Synthesis of Activation-Parallel Convolution Structures for Neuromorphic Architectures

Seban Kim and Jaeyong Chung
Incheon National University
Incheon, Korea

***Abstract*—Convolutional neural networks have demonstrated continued success in various visual recognition challenges. The convolutional layers are implemented in the activation-serial or fully parallel manner on neuromorphic computing systems. This paper presents an unrolling method that generates parallel structures for the convolutional layers depending on a required level of parallel processing. We analyze the resource requirements for the unrolling of the two-dimensional filters, and propose methods to deal with practical considerations such as stride, borders, and alignment. We apply the propose methods to practical convolutional neural networks including AlexNet and the generated structures are mapped onto a recent neuromorphic computing system. This demonstrates that the proposed methods can improve the performance or reduce the power consumption significantly even without area penalty.**

***Index Terms*—Deep Learning, Deep Neural Networks, Machine Learning, Neuromorphic Engineering**

I. Introduction

Machine intelligence has been improving significantly with growing interest in deep learning. Among others, convolutional deep neural networks have demonstrated astonishing results in various visual recognition challenges, firing the enthusiasm for deep learning. The functionality of machines is now a match for that of humans in some specific applications [1]; however, the energy efficiency of today's computing platforms running deep learning models is not comparable to the human brain. For example, AlphaGo, the state-of-the-art Go program based on deep learning [2], ran on a server cluster with 1920 CPUs and 120 GPUs possibly consuming several hundreds kilowatt, but the professional Go player, Sedol Lee, just consumed about 20W during the match of the century.

The Von Neumann architecture employed in the conventional computing platforms uses a small number of complex, versatile processing elements (PEs) and tries to time-share them as much as possible. In order to improve the performance, we had increased the clock frequency using more pipelining stages but this backfired due to the unmanageable power dissipation [3]. The paradigm has then shifted toward parallel processing, which now becomes the key to energy-efficient high-performance computing. For a higher level of parallel processing, we need to add more PEs on a silicon die. However, the silicon area is still expensive despite technology scaling, and some processors employ lightweight, less versatile PEs to increase the number of PEs. As the number of PEs increases, it becomes difficult to feed data to all the increased number of processing elements since the memory bandwidth is limited [4] and scheduling becomes complex. In this situation, the number of cores is limited up to a few thousands.

The extreme case of parallel processing is found in our brain. Our brain has about 100 billion neurons and over 100 trillion synapses. Synapses serve as memories as well as PEs. Neuromorphic engineering can be performed at many different levels and areas, and we can also mimic how so many PEs operate together. Recent neuromorphic architectures [5], [6] equip with a huge number of low-cost processing elements and place the PEs and memories in proximity of each other. Then, the units and connections of a neural network are mapped into silicon neurons and synapses, respectively. This approach is very different from the traditional architecture, where the computational operations of the neural network are scheduled to PEs.

Convolution is one of the most fundamental operations in signal processing, and can be performed using digital hardware filters. Parallel processing is a common way to improve the throughput of the filters [7]. However, in practice, replicating hardware has not been very popular because of the area overhead [8]. For example, even a single multiplier takes up large area, and thus practical development and research have focused on time-sharing of multipliers. However, in neuromorphic architectures, tiny PEs such as memristors are available, and parallel processing is likely to become practical in the near future.

Convolutional neural networks require a number of two-dimensional (2D) convolution operations. The 2D convolution with a $K \times K$ kernel for an image of $W \times H$ pixels can be reduced to a matrix-vector multiplication by unrolling the kernel. The matrix is roughly of the size $WH \times WH$ and approximately has WHK^2 non-zero elements. This matrix-vector multiplication is performed in the fully parallel manner in most neuromorphic architectures, and will require a huge number of synapses even if the neuromorphic architectures exploit the sparseness well. The huge number of synapses still becomes a burden even if very low-cost PEs are available. In addition, we may also require a huge buffer for the entire image, and the entire inputs may not be available at the same time due to the limited I/O bandwidth. On the other hand, the 2D convolution can also be performed in a word-serial manner as 2D convolution hardware does. In this case, K^2

This research was supported by Basic Science Research Program (NRF-2014R1A1A2A16055253) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education.

