Circuits and Systems, 2017, 8, 149-170
http://www.scirp.org/journal/cs
ISSN Online: 2153-1293
ISSN Print: 2153-1285

A Multithreaded CGRA for Convolutional Neural Network Processing

Kota Ando*, Shinya Takamaeda-Yamazaki, Masayuki Ikebe, Tetsuya Asai, Masato Motomura

Hokkaido University, Sapporo, Japan
Email: *andou@lalsie.ist.hokudai.ac.jp

How to cite this paper: Ando, K., Takamaeda-Yamazaki, S., Ikebe, M., Asai, T. and Motomura, M. (2017) A Multithreaded CGRA for Convolutional Neural Network Processing. *Circuits and Systems*, **8**, 149-170.
https://doi.org/10.4236/cs.2017.86010

Received: April 26, 2017
Accepted: June 26, 2017
Published: June 29, 2017

Copyright © 2017 by authors and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

Abstract

Convolutional neural network (CNN) is an essential model to achieve high accuracy in various machine learning applications, such as image recognition and natural language processing. One of the important issues for CNN acceleration with high energy efficiency and processing performance is efficient data reuse by exploiting the inherent data locality. In this paper, we propose a novel CGRA (Coarse Grained Reconfigurable Array) architecture with time-domain multithreading for exploiting input data locality. The multithreading on each processing element enables the input data reusing through multiple computation periods. This paper presents the accelerator design performance analysis of the proposed architecture. We examine the structure of memory subsystems, as well as the architecture of the computing array, to supply required data with minimal performance overhead. We explore efficient architecture design alternatives based on the characteristics of modern CNN configurations. The evaluation results show that the available bandwidth of the external memory can be utilized efficiently when the output plane is wider (in earlier layers of many CNNs) while the input data locality can be utilized maximally when the number of output channel is larger (in later layers).

Keywords

CNN, Convolutional Neural Network, Deep Learning, Multithreaded Architecture, CGRA

1. Introduction

Convolutional neural networks (CNNs) are attracting much attention by achieving high accuracy in various applications such as image recognition, natural language processing, object detection. CNN is a type of neural networks which employs convolution operation as a feature extraction method, where the weights of the convolution are also self-obtained through the training of the network. This "trainable" feature extraction algorithm is the key of high recognition accuracy of the CNNs.

CNNs are computationally intensive, many kinds of hardware acceleration such as GPGPU computation or ASIC/FPGA-based implementations [1] [2] have been utilized to process CNNs with an acceptable throughput and efficiency. GPGPU parallel computation [3] is used for server-side off-line CNN training. The high performance and it programmable feature allows for software-based development, but its large energy consumption is not suitable for mobile applications. ASIC approach can be the most efficient solution with the optimal design for an application. However, the disadvantage of an ASIC solution is the difficulty with adapting to new methods of an ever-evolving deep learning algorithm. FPGA-based implementations are also used to balance the performance and availability with the optimized architectures for the demand, but its effective power efficiency is not so high.

The difference of computational structures between convolutional and fully-connected layers prevents the accelerators to process the entire CNN efficiently. In addition, the convolutional layers in a single CNN may have quite different memory (data transfer) requirements.

In this paper, we analyze the data access patterns of CNN layers and find out a reconfigurable architecture that has the ability of changing data access/operation patterns layer by layer in order to utilize the locality of the layers. We propose a novel CGRA (Coarse Grained Reconfigurable Array) accelerator with time-domain multithreading, which exploits the locality of the input data and conceals the latency of the external memory access. Finally, we evaluate the benefits of the multithreading by estimating the data reusability and the hardware requirements for our experimental architecture. As a result, the efficiency and usability of the multithreaded processing in CGRA are proven, and the practical and quantitative methods for building/using multithreaded CGRA neural network accelerators are shown.

This paper is based on our previous work [4]. The followings are the main supplementary contribution of this paper: we considered some practical architectures with feasible SRAMs, analyzed the performance and complexity of these configurations, and also indicated the practical design points of the presented architecture based on the application requirements.

The purpose of this paper is to propose a CNN accelerator architectue and design for embedded applications. This paper is organized as follows: in Section 2, we analyze the computation pattern of CNNs. In Section 3, we discuss the localities of the CNN layers and propose an ideal multithreaded architecture to exploit them. In Section 4, we consider two feasible architectures with multithreaded accumulators. And then in Section 5 we evaluate the performances, arithmetic intensities, and hardware requirements of the architectures on several situations. In Section 6, we present some related studies. Then in Section 7, and we finally conclude this paper.

2. Computation Model of CNN Layers

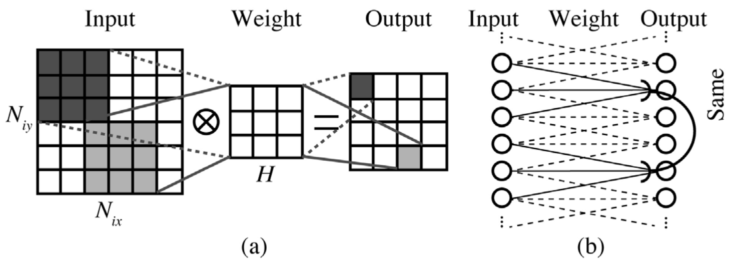
A CNN includes the layers operate 2-dimensional convolution between the inputs and the weights (kernels), called "convolutional" (CONV) layers, which behave as feature extraction that the pixels having the similar pattern to the weight pattern are amplified. The weights of the CNN are trained to have the reasonable patterns for the task, and this self-tuned feature extraction is the secret the CNNs succeed in the natural data recognition/classification. Typically a CNN might have the "fully-connected" (FC) layers after the CONV layers, and these layers operate vector transformation and work as classifier like the classical multi-layer perceptron.

In this section, we discuss the difference and the similarity of the computational structures and the data paths of the two types of layers.

First, we take a look at the computation of the CONV layer with single input/output channel.

An output pixel of the CONV layer is calculated using the weights shaped $H \times H$ and the input partial area which has the same shape, as shown in Figure 1(a). While the partial input area (sometimes called "window") slides over the entire input plane with stride, the weight values are common among all output elements. When we denote this 2-d convolution into the classical neural net style, each output neuron has a few connections to the input neurons of the particular location and all the output neurons has the same synapse weight values (Figure 1(b)).

The computation of the CONV layer consists of the same number of the multiplication and summation. Since all the output elements share the weights, each element of the output uses all of the weights. Then, the corresponding input partial area is uniquely determined by the location of the output element and the size of the weight matrix(H). Input pixels used by each output element are determined when we pick one weight element. Therefore, the convolution can be



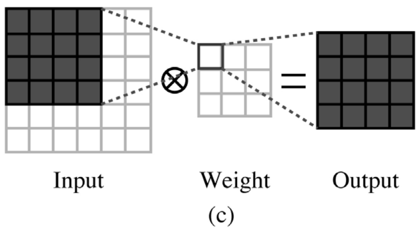


Figure 1. The structure of the convolutional layer (a) (b) and the data dependency (c).

computed by selecting a weight element one by one and multiplying the corresponding input area for all output elements. This one operation requires only one weight value and as many the input pixels as the output elements, as shown Figure 1(c).

From another view, as shown as black solid lines in Figure 2, an input element has inference to elements of all output channels. And, like broken lines in the figure, a weight element is shared among all output/input elements. In a CONV layer, both input and weight elements have forms of the data reusability.

The computation of an FC layer is denoted as the multiplication of the input vector and the weight matrix. The name “fully-connected” layer comes from that an output element is calculated using all of the input elements via the weights (Figure 3(b)). The value of an output element is the dot product of a row of the weight and the input vector, as Figure 3(a).

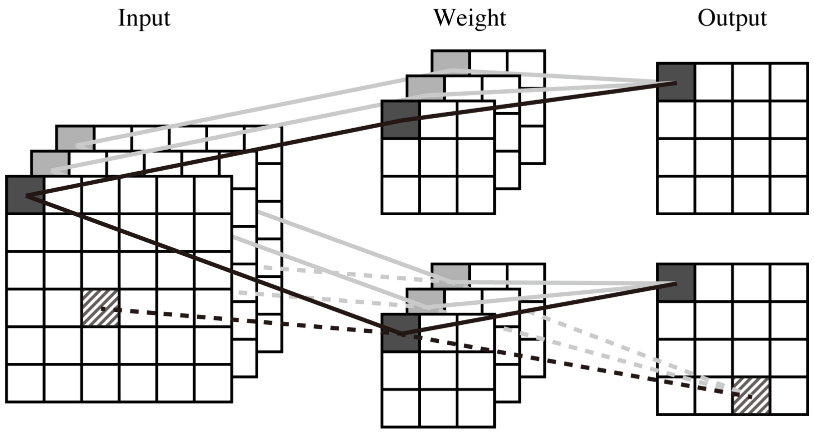


Figure 2. Data reusability of convolutional layers.

