Striping Input Feature Map Cache for Reducing off-chip Memory Traffic in CNN Accelerators

Rastislav Struharik and Vuk Vranjkovic, *Member, IEEE*

***Abstract* — Data movement between the Convolutional Neural Network (CNN) accelerators and off-chip memory is critical concerning the overall power consumption. Minimizing power consumption is particularly important for low power embedded applications. Specific CNN computes patterns offer a possibility of significant data reuse, leading to the idea of using specialized on-chip cache memories which enable a significant improvement in power consumption. However, due to the unique caching pattern present within CNNs, standard cache memories would not be efficient. In this paper, a novel on-chip cache memory architecture, based on the idea of input feature map striping, is proposed, which requires significantly less on-chip memory resources compared to previously proposed solutions. Experiment results show that the proposed cache architecture can reduce on-chip memory size by a factor of 16 or more, while increasing power consumption no more than 15%, compared to some of the previously proposed solutions.**

***Keywords* — Cache memory, convolutional neural network, energy-efficiency, low-power computing.**

I. Introduction

STATE-OF-THE-ART convolution neural networks (CNNs) are constructed from a large number of layers, in some cases even more than a thousand [1], to reach ever-higher accuracy. However, this comes at a price of increasing computational load and memory bandwidth, which even more importantly, increases power consumption. Increased power consumption is a serious problem, especially when CNNs are deployed in embedded applications, where solutions with power consumption above 1W are rarely acceptable. In order to solve these problems, many different solutions have been proposed, including conventional CPUs [2], GPUs [3], [4], ASICs [5], [6], and FPGAs. ASICs are natural candidates for offering low power solutions, but they usually lack flexibility in supporting the newest CNN architectures. On the other hand, reconfigurable computing platforms like FPGAs are providing both energy-efficiency and flexibility at the same time. As a result, FPGAs have become an attractive candidate for low-power CNNs acceleration.

Most FPGA-based accelerators process CNNs layer by layer[7]-[12], using single-layer data-flows, in which both input and output feature maps are stored in the off-chip memory due to their large size. Such an approach causes a lot of data transfer between the CNN accelerator and the off-chip memory. This problem is partially solved by the solution proposed in [13]. In contrast with the solution presented in [13], which focuses on minimizing data transfer from off-chip memory during layer swapping, this paper is concerned with minimizing data transfer during single layer processing.

Our work was motivated by results presented in [14], which show that significantly less power is consumed during transfers from on-chip memory to CNN accelerator (5pJ/access), compared to data transfers between CNN accelerator and external memory (640pJ/access). During the processing of a convolutional layer in particular, which is the most time-consuming operation in CNNs, significant reuse of input feature map data is possible by intelligent caching, thus reducing power consumption.

This paper presents a new version of Input Feature Map Cache, compared with previously published version in [15], which significantly reduces the required size of cache memory (with the reduction factor of 16 and more), with a minimal increase in power consumption (a maximum increase of no more than 15%).

The rest of this paper is organized as follows. In Section II a basic idea about improved input feature map stick buffer is presented. Details about the implementation of the new version of the input stick buffer are described in Section III. Finally, Section IV presents the results of experiments using five standard CNNs architectures: MobileNet v1, SqueezeNet, ResNet-18, ResNet-50, and Inception v3. The conclusion is also given in Section IV.

II. Input Feature Map Striping Stick Buffer

Let us first introduce some definitions. Let the **Input Feature Map** (IFM) be a 3D set of values making the input volume of a CNN layer. Input to the first CNN layer is usually not called an IFM, but rather an **input instance**.

A 3D region of $K_H \cdot K_W \cdot K_D$IFM points, where K_H, K_W, and K_D are kernel height, width, and depth, respectively, from the IFM that is directly connected to one neuron in the

Paper received May 05, 2020; revised November 10, 2020; accepted November 17, 2020. Date of publication December 25, 2020. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Vujo Drndarević.

This paper is revised and expanded version of the paper presented at the 27th Telecommunications Forum TELFOR 2019 [17].