J. Chung (E-mail: jychung@inu.ac.kr) is the corresponding author of the paper.

synapses are just needed, but it will take $W \times H$ time steps. Thus, a compromise between the two extremes is needed. There exists strong theoretic foundations for parallel one-dimensional FIR filters [7], but there are fewer discussions for the two-dimensional case. Practical considerations such as border handling and stride in the parallel structures have little been studied in the literature.

In this paper, we propose an algorithm that synthesizes activation-parallel 2D filters for a required level of parallel processing and discuss the practical issues when it is applied to real-world convolutional neural networks. The major contributions of this paper can be summarized as follows:

- We analyze the resource requirements when 2D filters are unrolled.
- We discuss how the practical issues such as stride, border handling, and alignment can be dealt with when we apply unrolling to the filters for convolutional neural networks.
- We apply the propose method to a recent neuromorphic architecture and demonstrate that a neuromorphic computing system can really achieve a surprising throughput at ultra-low power.

II. Related Works

Parallel FIR filters have been extensively studied and it is well known that parallel processing structures can be automatically created by unrolling a given data flow graph [7]. Although the multidimensional synchronous dataflow (MDSDF) model has been proposed in [9], most studies in (multi-rate) signal processing deal with one-dimensional signals and there is little discussion about unrolling a MDSDF graph to our best knowledge. The parallel processing structures obtained by unrolling may be naive, and the hardware cost increases linearly in the level of parallel processing. Fast FIR algorithm (FFA) based polyphase decomposition [8] can reduce the hardware cost significantly, but it is effective when the tap size is large enough. Thus, its use may not be desirable for image filters, where kernel sizes are small usually.

Another common way to perform convolution in parallel is to divide a long signal into pieces and to convolve each in parallel. The convolution of each piece can also be performed via dot product in the time domain or via element-wise multiplication in the frequency domain using the discrete fourier transform (DFT). In order to obtain the linear convolution from the cyclic convolution of the frequency domain, the overlap-add method or the overlap-save method is commonly used. However, approaches of this type do not fit neuromorphic architectures well.

In this paper, we deal with parallel processing structures in two-dimensional cascaded filters and discuss all the practical considerations such as stride, border handling, and alignment. To our best knowledge, there is no well-established theory dealing with all these issues.

III. Preliminaries

The neuron model commonly used in the machine learning community is called the perceptron. In this basic model, a neuron (a.k.a., unit) performs the computation

$$y = f(\sum_{i}^{N} w_i x_i + b) \qquad (1)$$

where y is the *activation* of the neuron, N is the number of inputs, x_i is the input activation, w_i is the weight, b is the bias, and f is the activation function. A neural network is a composition (network) of perceptrons. Convolutional neural networks (CNNs) are multi-layer neural networks, and usually take an image as input. A CNN is made up of two main types of layers: convolutional layers and fully connected layers. Neurons in a convolutional layer are arranged in a three-dimensional space. The dimensions are height and width (the spatial dimensions), and depth. For example, input neurons activated by the pixels of each channel of an input image can be laid out according to vertical and horizontal positions of the pixels and the channels. Each convolutional layer of a CNN transforms the input activations, represented by a 3D volume, into a 3D output volume. Consider a convolutional layer that takes N activation maps of size $H \times W$ and produces M activation maps of size $H' \times W'$. Each input activation map is convolved with a $K_h \times K_w$ kernel and the resulting maps are added up element-wise, yielding an output activation map. This is repeated M times, leading to the M activation maps. Thus, a convolutional layer requires $N \times M$ 2D convolutions. The convolutional layer is depicted in Figure 1. The convolutional layer can be re-arranged in a one-dimensional space, forming a $WH \times W'H'$ layer, and this can be implemented in a neuromorphic architecture directly. However, it can also be converted into a time delay neural network (TDNN) that performs the same task in an activation-serial manner. The time delay neural network is equivalent to a dataflow graph representing a two-dimensional FIR filter. In the TDNN, an activation map is serialized into a stream of activations and it takes $H \times W$ time steps to complete the processing. An example is shown in Figure 2.