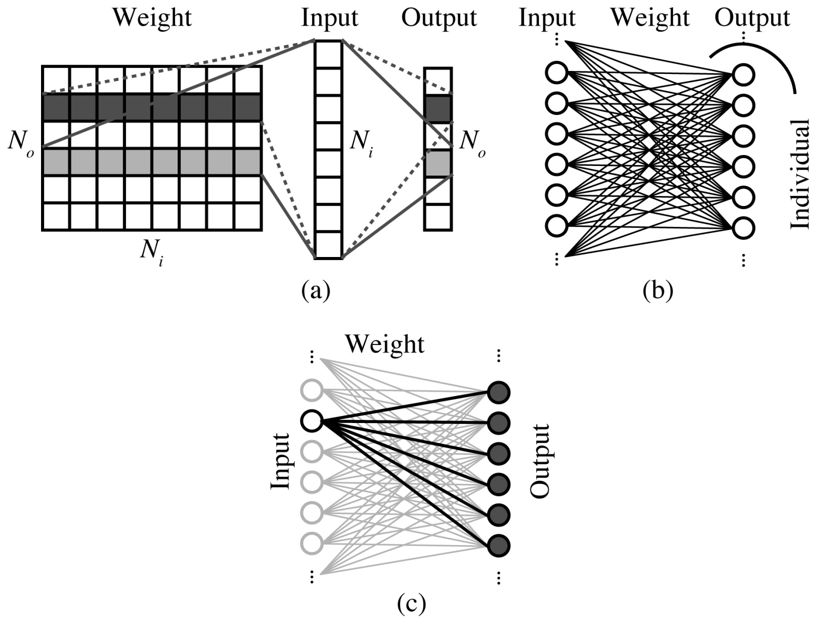


Figure 3. The structure of the fully-connected layer (a) (b) and the data dependency (c).

Similar to the CONV layers, the dominant operation of the FC layers is the multiplication and the addition. An output element is connected to all of the input element via their weight, as we discussed above, therefore an input element has the influence to all output elements. The computation of the FC layer can be done by picking an input element and the weights corresponding to the input and all output elements. One input element and as many weight values as output elements are needed in one operation cycle (Figure 3(c)).

Here, we look at the data reusability of an FC layer (Figure 4). An input element FC layer is used by all output elements, however, in contrast to CONV layers, weight elements have no chance to be reused. This means that, generally, the data size of the weights of an FC layer is larger than that of the inputs.

3. Multithreaded CGRA Architecture for Input Value Reuse

The discussion of the previous section shows that the dominant operation of the CONV and FC computation is commonly the multiply-add operation but data sizes of the operand are reversed. In this section, we propose the basic idea of the flexible architecture which does not need any difference of the hardware structure through the entire CNN computation.

3.1. Base Architecture

For the multiply-add (MULADD) operation, the main calculation of the CONV and FC layers, we construct simple multiply-accumulator-based array processor. This is an output-parallel system where a processor engine (PE) handles an output element. According to the above discussion, we need to distribute a common value to all PEs and individual data to each of PEs per one clock cycle.

In the CONV processing, we feed multiple input elements and a shared weight value into the PE array (Figure 5(a)). When the output plane is mapped to the array, the input partial area connected to the outputs through a weight is the same shape as the output plane.

The basic processing flow of the single-channel convolution is shown in Figure 6. Typically, the input and output have multiple channel (e.g. RGB color

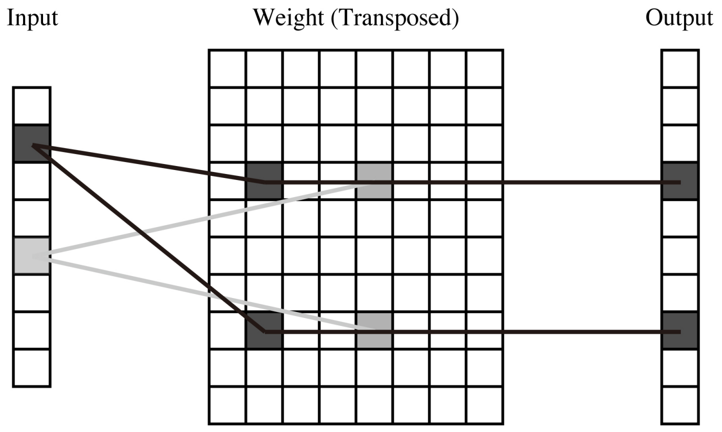


Figure 4. Data reusability of fully-connected layers.

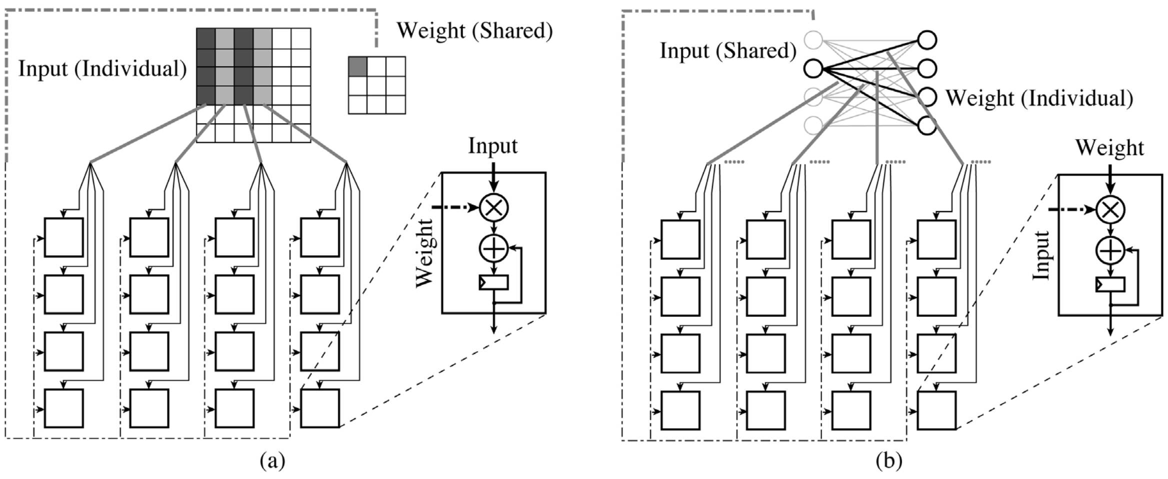


Figure 5. Data delivery of the convolutional (a) and fully-connected layers (b).

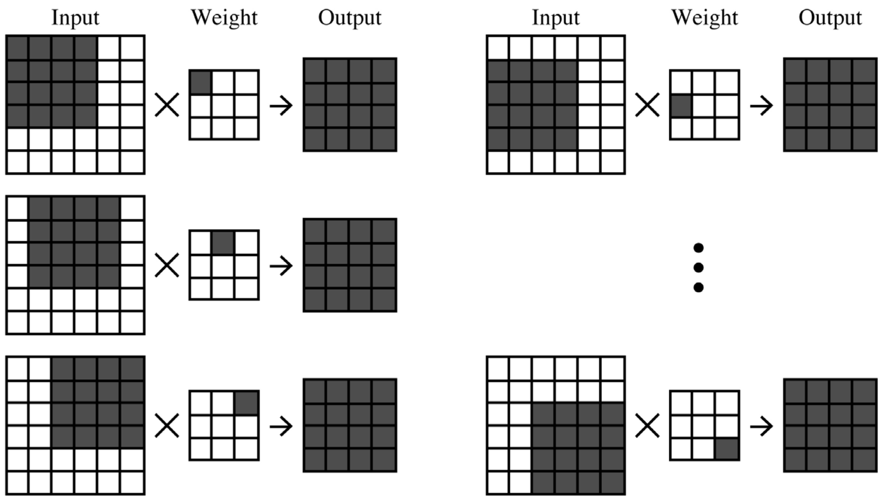


Figure 6. Basic procedure of a convolutional layer.

space for the input image or features for the hidden layers), convolution becomes multi-channel (that is, although the CONV layers have 2-d topology in the plane the channels of the input and the output are "fully-connected" so as to express the combination of the features), but let us think about the case the input and the output have only one channel here. Let the parallelism of the array $P_x \times P_y$, the size of the weight plane $H \times H$, the stride of the convolution s (in this example, $P_x = P_y = 4$, $H = 3$, $s = 1$.) The operation for one clock cycle is feeding one weight value into the shared input, cropping $P_x \times P_y$ partial area from the input and feeding it through the individual input. After $H \times H$ cycles (one operation per one weight), then the convolution is done. In this example, the processor array covers whole the output plane, but in the case the number of PEs is not sufficient the output plane divided into blocks and the array executes the time-division processing.