This work is partially supported by the following grant: „Innovativeelectronic components and systems based on inorganic and organictechnologies embedded in consumer goods and products“, pr. num.TR32016, Serbian Ministry of Education and Science.

Rastislav J.R.Struharik, University of Novi Sad, Faculty of Technical Sciences, TrgDositeja Obradovića 6, 21000 Novi Sad (phone +381-21-485-2537,email: rasti@uns.ac.rs.

Vuk Vranjkovic, University ofNovi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6,21000 Novi Sad (email: bykbpa@gmail.com).

current CNN layer will be designated as the **IFM bundle**.

A 3D region of $1 \cdot 1 \cdot K_D$ points within the IFM bundle will be called the **IFM stick**. Every IFM bundle is composed of a number of IFM sticks. For example, a $3 \cdot 3 \cdot K_D$IFM bundle is composed of 9 IFM sticks, each being a 3D region of $1 \cdot 1 \cdot K_D$ points, as shown in Fig. 1.

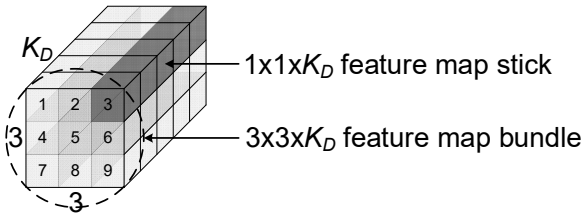

Fig. 1. Definitions of IFM stick and IFM bundle.

During the processing of convolutional and pooling layers, IFM is traversed in such a way that there is a significant overlap between adjacent IFM bundles, as shown in Fig. 2. This happens because convolutional kernels and pooling areas usually have horizontal and vertical stride values that are smaller than the kernel/pooling area size. This is illustrated in Fig. 2, where IFM is traversed using a kernel of size 3x3, with horizontal and vertical stride values of one.

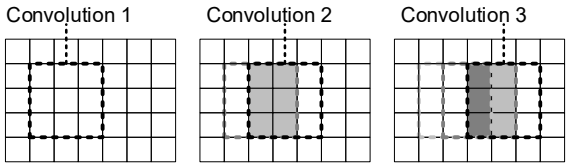

Fig. 2. IFM traversal using 3x3 convolutional kernel.

The left-most picture from Fig. 2 shows a region of IFM which is used to compute Convolution 1. As the kernel is shifted in the horizontal direction one IFM point to the right, since the horizontal stride in this example is set to one, a new IFM bundle is selected, which will be used to compute Convolution 2, as shown in the middle picture from Fig. 2. By comparing the IFM bundles that will be used to compute Convolutions 1 and 2 we can observe that there is a significant overlap between them, as indicated by the gray IFM sticks in Fig. 2. Convolution 1 and 2 IFM bundles share a total of six IFM sticks. As the convolutional kernel is shifted for one additional position in the horizontal direction, to select the IFM bundle that will be used in the computation of Convolution 3, there is still a significant overlap even with Convolution 1 IFM bundle (three IFM sticks shown in dark gray color), and even larger overlap with Convolution 2 IFM bundle (six IFM stick shown in light and dark gray color).

This significant overlap between adjacent IFM bundles offers an opportunity to optimize memory transfer between CNN accelerator and off-chip memory, by using cache memory. Instead of loading every IFM stick from a given IFM bundle directly from the off-chip memory, previously used IFM sticks can be stored locally, in the on-chip cache, and loaded from there, thus saving power and possibly even reducing latency of memory access.

There are many different ways how this IFM stick cache can be designed. One possible design was proposed in [15]. In the case of the design proposed in [15] size of IFM stick cache must be big enough to store $K_H \cdot IFM_W$IFM sticks at any time, where IFM_W is the width of IFM of the current layer, as shown in Fig. 3. a). Please notice that every IFM stick is K_D IFM points deep. Since sizes of kernels and IFMs for different CNN layers usually vary, to support all these IFMs, IFM stick cache size has to be selected in a way to satisfy the following constraint

$$ISB_{Size} \geq \max_{\substack{\in network \\ j \in layer \\ k \in kernel}} \left\{ K_{H_{i,j,k}} \cdot IFM_{W_{i,j}} \cdot K_{D_{i,j,k}} \right\} \qquad (1)$$

From formula (1) it can be observed that the design of IFM cache from [15] is not universal, in a sense that it cannot support arbitrary sized IFMs and kernels, and even supporting only a fraction of IFM or kernel sizes requires large cache memories, which can be prohibitive in many embedded applications.