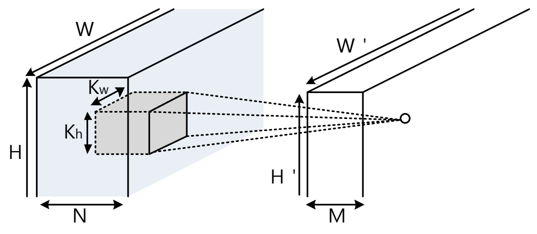


Fig. 1: Neurons in convolutional layers are arranged in a 3D space and have spatially local connections.

IV. Activation-Parallel 2D Filters

A 2D filter with a $K_h \times K_w$ kernel is applied to an input map of $H \times W$ activations. We parallelize the processing of the filter by unrolling so that it processes multiple activations in parallel. A parallel 2D filter processes a $L_h \times L_w$ *block* of the input map in a time step. The level of the parallel processing is defined by the size of the block. The filter is referred as the $L_h \times L_w$-parallel 2D filter. This parallel filter

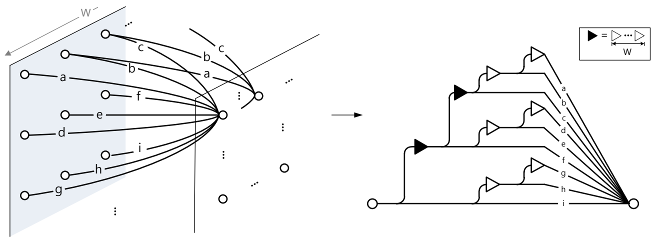


Fig. 2: Convolutional layers are converted into a TDNN performing activation-serial filtering.

will generate a block of $L_h \times L_w$ activations in a time step. Each output activation is generated from a window of $K_h \times K_w$ input activations, which will be referred as the *kernel window*. All the output activations produced in a time step are generated from a window of $G_h \times G_w$ input activations, which will be referred as the *group window*. The group window is the smallest bounding box enclosing all of the kernel windows. Whenever we try to avoid repetition for each dimension, we use K, L, G, etc instead of K_h, K_l, L_h, L_l, G_h, etc. The size of a dimension of the group window is given by

$$G = K + L - 1. \tag{2}$$

Figure 3 depicts the initial position of the group window. Filters are typically modeled as a dataflow graph in signal

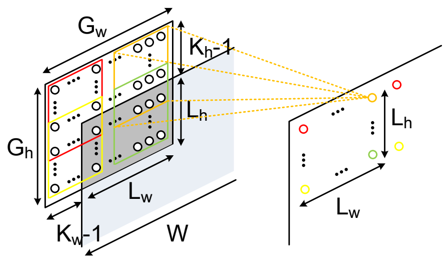


Fig. 3: Activation-parallel filters take $L_h \times L_w$ activations in a time step and generates $L_h \times L_w$ activations at the same time. The other activations in the group window are obtained from delays, a special type of memories.

processing literature. However, we will model them as time delay neural networks since we will use them in neuromorphic architectures. Nonetheless, they can be converted to dataflow graphs very easily for use in other conventional systems. Delays are a type of short-term memory, and a chain of delays performs time-to-space conversion. It allows us to bring past activations in a stream of activations to the present. A *line delay* is a chain of W/L_w delays. We use a line delay (*activation delay*) to bring a previous activation in the vertical (horizontal) direction to the current from an activation stream. To generate the $L_h \times L_w$ activations in a time step, all the activations in the group window should be available at the same time, but we only have the $L_h \times L_w$ activations from the input neurons. The other activations in the group window become available by delays. Figure 4 shows a time delay neural network representing the parallel filter. Let $I(m,n)$ be the activation at the spatial location (m,n). Let $\backslash$ denote