On the other hand, in the FC layer an input, not a weight, is used by multiple elements. An input element of the FC layer and the weight values for each PE are provided at one time (Figure 5(b)). The weight values to each output are one row of the weight matrix when an input neuron is picked, we distribute weights to all PEs very similarly to the CONV processing.

The basic computation for the FC layer is shown in Figure 7. The number of weights is $N_i \times N_o$, where N_i and N_o is the number of input and output elements, respectively. One PE calculates one output element. We feed an input element into the array via the common input in a clock cycle, while the weights for the input are fed through the individual input. The computation of the FC layer completes N_i cycles.

3.2. Multithreaded Accumulator for Data Reusing

Lower data transfer per operation is the better in point of the computational time and efficiency. There are two opportunities for reducing the data accesses.

First is the sequential locality of the input partial area of CONV layers. A pixel in an input channel is referenced $H \times H$ times, the operations of the H weights in each row are consecutive. As shown in Figure 8, partial inputs referenced by the next weight are the same except the last one column. In the same weight row, the input elements required by the PEs are the input values the adjoining PE loaded in the previous cycle. If these already-loaded elements are reused, the data requests decrease to at most $1/H$ (in the case there is no padding operations). Figure 8 indicates this idea, where the grayed input area is the pixels to be reused.

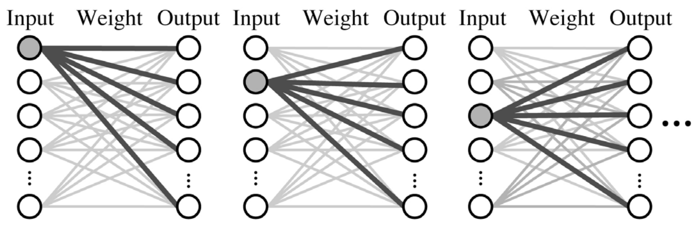


Figure 7. Basic procedure of a fully-connected layer.

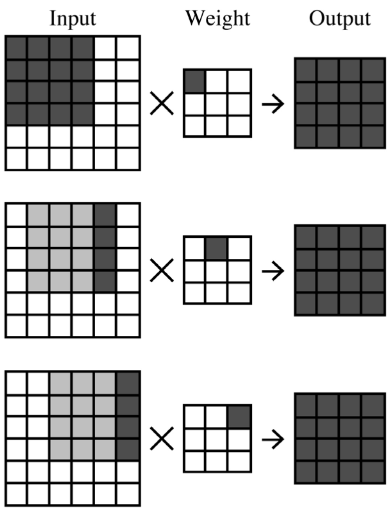


Figure 8. Sequential locality of the convolutional layer processing. The light-grayed elements can be reused.

Moreover, the temporal locality of the input of the multiple channel convolution is also observed. Let the number of input channels C_i, the number of the output channels C_o. One partial input area is used C_o times because all output channels (features) are computed from the combination of the all input channels. According to this, once the input elements are fetched for the computation of the first output channel, we can skip loading the same inputs for following output channels, as shown in Figure 9. When we utilize a T-word register file as accumulator in a PE, this data reusing enables the PE array to reduce data accesses at most $1/T$.

4. Row-Wise Data Feeding

We have explained the computation procedure of CONV and FC layers using an ideal processor array with individual and shared inputs, and the data reusability of CNNs on the array system. In this section, we build a feasible arrayed architecture with the multithreaded accumulator and the row-wise data delivery feature to exploit the locality the layers originally have.

4.1. Shift-Based Data Supply

Although a square partial input area is needed in every clock cycle in the CONV processing, the square area is not memory-friendly because it requires discontinuous memory access. In the ideal architecture we indicated in the previous

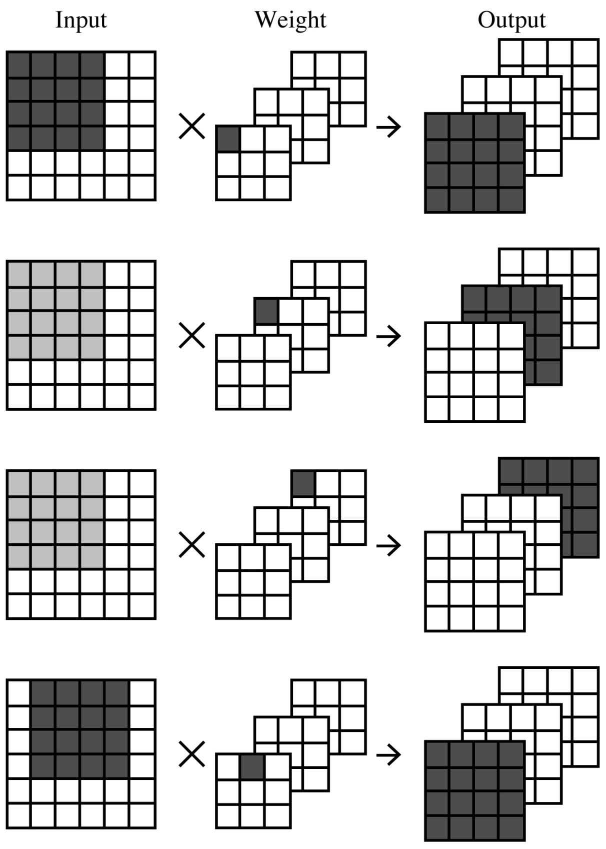


Figure 9. Temporal locality of the multi-channel convolution.

section, we assumed that the input partial area is fetched within a cycle to the PE array. This requires a very wide-band internal buffer. For a practical implementation, we propose a shift-based row-wise memory system, aiming simpler access pattern and a compact architecture design.

Since memory address of the CONV operations with output-parallel spatial array is continuous in the weight row, row-buffers where each buffer retains an input row can be appropriate (Figure 10). With admitting the latency to fill PEs, this idea is light-weighted since only the PEs on the most left column must have the "individual inputs" and the other PEs acquire the input data from their left-side PEs via the "forwarding bus". Note that the overhead of filling all PEs sequentially through the forwarding bus will be discussed later in Section 4.2.

The row-wise buffers are not sufficient because the CONV operations require H input rows per PE and $(P_y + H - 1)$ rows in the entire PE array (P_y rows (colored input values in Figure 10) are used for the first weight row, and the additional $H - 1$ rows (white input cells in Figure 10) are loaded for the remaining weight rows). Therefore we need a mechanism for selecting which row buffer should be connected to the left column. There are two solutions to this, using multi-port or multi-bank memories.

Figure 11(a) and Figure 12 show a system using the P_y-W P_y-R multi-port SRAMs, where the inputs can be read from any row. Compared to the basic idea (Figure 5), the "individual" input and output buses moved from PEs to row buffers. This is the solution with the most few internal memory accesses that each input row is loaded to buffer only once and is read by all PE rows using them (at most H rows) (Figure 13(a)), however the multi-port SRAMs cost $O(P_y^2)$ area.

Figure 11(a) and Figure 12 indicate a solution with the $(P_y + H - 1)$ rows of the 1W2R multi-bank SRAMs and the looped-back forwarding bus connected to the next row SRAMs. The input of the row buffers are connected to the external memory and to the forwarding output of the right column of the adjoining row (Figure 13(b)). This costs only $O(P_y)$ resource and remains the minimal external memory access, but the internal buffer write accesses increase compared to the multi-port solution, since the same data as the adjoining row used are

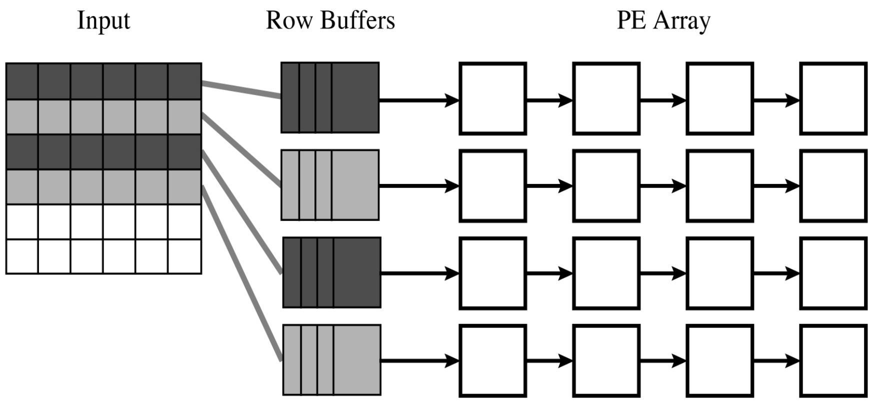


Figure 10. Inpout data mapping on the shift-based array.