In this paper, we propose a new organization of IFM stick cache, based on the idea of IFM striping, which can support arbitrarily sized IFMs while requiring significantly less on-chip memory resources. The idea of IFM striping is illustrated in Fig. 3. b).

When using IFM striping, the original IFM is partitioned into a number of vertical IFM stripes. In this case, instead of traversing the IFM as a whole, it is traversed one stripe at a time. Because of this, IFM stick cache size can be made smaller, since it must be able to store only $K_H \cdot IFM\_Stripe_W$ IFM sticks at a time, where IFM_Stripe_W is the width of individual IFM stripe.

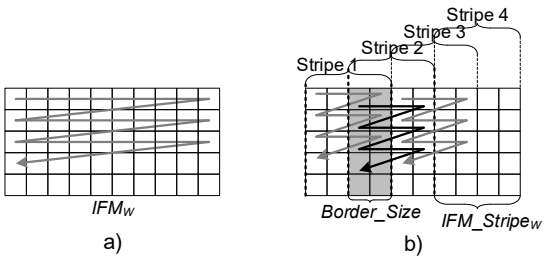

Fig. 3. (a) IFM stick cache from [15];
(b) IFM striping stick cache.

Dividing IFM into stripes provides an extra parameter, width of the IFM stripe, IFM_Stripe_W, which can be used to control the size of IFM stick cache memory. Theoretically, when using IFM striping technique of traversing the IFM, IFM stick cache can be made as small as the one needed to store only one IFM bundle, corresponding to the largest 3D kernel from the selected set of CNNs that should be supported,

$$ISB_{Min\_Size} = \max_{\substack{i \in network \\ j \in layer \\ k \in kernel}} \left\{ K_{H_{i,j,k}} \cdot K_{W_{i,j,k}} \cdot K_{D_{i,j,k}} \right\} \qquad (2)$$

Please notice that making IFM stick cache as small as possible would not be recommended since in that case the possibility to reuse IFM sticks would be lost and IFM stick cache would become useless. Therefore, finding an optimal size of the IFM stick cache is of great importance when designing a CNN hardware accelerator. On one side IFM stick cache should be made as small as possible, to save on-chip memory resources. On the other side, as the IFM stick

cache gets larger, there is more opportunity for IFM stick reuse. This will reduce the required number of DRAM memory accesses, reducing the power consumption, and CNN processing time.

Please notice that using IFM striping requires repeated re-loading of IFM sticks, located in the border region of adjacent IFM stripes, from the main memory. In case when IFM stick cache is large enough to satisfy equation (1) every IFM stick is loaded from main memory exactly once. But, in case of using IFM striping, all IFM sticks that lay in the border region of adjacent IFM stripes will be loaded at least twice. Size of the border region is determined by the size of kernel horizontal stride value, S_H, and kernel width, as the following equation shows

$$Stripe_{Border\_Size} = K_W - S_H \quad (3)$$

As a consequence of this, increasing the number of IFM stripes original IFM is divided into will increase the number of IFM sticks that will have to be loaded multiple times from the main memory, therefore increasing the power consumption and possibly latency. However, as we will show in Section IV, this power consumption increase is fairly moderate compared with the decrease in memory size required to build IFM stick cache, at least when standard CNN networks are being considered.

III. Implementation of IFM Stick Buffer

IFM stick buffer is part of a larger module, called Input Stream Manager (ISM), which is used to stream input data to the actual Data Computing (DC) module, used to compute numerical operations defined in various CNN layer types, like convolution, pooling, etc. Please notice that the proposed ISM module can be used together with many different implementations of the DC module. For example, the proposed ISM module has been successfully integrated within the CoNNA CNN accelerator [16].