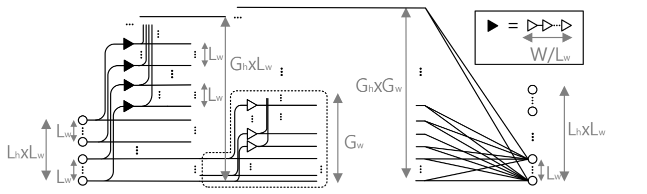


Fig. 4: A time delay neural network performs $L_h \times L_w$-parallel filtering.

the integer division. The time step is denoted by t. Let $i \equiv t\backslash(W/L_w)$ and $j \equiv mod(t, W/L_w)$. Let

$$s_{m,n}(t) \equiv I(L_h i + m, L_w j + n), \tag{3}$$

for $m = 0, ..., L_h - 1$ and $n = 0, ..., L_w - 1$. The $L_h \times L_w$ input neurons take the activations $s_{m,n}(t)$ at the time step t where $m = 0, ..., L_h - 1$ and $n = 0, ..., L_w - 1$. Then, we have

$$s_{-k-1,n}(t) \equiv s_{q,n}(t - pW/L_w) = I\left(L_h(i-p) + q, L_w j + n\right), \tag{4}$$

where $p = k\backslash L_h + 1, q = L_h - 1 - mod(k, L_h)$, for $k = 0, ..., K_h - 1$ and $n = 0, ..., L_w - 1$. The line delays allow us to obtain additional $(K_h - 1) \times L_w$ activations and the window of currently available activations are extended to the top. We also have

$$s_{m,-k-1}(t) \equiv s_{m,q}(t-p) = I(L_h i + m, L_w(j-p) + q), \tag{5}$$

where $p = k\backslash L_w + 1, q = L_w - 1 - mod(k, L_w)$, for $k = 0, ..., K_w - 1$ and $m = -K_h + 1, ..., L_h$. The activation delays bring additional $G_h \times (K_w - 1)$ activations to the current time step and the window is extended to the left. Then, we have $G_h \times G_w$ activations $s_{m,n}(t)$ at the time step t, where $m = -K_h + 1, ..., L_h$ and $n = -K_w + 1, ..., L_w$. Figure 5 depicts how the window of currently available activations is extended in the case of the block size 1×2. The number of the line

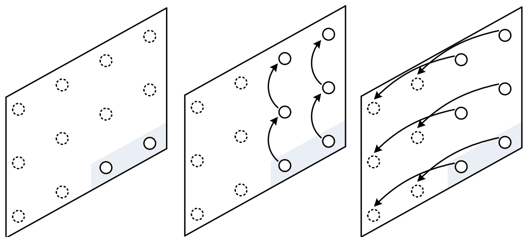


Fig. 5: The window of currently available activations are extended vertically first and then horizontally.

delays added is

$$N_{linedelays} = (K_h - 1)L_w. \tag{6}$$

The number of the activation delays added is

$$N_{actdelays} = (K_w - 1)(K_h + L_h - 1). \tag{7}$$

Then, the number of the total delays becomes

$$\begin{aligned} N_{delays} &= (W/L_w)N_{linedelays} + N_{actdelays} \\ &= W(K_h - 1) + (K_w - 1)(K_h + L_h - 1). \end{aligned} \tag{8}$$

It is interesting to note that the number of the total delays does not depend on L_w.