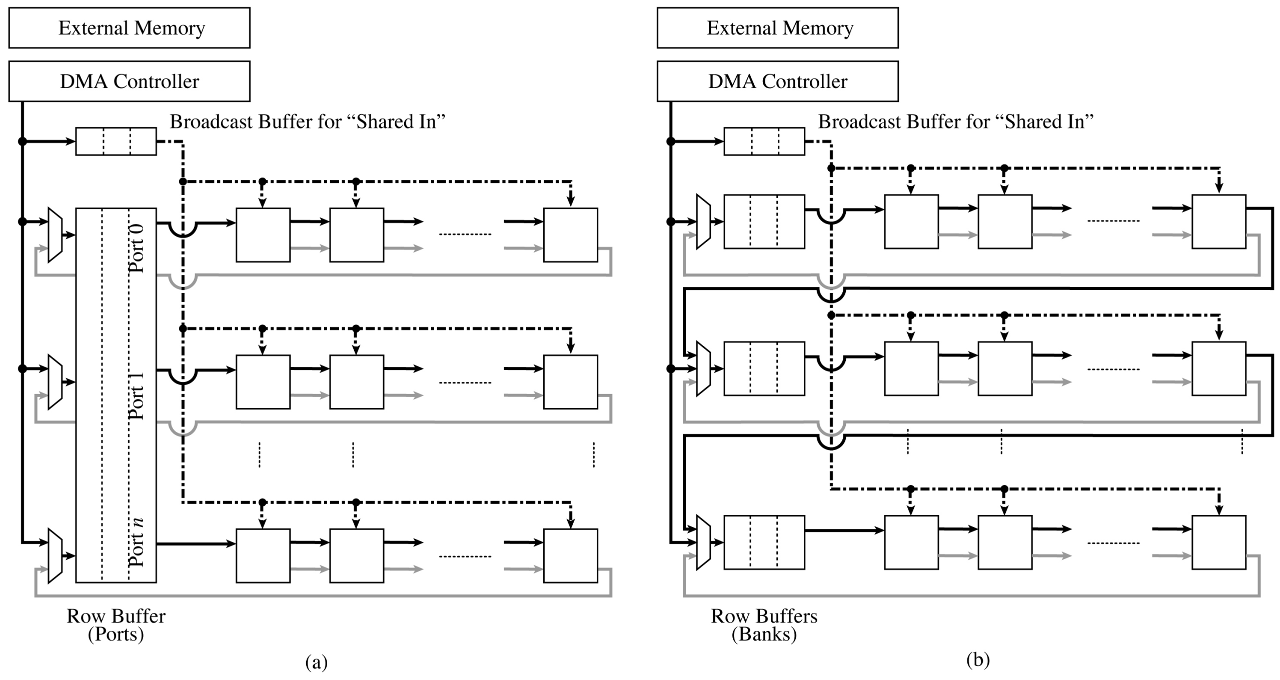


Figure 11. Shift-based architectures with (a) "multi-port" and (b) "multi-bank"' row buffers.

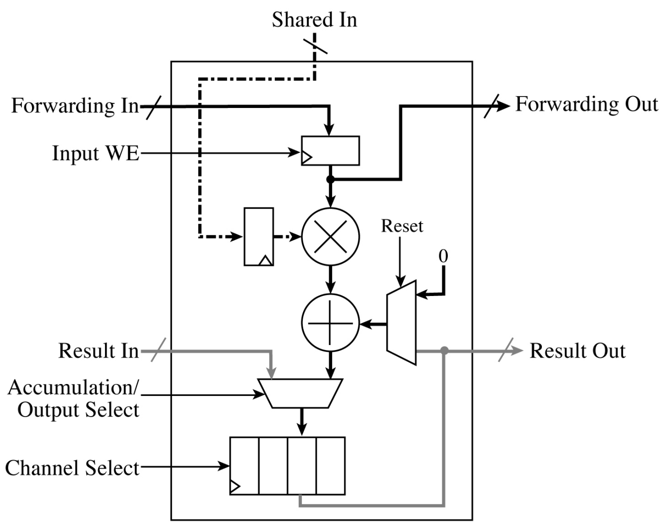


Figure 12. A PE for shift-based array (Figure 11(a) and Figure 11(b)).

written to the row buffers again from the forwarding output of the most right column.

4.2. Deep-Multithreaded Accumulator with SRAM

When we employ the row-wise buffers and the shift-based data delivery instead of the ideal fully-parallel memory system, the overhead for filling all PE columns occurs prior to the multiplication of the first element of the weight row. To overcome this, the time-domain multithreaded partial sum accumulation could be effective. Generally, many CNN implementations contain many-channel and

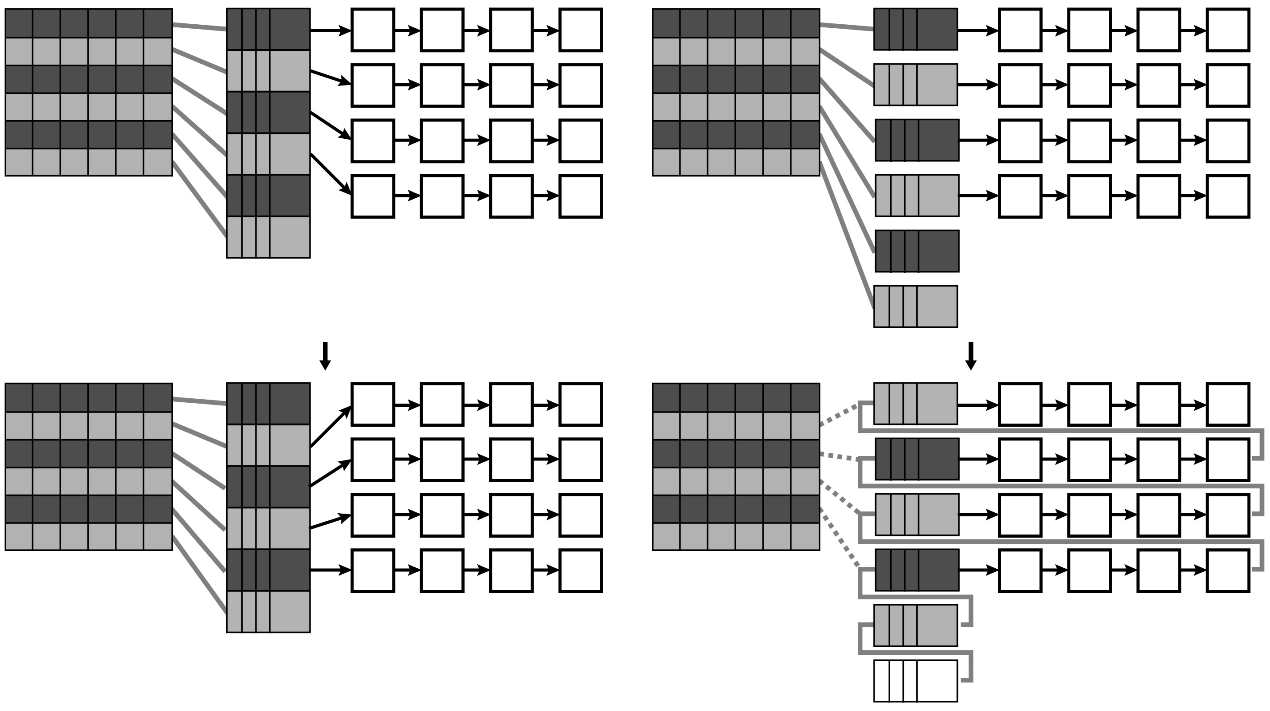


Figure 13. Data mapping and row selecting on (a) the "multi-port" and (b) the "multi-bank" systems.

small-plane CONV layers in deeper part of them, the overhead for sequential transfer will be relatively small, because an input connected to more output channels could be used more times and the data load is reduced by multithreading.

