation.

4.2 Accelerator Implementation

An optimization program tool is first developed to generate the optimized parameters (T_n, T_m, T_r, T_c) for the baseline CLP and the CLP based on the flexible on-chip memory architecture. The program enumerates all the possible combinations and selects a parameter set with a minimum number of execution cycles that satisfies the target FPGA resource budget and bandwidth constraint. The tool uses the models in [2] to calculate the execution cycles, the memory bandwidth requirement, DSP slices and BRAMs usage for each set of the parameters. It takes as input a file containing the descriptions of each CNN layer, a target FPGA resource budget profile, and the maximum memory bandwidth. After the generation of the optimized parameters for both accelerators, the values are used in parameterized implementations of the accelerators. The accelerators are implemented in the high-level-synthesis (HLS). Vivado HLS 2017.1 is used to compile the HLS implementations with the fixed parameters to synthesizable Verilog. We use HLS pragma and directives to instruct the compiler to implement the architectural structure of the accelerators. Both accelerators are operated in 32-bit floating point (FP32) and 16-bit fixed-point arithmetic. Also, the accelerators use a separate buffer for pre-loading shortcut connections in residual DCNNs to reduce the off-chip traffic.

A host CPU controls and initializes the accelerators. The host CPU is implemented as a MicroBlaze soft core processor for Xilinx FPGAs. The accelerators are connected to the host CPU through an AXI4-lite bus as a slave to receive the control commands and the parameters for each CNN layer. Four separate AXI4 ports for loading/storing of IFMs, weights, shortcuts, and OFMs are used to connect the accelerators through an AXI interconnect to the memory interface controller.

5 EVALUATION

To show the effectiveness of the flexible memory in reducing off-chip feature map traffic during CNN layers processing, two accelerators based on the baseline and the proposed flexible on-chip memory architecture are evaluated and compared to process different DCNNs with different range of CNN layers from 16 to 151 (VGGNet-E, ResNet-34, ResNet-50, ResNet-152) on two contemporary FPGA chips (Xilinx Virtex UltraScale+ VU9P and VU13P). We used the optimization tool to generate the accelerator's parameters for a given FPGA and DCNN under the same on-chip memory size constraint for fair comparison.

5.1 32-bit Floating-Point

In our first evaluation, we run ResNet-152 on the Virtex UltraScale+ VU9P FPGA. Table 2 shows the optimized parameters (T_n, T_m) generated by the tool to meet the memory bandwidth constraint and target frequency, along with the throughput. Since the deep ResNets have considerable numbers of shortcut connections, both implemented accelerators use a dedicated buffer to preload and store the shortcut connections to facilitate the ResNets processing. The shortcut connection buffer is implemented by the URAM blocks, while the BRAMs are used to implement the IFM and the OFM buffers, and the weights buffer are implemented using distributed RAMs, with 32-bit floating point arithmetic in both accelerators. Table 3 shows the FPGA resource utilization. Our experiment results show that the CLP with the flexible on-chip memory architecture has 16% lower feature map traffic between the accelerator and the off-chip memory during CNN layer evaluation, compared with the baseline CLB. The proposed architecture achieves a significant reduction in URAM usage (and some reduction in BRAM usage), with only slight increased use of DSPs, LUTs, and FFs.

Table 2: Accelerators comparison for ResNet-152

Accelerator Type	T_n	T_m	Frequency (MHZ)	Memory Bandwidth (GB/s)	Throughput (cycles per Image)
Baseline	8	128	100	10.4	12,641,195
Flexible	8	128	100	10.4	12,830,812

Table 3: FPGA resource usage

Accelerator (Tn, Tm)	URAMs	BRAM 18K	DSP Slices	FFs	LUTs
Baseline (8,128)	512	2,294	5,461	693,247	523,789
Flexible (8,128)	256	2,176	5,627	719,904	540,803
Available	960	4,318	6,840	2,364,480	1,182,240

5.2 Scalability of Flexible Memory Sub-System

We also extend our experiments and target a more advanced FPGA chip (VU13P FPGA) to show the scalability and advantage of using the proposed flexible memory architecture. The accelerators are evaluated to process four DCNNs with a different range of CNN layers. The networks are selected in order to show the advantage of the proposed design as deeper networks are used. 32-bit floating point arithmetic is used for data representation in the networks. Fig. 6 shows the off-chip feature map traffic between the accelerator and the external memory. The accelerator based on the proposed architecture is able to reduce the off-chip feature map traffic for VGGNet-E, ResNet-34, ResNet-50 and ResNet-152 by 11%, 14.5%, 23.6%, and 40%, respectively. This indicates that, as deeper networks are used to meet the accuracy requirement, the proposed design will likely be more effective to reduce the off-chip traffic.