A. Extension for Stride, Arbitrary Input Size, and Borders

It may be straightforward to unfold a given structure for parallel processing, but there are many non-trival issues in practical implementation of the parallel structure, especially for use in recent convolutional neural networks. For example, we may need a special handling for the case that H (W) is not a multiple of L_h (L_w). In addition, we have assumed that the stride of the convolution is 1. However, non-one stride is commonly used for the first convolution layer and max-pooling in recent convolutional neural networks (CNNs). Besides, 2D filtering is cascaded in CNNs, and the size of the input maps changes from layer to layer. We extend the proposed method to deal with those practical issues. We will generate a parallel structure for a convolution with stride S. This convolution is a stage in cascaded filters. For simplicity, we assume that $K \geq S$. We also suppose that L is a multiple of S if $L \geq S$ and S is a multiple of L otherwise. We will discuss the general case in the detail in the next subsection. When stride is not 1, the output block size becomes different from the input block size. Let $L'_h \times L'_w$ denote the output block size. Considering stride, the size of a dimension of the group window becomes

$$G = K + S(L' - 1). \tag{9}$$

The 2D filter with stride S downsamples by a factor of S, so we have

$$L' = \begin{cases} L/S & \text{if } L \geq S \\ 1 & \text{if } L < S. \end{cases} \tag{10}$$

We now extend the proposed method to handle the input of arbitrary size and the borders. For that, we introduce the *bleed*, which is the area to be trimmed out. We suppose that the input and the output of the parallel filter include the bleed. For the bleed, the values of the top B rows and the left B columns of the input map are meaningless. These values may be intentionally padded to make the input size a multiple of the block size, or may have been invalidated for a convolution in a previous stage. For the cascaded convolutions, we propagate the bleed without cropping it right away and it is trimmed out when $L' = 1$. When we handle the border with cropping, the output bleed becomes

$$B' = (B + K - 1)/S \tag{11}$$

if $L' > 1$. If $L' = 1$, $B' = 0$. Since the border handling affects the bleed, not the size of the output, the size of the input maps change only due to stride. Since the output size is simply W/S for the bleed, the output size becomes a multiple of L'. This means that if we make the input size for the first convolution a multiple of the block size by using the bleed, the alignment of the block is automatically taken care of in the subsequent convolutions.

When stride is not one, the positions of the group window depend on several parameters. We consider the group window of stride 1 even when we deal with non-one stride. We call it the *pan-group window*. The group window is displaced by $mod(B+K-1, \min(S, L))$ to the right owing to the bleed and the border handling, and the group window is smaller than the pan-group window by $min(S, L) - 1$. Thus, the right-bottom corner of the group window is away from that of the pan-group window by the margin

$$M = \min(S, L) - 1 - mod(B + K - 1, \min(S, L)) \tag{12}$$

as shown in Figure 6.

TABLE I: The number of the computational resources required depends on the output block size, not the input block size.

Impl. Type	Resource Type	Number of Resources
Conventional	$N_{multipliers}$	$K_h K_w L'_h L'_w$
	N_{adders}	$(K_h K_w - 1) L'_h L'_w$
	$N_{registers}$	$K_h K_w$
Neuromorphic	$N_{synapses}$	$K_h K_w L'_h L'_w$
	N_{neuron}	$L'_h L'_w$

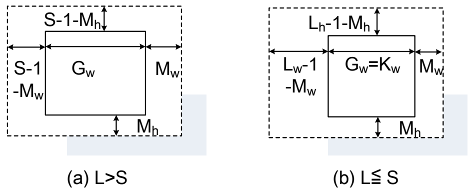


Fig. 6: The dotted box indicates the pan-group window. The group window is a smaller window within the pan-group window.

Some of the bleed may be replaced by zeros for zero padding and in order to pad zeros at the right and the bottom we can also decrease the left (top) bleed and increase the right (bottom) bleed by adding activation (line) delays. The time steps for the bottom bleed and some of the right bleed are borrowed from those for the next activation map.

The overall algorithm is summarized in Algorithm 1. The size of the window of currently available activations is $L_h \times L_w$ initially (line 3). We extend this window vertically to the top using line delays until the height of the window becomes $G_h + M_h$ (line 4-6). Then, we extend it horizontally to the left using activation delays until the width becomes $G_w + M_w$ (line 8-9).