Now, we try to extend the multi-channel accumulator from a register to an SRAM buffer. An SRAM can have more channels than a register array; this helps us to gain the number of operations per row (*i.e.* arithmetic intensity). Though SRAM accumulators require more area, it is feasible because the most layers of a CNN have smaller plane and more channels, so the number of PEs and threads (accumulators) could be actually less and more respectively.

5. Evaluation

5.1. Setup

We evaluate the performance, the arithmetic intensity of the proposed architecture. Then the complexity of SRAM types for row selection is discussed.

In this section, the input plane size is $N_{ix} \times N_{iy} \times C_i$, the output plane is $N_{ox} \times N_{oy} \times C_o$, the kernel size is $H \times H$ ($H^2 C_i C_o$ elements for all input/output channel), the number of PEs is $P_x \times P_y$, and the number of thread is T. The system operates at clock frequency f and the data bit precision is b.

5.2. Performance

We evaluated the processing time and data rate of the architecture (Table 1).

In the CONV processing, we pick all weights from all input/output channels

Table 1. Processing time [cycles] and data rate [bps] of the architecture. Bit precision *b*, clock frequency *f*, array $\left(P_x \times P_y\right)$-parallel, accumulator threads *T*, weight size $H \times H$.

	CONV	FC
Pre-requirements	Out $N_{ox}N_{oy}C_o$, In $N_{ix}N_{iy}C_i$	Out N_o, In N_i
Processing Time	$\left(P_x+H-1\right)HC_oC_i\left\lceil\frac{N_{ox}}{P_x}\right\rceil\left\lceil\frac{N_{oy}}{P_y}\right\rceil$	$N_iP_x\left\lceil\frac{N_o}{P_xP_y}\right\rceil$
Input Rate	$bf\frac{\left(P_y+H-1\right)N_{ix}\left\lceil\frac{C_o}{T}\right\rceil}{\left(P_x+H-1\right)HC_o\left\lceil\frac{N_{ox}}{P_x}\right\rceil}$	$bf\frac{\left(1+P_xP_y\right)}{P_x}$
Output Rate	$bf\frac{N_{ox}N_{oy}}{\left(P_x+H-1\right)HC_i\left\lceil\frac{N_{ox}}{P_x}\right\rceil\left\lceil\frac{N_{oy}}{P_y}\right\rceil}$	bfP_y/N_i

one by one with each PE processing an output element ($\left(P_x+H-1\right)HC_oC_i$ cycles; it takes P_x cycles per row for filling the array and $\left(H-1\right)$ cycles for loading all the weight columns). Time-division processing is taken if the output size exceeds the number of PEs ($\left\lceil\frac{N_{ox}}{P_x}\right\rceil\left\lceil\frac{N_{oy}}{P_y}\right\rceil$ cycles). The processing time is:

$$\left(P_x+H-1\right)HC_oC_i\left\lceil\frac{N_{ox}}{P_x}\right\rceil\left\lceil\frac{N_{oy}}{P_y}\right\rceil \quad [\text{Cycles}] \tag{1}$$

The data transfer is buffered so as to fetch all pixels required by a row-wise processing block $\left(\left(P_y+H-1\right)N_{ix}\right)$, the input data size is:

$$\left(P_y+H-1\right)N_{ix}\left\lceil\frac{N_{oy}}{P_y}\right\rceil\left\lceil\frac{C_o}{T}\right\rceil C_i \quad [\text{Data}] \tag{2}$$

The input data rate is denoted as:

$$bf\frac{\left(P_y+H-1\right)N_{ix}\left\lceil\frac{C_o}{T}\right\rceil}{\left(P_x+H-1\right)HC_o\left\lceil\frac{N_{ox}}{P_x}\right\rceil} \quad [\text{bps}] \tag{3}$$

The output data size and rates are:

$$N_{ox}N_{oy}C_o \quad [\text{Data}] \tag{4}$$

$$bf\frac{N_{ox}N_{oy}}{\left(P_x+H-1\right)HC_i\left\lceil\frac{N_{ox}}{P_x}\right\rceil\left\lceil\frac{N_{oy}}{P_y}\right\rceil} \quad [\text{bps}] \tag{5}$$

In the FC processing, it takes

$$N_iP_x\left\lceil\frac{N_o}{P_xP_y}\right\rceil \quad [\text{Cycles}] \tag{6}$$

(It takes P_x cycles per an input element for weight preloading, and $\left\lceil \frac{N_o}{P_xP_y} \right\rceil$ is the number of time-division processing blocks), requiring

$$N_i\left(P_xP_y+1\right)\left\lceil \frac{N_o}{P_xP_y} \right\rceil \quad [\text{Data}] \tag{7}$$

inputs (while $\left(P_xP_y\right)$ weights are needed for each of N_i input elements) and

$$P_xP_y\left\lceil \frac{N_o}{P_xP_y} \right\rceil \quad [\text{Data}]$$

outputs. The input and output data rates are:

$$bf\frac{\left(1+P_xP_y\right)}{P_x} \quad [\text{bps}] \tag{8}$$

$$bfP_y/N_i \quad [\text{bps}] \tag{9}$$

We evaluated the theoretical processing time and data rates of the proposed ("multi-port" and "multi-bank") architecture with AlexNet [5], as shown in Table 2. The number of PEs is 256 ($P_x = P_y = 16$), the bitwidth is $b = 16$. The table shows that this system can process AlexNet in 380 msec (2.6 fps) in $f = 200\ \text{MHz}$.

According to these results, the required data rate in CONV layers is 16 MB/s at most. For the embedded systems this is an acceptable result.

"Util." means "PE Utilization", the ratio of the averaged number of operating PEs to the number of PEs through the layer processing. Total PE utilization is calculated as:

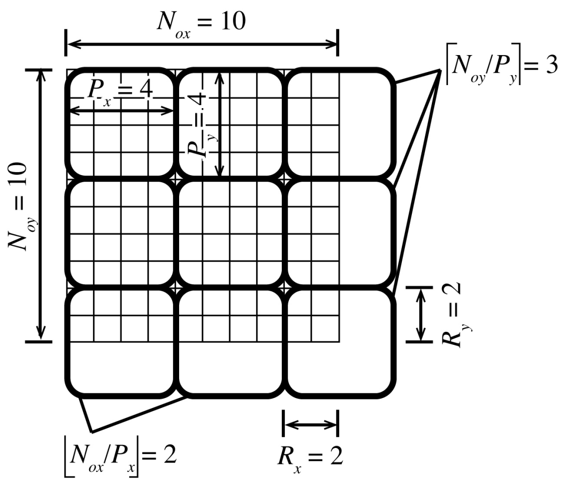
$$\begin{aligned}&(\text{Space Ratio})\times(\text{Time Ratio})\\ &=\frac{P_xP_y}{P_xP_y}\frac{\left\lfloor \frac{N_{ox}}{P_x}\right\rfloor\left\lfloor \frac{N_{oy}}{P_y}\right\rfloor}{\left\lceil \frac{N_{ox}}{P_x}\right\rceil\left\lceil \frac{N_{oy}}{P_y}\right\rceil}+\frac{R_xP_y}{P_xP_y}\frac{\left(\left\lceil \frac{N_{ox}}{P_x}\right\rceil-\left\lfloor \frac{N_{ox}}{P_x}\right\rfloor\right)\left\lfloor \frac{N_{oy}}{P_y}\right\rfloor}{\left\lceil \frac{N_{ox}}{P_x}\right\rceil\left\lceil \frac{N_{oy}}{P_y}\right\rceil}+\frac{R_xR_y}{P_xP_y}\frac{\left(\left\lceil \frac{N_{ox}}{P_x}\right\rceil-\left\lfloor \frac{N_{oy}}{P_y}\right\rfloor\right)\left(\left\lceil \frac{N_{oy}}{P_y}\right\rceil-\left\lfloor \frac{N_{oy}}{P_y}\right\rfloor\right)}{\left\lceil \frac{N_{ox}}{P_x}\right\rceil\left\lceil \frac{N_{oy}}{P_y}\right\rceil}\end{aligned} \tag{10}$$

Table 2. Evaluation of the proposed architecture using AlexNet [5].