5.3 16-bit Fixed-Point

There is an increasing trend to use compact data representations in DCNNs to improve the efficiency [3]. Therefore, in order to investigate the impact of using a compact data representation on the effectiveness of the flexible memory sub-system, we conduct further experiments for 16-bit fixed-point on a network with a shallow depth (VGGNet-E). Using a shallow depth network with a compact data type can highlight the efficiency of the flexible on-chip memory architecture more clearly as the proposed design achieves a better performance for deeper networks. Again, our developed optimization tool is used to generate the parameters of both accelerators for the VGGNet-E on VU9P FPGA in 16-bit fixed-point. The generated parameters are used to customize the parameterized HLS implementations of both accelerators. Experiment results show that, for the 16-bit fixed-point data type, the accelerator based on the proposed architecture can reduce the feature map traffic by nearly 50% for VGGNet-E and reduce the usage of BRAMs by 23%, compared with the baseline. This shows the capability of the proposed approach for deep networks with compact data types. The accelerators based on the approach and the baseline schemes can reach a throughput of 5,342,237 cycles/image and 5,136,768 cycles/image, respectively.

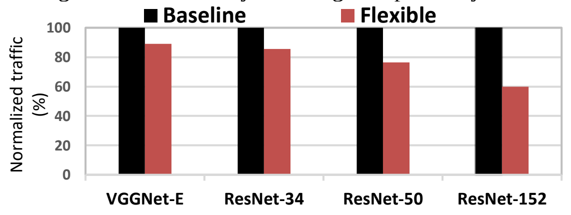

Fig. 6: Normalized off-chip feature map traffic in FP32

6 RELATED WORK

There is an increasing amount of work focusing on accelerating DCNNs on FPGAs [2,4-7]. However, all of them suffer from the static bank assignment problem, and the accelerator cannot effectively reuse feature maps for the next layer CNN processing. For example, the most recent work on the FPGA-based CNN acceleration [2] proposes a solution for the under-utilization problem of DSP slices (the arithmetic units in the CLP); whereas the proposed flexible memory architecture focuses on off-chip feature map traffic reduction, thus complementing [2]. Specifically, the solution in [2] tackles the under-utilization problem by partitioning FPGA resources among multiple accelerators (multi-CLP) with different smaller sizes. In case that the same CLP is used to process adjacent CNN layers in the multi-CLP approach, the flexible memory architecture can be used in each accelerator of the multi-CLP approach to reduce the off-chip feature map traffic as well as the overall off-chip feature map traffic generated by all the CLPs. In another recent example [4], a flexible buffering scheme is presented for the CLP to balance the off-chip bandwidth between feature maps and weights by choosing an optimal batch size [4]. However, the accelerator still suffers from the static bank assignment and the proposed architecture can be used to reuse feature maps. Another example is the fused-CNN accelerator discussed previously [9]. Other earlier works such as [11, 12] focus on the two-dimensional convolution engine, including the order of fetching data for processing and the data caching, but not for 3D convolution structures in deep networks.

7 CONCLUSION

With the objective of utilizing reusable data during the computational transition between CNN layers, we propose a flexible on-chip memory architecture with a bank management scheme. The key motivation behind our design is to address the static assignment of memory banks and more efficient utilization of large on-chip memories in modern FPGAs. In the proposed architecture, each individual memory bank can set its status dynamically during CNN processing and computational transition to the next layer. Experiment results show that using the proposed architecture can reduce the feature map traffic by 40% for a deep network such as ResNet-152 in 32-bit floating point on the Xilinx VU13P FPGAs, and reduce the off-chip feature map traffic by 50% in a more compact data type on the Xilinx VU9P FPGA.

Acknowledgment: This research was supported, in part, by the National Science Foundation grants #1619456 and #1619472.

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2011. Deep Residual Learning for Image Recognition. In *the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] Y. Shen, M. Ferdman, and P. Milder. 2017. Maximizing CNN Accelerator Efficiency Through Resource Partitioning. *(ISCA '17).*

[3] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Gee Hock Ong, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, and Guy 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)*.

[4] Yongming Shen, Michael Ferdman, and Peter 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)*.

[5] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong 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)*.

[6] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason 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)*.

[7] Lili Song, Ying Wang, Yinhe Han, Xin Zhao, Bosheng Liu, and Xiaowei 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)*.

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

[9] 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)*.

[10] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (Jan 2017).

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

[12] Maurice Peemen, Arnaud AA Setio, Bart Mesman, and Henk Corporaal. 2013. Memory-centric accelerator design for Convolutional Neural Networks. In Proceedings of the 31st *IEEE International Conference on Computer Design (ICCD '13)*.