Input stream data usually consists of data computing module configuration data, input image data or IFM data, and convolutional kernel coefficient values or fully-connected weight values, depending on the type of CNN layer that is currently being processed. All CNN-related data is received from external DRAM memory, which is then either stored inside the IFM Stick Buffer (ISB) module, in case of the input feature map data, or routed directly to the appropriate compute modules inside the data computing module, in the case of convolutional coefficients or fully-connected weights data. Fig. 4. a) presents details about the internal organization of the ISM module, with all major sub-modules shown.

ISM module is composed of the following major modules:

- IFM Stick Buffer (ISB) cache – used for storing cached IFM sticks.
- Stick Valid (SV) memory – used to store information is the current IFM stick, located at the same position in the ISB memory, valid for reading or not.
- Read Controller (RC) – used to read selected IFM stick from the ISB cache and transfer it to the DC module.
- Write Controller (WC) – used to write selected IFM stick from external DRAM memory to appropriate position inside the ISB cache.
- Input Stream Router (ISR) – used to select which input data stream will be sent to the DC module.

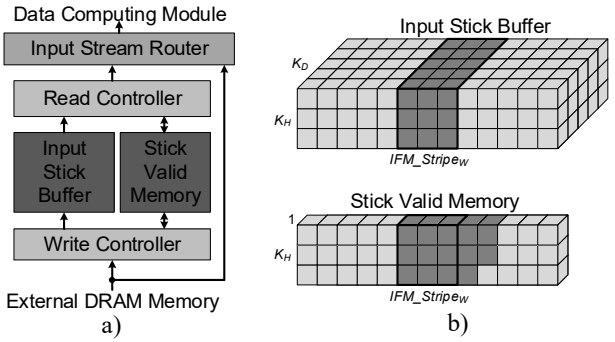

Fig. 4. a) Proposed ISM module;
b) Principle of operation.

The central module of the ISM is the IFM Stick Buffer cache memory module. This cache memory is used for storing selected IFM sticks from the input feature map, which is being processed by the current CNN layer. As explained in Section II, during IFM processing by convolutional and pooling layers, IFM sticks are being repeatedly reused during the calculation of adjacent convolutions/pooling operations, as convolutional/pooling kernels slide over the IFM. This opens a possibility to minimize data movement between the CNN accelerator and external memory by caching IFM sticks that will be reused during adjacent convolutions/pooling calculation steps. ISB module acts as this local on-chip cache of selected IFM sticks. ISB stores selected IFM sticks, 1x1xK_D sections of the IFM, which will be used in the upcoming convolution calculation operations, as shown in Fig. 4. b).

Each time the same IFM stick is needed in the current convolution calculation operation, instead of re-fetching it from external DRAM memory, it is fetched from the ISB cache module, reducing the number of DRAM data transfers, thus saving power. Once all convolution operations involving a given IFM stick are computed, the IFM stick can be removed from the ISB cache and the next IFM stick can be loaded in its place from external DRAM memory.

Read Controller and Write Controller modules are charged with correct manipulation of IFM sticks stored inside the ISB cache. WC module horizontally sweeps the current IFM stripe, line by line, as shown in Fig. 5, and writes IFM sticks into appropriate positions inside the ISB cache module.

On the other hand, the RC module reads selected IFM sticks from the ISB memory and transfers them to the DC module. RC module reads IFM sticks in a different order from the order that the WC module uses to write them in the ISB cache. While WC traverses IFM stripe horizontally, loading a complete line from IFM stripe before moving to the next one, RC traverses ISB memory in a more localized manner, which depends on the frontal shape of convolutional/pooling kernel, as shown in Fig. 5 in the case

of a 3x3 kernel.