#	Type	Weight Shape	Output Shape	Cycles	Rate [GB/s]	Util. [%]
1	CONV	$(11^2\times3\times48)\times2$	$(224\times224\times48)\times2$	16,144,128	0.013	100.0
2	CONV	$(5^2\times48\times128)\times2$	$(27\times27\times128)\times2$	19,660,800	0.009	73.9
3	CONV	$3^2\times(128\times2)\times(192\times2)$	$(13\times13\times192)\times2$	21,233,664	0.005	71.2
4	CONV	$(3^2\times192\times192)\times2$	$(13\times13\times192)\times2$	3,981,312	0.010	66.0
5	CONV	$(3^2\times192\times128)\times2$	$(13\times13\times128)\times2$	2,654,208	0.016	66.0
6	FC	43264×4096	4096	11,075,584	6.425	100.0
7	FC	4096×4096	4096	1,048,576	6.425	100.0
8	FC	4096×1000	1000	262,144	6.425	97.7
Total				76,060,416		

where the remainder pixels of the time-division blocks are $R_x = N_{ox} - P_x \lfloor N_{ox}/P_x \rfloor$ and $R_y = N_{oy} - P_y \lfloor N_{oy}/P_y \rfloor$, as illustrated in Figure 14. If the width/height of the output plane can be divided by the width/height of the PE array, all the PEs are utilized through all processing time of the layer, the utilization ratio is 100%. Therefore, if we design a CNN to run on the CGRA, we should set the number of output elements to a multiple of that of PEs in order to maximize the computational efficiency.

Figure 15 and Figure 16 show the relation between the problem size and processing time/resource, where the number of processors is $P_x = P_y = 16$. The processing time is in proportion to the number of output channels and to the output plane size, while required data rate is almost independent from the problem size. When the problem becomes larger the processing time increases



$N_{ox} = 10$
$\lceil N_{oy}/P_y \rceil = 3$
$P_x = 4$
$P_y = 4$
$N_{oy} = 10$
$R_y = 2$
$\lfloor N_{ox}/P_x \rfloor = 2$
$R_x = 2$

Figure 14. Example of dividing the output plane into the processing blocks. In the case of $N_{ox} = N_{oy} = 10$ and $P_x = P_y = 4$, the remainder pixels are $R_x = R_y = 2$, the number of fully utilized blocks is $\lfloor N_{ox}/P_x \rfloor \lfloor N_{oy}/P_y \rfloor$, total blocks are $\lceil N_{ox}/P_x \rceil \lceil N_{oy}/P_y \rceil$.

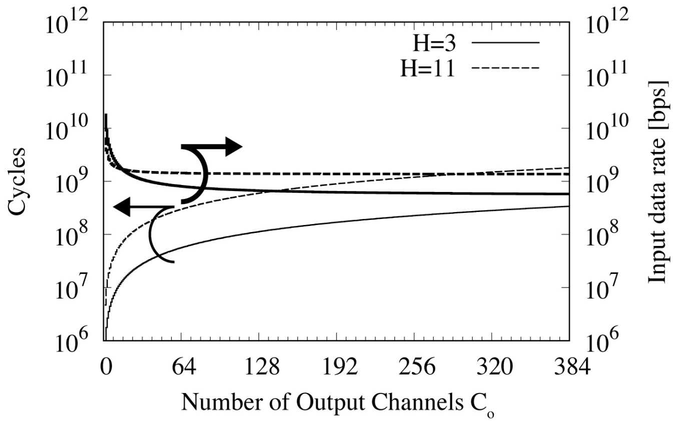


Figure 15. Processing time and output bitrate over number of output channels. ($C_i = 256, N_{ox} = N_{oy} = 128$).

but required resources such as data rate are independent, the availability is retained over the problem size.

Figure 17 shows the processing time versus the size of the input in CONV processing. There is no difference between the processing times of the "multi-port" and the "multi-bank" row-wise SRAM systems because the processing method that the row SRAMs feed the input data into each row sequentially is common among these two ideas. Compared to the "ideal" system (all data used in the PE array are provided at once, we described in Section 3), row-buffer system take more processing time, however the arithmetic intensities of these systems are the same because the required numbers of operations and input/weight data are not different (Figure 18). And more, as Figure 19, the required data input rate for row-wise system is lower than the base idea and is constant for the output size (if the output size becomes larger, both the processing time and required data increases, it results in the almost same data rate).

5.3. Arithmetic Intensity

Arithmetic intensities of some situations are indicated in Figure 20. The broken

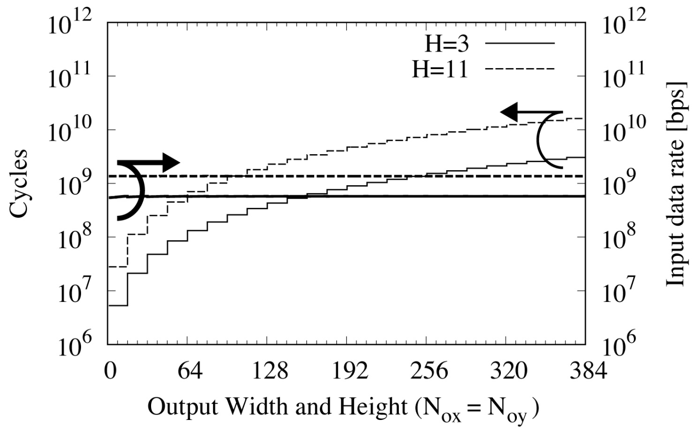


Figure 16. Processing time and output bitrate over numbers of output rows and columns ($C_i = 256, C_o = 384$).

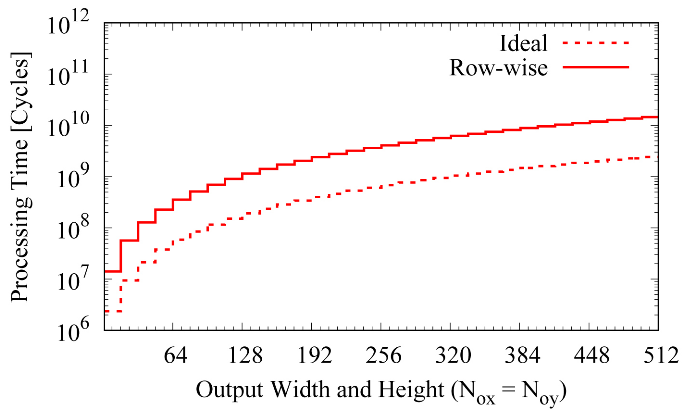


Figure 17. Output size and processing time.
$H = 3, C_i = 512, C_o = 512, P_x = P_y = 16, T = 512$.

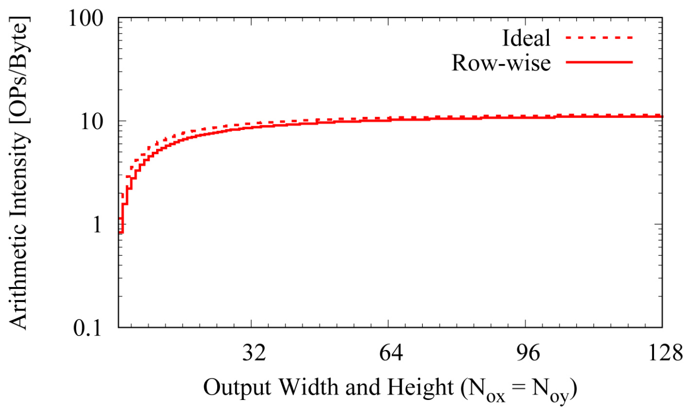


Figure 18. Output size and arithmetic intensity. $H = 3, C_i = 512, C_o = 512, P_x = P_y = 16, T = 512$.

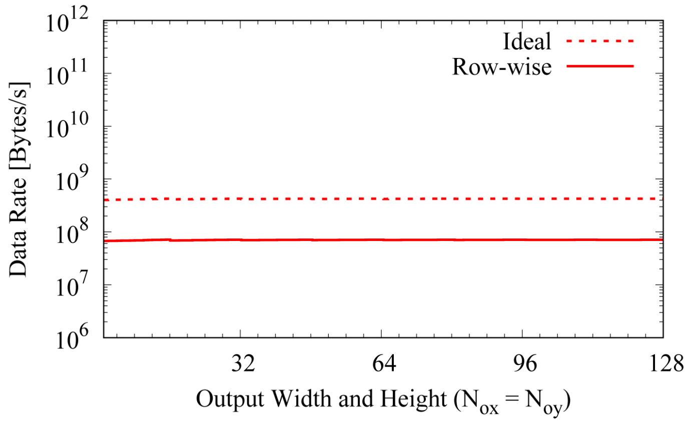


Figure 19. Output size and input data rate. $H = 3, C_i = 512, C_o = 512, P_x = P_y = 16, T = 512$.