The number of the line delays added is

$$N_{linedelays} = (G_h + M_h - L_h) L_w. \tag{13}$$

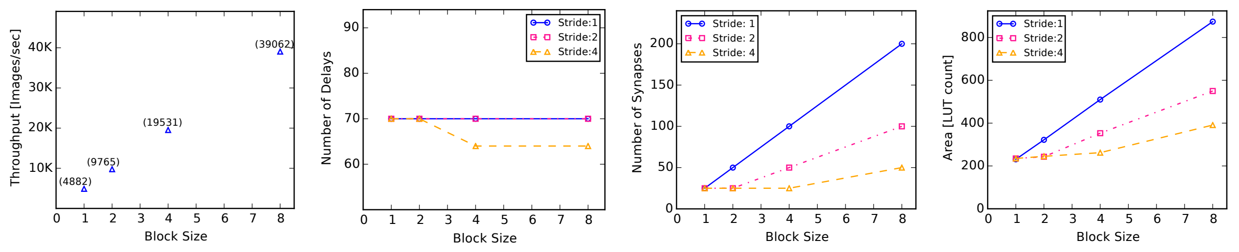
The number of the activation delays added is

$$N_{actdelays} = (G_w + M_w - L_w) G_h. \tag{14}$$

Once the activations in the group window are available at the same time by the structure generated by the algorithm, each output is generated by using the activations in its kernel window. In neuromorphic architectures, each output is generated by a neuron connected to $K_h K_w$ synapses, so it requires $K_h K_w L'_h L'_w$ synapses and $L'_h L'_w$ neurons in total. For the remaining structure, the number of the resources required in conventional and neuromorphic implementations are summarized in Table I.

B. Non-Integer Output Block Size

If the block size is not a multiple of the stride, or the stride is not a multiple of the block size, we can increase the output



(a) (b) (c) (d)

Fig. 7: Parallel filters improve the performance at the cost of circuit area. However, if stride is not one, it can improve the performance without any cost by increasing the utilization.

Algorithm 1 GenerateDelayStructure

```
Input: $L_h \times L_w$ input nodes
 1: Let $F$ be the two-dimensional array of nodes
 2: Let $N_h$ and $N_w$ be the height and the width of $F$
 3: Initialize $N$ with $L_h \times L_w$ input nodes
 4: while $N_h < G_h + M_h$ do
 5:     Expand $F$ vertically to the top by adding line delays
 6: end while
 7: while $N_w < G_w + M_w$ do
 8:     Expand $F$ horizontally to the left by adding activation delays
        for $G_h$ rows
 9: end while
10: Remove $M_h$ rows in $N$ from the bottom
11: return $F$
```

block size by setting $L' = lcm(L, S)/S$ and decrease the sampling rate instead. However, in this case, the input size should also be a multiple of $lcm(L, S)$ for the output to be a multiple of L'. If we encounter this case in a middle of the cascaded convolutions, and the input size is not such a multiple, then we can pad more bleed into the initial input, or remove some of the bleed in a previous stage.

V. Experimental Results

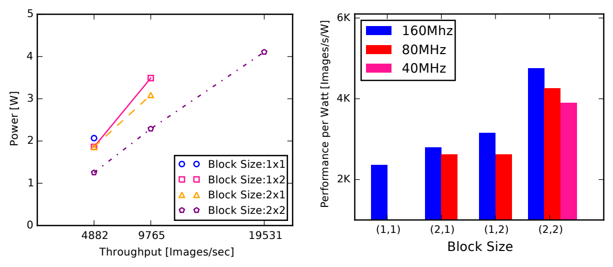
We have implemented the proposed algorithm in Python. It takes a convolutional neural network and generates a time delay neural network (TDNN). We adopt the neuromorphic architecture proposed in [6]. Each unit (connection) in the TDNN is implemented into a hardware neuron (synapse) based on a bit-serial adder (multiplier). Each delay in the TDNN is also implemented in a hardware delay based on a shift-register. Then, there is an one-to-one mapping between the TDNN and the processing elements, and the TDNN is converted into a huge feedforward logic circuit described in Verilog where constants are hard-coded. The circuit is synthesized using Xilinx Vivado and is mapped into Xilinx Virtex 7 2000T FPGA. No DSP slice and block RAM are used.