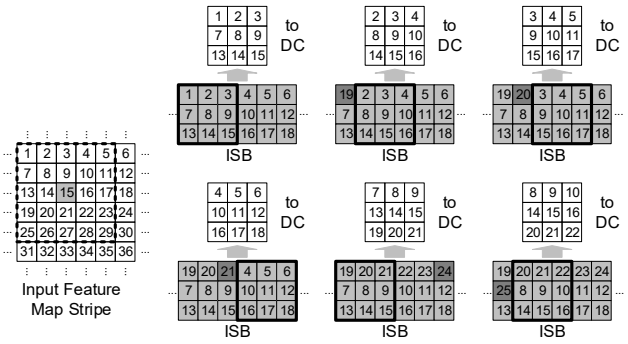

Fig. 5. Process of traversing IFM stripe.

Since the speed at which WC module writes IFM sticks into the ISB cache can differ from the speed at which the RC module reads IFM sticks from the ISB cache because data processing throughputs of the memory subsystem and DC module can be different, a mechanism that ensures consistency of ISB cache content needs to be devised. This is the purpose of the Stick Valid memory module.

SV memory is $IFM_W \cdot K_H \cdot 1$ bits large, as shown in Fig. 4. b). For each IFM stick from the ISB cache there is a corresponding bit in the SV memory, indicating is that particular IFM stick valid for reading or not. Each time WC module needs to write a new IFM stick into a selected location inside the ISB cache, it first checks is the corresponding bit from the SV memory cleared, which indicates that all convolutions that use the current IFM stick stored at the selected location inside the ISB cache have been computed, meaning that it can be replaced with the new IFM stick. If this is the case, WC writes a new IFM stick in the selected location inside the ISB cache and sets the corresponding bit inside SV memory. This situation is shown in Fig. 5, where dark gray boxes represent IFM sticks that have been replaced during the IFM processing. Otherwise, the WC module waits until the corresponding bit in SV memory is cleared.

When the RC module wants to read the IFM stick from the ISB cache, it first checks is the selected IFM stick valid for reading or not, by checking the corresponding bit from the SV memory. If the bit is set, this means that the selected IFM stick data is valid and the RC module can transfer it to the DC module, otherwise the RC module has to wait until the WC module certifies that particular IFM stick data is valid.

The operation of the ISB cache is shown in Fig. 5, which shows a segment of IFM stripe, 6x6 IFM sticks in size. If we assume that this IFM stripe is being processed by a 3x3 kernel with horizontal and vertical strides of one, then each IFM stick will be used in the process of calculating of up to nine different convolutions. Please notice that IFM sticks located at borders of the IFM stripe will be used in a smaller number of convolution computation operations. For example, IFM stick no. 1 will be used in only one convolution computation operation, IFM stick no. 2 will be used in two convolution computation operations, IFM sticks no. 3 and 4 in three convolution computation operations, and so on.

Without ISB cache, each IFM stick would have to be loaded from DRAM memory up to nine times, but by using ISB cache it is only necessary to load each IFM stick only once, or twice if it lies in the border zone as shown in Fig. 3. b), from external DRAM. Fig. 5 also shows the concept of replacing already used IFM sticks within the ISB module with the new ones. New IFM sticks that are being written in the ISB cache by the WC module are shown in dark grey color in Fig. 5.

IV. Experiments

In order to investigate how the amount of total DRAM data transfer size depends on the size of Input Stick Buffer cache, experiments using five popular deep CNNs, MobileNet v1, SqueezeNet, ResNet-18, ResNet-50, and Inception v3, have been performed. As noted earlier, the DRAM data movement represents a major part of the total power consumption when processing CNN networks, [14]. Experiments were performed by simulating the RTL model of the ISB cache, while processing different CNNs and measuring the number of DRAM memory accesses under different sizes of ISB cache, using Xilinx Vivado simulator.

Table 1: Total DRAM Memory Accesses.