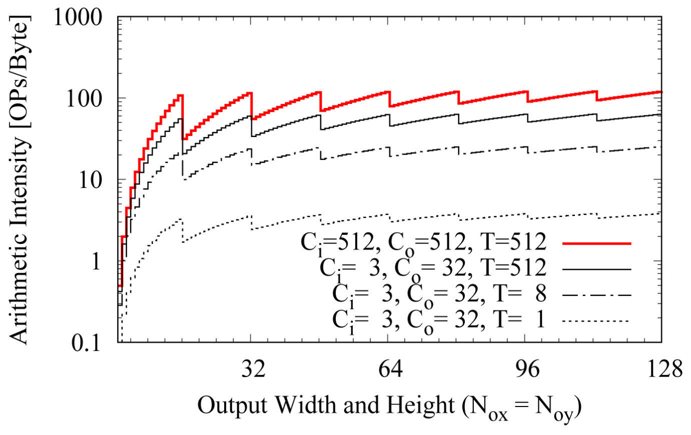


Figure 20. Output size and arithmetic intensity in some situations. $P_x = P_y = 16, H = 3$.

line is for 1-thread (*i.e.* non-multithreaded register) accumulator, the dotted line for 8-thread, and the solid lines are for 512-thread memory. The red line is in $C_i = C_o = 512$ case, assuming the deeper layer of a CNN, while the black ones are $C_i = 3, C_o = 32$ assuming the input layer. This tells us that, highly-multithreaded system works better especially in the layers with more output channels (*i.e.* the deeper layers) of CNNs, since the multithreaded computation reduces the data load from the external RAMs by exploiting the input data reusability among the output channels. If the problem size is bigger than the spatial array size, the time-divided processing is needed, but the arithmetic intensity is saturated since the required computation and data transfer per output element do not dramatically vary.

5.4. Complexity

We introduced two types of the row-wise SRAM buffers with "multi-port" and "multi-bank" data selection in Sections 4.1, and estimated the performance of the row-wise shift architectures in Section 5.3. Let us discuss the complexity of the ways of data selecting, "multi-port" and "multi-bank" SRAMs.

As described in Section 4.1, a PE row requires H input rows. To select which row buffer to connected to a PE row, two types of the buffer system have been proposed; "multi-port" SRAM that any PE row can have connection to any row buffer, and "multi-bank" one that row selection can be done by moving data internally.

For "multi-port" SRAM, we utilize a $\left(P_y + H - 1\right)$-read/write SRAM as the row buffer. The input data for an input row are stored only once and are read by at most H PE rows (*i.e.* H output rows), so the number of buffer write accesses is minimized (once per input element). However, an n-read SRAM costs hardware resource by $O\left(n^2\right)$, generally. For row buffers, the capacity (costs linearly) and the number of ports (quadratically) are both in $O\left(P_y\right)$, so this costs $O\left(P_y^3\right)$ area.

In the "multi-bank" system, on the other hand, we use a $\left(P + H\right)$ sets (or "banks") of simple 1-write 1-read SRAM. This costs $O\left(P_y\right)$ resource for $O\left(P_y\right)$ rows. After finishing loading input data for a CONV process, every clock cycle the data are fed into each row via the read port of SRAM, while the write port of SRAM acquires the data overflowed from the forwarding output of the right PE on the previous row. Therefore, although the input data are transferred from the external memory to the system only once, just like the "multi-port" implementation, the internal memory write is taken for at most H times per input element as the loaded data flows over the array.

Table 3 shows the area [mm^2] and energy [nJ] specification of the "multi-port" and "multi-bank" 16-bit SRAMs (512 words per row) calculated using CACTI simulation [6] [7]. We assume that 2 more ports/banks than P_y are needed for interleaved read access during the multithreaded processing in the array in this simulation. The energy is the simulated value of a CONV operation of $H = 3, C_i = C_o = 1$, including the internal data moving. We supposed the power consumption of the write accesses of the SRAM is equal to the read accesses [8]. The actual capacity and number of the ports/banks should be determined considering the array size (P_x, P_y), the throuphput/latency of the

Table 3. Comparison of "multi-port" and "multi-bank" implementations.

P_y	Multi-port			Multi-bank		
	#Ports	Area	Energy	#Banks	Area	Energy
2	4	0.694	0.706	4	0.085	0.103
4	8	4.726	5.041	6	0.127	0.279
6	8	4.726	9.410	8	0.169	0.535
14	16	41.332	105.697	16	0.338	2.387

external memory system, and the target application (mainly H). Seeing the result of this calculation, although the "multi-bank" memories require more write accesses, the energy and area are still lower than "multi-port" memories.

5.5. Discussion

In CONV processing, the array requires P_y input data at the first output channel in the multithreaded accumulators, therefore the input data rate requirement of the array (internal memory bandwidth) is $P_y/T\,[\text{Words/Cycle}]$. Note that the data rate may be $P_y/C_o\,[\text{Words/Cycle}]$ if the number of output channels is less than that of accumulator threads. And more, if the read interval T or C_o is greater than P_xP_y, the number of input values of the whole array, even a 1W1R buffer with b-bit width and the shift registers for whole the PE array can be used.

The P_xP_yT output values of the CONV processing become valid every H^2C_iT cycles, the required output data rate is $\frac{P_xP_y}{H^2C_i}[\text{Words/Cycle}]$. For the weights, every clock cycle the array multiply-accumulates the inputs with one weight value, so weight data rate is $1[\text{Words/Cycle}]$ constantly. This means that higher input bandwidth is required when the output channels or the threads are fewer, and that the output rate requirement gets severe if the input channels are fewer or the filter kernels are smaller.

Then we discuss the optimal configuration of the internal SRAMs. To get over the latency of loading input data from the external memories, the processing time must exceed the memory access time, we need row SRAMs of enough depth that the solid partial input area can be read from the external memory behind the multithreaded computation. At this point, if the number of threads is large, the SRAM buffer size should be at least $2P_x$ words per row (needs "2" for independent read/write accesses). The data rate of the internal read access is $P_y/T\,[\text{Words/Cycle}]$, as described above, required data rate of the external bus is also $P_y/T\,[\text{Words/Cycle}]$. When this data rate requirement is met, the time for the data acquirement can be covered by the processing time of the array.

The area of the PE array is in proportion to the number of multithreading accumulators T, while the external bandwidth or the area of the row buffers is in proportion to the array height P_y and in inverse proportion to T. The processing time is in inverse ratio to P_xP_y and direct to T. For planning the practical CGRA system, these observation should be in consideration on demand from the applications.

6. Related Work

AlexNet [5] proved the high accuracy of the modern CNNs on image recognition in 2012. This firstly appeared in ILSVRC [9], the competition for the accuracy and adaptability of neural networks on image recognition; the competition has been held every year, many CNNs are invented.

Various kinds of neural network accelerators have been developed. Zhang *et al.* analysed the data reusability and the arithmetic intensity of input- and output-parallel architectures built on an FPGA, and provided a method to optimize it when a target application is given [2]. Tanomoto *et al.* implemented EMAX [10], a CNN on the CGRA with multiple local memory banks; the main difference from our work is the data stationary. FPGA implementations of CNN coprocessors using DSP blocks were also proposed [11] [12]. LUT-based FPGA implementation [13] was proposed; this transforms the weight values using a mathematical techniques to replace the multipliers with the LUT and the adders. Chakradhar *et al.* proposed mapping a CNN into a coprocessor with the dynamic reconfiguration techniques [14].

For an example of purpose-built architecture, MIT Eyeriss [1] is a CNN-targeted ASIC system with spatial processor array, which employs an NoC multicasting and a hierarchical memory system aiming at maximizing the PE utilization and exploiting the locality of the input data. Shindo *et al.* model general multicore neural network accelerators with local memories, and discuss the efficient way to map neural networks onto them [15]. KU Leuven's accelerator [16] is a SIMD array system with dynamic voltage and bit precision control, aiming for low-power mobile applications. The accelerator proposed by KAIST [17] is a CNN accelerator which employs principal component analysis for the weights of convolutional layers to minimize the data size read from external memory. ShiDianNao [18] and its previous work DaDianNao [19] also retain the weight values in the internal buffers and employ spatial-mapped neural function unit. On the whole, including our work, the trend of the CNN accelerators is reducing external memory access and maximally utilizing the locality of the data for low-power embedded applications.