First, we randomly generate a 2D filter with a 3×3 kernel for a map of 32×32 activations, and we set $L_h = 1$ and $S_h = 1$ and vary L_w and S_w. In the neuromorphic architecture, there is no other performance bottleneck such as off-chip memory access and on-chip array access, and the actual throughput increases as the block size increases as shown in Figure 7(a). Figure 7(b) shows the number of delays when the block size (i.e., the level of parallel processing) varies. The block size has almost no impact on the number of delays, but depending on the margin of the group window, the number of delays can even be reduced. Figure 7(c) shows the number of synapses as the block size increases. As long as the stride is smaller than or equal to the block size, the number of synapses does not change. Thus, we can obtain almost free performance improvements until then. This is possible because the synapses are under-utilized when the block size is smaller than the stride. Figure 7(d) shows the synthesis results of the filters with different block sizes. The figure is very well correlated with Figure 7(c) because the synapses occupy most area of the circuit.

Next, we use the 6-layer convolutional neural network described in Table II for the CIFAR-10 task. This network achieves 84.43% classification accuracy and is simplified to a sparse network with 18K parameters using the method of [10] and the accuracy becomes 80.96%. After quantizing the

TABLE II: When the 1×1 block size meets stride $S > 1$, the utilization decreases $1/S^2$.

Layer Type	Min-Area Design [6]		2×2-Parallel Design			
	Input Size	Utilization	Block Size	Input Size	Util.	Bleed
3×3 conv	32	1	2×2	32	1	0
3×3 conv	30	1	2×2	32	1	2
4×4 pool-s2	28	1	2×2	32	1	4
3×3 conv	13	0.25	1×1	13	1	0
3×3 conv-p1	11	0.25	1×1	11	1	0
4×4 pool-s2	11	0.25	1×1	11	1	0
fc	4	0.0625	1×1	4	0.25	0
fc	1	0.0625	1×1	1	0.25	0



(a) (b)

Fig. 8: The energy-efficiency increases as the block size increases. The high-frequency designs show higher efficiency since lowing the operating frequency scales the dynamic power only.

TABLE III: A time delay neural network is converted into a logic circuit and it is implemented in two different flavors. The convolutional layers of AlexNet are executed at unprecedented energy-efficiency.

Datasets	Block Size	TDNN Model					Maximum Performance Design					Minimum Power Design				
		Delay			Neuron	Synapse	Area	Images	Power (W)			Area	Images	Power (W)		
		Line	Act.	Total			(LUTs)	/sec	Dyna.	Static	Total	(LUTs)	/sec	Dyna.	Static	Total
CIFAR 10	1×1	846	4366	18973	4091	16151	109169	4882	1.4	0.668	2.068	109169	4882	1.4	0.668	2.068
	1×2	1080	4114	18721	5412	19443	131899	9764	2.4	0.689	3.089	135112	4882	1.2	0.661	1.861
	2×2	954	4160	16751	8052	26027	169462	19528	3.4	0.707	4.107	169759	4882	0.6	0.653	1.253
ILVSRC 2012	1×1	2207	29221	87678	17470	90110	495232	97	4.4	1.007	5.407	495232	97	4.4	1.007	5.407
	1×2	2237	23531	82018	17688	90118	496217	194	4.4	1.003	5.403	496098	97	2.2	0.791	2.991
	2×2	2231	23531	81334	17694	90134	494769	388	5	1.008	6.008	494704	97	1	0.713	1.713
	4×4	2261	23465	79900	17606	90230	494777	1552	6.2	1.021	7.221	494630	97	0.2	0.663	0.863