ISB SIZE	Number of DRAM memory accesses				
	MobileNet v1	Inception v3	ResNet 18	ResNet 50	SqueezeNet
131072	3618340	12284677	4686905	14272064	2146998
65536	3618340	12284677	4708409	14272064	2146998
32768	3618340	12317461	4837433	14272064	2153974
16384	3654180	12424797	5140265	14273840	2169734
8192	3724069	12905239	5153945	14367904	2199510
4096	3924772	13392870	5178820	14596560	2265692
2048	4070148	14224984	5250042	14996496	2359824

Table 1 shows, how the size of ISB cache, expressed as the number of IFM points, affects the required number of memory accesses to the DRAM memory. The current implementation of IFM stick cache uses a 64-bit AXI-Full interface for accessing DRAM, so every access transfers 64-bits. Every IFM point is represented with 16-bits, so during each memory access, four IFM points are being transferred.

Figs. 6-10 show the relative increase in total DRAM read access numbers compared to a scenario where every IFM from selected CNN is traversed using only one stripe, which corresponds with the ISB cache design presented in [15], for every CNN network used in the experiments.

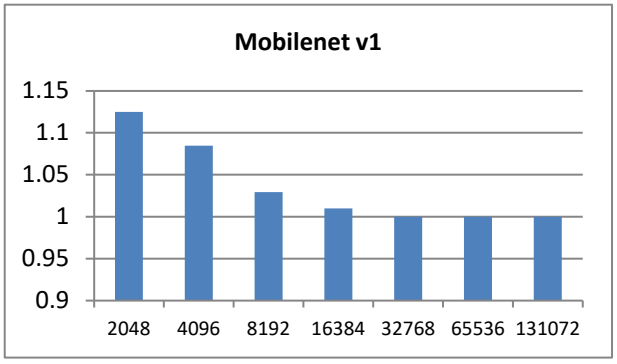

Fig. 6. The relative increase of DRAM memory accesses vs ISB cache size, in case of MobileNet v1 CNN.

In the case of MobileNet v1, we can observe that even when ISB cache size is reduced by a factor of 16, to a size of 2048 IFM points, a relative DRAM data movement increase is only 12.49%. As we increase the size of the ISB cache, the relative DRAM data movement quickly decreases to values that are below 5%, as shown in Fig. 6.

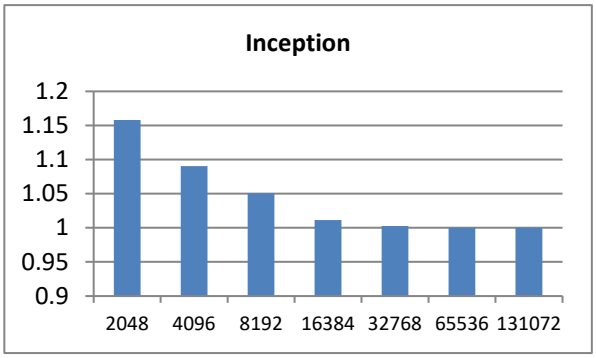

Fig. 7. The relative increase of DRAM memory accesses vs ISB cache size, in case of Inception v3 CNN.

For the Inception v3, the relative increase in the DRAM data movement overhead is slightly higher. For the ISB cache size of 2048 IFM points it reaches a value of 15.79%, but please notice that in this case, ISB cache size is reduced 32 times, compared with the reference configuration. Similar to the MobileNet v1 CNN, as we increase the size of ISB cache, the relative increase in the DRAM overhead decreases sharply to values that are below 5%, as shown in Fig. 7.

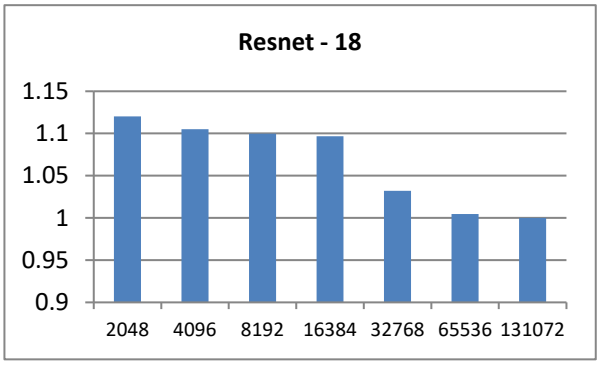

Fig. 8. The relative increase of DRAM memory accesses vs ISB cache size, in case of ResNet-18 CNN.