Some researches are trying to reduce the data transference and the multiply operations. Recently "binarized" neural networks were proposed [20] [21], where a weight value is only 1-bit, which replace the multiplications with simple bit-wise operations. Exploiting redundancy of the neural networks, the computation costs are drastically reduced while keeping the recognition/classification accuracy. The hardware binary neural network accelerators with high-throughput massively-paralleled processors have been developed [22] [23].

Commercial systems for neural network processing have been working. Google [24] and Microsoft [25] developed their own hardware/FPGA neural network processors and they are using them for their commercial services. NVIDIA's GPUs have become a very popular way to train/examine CNNs on the server with official libraries [3] both in academic/busuiness purposes.

7. Conclusion

We proposed a CGRA architecture with time-domain multithreading for exploiting input data locality. This architecture employs the multithreaded PE accumulators for obtaining multiple in-flight operations and the row-wise data feeding for reusing the input of the neighboring PEs. Our evaluation proved that the proposed architecture is suitable for both deeper and shallower layers in general CNNs. The multithreading in the PEs improves the arithmetic intensity by an order of magnitude. We also showed the quantitative way to determine the specification of the architecture on demand from the applications. It indicated that the architecture requires only a little bandwidth of the external memory, this feature is useful for embedded machine learning applications. Though we showed the advantage of the architecture through the evaluation of the performance, the energy efficiency evaluation is desired to develop a more efficient architecture. In order to realize more useful accelerators, we should consider the co-design of the CNN algorithms and the architectures.

Acknowledgements

This work is supported in part by the JST ACCEL program. We thank the editor and the anonymous reviewers for their useful comments.

References

[1] Chen, Y.H., Krishna, T., Emer, J. and Sze, V. (2016) 14.5 Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. 2016 *IEEE International Solid-State Circuits Conference* (*ISSCC*), San Francisco, 31 January-4 February 2016, 262-263. https://doi.org/10.1109/ISSCC.2016.7418007

[2] Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B. and Cong, J. (2015) Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks. *Proceedings of the* 2015 *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays in FPGA*'15, Monterey, 22-24 February 2015, 161-170. https://doi.org/10.1145/2684746.2689060

[3] NVIDIA. NVIDIA CUDNN—GPU Accelerated Deep Learning. https://developer.nvidia.com/cudnn

[4] Ando, K., Orimo, K., Ueyoshi, K., Ikebe, M., Asai, T. and Motomura, M. (2016) Reconfigurable Processor Array Architecture for Deep Convolutional Neural Cetworks. *The* 20*th Workshop on Synthesis and System Integration of Mixed Information Technologies* (*SASIMI*), Kyoto, 24-25 October 2016, R3-7.

[5] Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012) Imagenet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 1097-1105.

[6] HP Labs: CACTI. Hewlett-Packard Development Company, L.P. http://www.hpl.hp.com/research/cacti/

[7] CACTI 5.3. Hewlett-Packard Development Company, L.P. http://quid.hpl.hp.com:9081/cacti/sram.y

[8] Noguchi, H., Nomura, K., Abe, K., Fujita, S., Arima, E., Kim, K., Nakada, T., Miwa,

S. and Nakamura, H. (2013) D-MRAM Cache: Enhancing Energy Efficiency with 3T-1MTJ DRAM/MRAM Hybrid Memory. 2013 *Design*, *Automation Test in Europe Conference Exhibition* (*DATE*), 1813-1818. https://doi.org/10.7873/DATE.2013.363

[9] ImageNet. Stanford Vision Lab, Stanford University, Princeton University. http://www.image-net.org

[10] Tanomoto, M., Takamaeda-Yamazaki, S., Yao, J. and Nakashima, Y. (2015) A CGRA-Based Approach for Accelerating Convolutional Neural Networks. 2015 *IEEE* 9*th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip*, Turin, 23-25 September 2015, 73-80.

[11] Farabet, C., Poulet, C., Han, J.Y., LeCun, Y. (2009) CNP: An FPGA-Based Processor for Convolutional Networks. 2009 *International Conference on Field Programmable Logic and Applications*, Prague, 31 August-2 September 2009, 32-37. https://doi.org/10.1109/FPL.2009.5272559

[12] Rahman, A., Lee, J. and Choi, K. (2016) Efficient FPGA Acceleration of Convolutional Neural Networks Using Logical-3D Compute Array. 2016 *Design*, *Automation Test in Europe Conference Exhibition* (*DATE*), Dresden, 14-18 March 2016, 1393-1398. https://doi.org/10.3850/9783981537079_0833

[13] Nakahara, H. and Sasao, T. (2015) A Deep Convolutional Neural Network Based on Nested Residue Number System. 2015 25*th International Conference on Field Programmable Logic and Applications* (*FPL*), Lausanne, 2-4 September 2015, 1-6. https://doi.org/10.1109/FPL.2015.7293933

[14] Chakradhar, S., Sankaradas, M., Jakkula, V. and Cadambi, S. (2010) A Dynamically Configurable Coprocessor for Convolutional Neural Networks. *Proceedings of the* 37*th Annual International Symposium on Computer Architecture in ISCA*'10, Saint-Malo, 19-23 June 2010, 247-257. https://doi.org/10.1145/1815961.1815993

[15] Shindo, S., Ohba, M., Tsumura, T. and Miwa, S. (2016) Evaluation of Task Mapping on Multicore Neural Network Accelerators. 2016 *Fourth International Symposium on Computing and Networking* (*CANDAR*), Hiroshima, 22-25 November 2016, 415-421. https://doi.org/10.1109/CANDAR.2016.0078

[16] Moons, B. and Verhelst, M. (2016) A 0.3-2.6 TOPS/W Precision-Scalable Processor for Real-Time Large-Scale ConvNets. 2016 *Symposium on VLSI Circuits Digest of Technical Papers* (*VLSI*), 178-179.

[17] Sim, J., Park, J.-S., Kim, M., Bae, D., Choi, Y. and Kim, L.-S. (2016) 14.6 A 1.42TOPS/W Deep Convolutional Neural Network. Recognition Processor for Intelligent IoE Systems. In 2016 *IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, 264-265.

[18] Du, Z.D., Fasthuber, R., Chen, T., Ienne, P., Li, L., Luo, T., Feng, X., Chen, Y. and Temam, O. (2015) ShiDianNao: Shifting Vision Processing Closer to the Sensor. *Proceedings of the* 42*nd Annual International Symposium on Computer Architecture in ISCA*'15, Portland, 13-17 June 2015, 92-104. https://doi.org/10.1145/2749469.2750389

[19] Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N. and Temam, O. (2014) DaDianNao: A Machine-Learning Supercomputer. 2014 47*th Annual IEEE/ACM International Symposium on Microarchitecture*, Cambridge, 13-17 December 2014, 609-622. https://doi.org/10.1109/MICRO.2014.58

[20] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R. and Bengio, Y. (2016) Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or −1. ArXiv e-Prints.

[21] Rastegari, M., Ordonez, V., Redmon, J. and Farhadi, A. (2016) XNOR-Net: Image-

Net Classification Using Binary Convolutional Neural Networks. ArXiv e-Prints.

[22] Nakahara, H., Yonekawa, H., Iwamoto, H. and Motomura, M. (2017) A Batch Normalization Free Binarized Convolutional Deep Neural Network on an FPGA (Abstract Only). *Proceedings of the* 2017 *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays in FPGA*'17, Monterey, 22-24 February 2017, 290. https://doi.org/10.1145/3020078.3021782

[23] Andri, R., Cavigelli, L., Rossi, D., Benini, L. and Yoda, N.N. (2016) An Architecture for Ultra-Low Power Binary-Weight CNN Acceleration. ArXiv e-Prints.

[24] Google. Google Supercharges Machine Learning Tasks with TPU Custom Chip. https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html

[25] Microsoft. Project Catapult. https://www.microsoft.com/en-us/research/project/project-catapult/

Scientific Research Publishing

Submit or recommend next manuscript to SCIRP and we will provide best service for you:

Accepting pre-submission inquiries through Email, Facebook, LinkedIn, Twitter, etc.
A wide selection of journals (inclusive of 9 subjects, more than 200 journals)
Providing 24-hour high-quality service
User-friendly online submission system
Fair and swift peer-review system
Efficient typesetting and proofreading procedure
Display of the result of downloads and visits, as well as the number of cited articles
Maximum dissemination of your research work

Submit your manuscript at: http://papersubmission.scirp.org/
Or contact cs@scirp.org