weights to 5 bits, it becomes 80.61%. Then, we convert the convolutional layers into the activation-serial (baseline) and the proposed activation-parallel structures varying the block size and the whole network turns into a TDNN. Table II shows two different designs for the 6-layer convolutional neural network and the min-area, baseline design is the one proposed in [6]. In the layer type, 's' stands for stride and 'p' means zero padding. All the layers have stride 1 and no zero padding unless otherwise stated. In the min-area design, the hardware utilization decreases $1/S^2$ for stride S. Considering that, if we set the block size of the first layer to the sum of the non-one strides in the net, we can even make a full-utilization design. In the activation-parallel design, the input for each layer includes the bleed and the input size does not change due to the border handling. The block size, the input size and the bleed decrease when non-one stride is encountered as in the third layer. When the output block size becomes one, the bleed is trimmed out.

Table III shows the statistics of the TDNNs and the FPGA implementation results. For the 1×2 design, the number of the line delays increase, but the number of delays in a line delay decreases in the parallel structures. As the block size increases, the total number of delays decreases and it can happen when we generate the parallel structures for non-one stride. The dynamic power is measured at 220V AC and the efficiency of the AC-DC conversion is around 70%. All the dynamic power measurements include the inefficiency of on-board regulators as well. The static power is obtained from the power reports of Vivado. When the block size increases, we can increase the throughput by running the designs at the same frequency (Max. Performance Designs). Or, we can reduce the power consumption while maintaining the performance by decreasing the clock frequency. (Min. Power Designs). The baseline design is under-utilized because of pooling as shown in Table II and increasing the block size improves the utilization, and hence the energy-efficiency. As the block size increases, the number of LUTs increases slowly because only the layers earlier than the first pooling are parallelized. Figure 8 shows the energy-efficiency of the CIFAR-10 designs.

To validate the proposed method further, we use the convolutional layers of AlexNet [11] for the ImageNet large-scale natural image classification task. We use AlexNet without the local response normalization layers, and the baseline model achieves 52.46% top-1 accuracy. To use this network in a neuromorphic architecture, we simplify the convolutional layers until it has 100K parameters and the accuracy becomes 50.36%. After quantizing the weights to 5 bits, the accuracy becomes 49.23%. Table III shows the implementation results for various block sizes. In AlexNet, the first convolutional layer has stride of 4 and we can increase the performance or decrease the power consumption with almost no area penalty, achieving unprecedented energy efficiency using off-the-shelf FPGAs.

VI. Conclusion

We have proposed the methods to synthesize parallel structures for the convolutional layers of recent convolutional neural networks. We have dealt with practical issues such as border handling and alignment by introducing the bleed. The neuromorphic computing system based on activation-serial structures already demonstrated high energy-efficiency and performance, but the activation-parallel structures have allowed us to show that the nearly bottleneck-free architecture can have unparalleled efficiency and performance.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[2] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[3] Jeff Parkhurst, John Darringer, and Bill Grundmann. From single core to multi-core: preparing for a new exponential. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 67–72. ACM, 2006.

[4] Doug Burger, James R Goodman, and Alain Kägi. *Memory bandwidth limitations of future microprocessors*, volume 24. ACM, 1996.

[5] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[6] Jaeyong Chung, Taehwan Shin, and Yongshin Kang. Insight: A neuromorphic computing system for evaluation of large neural networks. *arXiv preprint arXiv:1508.01008*, 2015.

[7] Keshab K Parhi. *VLSI digital signal processing systems: design and implementation*. John Wiley & Sons, 2007.

[8] Yu-Chi Tsao and Ken Choi. Area-efficient vlsi implementation for parallel linear-phase fir digital filters of odd length based on fast fir algorithm. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(6):371–375, 2012.

[9] Praveen K Murthy and Edward A Lee. Multidimensional synchronous dataflow. *IEEE Transactions on Signal Processing*, 50(8):2064–2079, 2002.

[10] Jaeyong Chung and Taehwan Shin. Simplifying deep neural networks for neuromorphic architectures. In *Proceedings of the 53rd Annual Design Automation Conference*, page 126. ACM, 2016.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.