For ResNet-18 CNN, the situation is similar as with the MobileNet v1 CNN. For the ISB cache size of 2048 IFM points, there is the highest relative increase in the DRAM data movement, of 12.01%. Similarly to all CNN networks that have been used in the experiments, as the size of ISB cache is increased, there is a sharp decrease in the relative DRAM access overhead, to a value that is below 5%, as shown in Fig. 8.

Interestingly, for ResNet-50 CNN, the maximum relative DRAM access overhead increase is significantly smaller, compared with all other used CNN networks and even with the ResNet-18 CNN which belongs to the same family. The maximum relative DRAM access overhead increase for ResNet-50 CNN is only a 5.08% increase when ISB cache is 2048 IFM points large. In the case of ResNet-50, as the size of the ISB cache is an increase, the relative DRAM access overhead quickly falls below 2%, as shown in Fig. 9.

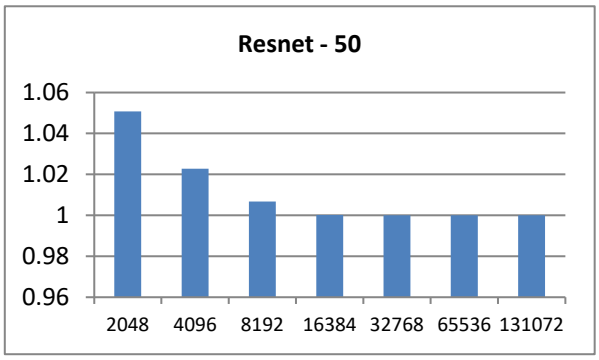

Fig. 9. The relative increase of DRAM memory accesses vs ISB cache size, in case of ResNet-50 CNN.

Finally, in the case of SqueezeNet CNN, the largest DRAM data movement increase is 9.91%, when ISB cache size is 2048 IFM points large. The trend of quick reduction in relative DRAM access overhead when ISB cache size is increased is visible for SqueezeNet CNN also, as can be observed in Fig. 10.

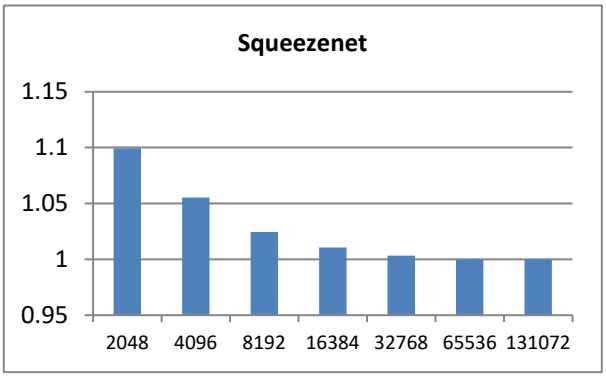

Fig. 10. The relative increase of DRAM memory accesses vs ISB cache size, in case of SqueezeNet CNN.

Based on the results of performed experiments, it can be observed that using IFM striping technique on-chip cache memory for storing IFM points can be reduced from 16 to 32 times, compared to some of the previously proposed solutions [15], while at the same time increasing power consumption due to the additional DRAM data movement by no more than 15%, for all CNN networks that were used in the experiments.

Table 2 presents the implementation results for the ISM module, shown in Fig. 4, for various sizes of ISB cache, ranging from 2048 IFM points to 131072 IFM points. These results we obtained after performing synthesis and implementation steps using Xilinx Vivado 2018.3 software tool using default tool settings.

As can be seen from Table 2, all instances of the ISM module require very little hardware resources, when LUTs and DSPs are concerned. The situation is different when BRAM usage is concerned, where a significant increase can be observed as the size of the ISB cache is increased.

TABLE 2: UTILIZATION.

ISB SIZE	BRAM	LUT	DSP
131072	64	1275	13
65536	35	1186	13
32768	21	1133	13
16384	14	1067	13
8192	10.5	1050	13
4096	8.5	1023	13
2048	7.5	1017	13

When BRAM resources are concerned, from Table 2 it can be observed that there is an almost exponential increase in required BRAMs to implement the ISM module as the ISB cache size is increased. This observation was the main motivation for developing an ISM module with the "IFM striping" capability.

From Table 2 it can be seen that there is no increase in required DSP blocks as the size of ISB cache is enlarged from 2048 to 131072 IFM points. This was expected, since DSP blocks are not used to build ISB cache, so their number should remain constant.

Finally, from Table 2, it can be seen that there is an increase in required LUT number as the size of ISB cache increases. However, the maximum LUT increase is no more than 25%, compared to the situation when the ISB cache size is 2048 IFM points.

The implementation of the ISB cache reduces latency access to memory. For hardware experiments, we used the Xilinx ZCU102 development board. On the board, the ISM implementation can work on 245 Mhz frequency. For that frequency, the latency from the ISM to the DDR memory is at least 34 cycles, and latency to the ISB cache is one cycle. Each time that the ISM access the data inside ISB, the latency is reduced from 34 cycles to 1.

The latency needed at the beginning of classification for filling of ISB cache is negligible for the large IFMs. Only one stick is required for the start of calculation, so the average time for memory access approaches one cycle.

REFERENCES

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

[2] A. Rodriguez, "Intel Processors for Deep Learning Training", November 2017. [Online]. Available: https://software.intel.com/en-us/articles/intel-processors-for-deep-learning-training

[3] D. Franklin, "NVIDIA Jetson TX2 Delivers Twice the Intelligence tothe Edge," march 2017. [Online]. Available: https://developer.nvidia.com/blog/jetson-tx2-delivers-twice-intelligence-edge/

[4] A. Frumusanu, "The Samsung Galaxy S9 and S9+ Review: Exynos and Snapdragon at 960fps," March 2018. [Online]. Available: https://www.anandtech.com/show/12520/the-galaxy-s9-review

[5] "Edge TPU," 2019. [Online]. Available: https://cloud.google.com/edge-tpu/

[6] J. Hruska, "New Movidius Myriad X VPU Packs a Custom NeuralCompute Engine," August 2017. [Online]. Available: https://www.extremetech.com/computing/254772-new-movidius-myriad-x-vpu-packs-custom-neural-compute-engine

[7] 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).

[8] E. Nurvitadhi, G.Venkatesh, J.Sim, D. Marr, R. Huang, J. G. H.Ong, Y.T.Liew, K.Srivatsan, D. Moss, S.Subhaschandra, and G. Boudoukh, 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), 2017.

[9] Y.Shen, M.Ferdman, and P. Milder, 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), 2017.

[10] 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, Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In Proceedings of the24thACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16), 2016.

[11] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, 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), 2015.

[12] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. L, 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), 2016.

[13] A.Azizimazreah,C. Lizhong, "Flexible On-chipMemory Architecture for DCNN Accelerators.", The First International Workshop on Architectures for Intelligent Machines (AIM 2017)

[14] M. Horowitz. Energy table for 45nm process, Stanford VLSI wiki. [Online]. Available:https://sites.google.com/site/seecproject

[15] D. Rakanovic, A.Erdeljan, V. Vranjkovic, B. Vukobratovic, P. Teodorovic, and R. Struharik, Reducing off-chip memory traffic in deep CNNs using stick buffer cache, In Proceedings of the 25th Telecommunication Forum (TELFOR), 2017.

[16] R. Struharik, B. Vukobratović, A. Erdeljan, and D. Rakanović, "CoNNA – Hardware accelerator for compressed convolutional neural networks", Microprocessors and Microsystems, vol. 73, March 2020, 102991.

[17] R. Struharik and V. Vranjkovic, "Stick Buffer Cache v2: Improved Input Feature Map Cache for Reducing off-chip Memory Traffic in CNN Accelerators," *2019 27th Telecommunications Forum (TELFOR)*, Belgrade, Serbia, 2019, pp. 1-4.